

USER GUIDE

# Essential Studio for Blazor

---

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

Diagram.....	45
Getting Started with Blazor Diagram Component .....	45
Importing Syncfusion Blazor component in the application.....	45
Adding component package to the application.....	45
Add SyncfusionBlazor service in Startup.cs .....	45
Adding Diagram component to the Application .....	46
Adding Nodes and Connectors.....	46
See Also .....	50
Nodes .....	51
Actions of nodes in Blazor Diagram Component .....	51
Positioning in Blazor Diagram Component .....	58
Appearance in Blazor Diagram Component.....	60
Interaction in Blazor Diagram Component .....	73
Events in Blazor Diagram Component .....	79
Shapes in Blazor Diagram Component.....	82
Text .....	82
Image.....	83
Image alignment .....	85
HTML.....	87
Basic shapes .....	88
Path .....	89
Flow Shapes .....	90
Bpmn Shapes.....	91
BPMN Shapes in Blazor Diagram Component .....	91
BPMN Event in Blazor Diagram Component.....	92
BPMN Gateway in Blazor Diagram Component.....	95
BPMN Activity in Blazor Diagram Component.....	97
BPMN Data Object in Blazor Diagram Component.....	111
BPMN Datasource in Blazor Diagram Component.....	113
BPMN Text Annotation in Blazor Diagram Component.....	114
BPMN Group in Blazor Diagram Component.....	115
BPMN Connectors in Blazor Diagram Component .....	116
UML Diagram Shapes in Blazor Diagram Component .....	120
UML Class Diagram Shapes .....	120
Class .....	120

Interface .....	121
Enumeration .....	122
Connector shapes .....	123
Relationships.....	123
Association .....	124
Aggregation.....	125
Composition .....	125
Dependency .....	126
Inheritance .....	127
Multiplicity .....	127
UmlActivity diagram.....	129
Connectors.....	133
Actions of Connectors in Blazor Diagram Component .....	133
Segments in Blazor Diagram Component .....	146
Customization in Blazor Diagram Component.....	153
Interaction in Blazor Diagram Component .....	165
Events in Blazor Diagram Component .....	174
Group in Blazor Diagram Component.....	176
Create group .....	177
Add group when initializing diagram .....	177
Add group at runtime .....	179
Group from SymbolPalette .....	180
Update position at runtime .....	182
Appearance .....	184
Interaction.....	184
See Also .....	184
Annotations.....	184
Actions of annotation in Blazor Diagram Component .....	184
Annotation for node in Blazor Diagram Component .....	189
Annotation for Connector in Blazor Diagram Component .....	196
Appearance in Blazor Diagram Component.....	203
Interaction in Blazor Diagram Component .....	217
Events in Blazor Diagram Component .....	224
Ports .....	226
Actions of port in Blazor Diagram Component .....	226

Positioning in Blazor Diagram Component .....	233
Appearance in Blazor Diagram Component.....	240
Interaction in Blazor Diagram Component .....	243
Constraints in Blazor Diagram Component.....	246
Diagram constraints .....	246
Node constraints .....	247
Connector constraints.....	250
Port constraints.....	252
Annotation constraints .....	253
Selector constraints .....	254
Snap constraints.....	256
Boundary constraints.....	257
Inherit behaviors.....	258
Bitwise operations .....	259
Add operation .....	259
Remove Operation .....	260
Check operation .....	260
Interaction in Blazor Diagram Component .....	260
Selection.....	260
Single selection .....	260
Selecting a group.....	261
Multiple selection .....	261
Select/Unselect elements using program .....	262
Select entire elements in diagram programmatically.....	262
Drag.....	262
Resize .....	263
Rotate.....	263
Connection editing.....	264
End point handles .....	264
Straight segment editing.....	264
Orthogonal thumbs.....	265
Drag and drop nodes over other elements.....	265
User handles .....	265
Alignment.....	266
Offset.....	266



Side.....	266
Horizontal and vertical alignments .....	266
Margin.....	266
Notification for the mouse button clicked.....	266
Appearance .....	267
Zoom pan .....	267
Keyboard .....	268
See Also .....	269
Tools in Blazor Diagram Component .....	269
Drawing tools .....	269
Shapes .....	269
Connectors .....	270
Text .....	271
Polygon shape .....	272
Tool selection .....	273
Gridlines in Blazor Diagram Component.....	275
Customize the gridlines visibility.....	275
Appearance .....	275
Line intervals.....	276
Snapping.....	277
Page Settings in Blazor Diagram Component .....	279
Page size and appearance .....	279
Set background image.....	280
Multiple page and page breaks.....	281
Boundary constraints.....	282
Scroll Settings in Blazor Diagram Component .....	283
Get current scroll status.....	283
Define scroll status.....	283
Update scroll status .....	283
Autoscroll .....	284
Autoscroll border .....	284
Scroll limit .....	284
Scroll Padding.....	285
Scrollable Area .....	285
UpdateViewport.....	286

Data Binding in Blazor Diagram Component .....	286
Local data .....	286
See Also .....	288
Layout.....	288
Automatic Layout in Blazor Diagram Component .....	288
Hierarchical Layout in Blazor Diagram Component .....	289
Organizational Chart in Blazor Diagram Component.....	296
Mind Map layout in Blazor Diagram Component .....	310
Radial Tree Layout in Blazor Diagram Component .....	312
Symmetric layout in Blazor Diagram Component.....	315
Line Distribution in Blazor Diagram Component .....	319
Linear Arrangement in Blazor Diagram Component.....	321
Accessibility in Blazor Diagram Component .....	322
Commands in Blazor Diagram Component.....	323
Alignment commands .....	323
Distribute .....	324
Sizing Commands .....	325
Clipboard.....	325
Grouping .....	326
Z-Order command.....	326
BringToFront command .....	326
SendToBack command.....	327
MoveForward command .....	327
SendBackward command.....	327
Zoom .....	328
Nudge command.....	328
Nudge by using arrow keys .....	329
BringIntoView .....	329
BringToCenter .....	329
FitToPage command .....	330
Undo and Redo command .....	330
Command manager.....	331
See Also .....	334
Undo Redo in Blazor Diagram Component.....	334
Undo and redo .....	334

Undo/redo through shortcut keys .....	334
Undo/redo through public APIs .....	334
History change event .....	337
Events.....	337
Created.....	337
Clicked .....	337
ContextMenuitemClicked .....	338
OnContextMenuOpen.....	338
DataLoaded .....	339
OnDoubleClick.....	339
DragEnter .....	340
DragLeave .....	340
OnDrop.....	341
HistoryChanged.....	341
MouseEnter.....	342
MouseLeave .....	342
OnPositionChange.....	343
PropertyChanged .....	343
OnRotateChange .....	344
SelectionChanged .....	344
OnSizeChange .....	345
TextEdited .....	345
Native events .....	346
Virtualization in Blazor Diagram Component .....	347
Virtualization in Diagram .....	347
Serialization in Blazor Diagram Component .....	347
Save .....	347
Load.....	348
Prevent Default Values .....	348
Printing and Exporting in Blazor Diagram Component .....	348
Exporting options .....	349
File Name .....	349
Format.....	349
Margin.....	349
Mode .....	350

Region .....	350
Custom bounds .....	351
Export diagram with stretch option .....	352
Print.....	352
Tooltip in Blazor Diagram Component.....	353
Default tooltip.....	353
Common tooltip for all nodes and connectors .....	354
Tooltip for a specific node/connector.....	355
Tooltip alignments .....	356
Tooltip animation.....	358
Layers in Blazor Diagram Component.....	359
Visible.....	360
Lock .....	361
Objects .....	362
AddInfo.....	363
Context Menu in Blazor Diagram Component.....	367
Default context menu .....	367
Custom context menu.....	368
Events.....	372
Ruler in Blazor Diagram Component .....	376
Adding Rulers to the Diagram.....	376
Customizing the Ruler .....	377
User Handles in Blazor Diagram Component.....	379
Initialization an userhandle.....	379
Customization .....	381
Events.....	386
Fixed user handles .....	388
Initialization an fixed user handles .....	388
Customization the fixed user handle .....	388
Customizing the node fixed user handle .....	391
Customizing the connector fixed user handle .....	393
FixedUserHandle Event.....	396
CSS Structure in Blazor Diagram Component .....	397
Customizing the connector end point handle.....	397
Customizing the connector end point handle when connected.....	397

Customizing the connector end point handle when disabled .....	397
Customizing the bezier connector handle .....	398
Customizing the bezier connector line .....	398
Customizing the resize handle .....	398
Customizing the selector pivot line.....	398
Customizing the selector border.....	398
Customizing the rotate handle .....	399
Customizing the symbolpalette while hovering .....	399
Customizing the symbolpalette when selected .....	399
Customizing the ruler.....	399
Customizing the ruler overlap.....	399
Customizing the text edit.....	399
Customizing the text edit on selection .....	400
Symbol Palette in Blazor Diagram Component.....	400
Create symbol palette.....	400
Add palettes to SymbolPalette .....	400
Customize the palette header .....	403
Stretch the symbols into the palette .....	403
Add/Remove symbols to palette at runtime .....	404
Customize the size of symbols .....	404
Symbol preview.....	405
Default settings .....	406
Adding symbol description for symbols in the palette .....	408
Palette interaction .....	410
Escape Key function .....	410
See Also .....	410
Overview Control in Blazor Diagram Component.....	410
Create overview .....	410
Zoom and Pan .....	413
Diagram Methods in Blazor Diagram Component.....	414
Nodes .....	414
Connectors .....	423
Annotations.....	427
Ports .....	431
Print and exporting .....	433

Save and load the diagram.....	435
Layers .....	436
Layout.....	447
Sizing Commands .....	447
Alignment commands .....	450
Distribution commands.....	452
Clipboard commands .....	453
Grouping commands.....	455
Order commands .....	457
Interaction.....	459
View Port.....	469
History .....	479
Tool tip .....	483
Property change.....	484
How To .....	484
Styling And Appearance in Blazor Diagram Component.....	484
Blazor WebAssembly Diagram in Blazor Diagram Component.....	485
Diagram Component.....	493
Getting Started in Blazor Diagram Component .....	493
Importing Syncfusion Blazor component in the application.....	493
Adding component package to the application .....	493
Add SyncfusionBlazor service in Startup.cs .....	493
Adding Diagram component to the Application .....	494
Adding Nodes and Connectors.....	494
See Also .....	497
Nodes .....	497
Node in Blazor Diagram Component .....	497
Appearance of a Node in Blazor Diagram Component .....	503
Node Interaction in Blazor Diagram Component.....	514
Events and Constraints in Blazor Diagram Component .....	521
Positioning a node in Blazor Diagram Component .....	526
Node Shapes in Blazor Diagram Component .....	530
Image node .....	530
HTML template shape.....	535
Basic shapes .....	537

Path shape.....	538
Flow Shapes .....	540
SVG template shape.....	541
Connectors.....	542
Connector in Blazor Diagram Component .....	542
Segments in Blazor Diagram Component .....	553
Customization in Blazor Diagram Component.....	563
Interaction in Blazor Diagram Component .....	576
Events in Blazor Diagram Component .....	581
NodeGroup in Blazor Diagram Component .....	586
Create NodeGroup .....	586
Add NodeGroup when initializing diagram.....	586
Add NodeGroup at runtime .....	589
Update position at runtime .....	592
Appearance .....	593
Interaction.....	593
See Also .....	593
Annotations.....	593
Annotation in Blazor Diagram Component.....	593
How to position node's annotation .....	598
How to position connector's annotation.....	605
Appearance in Blazor Diagram Component.....	610
Events in Blazor Diagram Component .....	622
Ports .....	623
Ports in Blazor Diagram Component.....	623
How to position node's port .....	632
Port appearance and positioning.....	641
Interaction in Blazor Diagram Component .....	644
Constraints in Blazor Diagram Component.....	646
Diagram constraints .....	646
Node constraints .....	647
Connector constraints.....	650
Port constraints.....	652
Annotation constraints .....	653
Selector constraints .....	654

Snap constraints.....	656
Boundary constraints.....	658
Inherit behaviors.....	659
Bitwise operations.....	660
Add operation.....	660
Remove Operation.....	660
Check operation.....	660
Interaction in Blazor Diagram Component.....	661
Selection.....	661
Single selection.....	661
Selecting a group.....	662
Multiple selection.....	662
Select/Unselect elements using program.....	663
Select entire elements in diagram programmatically.....	663
Drag.....	663
Resize.....	664
Rotate.....	665
Connection editing.....	666
End point handles.....	667
Straight segment editing.....	668
Orthogonal segment editing.....	668
User handles.....	670
Alignment.....	670
Offset.....	670
Side.....	670
Horizontal and vertical alignments.....	670
Margin.....	670
Notification for the mouse button clicked.....	670
Appearance.....	671
Zoom pan.....	671
Keyboard.....	672
See Also.....	673
Tools in Blazor Diagram Component.....	673
Drawing tools.....	673
Shapes.....	673



Connectors .....	675
Polygon shape .....	677
Tool selection .....	679
Gridlines in Blazor Diagram Component.....	681
Customize the Gridlines visibility .....	681
Appearance .....	682
How to create dot grid patterns .....	683
Line intervals.....	683
Snapping.....	684
Data Binding in Blazor Diagram Component .....	688
Local data .....	689
JSON Data.....	691
Remote Data .....	692
See Also .....	694
Layout.....	694
Automatic Layout in Blazor Diagram Component .....	694
Hierarchical layout in Blazor Diagram Component.....	695
Organizational Chart in Blazor Diagram Component.....	698
Mind map Layout in Blazor Diagram Component.....	709
Commands in Blazor Diagram Component.....	712
Alignment commands .....	713
Distribute .....	719
Sizing Commands .....	722
Clipboard.....	723
Grouping .....	724
Zoom .....	725
Nudge command.....	726
Nudge by using arrow keys .....	727
Undo and Redo command .....	727
Command manager.....	727
Undo Redo support in Blazor Diagram Component .....	731
Undo and redo .....	731
Undo/redo through shortcut keys .....	731
Undo/redo through public APIs .....	731
History change event .....	732

Track custom entry .....	732
User Handles for node, connector in Blazor Diagram Component .....	735
Initialization an user handle.....	735
Customization .....	737
Fixed user handles .....	741
Initialization an fixed user handles .....	741
Customization the fixed user handle .....	742
Customizing the node fixed user handle .....	744
Customizing the connector fixed user handle .....	747
FixedUserHandle Event.....	751
Symbol Palette in Blazor Diagram Component.....	752
Create symbol palette.....	752
Add palette to SymbolPalette.....	758
How to drag and drop symbols from palette to diagram .....	762
Customize the palette header .....	765
Add/Remove symbols to palette at runtime .....	766
Add/Remove palettes at runtime .....	766
Customize the size of symbols .....	767
SymbolDragPreviewSize.....	769
Default settings .....	771
Adding symbol description for symbols in the palette .....	771
Palette interaction .....	773
Escape Key function .....	773
See Also .....	773
Dialog .....	773
Getting Started with Blazor Dialog Component .....	773
Importing Syncfusion Blazor component in the application.....	773
Adding component package to the application .....	774
Add SyncfusionBlazor service in Startup.cs .....	774
Add Dialog component .....	775
Run the application .....	775
Modal dialog .....	775
Enable header .....	776
Render Dialog with buttons .....	777
See Also .....	778

Draggable in Blazor Dialog Component .....	778
Resizing in Blazor Dialog Component .....	779
Positioning in Blazor Dialog Component.....	780
Templates in Blazor Dialog Component.....	783
Header.....	783
Footer.....	783
See Also .....	787
Animation in Blazor Dialog Component.....	787
Style and appearance in Blazor Dialog Component.....	788
Customizing the dialog header .....	788
Customizing the dialog content .....	788
Customizing modal dialog overlay .....	788
Customizing the dialog resize icon.....	789
Customizing the dialog close button.....	789
Customizing the dialog footer button.....	789
Localization in Blazor Dialog Component .....	789
Accessibility in Blazor Dialog Component.....	790
Keyboard interaction .....	791
See Also .....	792
Events in Blazor Dialog Component.....	792
Created.....	792
Opened.....	793
OnOpen .....	793
Closed.....	793
OnClose .....	794
OnDragStart .....	794
OnDragStop.....	794
OnDrag .....	795
OnOverlayModalClick .....	795
OnResizeStart.....	795
Resizing .....	795
OnResizeStop .....	796
Destroyed.....	796
How To .....	796
Create nested Dialog in Blazor Dialog Component.....	796

Two-way binding using the visible property in Blazor Dialog Component .....	798
Position the Blazor Dialog in center of the page on scrolling .....	799
Render a dialog header dynamically in Blazor Dialog Component .....	800
Show Dialog with fullscreen in Blazor Dialog Component .....	801
Display a Dialog with custom position in Blazor Dialog Component .....	802
Prevent closing of modal Dialog in Blazor Dialog Component .....	802
Prevent the focus on the first element in Blazor Dialog Component .....	805
Open a Dialog on condition in Blazor Dialog Component .....	806
Customize the appearance in Blazor Dialog Component .....	808
Close Blazor Dialog Component when clicking outside of its region .....	810
Add an icons to Dialog buttons in Blazor Dialog Component .....	811
DocumentEditor .....	813
Getting Started .....	813
Features in Blazor DocumentEditor Component .....	813
Blazor DocumentEditor Component in Server Side App .....	814
Blazor DocumentEditor Component in WebAssembly App using Visual Studio .....	821
Opening a document in Blazor DocumentEditor Component .....	822
Opening a document from URL .....	822
Opening a document from Cloud .....	823
Opening a document from database .....	826
Opening a document from file system .....	827
Opening a document on control initialization .....	827
Saving document in Blazor DocumentEditor Component .....	828
Save document to server .....	828
Save document to database .....	829
Download document as a copy .....	830
Clipboard in Blazor DocumentEditor Component .....	830
Copy .....	830
Cut .....	830
Paste .....	831
Local paste .....	831
Undo and redo in Blazor DocumentEditor Component .....	831
Enable or disable history .....	831
Undo and redo .....	832
Stack size .....	832

Images in Blazor DocumentEditor Component .....	832
Image resizing .....	833
Changing size.....	833
Tables in Blazor DocumentEditor Component.....	833
Create a table.....	833
Insert rows .....	833
Delete table.....	834
Delete row.....	835
Delete column.....	835
Merge cells.....	835
Table of contents in Blazor DocumentEditor Component.....	835
Inserting table of contents.....	835
Update or edit table of contents .....	838
Headers and Footers in Blazor DocumentEditor Component .....	838
Go to header footer region.....	839
Header and footer distance .....	839
Close header footer region .....	839
Working with Text Formatting in Blazor DocumentEditor Component .....	840
Bold .....	840
Italic.....	840
Underline property .....	840
Strikethrough property .....	840
Superscript property .....	841
Subscript property .....	841
Size .....	841
Color.....	841
Font .....	842
Highlight color.....	842
Working with Paragraph Formatting in Blazor DocumentEditor Component.....	842
Indentation.....	842
Special indentation .....	842
Increase indent .....	842
Decrease indent .....	842
Text alignment .....	843
Line spacing and its type .....	843

Paragraph spacing.....	843
Working with Styles in Blazor DocumentEditor Component.....	843
Styles definition overview.....	843
Default style .....	844
Style hierarchy .....	844
Defining new styles .....	845
Defining a character style .....	845
Defining a paragraph style .....	845
Defining a linked style .....	846
Applying a style .....	846
Working with Lists in Blazor DocumentEditor Component .....	847
Create bullet list.....	847
Create numbered list .....	847
Clear list.....	847
Editing numbered list.....	847
Working with Table Formatting in Blazor DocumentEditor Component.....	848
Cell margins.....	848
Background color .....	849
Cell spacing.....	849
Cell vertical alignment.....	849
Table alignment .....	849
Cell width .....	849
Table width .....	850
Working with row formatting .....	850
Row height .....	850
Header row .....	851
Allow row break across pages.....	851
Working with Section Formatting in Blazor DocumentEditor Component .....	851
Page size.....	851
Page margins.....	851
Header distance .....	851
Footer distance .....	852
Find and Replace in Blazor DocumentEditor Component .....	852
Show or hide navigation pane .....	852
Search.....	852

Accessibility in Blazor DocumentEditor Component .....	853
Keyboard Shortcuts.....	853
Restrict editing in Blazor DocumentEditor Component .....	856
Read only.....	856
Allows changes to certain portion of the document .....	856
Dropdown Menu.....	857
Getting Started with Blazor Dropdown Menu Component.....	857
Importing Syncfusion Blazor component in the application.....	858
Adding component package to the application.....	858
Add SyncfusionBlazor service in Startup.cs .....	858
Adding component package to the application.....	859
Adding Dropdown Menu component to the application .....	859
Run the application .....	859
See Also .....	860
Events in Blazor Dropdown Menu Component .....	860
List of events supported .....	860
How to bind event to Dropdown Menu.....	860
Icons in Blazor Dropdown Menu Component.....	861
Dropdown Menu icons.....	861
See Also .....	863
Popup Items in Blazor Dropdown Menu Component.....	863
Icons.....	863
Separator.....	864
Navigations .....	865
Template .....	866
Accessibility in Blazor Dropdown Menu Component .....	866
ARIA attributes.....	866
Keyboard interaction .....	867
Styles and Appearances in Blazor Dropdown Menu Component.....	868
How To .....	868
Change caret icon in Blazor Dropdown Menu Component .....	868
Create Blazor Dropdown Menu with Rounded Corner.....	869
Create right-to-left Blazor Dropdown Menu Component .....	870
Customize icon and width in Blazor Dropdown Menu Component .....	871
Disable a Dropdown Menu in Blazor Dropdown Menu Component.....	872

Group dropdown listview items in Blazor Dropdown Menu Component .....	872
Hide dropdown arrow in Blazor Dropdown Menu Component .....	873
Open a dialog on popup item click in Blazor Dropdown Menu Component .....	874
Position popup open in Blazor Dropdown Menu Component .....	875
Create Dropdown List in Popup of Blazor Dropdown Menu Component .....	876
Add and Remove Items in Blazor Dropdown Menu Component .....	877
DropDown List.....	879
Getting Started with Blazor DropDown List Component.....	879
Importing Syncfusion Blazor component in the application.....	879
Adding component package to the application.....	879
Add SyncfusionBlazor service in Startup.cs .....	880
Adding DropDownList component to the application .....	880
Binding data source .....	880
Configure the popup list .....	881
See Also .....	882
Data Binding in Blazor DropDown List Component .....	882
Index Value Binding .....	883
Data Source in Blazor DropDown List Component .....	883
Binding local data.....	884
Binding remote data .....	886
Binding ExpandoObject.....	891
Binding DynamicObject.....	892
Binding ObservableCollection.....	893
Entity Framework.....	894
Grouping in Blazor DropDown List Component.....	897
Filtering in Blazor DropDown List Component .....	898
Custom Filtering.....	899
Templates in Blazor DropDown List Component.....	900
Item template .....	900
Value template.....	901
Group template.....	902
Header template .....	903
Footer template .....	904
No records template .....	905
Action failure template .....	906



Style and appearance in Blazor DropDown List Component .....	906
Customizing the appearance of wrapper element .....	907
Customizing the dropdown icon's color .....	907
Customizing the focus color .....	907
Customizing the outline theme's focus color .....	907
Customizing the disabled component's text color .....	907
Customizing the float label element's focusing color .....	908
Customizing the color of the placeholder text .....	908
Customizing the placeholder to add mandatory indicator(*).....	908
Customizing the background color of focus, hover, and active item's .....	908
Customizing the appearance of pop-up element .....	909
Virtualization in Blazor DropDown List Component .....	909
Localization in Blazor DropDown List Component.....	910
Blazor server side .....	910
Blazor WebAssembly .....	912
Accessibility in Blazor DropDown List Component .....	914
ARIA attributes .....	914
Keyboard interaction .....	915
Events in Blazor DropDown List Component .....	915
Blur .....	915
ValueChange .....	916
Closed.....	917
Created.....	917
Destroyed.....	918
Focus .....	918
OnOpen .....	919
OnClose .....	919
DataBound .....	920
Filtering .....	920
OnActionBegin .....	921
OnActionComplete.....	922
OnActionFailure .....	922
OnValueSelect.....	923
Opened.....	924
How To .....	924

DropDownList options with tooltip in Blazor DropDown List Component .....	924
FileManager .....	926
Getting Started with Blazor FileManager Component .....	926
Importing Syncfusion Blazor component in the application.....	926
Add Syncfusion Blazor service in Startup.cs (Server-side application) .....	930
Add Syncfusion Blazor service in Program.cs (Client-side application) .....	931
Adding File Manager component namespace to the application.....	931
Initialize the File Manager component .....	931
Initialize the service in controller.....	932
File download support .....	934
File upload support .....	935
Image preview support .....	935
Sample application.....	936
See Also .....	936
User Interface in Blazor FileManager Component .....	937
Toolbar .....	938
Files and folders navigation .....	938
View .....	939
Context menu.....	940
File Operations in Blazor FileManager Component.....	941
File operation request and response Parameters .....	941
File request and response contents.....	953
Action Buttons .....	954
Multiple File Selection in Blazor FileManager Component.....	956
Output.....	956
Drag and Drop in Blazor FileManager Component.....	956
Output.....	957
File System Providers in Blazor FileManager Component.....	957
ASP.NET Core file system provider .....	957
ASP.NET MVC 5 file system provider .....	958
ASP.NET Core Azure cloud file system provider .....	958
ASP.NET Core Amazon S3 cloud file provider .....	960
ASP.NET MVC Amazon S3 cloud file provider .....	960
File Transfer Protocol file system provider .....	961
SQL database file system provider.....	962

Google Drive file system provider.....	963
Node.js file system provider .....	963
Firebase file system provider .....	964
Accessibility in Blazor FileManager Component.....	967
ARIA attributes.....	967
Keyboard interaction .....	968
How To .....	968
Adding Custom Item To Context Menu in Blazor FileManager Component .....	968
Adding Custom Item To Toolbar in Blazor FileManager Component .....	969
File Upload .....	970
Getting Started with Blazor File Upload Component .....	970
Importing Syncfusion Blazor component in the application.....	970
Adding component package to the application .....	971
Add SyncfusionBlazor service in Startup.cs .....	971
Adding uploader component to the application.....	971
Run the application .....	971
Without server-side API endpoint .....	972
With server-side API endpoint.....	973
Configure allowed file types .....	974
See Also .....	975
Asynchronous Upload in Blazor File Upload Component .....	975
Multiple file upload.....	975
Single file upload.....	976
Auto upload.....	976
Sequential upload .....	977
Preloaded files .....	977
Chunk Upload in Blazor File Upload Component.....	978
Save and remove action for Blazor (ASP.NET Core hosted) application.....	978
Save action configuration in server-side blazor .....	980
Additional configurations.....	981
Resumable upload .....	981
Cancel upload.....	982
Localization in Blazor File Upload Component .....	982
Blazor server side .....	982
Blazor WebAssembly .....	985

File Source in Blazor File Upload Component .....	988
Directory upload .....	988
Drag and drop .....	990
Validation in Blazor File Upload Component .....	991
File type .....	991
File size .....	992
Events in Blazor File Upload Component .....	992
BeforeRemove .....	992
BeforeUpload .....	993
Created .....	993
FileSelected .....	993
OnActionComplete .....	994
OnCancel .....	994
OnChunkFailure .....	994
OnChunkFailed .....	995
OnChunkSuccess .....	995
OnChunkUploadStart .....	995
OnClear .....	995
OnFailure .....	996
OnFailed .....	996
OnFileListRender .....	996
OnRemove .....	997
OnResume .....	997
OnUploadStart .....	997
Paused .....	998
Progressing .....	998
Success .....	998
ValueChange .....	998
How To .....	999
Blazor File Upload Component in WebAssembly App using Visual Studio .....	999
Adding html attributes in Blazor File Upload Component .....	1004
Gantt Chart .....	1005
Getting Started with Blazor Gantt Chart Component .....	1005
Importing Syncfusion Blazor component in the application .....	1005
Adding component package to the application .....	1005

Add SyncfusionBlazor service in Startup.cs .....	1005
Adding Gantt Chart component to the application .....	1006
Binding Gantt Chart with Data .....	1006
Mapping Task Fields .....	1007
Defining Columns .....	1008
Enable Editing .....	1010
Enable Filtering .....	1011
Enable Sorting .....	1012
Enabling Predecessors or Task Relationships .....	1013
See Also .....	1015
Data Binding in Blazor Gantt Chart Component .....	1015
List Binding .....	1015
Remote Data .....	1023
Custom Binding in Blazor Gantt Chart Component .....	1026
Data binding .....	1027
Inject service into Custom Adaptor .....	1029
CRUD operation .....	1032
Virtualization in Blazor Gantt Chart Component .....	1035
Row Virtualization .....	1035
Limitations for Virtualization .....	1043
Columns in Blazor Gantt Chart Component .....	1043
Defining Columns .....	1043
Header template .....	1046
Format .....	1049
Reordering .....	1054
Resizing .....	1059
Column Template .....	1063
Column Menu .....	1066
Responsive Columns .....	1069
Change Tree / Expander column .....	1070
Show or Hide Columns dynamically .....	1073
Controlling Gantt Column actions .....	1075
Column Type .....	1077
Custom Columns .....	1077
Column Chooser .....	1080

Row .....	1083
Cell.....	1094
Toggle Selection .....	1103
Clear Selection .....	1105
Get Selected Row Indexes and Records.....	1107
Filtering in Blazor Gantt Chart Component .....	1109
Menu Filtering.....	1109
Managing Tasks in Blazor Gantt Chart Component .....	1122
Adding New Tasks .....	1122
Editing Tasks.....	1128
Cell Edit Type and its Params .....	1149
Disable Editing for Particular Column .....	1154
Deleting Tasks .....	1156
Entity FrameWork .....	1160
Indent and Outdent .....	1168
Troubleshoot: Editing works only when primary key column is defined.....	1170
Task Dependencies in Blazor Gantt Chart Component.....	1170
Task relationship types .....	1170
Define task relationship .....	1171
Predecessor offset with duration units.....	1173
Validate predecessor links on editing .....	1175
Sorting in Blazor Gantt Chart Component .....	1178
Sorting Column on Gantt Chart Initialization.....	1181
Sorting Column dynamically .....	1183
Clear all the Sorted Columns dynamically .....	1185
Sorting Events .....	1188
Sorting Custom Columns.....	1190
Scrolling in Blazor Gantt Component.....	1192
Set width and height.....	1192
Responsive with the parent container.....	1193
Context Menu in Blazor Gantt Chart Component.....	1194
Baseline in Blazor Gantt Chart Component .....	1197
Time Line in Blazor Gantt Chart Component .....	1199
Timeline View Modes.....	1199
Top Tier and Bottom Tier .....	1210

Timeline Cell Width .....	1217
Week Start Day Customization .....	1219
Customize Automatic Timescale Update action .....	1221
Zooming .....	1223
Zoom action by methods .....	1226
Holidays in Blazor Gantt Chart Component .....	1231
Resources in Blazor Gantt Chart Component .....	1234
Resource Collection .....	1234
Assign Resource .....	1237
Add / Edit Resource Collection .....	1241
Work in Blazor Gantt Chart Component .....	1245
Task type .....	1248
Tooltip in Blazor Gantt Chart Component .....	1251
Enable Tooltip .....	1251
Timeline Cells Tooltip .....	1255
Cell Tooltip .....	1258
Tooltip Template .....	1260
Templates in Blazor Gantt Chart Component .....	1270
Template Context .....	1270
GanttChartTemplates component .....	1270
Appearance Customization in Blazor Gantt Chart Component .....	1275
Taskbar customization .....	1275
Task labels .....	1278
Connector lines .....	1280
Customize rows and cells .....	1282
Grid lines .....	1284
Splitter .....	1286
Duration Unit in Blazor Gantt Chart Component .....	1290
Mapping the Duration Unit Field .....	1291
Defining Duration Unit along With Duration Field .....	1293
Scheduling Tasks in Blazor Gantt Chart Component .....	1295
Automatically Scheduled Tasks .....	1296
Manually Scheduled Tasks .....	1297
Custom .....	1300
Unscheduled Tasks .....	1302

Define Unscheduled Tasks in Data Source.....	1303
Working Time Range .....	1305
Weekend or Non-working Days .....	1307
Rows in Blazor Gantt Chart Component .....	1310
Row Height.....	1310
Expand or Collapse Row.....	1312
Drag and Drop.....	1319
Customize Rows .....	1324
Styling Alternate Rows .....	1327
Toolbar in Blazor Gantt Chart Component .....	1329
Built-in Toolbar Items.....	1330
Custom Toolbar Items.....	1332
Built-in and Custom Items in Toolbar .....	1335
Enable or Disable Toolbar Items .....	1337
Excel Export in Blazor Gantt Chart Component .....	1340
Customize the Excel Export.....	1342
Data Markers in Blazor Gantt Chart Component.....	1349
WebAssembly Performance in Blazor Gantt Component.....	1350
Avoid unnecessary component renders .....	1351
Accessibility in Blazor Gantt Chart Component .....	1352
WAI-ARIA.....	1352
Keyboard Navigation.....	1353
Globalization in Blazor Gantt Chart Component .....	1354
Localization .....	1354
Internationalization.....	1364
Right to Left (RTL).....	1364
Touch Interaction in Blazor Gantt Chart Component .....	1366
Tooltip .....	1366
Context Menu .....	1366
Sorting.....	1366
Column Resize .....	1367
Editing .....	1368
Selection.....	1369
Style And Appearance in Blazor Gantt Chart Component .....	1370
Events in Blazor GanttChart Component.....	1374



OnActionBegin .....	1374
OnActionComplete.....	1376
OnActionFailure .....	1378
Created.....	1380
OnLoad .....	1381
Destroyed.....	1383
TaskbarEdited .....	1385
RowDropped .....	1387
RowDataBound .....	1389
OnRowDragStart .....	1390
QueryChartRowInfo .....	1392
QueryCellInfo .....	1394
EndEdit .....	1396
BeforeTooltipRender.....	1398
SplitterResizeStart.....	1400
SplitterResized .....	1401
OnCellEdit.....	1403
SplitterResizing.....	1405
RowSelecting.....	1407
RowSelected.....	1409
RowDeselecting.....	1411
RowDeselected .....	1412
CellSelecting.....	1414
CellSelected.....	1416
CellDeselecting.....	1418
CellDeselected.....	1420
OnToolBarClick.....	1422
ColumnMenuClicked .....	1423
ContextMenuClicked .....	1425
ContextMenuOpen .....	1427
Collapsed.....	1429
Collapsing .....	1431
Expanding.....	1433
Expanded.....	1434
How To .....	1436

Hide chart part in Blazor Gantt Chart Component .....	1436
Open Add Edit Dialog Dynamically in Blazor Gantt Chart Component .....	1438
WebAssembly Gantt Chart in Blazor Gantt Chart Component .....	1440
HeatMap Chart.....	1455
Getting Started with Blazor HeatMap Chart Component.....	1455
Importing Syncfusion Blazor component in the application.....	1455
Add Syncfusion Blazor service in Startup.cs (Server-side application) .....	1459
Add Syncfusion Blazor service in Program.cs (Client-side application) .....	1460
Adding component package to the application .....	1460
Adding HeatMap component to the application .....	1461
Run the application .....	1462
See Also .....	1462
Working with data in Blazor HeatMap Chart Component.....	1462
Data adaptor .....	1462
Empty points .....	1464
Bubble heat map in Blazor HeatMap Chart Component .....	1465
Bubble attributes .....	1465
Rendering mode in Blazor HeatMap Chart Component .....	1470
Axis in Blazor HeatMap Chart Component .....	1471
Types .....	1471
Inversed axis.....	1475
Opposed axis.....	1476
Label formatting.....	1478
Axis intervals .....	1479
Axis label increment.....	1480
Palette in Blazor HeatMap Chart Component .....	1481
Palette types .....	1481
Legend in Blazor HeatMap Chart Component .....	1485
Legend types .....	1487
Placement .....	1488
Alignment.....	1490
Legend dimensions .....	1491
Smart Legend .....	1494
Legend Selection .....	1495
Appearance in Blazor HeatMap Chart Component .....	1497

Cell customizations .....	1497
Margin .....	1500
Title .....	1501
Data label .....	1502
Dimensions in Blazor HeatMap Chart Component .....	1506
Size for heat map .....	1506
Tooltip in Blazor HeatMap Chart Component .....	1508
Default tooltip .....	1509
Selection in Blazor HeatMap Chart Component .....	1510
In-place Editor .....	1511
Getting Started with Blazor In-place Editor Component .....	1511
Importing Syncfusion Blazor component in the application .....	1511
Adding component package to the application .....	1512
Add SyncfusionBlazor service in Startup.cs .....	1512
Add In-Place Editor component .....	1512
Render In-place Editor with popup .....	1513
Run the application .....	1514
Configuring DropDownList .....	1514
Integrate DatePicker .....	1515
Submitting data to the server (save) .....	1518
Refresh In-place Editor with modified value .....	1518
See Also .....	1520
List of Components in Blazor In-place Editor Component .....	1520
See Also .....	1526
Configuration in Blazor In-place Editor Component .....	1526
Rendering modes .....	1526
Event actions for editing .....	1528
Action on focus out .....	1530
Display modes .....	1532
See Also .....	1533
Buttons in Blazor In-place Editor Component .....	1533
See Also .....	1535
Server Actions in Blazor In-place Editor Component .....	1535
Data Binding in Blazor In-place Editor Component .....	1537
Local .....	1537

Integrate HTML5 Components in Blazor In-place Editor Component .....	1538
See Also .....	1539
Validation in Blazor In-place Editor Component.....	1539
Style and appearance in Blazor In-place Editor Component .....	1540
Customizing the In-place Editor text.....	1540
Customizing the In-place Editor action buttons .....	1541
Globalization in Blazor In-place Editor Component.....	1541
Localization .....	1541
Right to left .....	1543
Format.....	1544
Events in Blazor In-place Editor Component .....	1545
Created.....	1545
OnActionBegin .....	1545
OnActionSuccess.....	1545
OnActionFailure .....	1546
ValueChange .....	1546
Destroyed.....	1546
How To .....	1547
Dynamically move In-place Editor to edit mode in Blazor.....	1547
Disable the edit mode specifically in Blazor In-place Editor Component .....	1548
Custom Animation for popup mode in Blazor In-place Editor Component.....	1549
Input Mask .....	1550
Getting Started with Blazor Input Mask Component .....	1550
Importing Syncfusion Blazor component in the application.....	1551
Adding component package to the application.....	1551
Add SyncfusionBlazor service in Program.cs .....	1551
Adding MaskedTextBox component to the application .....	1552
Set the mask.....	1552
See Also .....	1552
Data Binding in Blazor Input Mask Component.....	1553
Dynamic Value Binding .....	1553
Mask Configuration in Blazor Input Mask Component.....	1553
Standard mask elements.....	1553
Custom mask elements.....	1555
Prompt character .....	1555

Accessibility in Blazor Input Mask Component.....	1556
Native Events in Blazor Input Mask Component .....	1556
Bind native events to MaskedTextBox.....	1556
Pass event data to event handler .....	1557
List of Native events supported .....	1557
Events in Blazor Input Mask Component.....	1558
Blur .....	1558
Created.....	1558
Destroyed.....	1558
Focus .....	1558
ValueChange .....	1559
ValueChanged .....	1559
How To .....	1559
Customize the UI appearance of the Blazor Input Mask Component .....	1559
Model Binding in Blazor Input Mask Component.....	1560
Kanban .....	1561
Getting Started with Blazor Kanban Component .....	1561
Importing Syncfusion Blazor component in the application.....	1561
Adding component package to the application.....	1562
Add SyncfusionBlazor service in Startup.cs .....	1562
Add Kanban component .....	1562
Run the application .....	1563
Populating cards.....	1563
Enable Swimlane .....	1564
See Also .....	1567
Data Binding in Blazor Kanban Component.....	1567
List binding .....	1567
Remote data.....	1571
Observable collection .....	1580
Columns in Blazor Kanban Component .....	1583
Single-key mapping .....	1583
Multi-key mapping .....	1585
Header text .....	1587
Header template .....	1588
Toggle columns .....	1593

Stacked headers .....	1598
Cards in Blazor Kanban Component .....	1600
Drag-and-drop.....	1600
Header.....	1600
Content .....	1603
Tags .....	1603
Left border color .....	1605
Custom class.....	1608
Template .....	1612
Selection.....	1615
Swimlane in Blazor Kanban Component.....	1618
Render swimlane row .....	1618
Custom row text.....	1620
Template .....	1623
Sorting.....	1626
Drag-and-drop.....	1629
Calculate cards count.....	1631
Enable frozen rows .....	1633
Workflow in Blazor Kanban Component.....	1636
Prevent transition across columns.....	1636
Prevent Drop actions .....	1638
Prevent Drag actions.....	1639
Sorting in Blazor Kanban Component.....	1642
SortBy.....	1642
Change the direction.....	1650
Card Editing in Blazor Kanban Component.....	1653
Default Dialog.....	1653
Custom Fields.....	1656
Dialog Template .....	1657
Prevent Dialog.....	1661
Persisting data in server.....	1663
Tooltip in Blazor Kanban Component .....	1668
Styling And Appearance in Blazor Kanban Component.....	1671
WIP Validation in Blazor Kanban Component.....	1673
Minimum card limit.....	1673

Maximum card limit .....	1673
Responsive Mode in Blazor Kanban Component.....	1676
Layouts .....	1676
Scrolling.....	1678
Selection.....	1679
Localization in Blazor Kanban Component .....	1680
Localization .....	1681
Right to left (RTL) .....	1683
Dimensions in Blazor Kanban Component.....	1686
Auto height and width .....	1686
Height and width in pixel .....	1688
Height and width in percentage .....	1691
Events in Blazor Kanban Component.....	1693
OnLoad.....	1694
ActionBegin .....	1694
ActionComplete .....	1694
ActionFailure .....	1694
CardClick.....	1695
CardDoubleClick.....	1695
CardRendered .....	1695
DataBinding.....	1696
DialogClose.....	1696
DialogOpen .....	1696
DragStart .....	1696
DragStop .....	1697
QueryCellInfo .....	1697
Linear Gauge .....	1697
Getting Started with Blazor Linear Gauge Component (SfLinearGauge).....	1697
Importing Syncfusion Blazor Linear Gauge component in the application .....	1698
Adding a component package to the application .....	1698
Adding SyncfusionBlazor Service in Startup.cs .....	1698
Initializing Linear Gauge component in the application .....	1699
Set pointer value .....	1699
Add a title for Linear Gauge .....	1700
Add ranges in the Linear gauge.....	1701

See also .....	1701
Dimensions in Blazor Linear Gauge Component.....	1702
Size for Linear Gauge .....	1702
Axes in Blazor Linear Gauge Component.....	1703
Setting the start value and end value of the axis.....	1703
Line Customization.....	1704
Ticks Customization .....	1705
Labels Customization .....	1707
Orientation.....	1711
Inverted Axis .....	1712
Opposed Axis.....	1712
Multiple Axes .....	1713
Ranges in Blazor Linear Gauge Component.....	1714
Customizing the range .....	1715
Setting the range color for the labels .....	1716
Multiple ranges .....	1717
Gradient Color.....	1718
Pointers in Blazor Linear Gauge Component.....	1720
Types of pointer .....	1720
Multiple pointers .....	1724
Pointer animation .....	1724
Gradient Color.....	1725
Annotations in Blazor Linear Gauge Component.....	1727
Adding annotation .....	1727
Customization .....	1729
Multiple annotations .....	1732
User Interaction in Blazor Linear Gauge Component .....	1734
Tooltip .....	1734
Pointer drag .....	1738
Print And Export in Blazor Linear Gauge Component.....	1739
Print.....	1739
Export.....	1740
Appearance in Blazor Linear Gauge Component.....	1742
Customizing the Linear Gauge area .....	1742
Setting up the Linear Gauge title .....	1743



Customizing the Linear Gauge container.....	1744
Fitting the Linear Gauge to the control.....	1746
Accessibility in Blazor Linear Gauge Component.....	1747
Globalization in Blazor Linear Gauge Component .....	1747
Globalization .....	1747
Events in Blazor Linear Gauge Component.....	1748
AnnotationRendering.....	1748
AxisLabelRendering.....	1749
Loaded.....	1749
OnDragEnd .....	1750
OnDragStart .....	1750
OnGaugeMouseDown.....	1751
OnGaugeMouseLeave .....	1751
OnGaugeMouseUp.....	1751
OnLoad.....	1752
OnPrint.....	1752
Resizing .....	1753
TooltipRendering .....	1753
ValueChange .....	1754
Methods in Blazor Linear Gauge Component.....	1754
SetPointerValue .....	1754
SetAnnotationValue .....	1755
RefreshAsync.....	1756
ListBox.....	1757
Getting Started with Blazor ListBox Component .....	1757
Importing Syncfusion Blazor component in the application.....	1757
Adding component package to the application.....	1757
Add SyncfusionBlazor service in Startup.cs .....	1758
Adding ListBox component to the application.....	1758
Binding data source .....	1758
Run the application .....	1759
See Also .....	1759
Accessibility in Blazor ListBox Component.....	1759
ARIA Attributes .....	1759
Keyboard interaction .....	1760

Data Binding in Blazor ListBox Component.....	1760
Local Data.....	1761
Remote Data .....	1763
Drag And Drop in Blazor ListBox Component .....	1764
Single ListBox .....	1764
Multiple ListBox .....	1765
Dual ListBox in Blazor ListBox Component .....	1766
Icons and Templates in Blazor ListBox Component .....	1768
Icons.....	1768
Templates.....	1770
Selection in Blazor ListBox Component .....	1772
Single selection .....	1772
Multiple selection .....	1773
CheckBox Selection .....	1774
Sorting and Grouping in Blazor ListBox Component.....	1775
Sorting.....	1775
Grouping .....	1776
Styles and Appearances in Blazor ListBox Component.....	1777
How To .....	1777
Add/Remove Items in Blazor ListBox Component .....	1777
Bind Change Events in Blazor ListBox Component .....	1779
Enable Scroller in Blazor ListBox Component .....	1780
Enable/Disable ListBox in Blazor ListBox Component.....	1781
Select Items in Blazor ListBox Component.....	1782
ListView .....	1783
Getting Started with Blazor ListView Component .....	1783
Importing Syncfusion Blazor component in the application.....	1784
Add Syncfusion Blazor service in Startup.cs (Server-side application) .....	1787
Add Syncfusion Blazor service in Program.cs (Client-side application) .....	1788
Adding ListView component namespace to the application .....	1788
Adding ListView component to the application.....	1788
Run the application .....	1789
See Also .....	1789
Data Binding in Blazor ListView Component .....	1789
Bind to local data .....	1790

Bind to remote data .....	1791
Entity Framework.....	1792
Grouping in Blazor ListView Component .....	1796
Check list in Blazor ListView Component.....	1797
Checkbox Position.....	1799
Nested list in Blazor ListView Component .....	1800
Customizing Templates in Blazor ListView Component.....	1804
Header Template .....	1804
Template .....	1805
Group template.....	1810
Virtualization in Blazor ListView Component .....	1812
Getting started .....	1812
CSS Structure in Blazor ListView Component .....	1815
Customizing ListView .....	1815
Customizing the list items.....	1816
Customizing ListView's header.....	1816
Customizing group header of ListView .....	1816
Customizing the hover state of ListView control.....	1816
Customizing selected item of ListView control.....	1816
Accessibility in Blazor ListView Component.....	1817
Keyboard interaction .....	1817
ARIA attributes.....	1820
How To .....	1821
Get selected items from Blazor ListView Component .....	1821
Add and remove list items in Blazor ListView Component.....	1822
Use dynamic templates in Blazor ListView based on device .....	1825
Create mobile contact layout using Blazor ListView.....	1831
Filter and search list items using Blazor ListView Component .....	1835
Blazor ListView Component with hyper-link navigation.....	1837
Chat window user interface using Blazor ListView Component .....	1839
Create dual list using Blazor ListView Component .....	1846
Customize Blazor ListView Component to Grid Layout .....	1851
Get selected items from listview template in Blazor ListView Component .....	1859
Trace events of listview in Blazor ListView Component .....	1861
Maps .....	1864

Getting Started with Blazor Maps Component.....	1864
Importing Syncfusion Blazor Maps component in the application.....	1864
Adding component package to the application.....	1864
Adding SyncfusionBlazor Service in Startup.cs .....	1864
Adding Maps component.....	1865
Adding GeoJSON data in Maps layer .....	1865
Bind data source .....	1866
Apply color mapping .....	1867
Adding data labels.....	1868
Adding title for Maps .....	1869
Enable legend.....	1870
Enable tooltip.....	1872
See also .....	1873
Populate Data in Blazor Maps Component.....	1873
Shape data .....	1873
Data source .....	1873
Data binding.....	1874
Fetching data from JSON file.....	1877
Layers in Blazor Maps Component .....	1879
Multilayer.....	1879
Sublayer .....	1879
Displaying different layer in the view .....	1880
See also .....	1881
Providers .....	1881
OpenStreetMap in Blazor Maps Component.....	1881
Bing Maps in Blazor Maps Component.....	1886
Other Maps in Blazor Maps Component .....	1892
Customization in Blazor Maps Component .....	1898
Setting the size for Maps .....	1898
Maps title .....	1899
Setting theme.....	1900
Customizing Maps container .....	1901
Customizing Maps area.....	1902
Customizing the shapes .....	1903
Setting color to the shapes from the data source .....	1904

Projection type.....	1905
Color Mapping in Blazor Maps Component.....	1906
Types of color mapping.....	1907
Multiple colors for a single shape .....	1911
Color for items excluded from color mapping .....	1912
Color mapping for bubbles .....	1913
Data labels in Blazor Maps Component.....	1915
Adding data labels.....	1915
Customization .....	1917
Smart labels.....	1919
Intersect action .....	1920
Adding data label as a template .....	1921
Markers in Blazor Maps Component .....	1922
Adding marker.....	1922
Adding marker template.....	1924
Customization .....	1925
Marker shapes .....	1926
Multiple marker groups .....	1928
Customize marker shapes from data source .....	1929
Marker zooming.....	1932
Marker clustering.....	1933
Customization of marker cluster.....	1935
Expanding the marker cluster .....	1936
Tooltip for marker .....	1938
See also .....	1939
Bubble in Blazor Maps Component .....	1939
Bubble shapes .....	1941
Customization .....	1943
Setting colors to the bubbles from the data source .....	1944
Setting the range of the bubble size .....	1945
Multiple bubble groups.....	1947
Enable tooltip for bubble .....	1949
Legend in Blazor Maps Component.....	1950
Modes of legend .....	1950
Positioning of the legend .....	1952

Legend for shapes .....	1953
Enable legend for bubbles .....	1967
Enable legend for markers .....	1968
Navigation Lines in Blazor Maps Component .....	1970
Customization .....	1970
Enabling the arrows .....	1971
Annotations in Blazor Maps component .....	1972
Annotation .....	1973
Annotation customization .....	1974
Multiple Annotation.....	1976
User Interactions in Blazor Maps Component.....	1977
Zooming .....	1977
Selection.....	1984
Highlight .....	1993
Tooltip .....	1999
See also .....	2005
Print and export in Blazor Maps Component .....	2005
Print.....	2005
Export.....	2006
State Persistence in Blazor Maps Component.....	2010
State Persistence.....	2010
Accessibility in Blazor Maps Component .....	2010
KeyBoard Navigation.....	2011
Globalization in Blazor Maps Component .....	2011
Globalization .....	2011
Numeric Format .....	2013
See also .....	2014
Localization in Blazor Maps Component.....	2014
See also .....	2017
Events in Blazor Maps Component .....	2017
AnimationCompleted.....	2017
AnnotationRendering.....	2018
BubbleRendering.....	2018
DataLabelRendering.....	2019
LayerRendering .....	2020

LegendRendering .....	2020
Loaded .....	2021
MarkerRendering .....	2021
MarkerClusterClick .....	2022
MarkerClusterMouseMove .....	2023
OnBubbleClick .....	2024
OnBubbleMouseMove .....	2025
OnClick .....	2026
OnDoubleClick .....	2027
OnItemHighlight .....	2027
OnItemSelect .....	2028
OnLoad .....	2028
OnMarkerClick .....	2029
OnMarkerMouseLeave .....	2029
OnMarkerMouseMove .....	2030
OnPan .....	2031
OnPanComplete .....	2031
OnPrint .....	2032
OnRightClick .....	2032
OnZoom .....	2033
OnZoomComplete .....	2033
Resizing .....	2034
ShapeHighlighted .....	2034
ShapeRendering .....	2035
ShapeSelected .....	2035
TooltipRendering .....	2036
Methods in Blazor Maps Component .....	2036
ShapeSelectionAsync .....	2036
Refresh .....	2037
PanByDirectionAsync .....	2037
ZoomByPosition .....	2038
ZoomToCoordinates .....	2039
How To .....	2039
Display geometry shapes in Bing maps in Blazor Maps Component .....	2039
Add different types of markers in Blazor Maps Component .....	2041

Change center position on zooming in Blazor Maps Component ..... 2045



## Diagram

### Getting Started with Blazor Diagram Component

This section briefly explains about how to include a Diagram 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 **Syncfusion.Blazor.Diagrams** 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.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.Diagrams** packages.

#### ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Diagrams
```

#### 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();  
    }  
}
```

**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 Diagram component to the Application

Diagram component can be rendered by using the `SfDiagram` tag helper in ASP.NET Core Blazor application. Add the Diagram component in any web page razor in the `Pages` folder. For example, the Diagram component is added in the `~/Pages/Index.razor` page.

The following example shows a basic Diagram component.

#### ASPX-CS

```
<SfDiagram Width="100%" Height="600px">  
</SfDiagram>
```

#### Adding Nodes and Connectors

Let us create and add a `nodes` with specific position, size, label and shape. Connect two or more nodes by using a `connectors`.

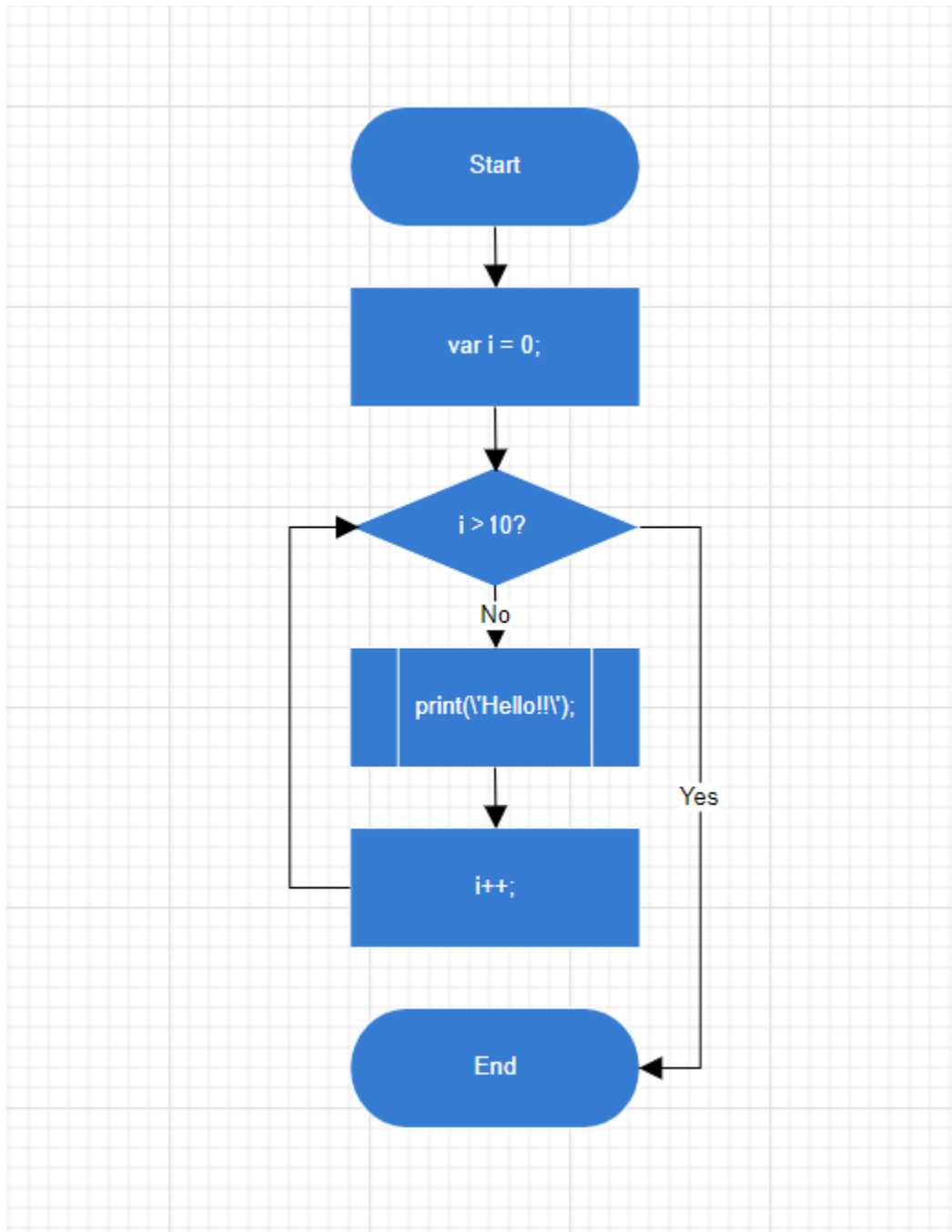
#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams  
@using System.Collections.ObjectModel  
@using DiagramShapes = Syncfusion.Blazor.Diagrams.Shapes  
@using DiagramSegments = Syncfusion.Blazor.Diagrams.Segments  
<SfDiagram Height="600px" Nodes="@NodeCollection"  
Connectors="@ConnectorCollection" NodeDefaults="@NodeDefaults"  
ConnectorDefaults="@ConnectorDefaults">
```

```
</SfDiagram>
@code
{
    int connectorCount = 0;
    // Reference to diagram
    SfDiagram diagram;
    // Defines diagram's nodes collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    // Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection { get;
    set; }
    // Defines default values for DiagramNode object
    public DiagramNode NodeDefaults { get; set; }
    // Defines default values for DiagramConnector object
    public DiagramConnector ConnectorDefaults { get; set; }
    protected override void OnInitialized()
    {
        InitDiagramModel();
    }
    private void InitDiagramModel()
    {
        InitDiagramDefaults();
        NodeCollection = new ObservableCollection<DiagramNode>();
        ConnectorCollection = new ObservableCollection<DiagramConnector>();
        CreateNode("Start", 50, FlowShapes.Terminator, "Start");
        CreateNode("Init", 140, FlowShapes.Process, "var i = 0;");
        CreateNode("Condition", 230, FlowShapes.Decision, "i < 10?");
        CreateNode("Print", 320, FlowShapes.PreDefinedProcess,
        "print('Hello!!')");
        CreateNode("Increment", 410, FlowShapes.Process, "i++;");
        CreateNode("End", 500, FlowShapes.Terminator, "End");
        DiagramConnectorSegment segment1 = new DiagramConnectorSegment()
        {
            Type = DiagramSegments.Orthogonal,
            Length = 30,
            Direction = Direction.Right
        };
        DiagramConnectorSegment segment2 = new DiagramConnectorSegment()
        {
            Type = DiagramSegments.Orthogonal,
            Length = 300,
            Direction = Direction.Bottom
        };
        DiagramConnectorSegment segment3 = new DiagramConnectorSegment()
        {
            Type = DiagramSegments.Orthogonal,
            Length = 30,
            Direction = Direction.Left
        };
        DiagramConnectorSegment segment4 = new DiagramConnectorSegment()
        {
            Type = DiagramSegments.Orthogonal,
            Length = 200,
            Direction = Direction.Top
        };
        CreateConnector("Start", "Init");
        CreateConnector("Init", "Condition");
    }
}
```

```
CreateConnector("Condition", "Print");
CreateConnector("Condition", "End", "Yes", segment1, segment2);
CreateConnector("Print", "Increment", "No");
CreateConnector("Increment", "Condition", null, segment3, segment4);
}
private void CreateConnector(string sourceId, string targetId, string label
= default(string), DiagramConnectorSegment segment1 = null,
DiagramConnectorSegment segment2 = null)
{
    DiagramConnector diagramConnector = new DiagramConnector()
    {
        Id = string.Format("connector{0}", ++connectorCount),
        SourceID = sourceId,
        TargetID = targetId
    };
    if (label != default(string))
    {
        var annotation = new DiagramConnectorAnnotation()
        {
            Content = label,
            Style = new AnnotationStyle() { Fill = "white" }
        };
        diagramConnector.Annotations = new
        ObservableCollection<DiagramConnectorAnnotation>() { annotation };
    }
    if (segment1 != null)
    {
        diagramConnector.Segments = new
        ObservableCollection<DiagramConnectorSegment>() { segment1, segment2 };
    }
    ConnectorCollection.Add(diagramConnector);
}
private void InitDiagramDefaults()
{
    DiagramNodeAnnotation defaultAnnotation = new DiagramNodeAnnotation()
    {
        Style = new AnnotationStyle()
        {
            Color = "white",
            Fill = "transparent"
        }
    };
    NodeDefaults = new DiagramNode()
    {
        Width = 140,
        Height = 50,
        OffsetX = 300,
        Annotations = new ObservableCollection<DiagramNodeAnnotation>() {
        defaultAnnotation },
        Style = new NodeShapeStyle() { Fill = "#357BD2", StrokeColor = "white" }
    };
    ConnectorDefaults = new DiagramConnector()
    {
        Type = DiagramSegments.Orthogonal,
        TargetDecorator = new ConnectorTargetDecorator() { Shape =
        DecoratorShapes.Arrow, Width = 10, Height = 10 }
    };
}
```

```
}  
private void CreateNode(string id, double y, FlowShapes shape, string label,  
bool positionLabel = false)  
{  
    DiagramNodeAnnotation annotation = new DiagramNodeAnnotation() { Content =  
    label };  
    if (positionLabel)  
    {  
        annotation.Margin = new NodeAnnotationMargin() { Left = 25, Right = 25 };  
    };  
    DiagramNode diagramNode = new DiagramNode()  
    {  
        Id = id,  
        OffsetY = y,  
        Shape = new DiagramShape() { Type = DiagramShapes.Flow, FlowShape = shape },  
        Annotations = new ObservableCollection<DiagramNodeAnnotation>() { annotation  
    }  
    };  
    NodeCollection.Add(diagramNode);  
}  
}
```



You can refer to our [Blazor Diagram](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Diagram example](#) to understand how to present and manipulate data.

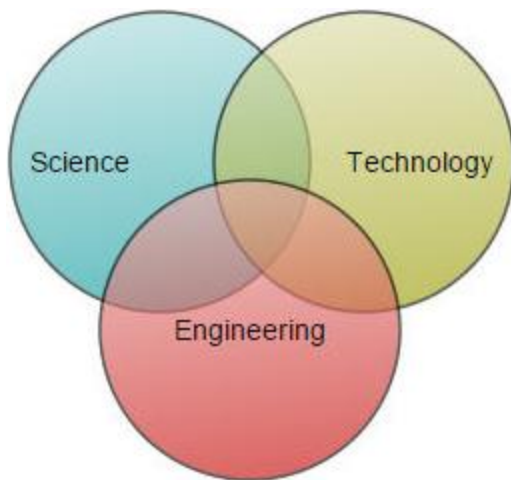
See Also

- [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](#)

## Nodes

### Actions of nodes in Blazor Diagram Component

Nodes are graphical objects that are used to visually represent the geometrical information, process flow, internal business procedure, entity, or any other kind of data and it represents the functions of a complete system in regards to how it interacts with external entities.



### Create node

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

### Add node through nodes collection

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in the nodes collection.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
        };
        // Add node
        NodeCollection.Add(node1);
    }
}
```

```
}
```



### Add node at runtime

- A node can be added in a diagram at runtime by using the [AddNode](#).
- The node's ID property is used to define the name of the node and it is further used to find the node at runtime and do any customization.

The following code shows how to add a node at runtime.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Add Node" @onclick="@AddNode" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    SfDiagram Diagram;
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes collection.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new NodeShapeStyle()
            {
                Fill = "#6BA5D7",
                StrokeColor = "white"
            }
        };
        NodeCollection.Add(node1);
    }
    // Add node at runtime
    public void AddNode()
    {
        DiagramNode Node1 = new DiagramNode()
        {
            Id = "New Node1",
```



```
OffsetX = 100,  
OffsetY = 100,  
Width = 100,  
Height = 100  
};  
Diagram.AddNode(Node1);  
}  
}
```

Also, the Node can be added at runtime by using the **Add** method.

### CSHARP

```
// Add node at runtime  
public void AddNode()  
{  
    DiagramNode Node2 = new DiagramNode()  
    {  
        Id = "New Node2",  
        OffsetX = 100,  
        OffsetY = 200,  
        Width = 100,  
        Height = 100,  
    };  
    NodeCollection.Add(Node2);  
}
```



#### *Add node from palette*

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

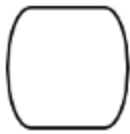
- Once you drag a node/connector from the palette to the diagram, the following events can be used to do the customization.
- When a symbol is dragged into a diagram from symbol palette, the [DragEnter](#) event gets triggered.
- When a symbol is dragged over a diagram, the [DragOver](#) event gets triggered.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

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@using DiagramShapes = Syncfusion.Blazor.Diagrams.Shapes
<div id="palette-space" style="width: 130px; float: left">
<SfSymbolPalette id="symbolPalette" Width="130px" Height="700px"
SymbolHeight="60" SymbolWidth="60" SymbolInfo="@SymbolInfo">
<SymbolPalettePalettes>
<SymbolPalettePalette Id="FlowShapePalette" Expanded="true"
Symbols="@FlowShapeList" Title="Flow Shapes">
</SymbolPalettePalette>
</SymbolPalettePalettes>
</SfSymbolPalette>
</div>
<div style="border: 1px solid #D7D7D7">
<SfDiagram ID="diagram" Height="400px" Width="1200px">
</SfDiagram>
</div>
@code
{
// Defines palette's flow-shape collection
public ObservableCollection<DiagramNode> FlowShapeList { get; set; }
// Defines Symbol info
public SymbolInfo SymbolInfo { get; set; }
protected override void OnInitialized()
{
// Sets the symbol info
SymbolInfo = new SymbolInfo() { Fit = true };
// Add shapes to palette
FlowShapeList = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Terminator",
// Set the symbol preview size
PreviewSize = new SymbolSizeModel() { Width = 100, Height = 100 },
// Sets the shape of the node
Shape = new DiagramShape()
{
Type = DiagramShapes.Flow,
FlowShape = FlowShapes.Terminator
}
}
};
}
```

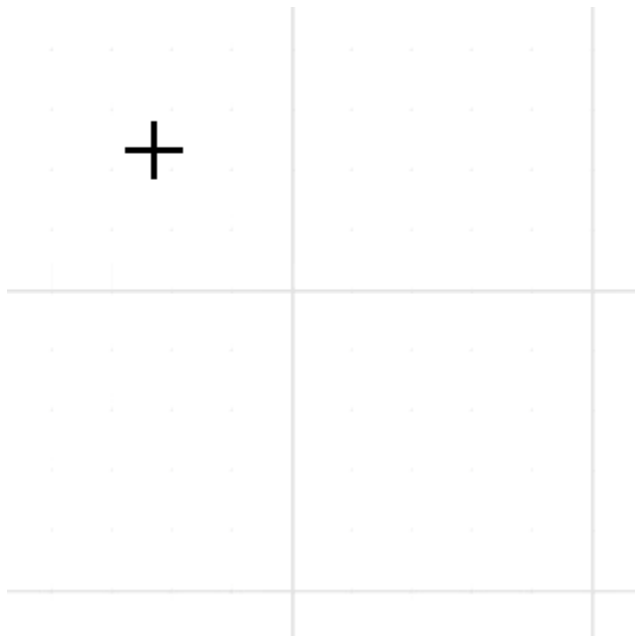
## Flow Shapes



### *Draw Node using drawing object*

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

For more information about drawing Node, refer to the [Draw Nodes](#).



### *Create node through data source*

Nodes can be generated automatically with the information provided through data source. The default properties for these nodes are fetched from default settings.

### *Remove node at runtime*

A node can be removed from diagram at runtime by using the [Remove](#) method.

The following code shows how to remove a node at runtime.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
```

```

@using System.Collections.ObjectModel
<input type="button" value="Remove Node" @onclick="@RemoveNode" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in node's collection.
DiagramNode node1 = new DiagramNode()
{
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
// Add node
NodeCollection.Add(node1);
}
// Remove Node at runtime
public void RemoveNode()
{
Diagram.Remove(NodeCollection[0]);
}
}

```

A Node can be removed from the diagram by using the native [RemoveAt] method. Refer to the following example that shows how to remove node at runtime.

### CSHARP

```

// Add node at runtime
public void RemoveNode()
{
NodeCollection.RemoveAt(0);
}

```

### Update node at runtime

You can change any node's properties at runtime. The following code sample shows how the annotation of the node changed at runtime.

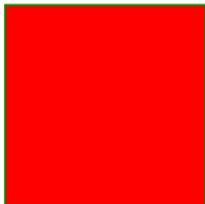
### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Update" @onclick="@UpdateNode" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
// reference of the diagram
SfDiagram Diagram;

```

```
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
    // A node is created and stored in nodes collection.
    DiagramNode node1 = new DiagramNode()
    {
        // Position of the node
        OffsetX = 250,
        OffsetY = 250,
        // Size of the node
        Width = 100,
        Height = 100,
        Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
    };
    // Add node
    NodeCollection.Add(node1);
}
// update node properties at run time
public void UpdateNode()
{
    Diagram.BeginUpdate();
    NodeCollection[0].Style.Fill = "red";
    NodeCollection[0].Style.StrokeColor = "green";
    Diagram.EndUpdate();
}
}
```



**Note:** You cannot reset the node collection directly to add or update the node collection.

*See Also*

- [How to add annotations to the node](#)
- [How to add ports to the node](#)
- [How to enable/disable the behavior of the node](#)
- [How to edit the node visual interface](#)

## Positioning in Blazor Diagram Component

### *Arrange the nodes*

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

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

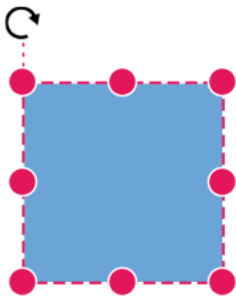
Pivot	Offset
-----	-----
(0.5,0.5)	OffsetX and OffsetY values are considered as the node's center point.
(0,0)	OffsetX and OffsetY values are considered as the top-left corner of the node.
(1,1)	OffsetX and OffsetY values are considered as the bottom-right corner of the node.

The following code shows how to change the **Pivot** value.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" @ref="@diagram" Nodes="@NodeCollection">
</SfDiagram>
@code {
    SfDiagram diagram;
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes array.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
            // Pivot of the node
            Pivot = new NodePivotPoint() { X = 0, Y = 0 }
        };
        NodeCollection.Add(node1);
    }
    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        if (firstRender)
```

```
{
//OnAfterRenderAsync will be triggered after the component rendered.
await Task.Delay(1500);
diagram.Select(new ObservableCollection<DiagramNode>() { diagram.Nodes[0] },
null);
}
}
}
```



### Flip

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

- [The Flip is also applicable for connectors](#)

The flip types are as follows:

- **HorizontalFlip** is used to change the element in the horizontal direction.
- **VerticalFlip** is used to change the element in the vertical direction
- **Both** that involves both vertical and horizontal changes of the element.

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code {
SfDiagram diagram;
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in nodes array.
DiagramNode node1 = new DiagramNode()
{
// Position of the node
OffsetX = 250,
OffsetY = 250,
```

```
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
// Flip the node in Vertical Direction
Flip = FlipDirection.Vertical,
Shape = new DiagramShape()
{
    Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
    BasicShape = BasicShapes.Triangle
}
};
// Add node
NodeCollection.Add(node1);
}
}
```



Before Flip



After Flip

---

The flip is also applicable for group and BPMN shapes.

---

*See also*

- [How to customize the node](#)
- [How to get events when they interact the node](#)

### Appearance in Blazor Diagram Component

The appearance of a node can be customized by changing its [Fill](#) color, [BorderColor](#), [BorderWidth](#), and [Shadow](#). The [Visible](#) property of the node enables or disables the visibility of the node.

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes array.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
```



```

OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
// Add node
Style = new NodeShapeStyle()
{
    Fill = "#6BA5D7",
    StrokeDashArray = "5,5",
    StrokeColor = "red",
    StrokeWidth = 2
},
};
NodeCollection.Add(node1);
}
}

```



### Gradient

The [Gradient](#) property of the node allows you to define and apply the gradient effect to the node. The gradient stop property defines the color and a position, where the previous color transition ends and a new color transition starts. The gradient stop's opacity property defines the transparency level of the region.

There are two types of gradients as follows:

- Linear Gradient
- Radial Gradient

### Linear gradient

[LinearGradient](#) defines a smooth transition between a set of colors (so-called stops) in a line. A linear gradient's X1, Y1, X2, Y2 properties are used to define the position (relative to the node) of the rectangular region that needs to be painted.

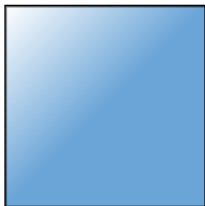
### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes array.
    }
}

```

```
DiagramNode node1 = new DiagramNode()
{
    // Position of the node
    OffsetX = 250,
    OffsetY = 250,
    // Size of the node
    Width = 100,
    Height = 100,
    Style = new NodeShapeStyle()
    {
        Gradient = new DiagramGradient()
        {
            //Start point of linear gradient
            X1 = 0,
            Y1 = 0,
            //End point of linear gradient
            X2 = 50,
            Y2 = 50,
            //Sets an array of stop objects
            Stops = new ObservableCollection<DiagramsGradientStop>()
            {
                new DiagramsGradientStop()
                {
                    Color = "white",
                    Offset = 0
                },
                new DiagramsGradientStop()
                {
                    Color = "#6BA5D7",
                    Offset = 100
                }
            },
            Type = GradientType.Linear
        }
    };
    NodeCollection.Add(node1);
}
```

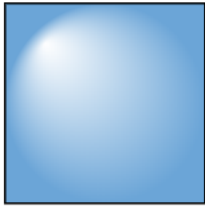


#### Radial gradient

[RadialGradient](#) defines a smooth transition between stops on a circle. A radial gradient properties are used to define the position (relative to the node) of the outermost or the innermost circle of the radial gradient.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
    DiagramNode node1 = new DiagramNode()
    {
        // Position of the node
        OffsetX = 250,
        OffsetY = 250,
        // Size of the node
        Width = 100,
        Height = 100,
        Style = new NodeShapeStyle()
        {
            Gradient = new DiagramGradient()
            {
                //Center point of inner circle
                Fx = 20,
                Fy = 20,
                //Center point of outer circle
                Cx = 50,
                Cy = 50,
                //Radius of a radial gradient
                R = 50,
                //Sets an array of stop objects
                Stops = new ObservableCollection<DiagramsGradientStop>()
                {
                    new DiagramsGradientStop()
                    {
                        Color = "white",
                        Offset = 0
                    },
                    new DiagramsGradientStop()
                    {
                        Color = "#6BA5D7",
                        Offset = 100
                    }
                },
                Type = GradientType.Radial
            }
        },
    };
    // Add node
    NodeCollection.Add(node1);
}
```



### Shadow

Diagram provides support to add [Shadow](#) effect to a node that is disabled, by default. It can be enabled with the constraints property of the node.

The following code shows how to draw shadow.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes array.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new NodeShapeStyle()
            {
                Fill = "#6BA5D7",
                StrokeColor = "white"
            },
            Constraints = NodeConstraints.Default | NodeConstraints.Shadow
        };
        NodeCollection.Add(node1);
    }
}
```



### Customizing shadow

The [Angle](#), [Distance](#), and [Opacity](#) of the shadow can be customized with the shadow property of the node.

The following code example illustrates how to customize shadow.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes array.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new NodeShapeStyle()
            {
                Fill = "#6BA5D7",
                StrokeColor = "white"
            },
            Constraints = NodeConstraints.Default | NodeConstraints.Shadow,
            // Custom Shadow of the node
            Shadow = new DiagramShadow()
            {
                Angle = 50,
                Opacity = 0.8,
                Distance = 9
            }
        };
        NodeCollection.Add(node1);
    }
}
```



*Icon*

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

---

Icon can be created only when the node has outEdges.

---

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

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

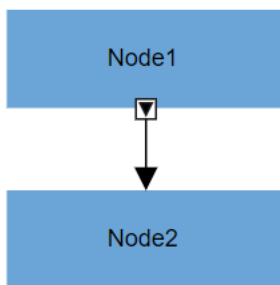
**ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection"
NodeDefaults="@NodeDefault" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>() { };
DiagramNode NodeDefault = new DiagramNode()
{
Width = 140,
Height = 50,
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
},
};
protected override void OnInitialized()
{
DiagramNode node1 = new DiagramNode()
{
Id = "Start",
OffsetX = 300,
OffsetY = 50,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Node1"
}
},
ExpandIcon = new NodeExpandIcon()
{
Shape = IconShapes.ArrowDown,
Width = 10,
Height = 10
},
CollapseIcon = new NodeCollapseIcon()
{
Shape = IconShapes.ArrowUp,
```

```

Width = 10,
Height = 10
};
NodeCollection.Add(node1);
DiagramNode node2 = new DiagramNode()
{
    Id = "Init",
    OffsetX = 300,
    OffsetY = 140,
    Annotations = new ObservableCollection<DiagramNodeAnnotation>()
    {
        new DiagramNodeAnnotation()
        {
            Content = "Node2"
        }
    },
};
NodeCollection.Add(node2);
DiagramConnector connector1 = new DiagramConnector()
{
    // Unique name for the connector
    Id = "connector1",
    // Source and Target node's name to which connector needs to be connected.
    SourceID = "Start",
    TargetID = "Init",
    Type = Segments.Orthogonal
};
ConnectorCollection.Add(connector1);
}
}

```



### Customizing expand icon

- Set the BorderColor, BorderWidth, and background color for an ExpandIcon using the BorderColor, BorderWidth, and Fill properties.
- Set a size for an expandIcon by using the Width and Height properties.
- The expand icon can be aligned relative to the node boundaries. It has Margin, Offset, HorizontalAlignment, and VerticalAlignment settings. It is quite tricky when all four alignments are used together but gives you more control over alignment.

### Customizing collapse icon

- Set the `borderColor`, `borderWidth`, background color for an `collapseIcon` using the [BorderColor](#), [BorderWidth](#), and [Fill](#) properties.
- Set a size for `collapseIcon` by using the [Width](#) and

[Height](#) properties.

- Like `expandIcon`, `collapseIcon` also can be aligned relative to the node boundaries. It has `Margin`, `Offset`, `HorizontalAlignment`, and `VerticalAlignment` settings. It is quite tricky when all four alignments are used together but gives you more control over alignment.

### Constraints

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

### Custom properties

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

The following code shows how to set the `AddInfo` value.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>();
    public List<NodeAddInfo> AddInfo = new List<NodeAddInfo>()
    {
        new NodeAddInfo()
        {
            Content = "NodeContent",
            ParentID = "diagram"
        }
    };
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes collection.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new NodeShapeStyle()
            {
                Fill = "#6BA5D7",
                StrokeColor = "white"
            },
            AddInfo = AddInfo
        }
    }
}
```



```
};  
// Add node  
NodeCollection.Add(node1);  
}  
public class NodeAddInfo {  
    public string ParentID;  
    public string Content;  
}  
}
```

**Note:** We can set any type of value for the AddInfo property.

#### Stack order

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

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

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams  
@using System.Collections.ObjectModel  
<SfDiagram Height="600px" Nodes="@NodeCollection">  
</SfDiagram>  
@code  
{  
    //Defines diagram's connector collection  
    public ObservableCollection<DiagramNode> NodeCollection;  
    protected override void OnInitialized()  
    {  
        NodeCollection = new ObservableCollection<DiagramNode>()  
        {  
            new DiagramNode(){  
                // Unique id of the node  
                Id = "Node1",  
                // Position of the node  
                OffsetX = 100,  
                OffsetY = 100,  
                // Size of the node  
                Width = 100,  
                Height = 100,  
                // Sets style of the node  
                Style = new NodeShapeStyle()  
                {  
                    Fill = "#6BA5D7",  
                    StrokeColor = "white"  
                },  
                //Specify the z-index value  
                ZIndex = 2,  
            },  
            new DiagramNode()  
            {  
                // Unique id of the node  
                Id = "Node2",  
                // Position of the node  
                OffsetX = 150,
```

```

OffsetY = 150,
// Size of the node
Width = 100,
Height = 100,
// Sets style of the node
Style = new NodeShapeStyle()
{
    Fill = "#6BA5D7",
    StrokeColor = "white"
},
//Specify the z-index value
ZIndex = 1,
}
};
}
}

```



### Data flow

You can find the flow in the diagram by using the [GetEdges](#) method. In this method, you can find what are all the connectors that are connected to the node.

### ASPX-CS

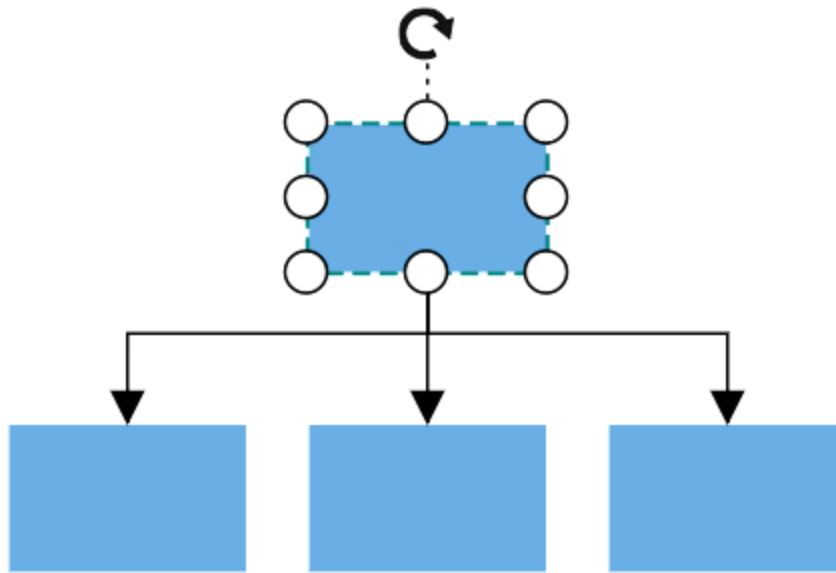
```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="UpdateEdges" @onclick="@UpdateEdges" />
<SfDiagram Height="600px" @ref="@Diagram" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" NodeDefaults="@NodeDefaults">
</SfDiagram>
@code{
// Reference of the diagram
SfDiagram Diagram;
// Define node and connector collection
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
// Set the default value of the node
DiagramNode NodeDefaults = new DiagramNode()
{
    // Size of the node
    Width = 80, Height = 50,
    // Style of the node
    Style = new NodeShapeStyle()

```

```
{
    Fill = "#6BA5D7",
    StrokeColor = "white"
},
};
protected override void OnInitialized()
{
    NodeCollection = new ObservableCollection<DiagramNode>() {
        new DiagramNode()
        {
            Id = "node1",
            // Position of the node
            OffsetX = 450,
            OffsetY = 100
        },
        new DiagramNode()
        {
            Id = "node2",
            // Position of the node
            OffsetX = 350,
            OffsetY = 200
        },
        new DiagramNode()
        {
            Id = "node3",
            // Position of the node
            OffsetX = 450,
            OffsetY = 200
        },
        new DiagramNode()
        {
            Id = "node4",
            // Position of the node
            OffsetX = 550,
            OffsetY = 200
        }
    };
    ConnectorCollection = new ObservableCollection<DiagramConnector>() {
        new DiagramConnector()
        {
            // Sets the unique id, source node, target node
            Id = "connector1",
            SourceID = "node1",
            TargetID = "node2",
            // Sets the type of the connector
            Type = Segments.Orthogonal
        },
        new DiagramConnector()
        {
            // Sets the unique id, source node, target node
            Id = "connector2",
            SourceID = "node1",
            TargetID = "node3",
            // Sets the type of the connector
            Type = Segments.Orthogonal
        },
        new DiagramConnector()
    }
}
```

```
{
// Sets the unique id, source node, target node
Id = "connector3", SourceID = "node1", TargetID = "node4",
// Sets the type of the connector
Type = Segments.Orthogonal
}
};
}
protected override async Task OnAfterRenderAsync(bool firstRender)
{
if (firstRender)
{
//OnAfterRenderAsync will be triggered after the component rendered.
await Task.Delay(1500);
// Select the node
Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[0] },
null);
}
}
public async Task UpdateEdges()
{
string NodeId = Diagram.SelectedItems.Nodes[0].Id;
// GetEdges method is used to get the connectors that connected to nodes.
string[] edges = await Diagram.GetEdges(NodeId, true);
for (int i = 0; i < edges.Length; i++)
{
// Get the connector from id
DiagramConnector connector = Diagram.GetConnector(edges[i]);
// Change the style of the connector
connector.Style.StrokeColor = "#1413F8";
connector.TargetDecorator.Style.StrokeColor = "#1413F8";
connector.TargetDecorator.Style.Fill = "#1413F8";
}
}
}
```



See also

- [How to get events when they interact the node](#)

### Interaction in Blazor Diagram Component

Diagram provides the support to drag, resize, or rotate the node interactively.

#### Select

A node can be select at runtime by using the [Select](#) method and clear the selection in the diagram using the [ClearSelection](#).

The following code explains how to select and clear selection in the diagram.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Select" @onclick="OnSelect">
<input type="button" value="UnSelect" @onclick="@UnSelect" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code {
// reference of the diagram
SfDiagram Diagram;
// To define node collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in nodes collection.
```

```
DiagramNode node1 = new DiagramNode()
{
    // Position of the node
    OffsetX = 250, OffsetY = 250,
    // Size of the node
    Width = 100, Height = 100,
    Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
// Add node
NodeCollection.Add(node1);
}
public void OnSelect()
{
    // Select the node
    Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[0] },
        null);
}
public void UnSelect()
{
    // clear selection in the diagram
    Diagram.ClearSelection();
}
}
```

And also the selection enable during the interaction.

- An element can be selected by clicking that element.
- When you select the elements in the diagram, the [SelectionChanged](#) event gets triggered and to do customization in this event.



### *Drag*

A node can be drag at runtime by using the [Drag](#) method.

The following code explains how to drag the node by using the drag method.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Drag" @onclick="OnDrag">
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code {
// reference of the diagram
SfDiagram Diagram;
// To define node collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in nodes collection.
DiagramNode node1 = new DiagramNode()
{
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
}
};
// Add node
NodeCollection.Add(node1);
}
public void OnDrag()
{
// Drag the node
Diagram.Drag(Diagram.Nodes[0], 10, 10);
}
}
```

And also the drag the node during the interaction.

- An object can be dragged by clicking and dragging it. When multiple elements are selected, dragging any one of the selected elements move all the selected elements.
- When you drag the elements in the diagram, the [OnPositionChange](#) event gets triggered and to do customization in this event.



### Resize

A node can be resize at runtime by using the [Scale](#) method.

The following code explains how to resize the node by using the scale method.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Resize" @onclick="OnResize">
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code {
// reference of the diagram
SfDiagram Diagram;
// To define node collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in nodes collection.
DiagramNode node1 = new DiagramNode()
{
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
}
};
// Add node
NodeCollection.Add(node1);
}
public void OnResize()
{
}
```



```
// Resize the node
Diagram.Scale(Diagram.Nodes[0], 0.5, 0.5, new PointModel() { X = 0, Y = 0
});
}
}
```

And also the resize the node during the interaction.

- Selector is surrounded by eight thumbs. When dragging these thumbs, the selected items can be resized.
- When one corner of the selector is dragged, the opposite corner will be in a static position.
- When a node is resized, the [OnSizeChange](#) event gets triggered.



While dragging and resizing, the objects are snapped towards the nearest objects to make better alignments. For better alignments, refer to [Snapping](#).

### Rotate

A node can be rotate at runtime by using the [Rotate](#) method. The following code explains how to rotate the node by using method.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Rotate" @onclick="OnRotate">
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code {
// reference of the diagram
SfDiagram Diagram;
// To define node collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in nodes collection.
DiagramNode node1 = new DiagramNode()
```

```
{
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
}
};
// Add node
NodeCollection.Add(node1);
}
public void OnRotate()
{
// Rotate the node
Diagram.Rotate(Diagram.Nodes[0], Diagram.Nodes[0].RotateAngle+10);
}
}
```

And also the rotate the node during the interaction.

- A rotate handler is placed above the selector. Clicking and dragging the handler in a circular direction lead to rotate the node.
- The node is rotated with reference to the static pivot point.
- Pivot thumb (thumb at the middle of the node) appears when rotating the node to represent the static point.



For more information about node interaction, refer to [Node Interaction](#).

*See also*

- [How to get events while the interact the node](#)
- [How to positioning the node](#)

- [How to customize the node](#)
- [How to interact the annotation in diagram](#)
- [How to interact the port in diagram](#)
- [How to interact the connector in diagram](#)

### Events in Blazor Diagram Component

Diagram provides some events support for node that triggers when interacting the node.

#### *Selection change*

The [SelectionChanged](#) events is triggered when select/unselect the node or connector. The [IBlazorSelectionChangeEventArgs](#) interface is used to get selection change event arguments.

The following code example explains how to get the selection change event in the diagram.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
  <DiagramEvents SelectionChanged="@SelectionChanged"></DiagramEvents>
</SfDiagram>
@code{
  // To define node collection
  public ObservableCollection<DiagramNode> NodeCollection = new
  ObservableCollection<DiagramNode>() { };
  protected override void OnInitialized()
  {
    // A node is created and stored in nodes collection.
    DiagramNode node1 = new DiagramNode()
    {
      // Position of the node
      OffsetX = 250,
      OffsetY = 250,
      // Size of the node
      Width = 100,
      Height = 100,
      // Apperence of the node
      Style = new NodeShapeStyle()
      {
        Fill = "#6BA5D7",
        StrokeColor = "white"
      }
    };
    // Add node
    NodeCollection.Add(node1);
    // SelectionChange event for diagram
    public void SelectionChanged(IBlazorSelectionChangeEventArgs args)
    {
      Console.WriteLine(args.NewValue.Nodes[0].Id);
    }
  }
}
```

### Position change

The [OnPositionChange](#) events is triggered when drag the node or connector in interaction. The [IBlazorDraggingEventArgs](#) interface is used to get position change event arguments.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
  <DiagramEvents OnPositionChange="@OnPositionChange"></DiagramEvents>
</SfDiagram>
@code{
  // To define node collection
  public ObservableCollection<DiagramNode> NodeCollection = new
  ObservableCollection<DiagramNode>() { };
  protected override void OnInitialized()
  {
    // A node is created and stored in nodes collection.
    DiagramNode node1 = new DiagramNode()
    {
      // Position of the node
      OffsetX = 250,
      OffsetY = 250,
      // Size of the node
      Width = 100,
      Height = 100,
      Style = new NodeShapeStyle()
      {
        Fill = "#6BA5D7",
        StrokeColor = "white"
      }
    };
    // Add node
    NodeCollection.Add(node1);
  }
  // Position change event for diagram
  public void OnPositionChange(IBlazorDraggingEventArgs args)
  {
    Console.WriteLine(args.NewValue.Nodes[0].Id);
  }
}
```

### Size change

The [OnSizeChange](#) events is triggered when resizing the node during the interaction. The [ISizeChangeEventArgs](#) interface is used to get size change event arguments.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
  <DiagramEvents OnSizeChange="@OnSizeChange"></DiagramEvents>
</SfDiagram>
@code{
  // To define node collection
```

```

public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
    // A node is created and stored in nodes collection.
    DiagramNode node1 = new DiagramNode()
    {
        // Position of the node
        OffsetX = 250,
        OffsetY = 250,
        // Size of the node
        Width = 100,
        Height = 100,
        Style = new NodeShapeStyle()
        {
            Fill = "#6BA5D7",
            StrokeColor = "white"
        }
    };
    // Add node
    NodeCollection.Add(node1);
}
// Size change event for diagram
public void OnSizeChange(ISizeChangeEventArgs args)
{
    Console.WriteLine(args.NewValue.Nodes[0].Id);
}
}

```

### Rotate change

The [OnRotateChange](#) events is triggered when resizing the node during the interaction. The [IRotationEventArgs](#) interface is used to get size change event arguments.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
    <DiagramEvents OnRotateChange="@OnRotateChange"></DiagramEvents>
</SfDiagram>
@code{
    // To define node collection
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes collection.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new NodeShapeStyle()

```

```
{
  Fill = "#6BA5D7",
  StrokeColor = "white"
};
// Add node
NodeCollection.Add(node1);
}
// Rotate change event for diagram
public void OnRotateChange(IRotationEventArgs args)
{
  Console.WriteLine(args.NewValue.Nodes[0].Id);
}
}
```

*See also*

- [How to interact the node in diagram](#)
- [How to get events when they interact the connector](#)
- [How to get events when they interact the annotation](#)

## Shapes in Blazor Diagram Component

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

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

<!-- markdownlint-disable MD033 -->

<!-- markdownlint-disable MD010 -->

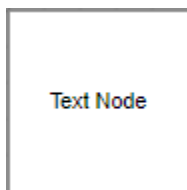
### Text

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize Diagram */
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
  //Initialize node collection with node
  ObservableCollection<DiagramNode> NodeCollection = new
  ObservableCollection<DiagramNode>()
  {
```

```
//Creates a text node
new DiagramNode()
{
    Id="node1",
    // Size of the node
    Height=100,
    Width=100,
    // Position of the node
    OffsetX=100,
    OffsetY=100,
    //Sets type of the shape as text
    Shape=new DiagramShape() {Type=Shapes.Text,Content="Text Node"}
};
```



### Image

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

The following code illustrates how an image node is created.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection" />
@code{
    //Initialize node collection with node
    ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>()
    {
        //Creates a image node
        new DiagramNode()
        {
            Id="node1",
            //Size of the node
            Height=100,
            Width=100,
            //Position of the node
            OffsetX=100,
            OffsetY=100,
            //Sets type of the shape as image
            Shape=new
            DiagramShape() {Type=Shapes.Image,Source="/diagram/images/syncfusion.png"}
        }
    };
};
```

}



### Base64 Encoded Image Into The Image Node:

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

## ASPX-CS

```
@using Fusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection"/>
@code{
//Initialize node collection with node
ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>()
{
//Creates a image node
new DiagramNode()
{
Id="node1",
//Size of the node
Height=100,
Width=100,
//Position of the node
OffsetX=100,
OffsetY=100,
//Sets type of the shape as image
Shape=new
DiagramShape() {Type=Shapes.Image,Source="data:image/gif;base64,R0lGODlhPQBEA
PeoAJosM//AwO/AwHvYZ/z595kzAP/s7P+goOXmv8+fhw/v739/f+8PD98fh/8mJl+fn/9ZWb8/P
zWlwv///6wWGbImApGTEMImIN9gUFCEm/gDALULDN8PAD6atYdCTX9gUNKlj8wZAKUsAOzZz+UMA
OsJAP/Z2ccMDA8PD/95eX5NWvsJCOVNQPtfX/8zM8+QePLl38MGBr8JCP+zs9myn/8GBqwpAP/Gx
gwJCpny78lzYLgjAJ8vAP9fX/+MjMUcAN8zm/9wcM8ZGCATEL+QePdZWf/29uc/P9cmJu9MTDImI
N+/r7+/vz8/P8VNVGNugV8AAF9fX8swMNgtAFldOICAgPNSUnNWSMQ5MBAQEJE3QPIGAM9AQMcGc
G9vb6MhJsEdGM8vLx8fh98AANIWAMuqeL8fABktePPQ0OM5OSydgF15jo+Pj/+pqcsTE78wMFNGG
LYmID4dGPvd3UBAQJmTkP+8vh9QUK+vr8ZWSHpzcJMILdwcLOGCHRQHxwcK9PT9DQ00/v70w5M
LypoG8wKOuwsP/g4P/Q0IcwKEswKMl8aJ9fX2xjdOtGRs/Pz+Dg4GImp8gIH0sKEAwKKmTiKZ8a
B/f39Wsl+LFt8dgUE9PT5x5aHBwcP+AgP+WltdgYMyZfytywz78AAAAAAD///8AAP9mzv///wAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACH5BAEAAKgALAAAAAA9AEQAAaj/AFEJHEiwo
MGDCBMqXMiwocAbBww4nEhxoyKUpzJGrMixogkfGUNqlNixJEIDB0SqHGmyJSojMlbKZOmyop0gm
3Oe2liTISKMOOpY7GnwY9CjIYcSRym0aVKSLme6nfq05QycVLPuhDrxBltYJUqNAq2bnWEBj6ZX
RuyxZyDRtqwnXvkACDD+euTeJmlKi7A73qNWtFiF+/ga95Gly2CJLDhweHMOUAAuOpLYDEgBxZ4
GRTlClfDnpkm+fOqd6DDjlazpITp0dtGCDhr+fVuCu3zlga49ijaokTZTo27uG7Gjn2P+hI8+PDPE
RoUB318bWbfAJf5sUNFCuGRTYUqV/3ogfXplrwLM6awJjiAAd2fm4ogXjjz56aypOOIde4OE5u/F9
xl99dlXnnGiHZWEYbGpsAEA3QXYnHwEFliKAgsqwJ8LPeiUXGwedCAKABACCN+EAlpYIIYaFlCdH
```



```

ytd51sGAJbo3onOpajiihlO92KHGaUXGwWjUBChjSPiWJuOO/LYIm4v1tXfE6J4gCSJEZ7YgRYUN
rkji9P55sF/ogxw5ZkSqIDaZBV6aSGYq/lGZplndkckz98xoICbTcIJGQAZcNmdmUc210hs35nCy
J58fgmIKX5RQGOZowxaZwYA+JaoKQwswGijBV4C6SiTUmpphMspJx9unX4KaimjDv9aaXOEBteBq
muuxgEHOlX6Kqx+yXqqBANsgCtit4FWQAEkrNbpq7HSOmtwag5w57GrmlJBASEU18ADjUYb3ADTi
nIttsGsb1oJFFa63bduimuqKB1keqwUhoCSK374wbujuvOSu4QG6UvxBRydcPksav++Ca6G8A6Pr1
x2kVMYHwsVxUALDq/krnrhPSOzXG1lUTIoffqGR7Goi2MAxbv6O2kEG56I7CSlRsEFKFVYovDJoI
RTg7sugNRDGqCJzJgcKE0ywc0ELm6KBCCJo8DIPFeCWNGcyqNFE06ToAfV0HBRgxsvLThHn1oddQ
MrXj5DyAQgjEHS AJMWZs3HPxT/QMbabi/iBCliMLEJKX2EEkomBAUCxRi42VDADxyTYDVogV+wS
ChqmKxEKCDAYFDFj4OmwbY7bDGdBhtrnTQYOigeChUmc1K3QTnAUfEgGFgAWt88hKA6aCRIXhxnQ
1yg3BCayK44EWdkUQcBBYEQChFXfCB776aQsG0BilQgQgE8qO26X1h8cEUep8ngRBnOy74E9QgRg
EAC8SvOfQkh7FDBDms43PmGoIiKUUEGkMEC/PJHgwx0xH74yx/3XnaYRJgMB8obxQW6kL9QYEJ0F
IFgByfIL7/IQAlvQwEpnAC7DtLNJCKUoO/w45c44GwCXiAFB/OXAATQryUxdN4LfFiwgjCNYg+kY
MIEFkCKDs6PKAIJouyGWMS1FSKJOMRB/BoIxYJIUXFUxNwoIkEKPAGCBZSQHQ1A2EWDfDEUvLyAD
j5AchSIQW6gul0bE/JG2VnZGfo4R4d0sdQoBAHhPjhIB94v/wRoRKQWGRHgrhGSQJxCS+0pCZbE
hAAOw==" }
}
};
}

```



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

Link 1: <http://stackoverflow.com/questions/4761711/local-image-in-canvas-in-chrome>

#### Image alignment

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

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

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection" />
@code{
//Initialize node collection with node
ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>()
{
//Creates a image node
new DiagramNode()
{
Id="node1",
//Size of the node
Height=100,
Width=100,

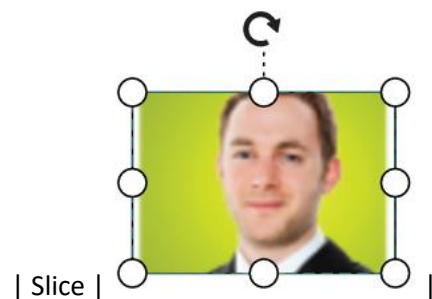
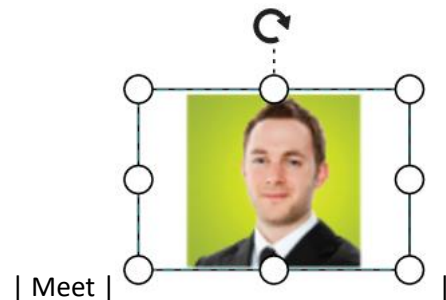
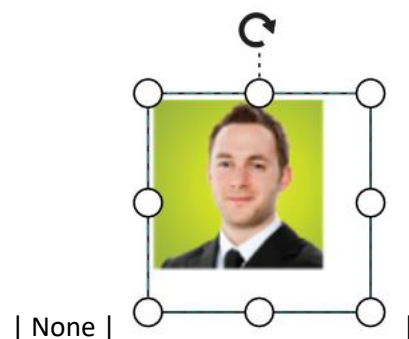
```

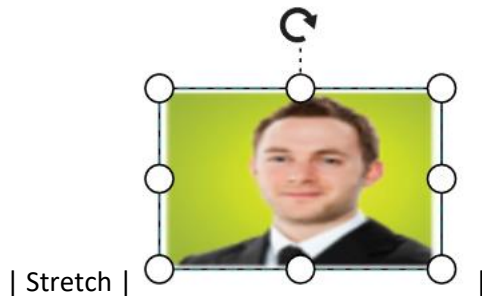
```
//Position of the node
OffsetX=100,
OffsetY=100,
//Sets type of the shape as image
Shape=new
DiagramShape() {Type=Shapes.Image,Source="/diagram/images/syncfusion.png",Scale=Stretch.Meet, Align = ImageAlignment.XMinYMin}
}
};
}
```

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

| Values | Images |

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





### HTML

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

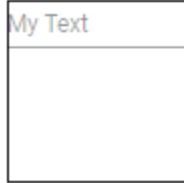
### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Add a Namespace for a syncfusion control used in Diagram HTML node */
@using Syncfusion.Blazor.Inputs
/* Initialize Diagram with node template */
<SfDiagram ModelType="@model" Height="600px" Nodes="@NodeCollection">
<DiagramTemplates>
<NodeTemplate>
@{
<SfTextBox Placeholder="My text"></SfTextBox>
}
</NodeTemplate>
</DiagramTemplates>
</SfDiagram>
@code{
public Type model = typeof(Node);
public class Node
{
public string Id { get; set; }
public double Width { get; set; }
}
//Initialize node collection with node
ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>()
{
//Creates an HTML node
new DiagramNode()
{
Id="node1",
//Size of the node
Height=100,
Width=100,
//Position of the node
OffsetX=400,
OffsetY=100,
//sets the type of the shape as HTML
Shape=new DiagramShape()
{
Type=Shapes.HTML,
```

```

}
}
};
}

```



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

### Basic shapes

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

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

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection" />
@code{
//Initialize node collection with node
ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>()
{
//Creates a basic shape node
new DiagramNode()
{
Id="node1",
//Size of the node
Height=100,
Width=100,
//Position of the node
OffsetX=100,
OffsetY=100,
//sets the type of the shape as basic
Shape=new DiagramShape() {Type=Shapes.Basic,BasicShape=BasicShapes.Rectangle}
}
};
}

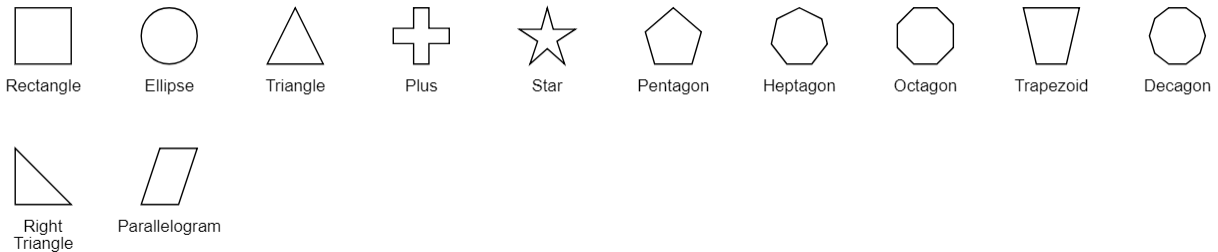
```

By default, the **Shape** property of the node is set as **basic**.

Default property for shape is Rectangle.

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

The list of basic shapes are as follows.



### Path

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize Diagram */
<SfDiagram Height="600px" Nodes="@NodeCollection" />
@code{
//Initialize node collection with node
ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>()
{
// Creates a path node
new DiagramNode()
{
Id="node1",
//Size of the node
Height=100,
Width=100,
//Position of the node
OffsetX=100,
OffsetY=100,
//Sets the type of the shape as path
Shape=new DiagramShape()
{
Type=Shapes.Path,
Data="M35.2441,25 L22.7161,49.9937 L22.7161,0.00657536 L35.2441,25 z
M22.7167,25 L-0.00131226,25 M35.2441,49.6337 L35.2441,0.368951 M35.2441,25
L49.9981,25"
}
}
};
}
```

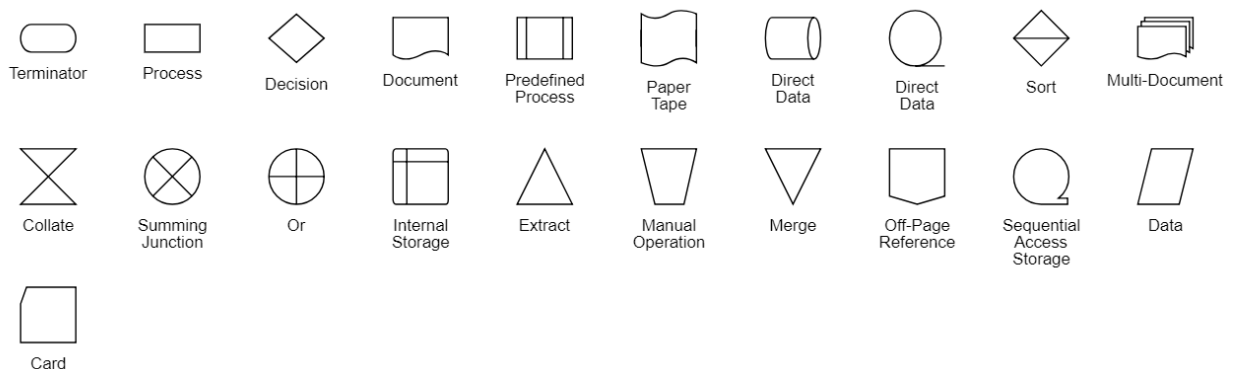
## Flow Shapes

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection" />
@code{
//Initialize node collection with node
ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>()
{
//Creates a flow shape node
new DiagramNode()
{
Id="node1",
//Size of the node
Height=100,
Width=100,
//Position of the node
OffsetX=100,
OffsetY=100,
//Sets the type of the shape as flow
Shape=new DiagramShape()
{
Type=Shapes.Flow,
FlowShape=FlowShapes.DirectData
}
}
};
}
```

The list of flow shapes are as follows.



## Bpmn Shapes

### BPMN Shapes in Blazor Diagram Component

BPMN(Business Process Model and Notation) shapes are used to represent the internal business procedure in a graphical notation and enable you to communicate the procedures in a standard manner. To create a BPMN shape, in the node property shape, type should be set to “Bpmn” and its shape should be set as any one of the built-in shapes.

The following code example explains how to create a simple business process.

#### ASPX-CS

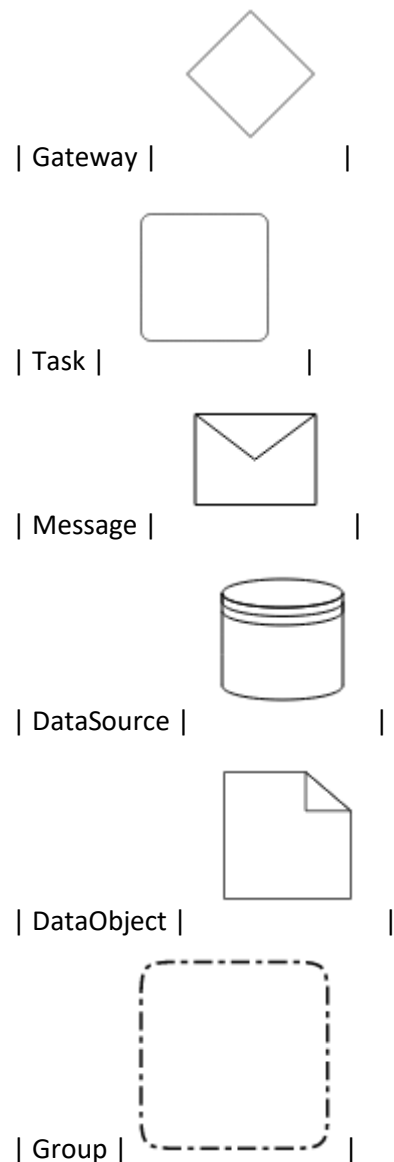
```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
// Position of the node
OffsetX = 100,
OffsetY = 100,
// Size of the node
Width = 100,
Height = 100,
// Unique Id of the node
Id = "node1",
Shape = new DiagramShape()
{
//Sets type to Bpmn
Type = Shapes.Bpmn
}
};
NodeCollection.Add(node);
}
}
```

The default value for the property [Shape](#) is “event”.

The list of BPMN shapes are as follows:

Shape	Image
-----	-----
Event	





### BPMN Event in Blazor Diagram Component

An [Event](#) is a common BPMN process model element that represents something that happens during a business process and is notated with a circle. The type of events are as follows:

- Start - indicates the beginning of the process and every business process start with an event.
- Intermediate - indicates the middle of the process.
- End - indicates the beginning of the process and every business process end with an event.

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
```



```

<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
//Unique Id of the node
Id = "node1",
Shape = new DiagramShape()
{
//Sets type to Bpmn and shape to Event
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Event,
// Set the event type to End
Event = new DiagramBpmnEvent() { Event = BpmnEvents.End }
}
};
NodeCollection.Add(node);
}
}

```

### BPMN event trigger

Event triggers are notated as icons inside the circle and they represent the specific details of the process. The Trigger property of the node allows you to set the type of trigger and by default, it is set to None. The following code example explains how to create a BPMN trigger.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,

```




























```

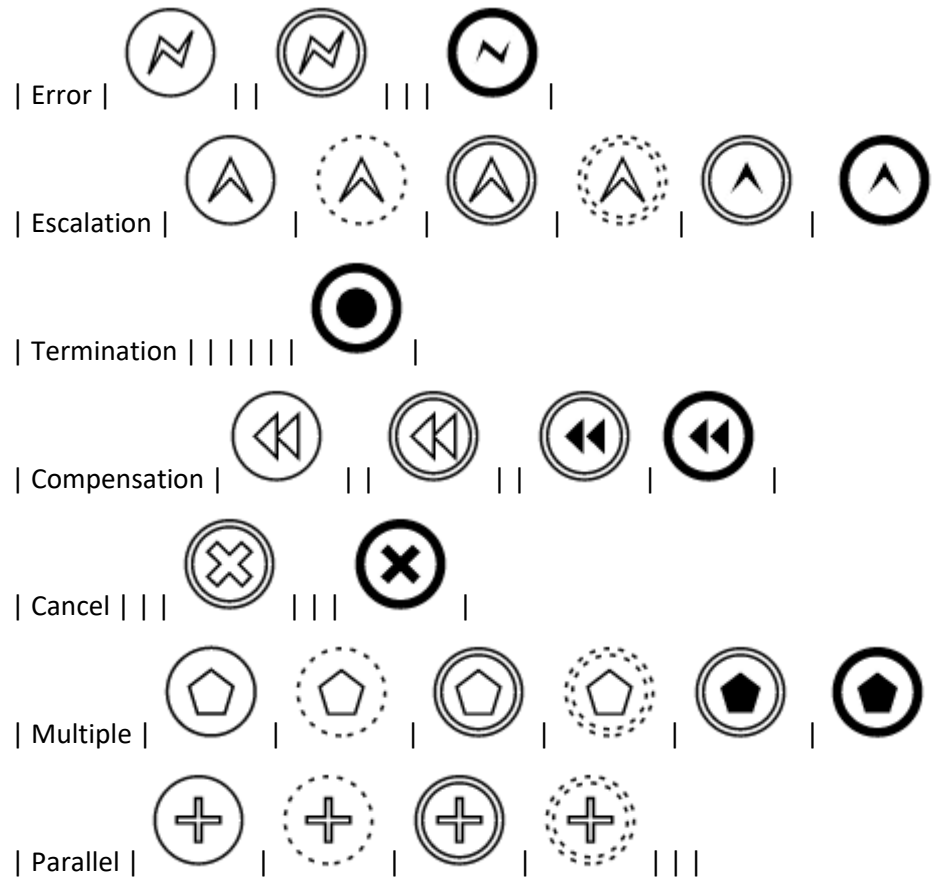
Height = 100,
//Unique Id of the node
Id = "node1",
Shape = new DiagramShape()
{
//Sets type to Bpmn and shape to Event
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Event,
// Set the event type to NonInterruptingIntermediate and set the trigger as
message
Event = new DiagramBpmnEvent()
{
Event = BpmnEvents.NonInterruptingIntermediate,
Trigger = BpmnTriggers.Message
}
}
};
NodeCollection.Add(node);
}
}

```

The following table illustrates the type of event triggers.

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

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



### BPMN Gateway in Blazor Diagram Component

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize Diagram */
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
```

```
//Unique Id of the node
Id = "node1",
Shape = new DiagramShape()
{
//Sets type to Bpmn and shape to Gateway
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Gateway,
//Sets type of the gateway to None
Gateway = new DiagramBpmnGateway()
{
Type = BpmnGateways.None
}
}
};
NodeCollection.Add(node);
}
```

---

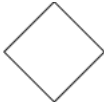
By default, the **Gateway** will be set to **None**.


---


There are several types of gateways as follows:

| Shape | Image |


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

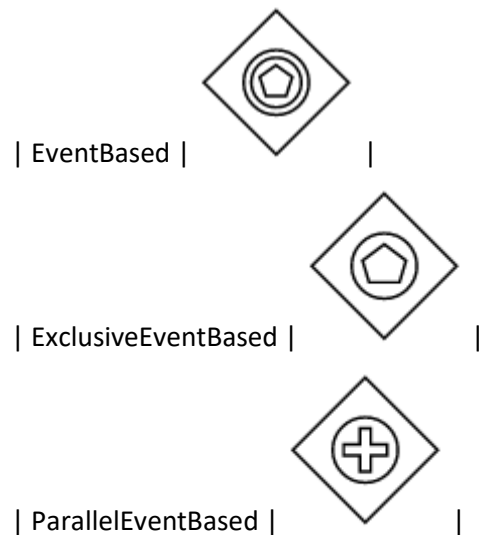
| None |  |

| Exclusive |  |

| Parallel |  |

| Inclusive |  |

| Complex |  |



### BPMN Activity in Blazor Diagram Component

The [Activity](#) is the task that is performed in a business process. It is represented by a rounded rectangle. There are two types of activities. They are listed as follows:

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
//Unique Id of the node
Id = "node1",
//sets the type of shape to Bpmn and shape to activity
Shape = new DiagramShape()
{
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Activity,
//Sets the activity type to task
Activity = new DiagramBpmnActivity() { Activity = BpmnActivities.Task },
```

```

}
};
NodeCollection.Add(node);
}
}

```

### BPMN activity task

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

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

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
//Unique Id of the node
Id = "node1",
//Sets the type to bpmn and shape to activity
Shape = new DiagramShape()
{
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Activity,
//Sets activity to Task
Activity = new DiagramBpmnActivity()
{
Activity = BpmnActivities.Task,
//Sets the type of the task to Send
Task = new DiagramBpmnTask() { Type = BpmnTasks.Send }
}
}
};
NodeCollection.Add(node);
}
}

```

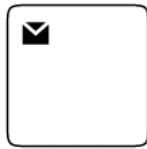
The various types of BPMN tasks are tabulated as follows.

| Shape | Image |

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



| Service |



| Send |



| Receive |



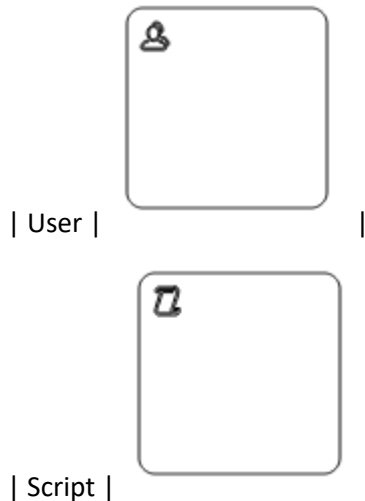
| Instantiating Receive |



| Manual |



| Business Rule |



### *BPMN activity sub process*

A [Sub-process](#) is a group of tasks that is used to hide or reveal details of additional levels using the **Collapsed** property.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize Diagram */
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
//Unique id of the node
Id = "node1",
Shape = new DiagramShape()
{
//Sets type to Bpmn and shape to Activity
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Activity,
Activity = new DiagramBpmnActivity()
{
//Sets activity to subprocess
Activity = BpmnActivities.SubProcess,
// Set collapsed of subprocess to true
SubProcess = new DiagramBpmnSubProcess() { Collapsed = true }
},

```



```

}
};
NodeCollection.Add(node);
}
}

```

The different types of subprocess are as follows:

- Event subprocess
- Transaction

#### Event sub process

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

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
// Position of the node
OffsetX = 100,
OffsetY = 100,
// Size of the node
Width = 100,
Height = 100,
// Unique id of the node
Id = "node1",
// Sets the type to Bpmn and shape to Activity
Shape = new DiagramShape()
{
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Activity,
//Sets activity to SubProcess
Activity = new DiagramBpmnActivity()
{
Activity = BpmnActivities.SubProcess,
//Sets the collapsed to true and type to Event
SubProcess = new DiagramBpmnSubProcess()
{
Collapsed = true,

```

```

Type = BpmnSubProcessTypes.Event,
//Sets event to Start and trigger to Message
Events = new ObservableCollection<DiagramBpmnSubEvent>()
{
    new DiagramBpmnSubEvent()
    {
        Event = BpmnEvents.Start, Trigger = BpmnTriggers.Message
    }
}
};
NodeCollection.Add(node);
}
}

```

### Transaction sub process

The **Transaction** is a set of activities that logically belong together that all contained activities must complete their parts of the transaction, otherwise the process is fail.

The execution result of a transaction is one of

- Successful Completion
- Unsuccessful Completion (Cancel)
- Hazard (Exception)

The **Events** property of subprocess allows you to represent these results as an event attached to the subprocess.

- The event object allows you to define the type of event by which the subprocess will be triggered. The name of the event can be defined to identify the event at runtime.
- The event's offset property is used to set the fraction or ratio (relative to parent) that defines the position of the event shape.
- The trigger property defines the type of the event trigger.
- You can also use define ports and labels to subprocess events by using the event's ports and labels properties.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
    NodeCollection = new ObservableCollection<DiagramNode>();
    DiagramNode node = new DiagramNode()
    {
        // Position of the node
    }
}
}

```

```

OffsetX = 100,
OffsetY = 100,
// Size of the node
Width = 100,
Height = 100,
// Unique id of the node
Id = "node1",
//Defines the type to BPMN and shape to activity
Shape = new DiagramShape()
{
    Type = Shapes.Bpmn,
    BpmnShape = BpmnShapes.Activity,
    //Sets the activity to subprocess
    Activity = new DiagramBpmnActivity()
    {
        Activity = BpmnActivities.SubProcess,
        //Sets collapsed to true and type to Transaction
        SubProcess = new DiagramBpmnSubProcess()
        {
            Collapsed = true,
            Type = BpmnSubProcessTypes.Transaction,
            //Sets offset and visible for cancel and offset for failure
            Transaction = new DiagramBpmnTransactionSubProcess()
            {
                Cancel = new CancelSubEvent()
                {
                    Visible = true,
                    Offset = new BpmnSubEventOffset() { X = 0.25, Y = 1 }
                },
                Failure = new FailureSubEvent()
                {
                    Offset = new BpmnSubEventOffset() { X = 0.75, Y = 1 }
                }
            },
        },
    };
    NodeCollection.Add(node);
}

```

### Process

[Processes](#) is an array collection that defines the children values for BPMN subprocess.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize Diagram */
<SfDiagram Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }

```

```
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = new DiagramNode()
{
Id = "Start",
Width = 50,
Height = 50,
Margin = new NodeMargin() { Left = 10, Top = 50 },
Shape = new DiagramShape()
{
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Event,
Event = new DiagramBpmnEvent() { Event = BpmnEvents.Start }
}
};
DiagramNode node2 = new DiagramNode()
{
Id = "End",
Width = 50,
Height = 50,
Margin = new NodeMargin() { Left = 200, Top = 50 },
Shape = new DiagramShape()
{
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Event,
Event = new DiagramBpmnEvent() { Event = BpmnEvents.End }
}
};
DiagramNode node3 = new DiagramNode()
{
Id = "Node1",
Width = 50,
Height = 50,
Margin = new NodeMargin() { Left = 100, Top = 200 },
Shape = new DiagramShape()
{
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Activity,
Activity = new DiagramBpmnActivity()
{
Activity = BpmnActivities.SubProcess,
SubProcess = new DiagramBpmnSubProcess()
{
Collapsed = false
}
}
},
Constraints = NodeConstraints.Default | NodeConstraints.AllowDrop
};
DiagramNode node4 = new DiagramNode()
{
Id = "ActivityProcessNode",
Width = 300,
```

```

Height = 300,
MaxHeight = 400,
MaxWidth = 400,
MinWidth = 200,
MinHeight = 200,
OffsetX = 200,
OffsetY = 200,
Shape = new DiagramShape()
{
    Type = Shapes.Bpmn,
    BpmnShape = BpmnShapes.Activity,
    Activity = new DiagramBpmnActivity()
    {
        Activity = BpmnActivities.SubProcess,
        SubProcess = new DiagramBpmnSubProcess()
        {
            Collapsed = false,
            Type = BpmnSubProcessTypes.Event,
            Processes = new string[] { "Start", "End", "Node1" }
        }
    },
    Constraints = NodeConstraints.Default | NodeConstraints.AllowDrop
};
NodeCollection.Add(node1);
NodeCollection.Add(node2);
NodeCollection.Add(node3);
NodeCollection.Add(node4);
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector1 = new DiagramConnector()
{
    Id = "Connector1",
    SourceID = "Start",
    TargetID = "Node1"
};
DiagramConnector connector2 = new DiagramConnector()
{
    Id = "Connector2",
    SourceID = "Node1",
    TargetID = "End"
};
ConnectorCollection.Add(connector1);
ConnectorCollection.Add(connector2);
}
}

```

### Loop

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

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

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel

```

```

@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
//Unique id of the node
Id = "node1",
//Defines the type to BPMN and shape to activity
Shape = new DiagramShape()
{
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Activity,
//Set the activity to subprocess
Activity = new DiagramBpmnActivity()
{
Activity = BpmnActivities.SubProcess,
//Sets collapsed to true and loop to standard
SubProcess = new DiagramBpmnSubProcess()
{
Collapsed = true,
Loop = BpmnLoops.Standard,
},
}
};
NodeCollection.Add(node);
}
}

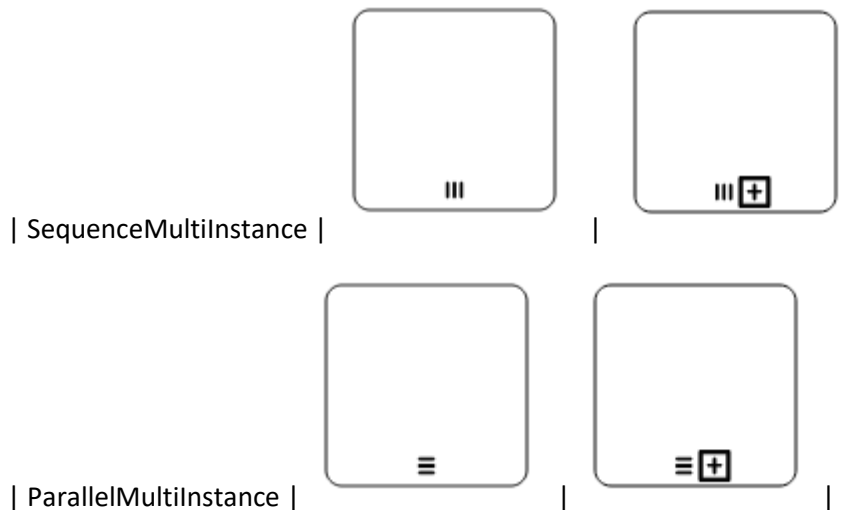
```

The following table contains various types of BPMN loops.

Loops	Task	Subprocess
-----	-----	-----



Standard	
----------	--



### Compensation

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize Diagram */
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = new DiagramNode()
{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
//Unique id of the node
Id = "node1",
//Defines the type to BPMN and shape to activity
Shape = new DiagramShape()
{
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Activity,
//Set the activity to task
Activity = new DiagramBpmnActivity()
{
Activity = BpmnActivities.Task,
//set compensation to true
Task = new DiagramBpmnTask()
{

```

```

Compensation = true,
},
}
};
DiagramNode node2 = new DiagramNode()
{
//Position of the node
OffsetX = 300,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
//Unique id of the node
Id = "node2",
//Defines the type to BPMN and shape to activity
Shape = new DiagramShape()
{
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Activity,
//Set the activity to SubProcess
Activity = new DiagramBpmnActivity()
{
Activity = BpmnActivities.SubProcess,
//Sets collapsed and compensation to true
SubProcess = new DiagramBpmnSubProcess()
{
Collapsed = true,
Compensation = true,
},
}
}
};
NodeCollection.Add(node1);
NodeCollection.Add(node2);
}
}

```

### Call

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

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize Diagram */
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()

```



```

{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
//Unique id of the node
Id = "node1",
//Defines the type to BPMN and shape to activity
Shape = new DiagramShape()
{
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.Activity,
//sets the activity to task
Activity = new DiagramBpmnActivity()
{
Activity = BpmnActivities.Task,
//Sets call to true
Task = new DiagramBpmnTask()
{
Call = true,
},
}
}
};
NodeCollection.Add(node);
}
}

```

### Ad-Hoc

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

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
}
}
}

```

```
//unique id of the node
Id = "node1",
//Defines the type to BPMN and shape to activity
Shape = new DiagramShape()
{
    Type = Shapes.Bpmn,
    BpmnShape = BpmnShapes.Activity,
    //Sets the activity to subprocess
    Activity = new DiagramBpmnActivity()
    {
        Activity = BpmnActivities.SubProcess,
        //sets collapsed and ad hoc to true
        SubProcess = new DiagramBpmnSubProcess()
        {
            Collapsed = true,
            Adhoc = true
        },
    }
};
NodeCollection.Add(node);
}
```

### Boundary

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
    //Defines diagram's node collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node = new DiagramNode()
        {
            //Position of the node
            OffsetX = 100,
            OffsetY = 100,
            //Size of the node
            Width = 100,
            Height = 100,
            //Unique Id of the node
            Id = "node1",
            //Sets type to Bpmn and shape to Activity
            Shape = new DiagramShape()
            {
                Type = Shapes.Bpmn,
```

```

BpmnShape = BpmnShapes.Activity,
//Sets activity to SubProcess
Activity = new DiagramBpmnActivity()
{
    Activity = BpmnActivities.SubProcess,
    //Sets collapsed to true and boundary to Call
    SubProcess = new DiagramBpmnSubProcess()
    {
        Collapsed = true,
        Boundary = BpmnBoundary.Call
    },
}
};
NodeCollection.Add(node);
}
}

```

The following table contains various types of BPMN boundaries.

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

### BPMN Data Object in Blazor Diagram Component

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

**ASPX-CS**

```

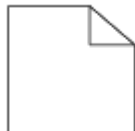
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize the Diagram*@
<SfDiagram ID="Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
//Unique Id of the node
Id = "node1",
//Sets type to Bpmn and shape to DataObject
Shape = new DiagramShape()
{
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.DataObject,
//Sets collection to true when Dataobject is not a Single instance
DataObject = new DiagramBpmnDataObject()
{
Collection = true,
Type = BpmnDataObjects.Input
}
}
};
NodeCollection.Add(node);
}
}

```

The following table contains various representation of the BPMN data object.

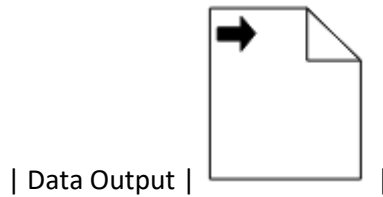
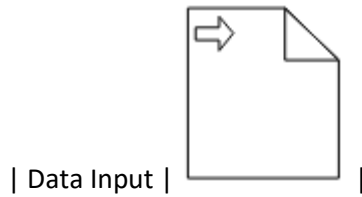
| Boundary | Image |

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



| Collection Data Object |

|

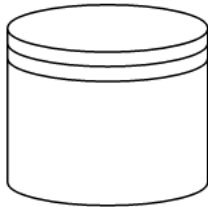


### BPMN Datasource in Blazor Diagram Component

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

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
//Unique Id of the node
Id = "node1",
//Sets type to Bpmn and shape to DataSource
Shape = new DiagramShape()
{
Type = Shapes.Bpmn,
BpmnShape = BpmnShapes.DataSource,
}
};
NodeCollection.Add(node);
}
}
```



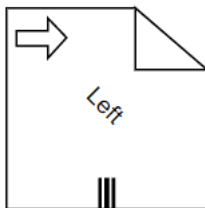
### BPMN Text Annotation in Blazor Diagram Component

- A BPMN object can be associated with a text annotation that does not affect the flow but gives details about objects within a flow. The annotation property of the node is used to connect an annotation element to the BPMN node.
- The annotation element can be switched from a BPMN node to another BPMN node simply by dragging the source end of the annotation connector into the other BPMN node.
- The annotation angle property is used to set the angle between the BPMN shape and the annotation.
- The annotation direction property is used to set the direction of the text annotation.
- To set the size for text annotation, use width and height properties.
- The annotation length property is used to set the distance between the BPMN shape and the annotation.
- The annotation text property defines the additional information about the flow object in a BPMN process.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Size of the node
Width = 100,
Height = 100,
//Unique Id of the node
Id = "node1",
```

```
//Sets type as Bpmn and shape as DataObject
Shape = new DiagramShape()
{
    Type = Shapes.Bpmn,
    BpmnShape = BpmnShapes.DataObject,
    //Sets collection as true when Dataobject is not a Single instance
    DataObject = new DiagramBpmnDataObject()
    {
        Collection = true,
        Type = BpmnDataObjects.Input
    }
},
//Sets the id, angle, and text for the annotation
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    new DiagramNodeAnnotation()
    {
        Id="Left",
        RotateAngle=45,
        Content="Left"
    }
};
NodeCollection.Add(node);
}
```



### BPMN Group in Blazor Diagram Component

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

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
```

```
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
    NodeCollection = new ObservableCollection<DiagramNode>();
    DiagramNode node = new DiagramNode()
    {
        //Position of the node
        OffsetX = 100,
        OffsetY = 100,
        //Size of the node
        Width = 100,
        Height = 100,
        //Unique Id of the node
        Id = "node1",
        //Sets type to Bpmn and shape to Group
        Shape = new DiagramShape()
        {
            Type = Shapes.Bpmn,
            BpmnShape = BpmnShapes.Group,
        }
    };
    NodeCollection.Add(node);
}
```



### BPMN Connectors in Blazor Diagram Component

The **BPMN Connectors** are lines that connect BPMN flow objects.

They are classified as follows.

- Association
- Sequence
- Message

#### Association

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






- Directional
- BiDirectional
- Default

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

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
//Unique Id of the connector
Id = "connector1",
// Start and end point of the connector
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 200 },
TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 200 },
//Sets the type to Bpmn, flow to Association and association to
bidirectional
Shape = new DiagramConnectorShape()
{
Type = ConnectionShapes.Bpmn,
BpmnFlow = BpmnFlows.Association,
Association = BpmnAssociationFlows.BiDirectional
}
};
ConnectorCollection.Add(connector);
}
}
```

The following table shows the visual representation of association flows.

Association	Image
Default	
Directional	
BiDirectional	

Note : The default value for the property Association is **Default**.

### Sequence

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

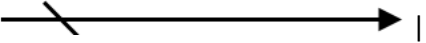
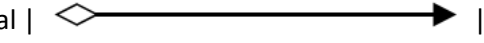
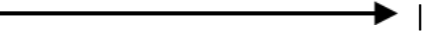
- Normal
- Conditional
- Default

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
//Unique Id of the connector
Id = "connector1",
// Start and end point of the connector
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 200 },
TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 200 },
//Sets type to Bpmn, flow to Sequence, and sequence to Conditional
Shape = new DiagramConnectorShape()
{
Type = ConnectionShapes.Bpmn,
BpmnFlow = BpmnFlows.Sequence,
Sequence = BpmnSequenceFlows.Conditional
}
};
ConnectorCollection.Add(connector);
}
```

The following table contains various representation of sequence flows.

Sequence	Image
Default	
Conditional	
Normal	

---

The default value for the property **Sequence** is **Normal**.

---

### Message

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

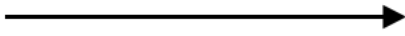

- InitiatingMessage
- NonInitiatingMessage
- Default


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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize Diagram */
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
//Unique Id of the connector
Id = "connector1",
// Start and end point of the connector
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 200 },
TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 200 },
//Sets type to Bpmn, flow to Message, and message to InitiatingMessage
Shape = new DiagramConnectorShape()
{
Type = ConnectionShapes.Bpmn,
BpmnFlow = BpmnFlows.Message,
Message = BpmnMessageFlows.InitiatingMessage
}
};
ConnectorCollection.Add(connector);
}
```

The following table contains various representation of message flows.

Message	Image
-----	-----
Default	
InitiatingMessage	

| NonInitiatingMessage |  |

The default value for the property **Message** is **Default**.

## UML Diagram Shapes in Blazor Diagram Component

### UML Class Diagram Shapes

Class diagram is used to represent the static view of an application. The class diagrams are widely used in the modelling of object-oriented systems because they are the only UML diagrams which can be mapped directly with object-oriented languages. Diagram supports to generate the class diagram shapes from business logic.

The UML class diagram shapes are explained as follows.

#### Class

- A class describes a set of objects that shares the same specifications of features, constraints, and semantics. To define a class object, you should define the classifier as [Class](#).
- Also, define the [Name](#), [Attributes](#), and [Methods](#) of the class using the class property of node.
- The attribute's [Name](#), [Type](#), and [Scope](#) properties allow you to define the name, data type, and visibility of the attribute.
- The method's [Name](#), [Parameters](#), [Type](#), and [Scope](#) properties allow you to define the name, parameter, return type, and visibility of the methods.
- The method parameters object properties allow you to define the name and type of the parameter.
- The following code example illustrates how to create a class.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initializes diagram control *@
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's Node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
Id = "Patient",
OffsetX = 200,
OffsetY = 200,
Shape = new DiagramShape()
{
Type = Syncfusion.Blazor.Diagrams.Shapes.UmlClassifier,
Classifier = ClassifierShape.Class,
//Define class object
ClassShape = new DiagramUmlClass()
{
Name = "Patient",
//Define class attributes
```

```

Attributes = new ObservableCollection<DiagramUmlClassAttribute>()
{
    new DiagramUmlClassAttribute() { Name = "accepted", Type = "Date" }
},
//Define class methods
Methods = new ObservableCollection<DiagramUmlClassMethod>()
{
    new DiagramUmlClassMethod()
    {
        Name = "getHistory", Type = "getHistory"
    }
}
};
//Add node
NodeCollection.Add(node);
}
}

```

## Interface

- An Interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations. To create an interface, define the classifier property as [Interface](#).
- Also, define the [Name](#), [Attributes](#), and [Methods](#) of the interface using the interface property of the node.
- The attribute's name, type, and scope properties allow you to define the name, data type, and visibility of the attribute.
- The method's name, parameter, type, and scope properties allow you to define the name, parameter, return type, and visibility of the methods.
- The method parameter object properties of name and type allows you to define the name and type of the parameter.
- The following code example illustrates how to create an interface.

## ASPX-CS

```

/* Initializes diagram control */
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's Node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
    NodeCollection = new ObservableCollection<DiagramNode>();
    DiagramNode node = new DiagramNode()
    {
        Id = "Patient",
        OffsetX = 200,
        OffsetY = 200,
        Shape = new DiagramShape()
    }
}
}

```

```

{
    Type = Syncfusion.Blazor.Diagrams.Shapes.UmlClassifier,
    Classifier = ClassifierShape.Interface,
    //Define interface object
    InterfaceShape = new DiagramUmlInterface()
    {
        Name = "Patient",
        //Define interface attributes
        Attributes = new ObservableCollection<DiagramUmlClassAttribute>()
        {
            new DiagramUmlClassAttribute() { Name = "owner", Type = "String[*]" }
        },
        //Define interface methods
        Methods = new ObservableCollection<DiagramUmlClassMethod>()
        {
            new DiagramUmlClassMethod()
            {
                Name = "deposit",
                Parameters = new ObservableCollection<DiagramMethodArguments>()
                {
                    new DiagramMethodArguments() { Name = "amount", Type = "Dollars" }
                }
            }
        }
    };
    //Add node
    NodeCollection.Add(node);
}
}

```

## Enumeration

- To define an enumeration, define the classifier property of node as [Enumeration](#). Also, define the name and members of the enumeration using the enumeration property of the node.
- You can set a name for the enumeration members collection using the name property of members collection.
- The following code example illustrates how to create an enumeration.

## ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initializes diagram control */
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    //Defines diagram's Node collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node = new DiagramNode()
    }
}

```

```

{
    Id = "Patient",
    OffsetX = 200,
    OffsetY = 200,
    Shape = new DiagramShape()
    {
        Type = Syncfusion.Blazor.Diagrams.Shapes.UmlClassifier,
        Classifier = ClassifierShape.Enumeration,
        //Define enumeration object
        EnumerationShape = new DiagramUmlEnumeration()
        {
            Name = "AccountType",
            //set the members of enumeration
            Members = new ObservableCollection<DiagramUmlEnumerationMember>()
            {
                new DiagramUmlEnumerationMember()
                {
                    Name = "Checking Account"
                },
                new DiagramUmlEnumerationMember()
                {
                    Name = "Savings Account"
                },
                new DiagramUmlEnumerationMember()
                {
                    Name = "Credit Account"
                }
            }
        };
        NodeCollection.Add(node);
    }
}

```

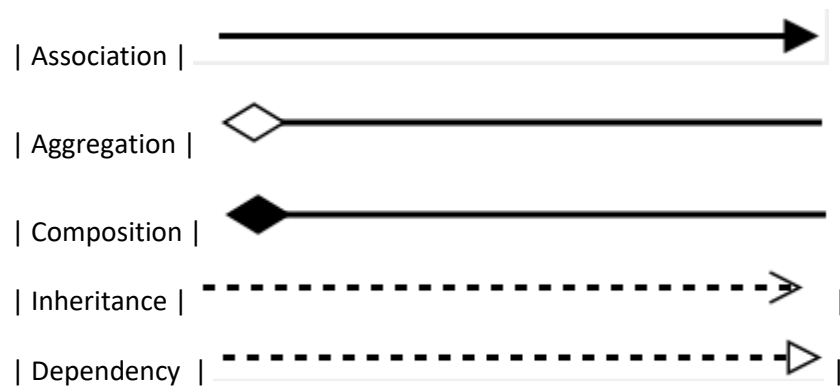
### Connector shapes

- The connector shape property defines the role or meaning of the connector.
- The different types of connector shapes are BPMN, [UMLClassifier](#) and [UMLActivity](#) and can render these shapes by setting the connector shape type property.
- The type of flow shapes in a BPMN process are sequence, association, and message.

### Relationships

- A relationship is a general term covering the specific types of logical connections found on class diagrams.
- The list of Relationships is demonstrated as follows.

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



### Association

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

1. Directional
2. BiDirectional

The association property allows you to define the type of association. The default value of association is [Directional](#). The following code example illustrates how to create an association.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initializes diagram control */
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
Id = "Connector1",
Type = Segments.Straight,
//Define connector start and end points
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 300 },
Shape = new DiagramConnectorShape()
{
Type = ConnectionShapes.UmlClassifier,
Relationship = ClassifierShape.Association,
//Define type of association
Association = BpmnAssociationFlows.BiDirectional
}
};
//Add connector
ConnectorCollection.Add(connector);
}
```



```
}
```

### Aggregation

Aggregation is a binary association between a property and one or more composite objects which group together a set of instances. Aggregation is decorated with a hollow diamond. To create an aggregation shape, define the relationship as “aggregation”.

The following code example illustrates how to create an aggregation.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initializes diagram control */
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
Id = "Connector1",
Type = Segments.Straight,
Shape = new DiagramConnectorShape()
{
Type = ConnectionShapes.UmlClassifier,
//Set an relationship for connector
Relationship = ClassifierShape.Aggregation
},
//Define connector start and end points
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 300 }
};
ConnectorCollection.Add(connector);
}
}
```

### Composition

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

The following code example illustrates how to create a composition.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initializes diagram control */
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
//Defines diagram's connector collection
```

```

public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
Id = "Connector1",
Type = Segments.Straight,
Shape = new DiagramConnectorShape()
{
Type = ConnectionShapes.UmlClassifier,
//Set an relationship for connector
Relationship = ClassifierShape.Composition
},
//Define connector start and end points
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 300 }
};
//Add connector
ConnectorCollection.Add(connector);
}
}

```

### Dependency

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

The following code example illustrates how to create an dependency.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initializes diagram control *@
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
Id = "Connector1",
Type = Segments.Straight,
Shape = new DiagramConnectorShape()
{
Type = ConnectionShapes.UmlClassifier,
//Set an relationship for connector
Relationship = ClassifierShape.Dependency
},
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },

```

```
TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 300 }  
};  
ConnectorCollection.Add(connector);  
}  
}
```

### Inheritance

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

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

The following code example illustrates how to create an inheritance.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams  
@using System.Collections.ObjectModel  
@* Initializes diagram control *@  
<SfDiagram Height="600px" Connectors="@ConnectorCollection">  
</SfDiagram>  
@code{  
    //Defines diagram's connector collection  
    public ObservableCollection<DiagramConnector> ConnectorCollection { get;  
    set; }  
    protected override void OnInitialized()  
    {  
        ConnectorCollection = new ObservableCollection<DiagramConnector>();  
        DiagramConnector connector = new DiagramConnector()  
        {  
            Id = "Connector1",  
            Type = Segments.Straight,  
            Shape = new DiagramConnectorShape()  
            {  
                Type = ConnectionShapes.UmlClassifier,  
                //Set an relationship for connector  
                Relationship = ClassifierShape.Inheritance  
            },  
            //Define connector start and end points  
            SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },  
            TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 300 }  
        };  
        ConnectorCollection.Add(connector);  
    }  
}
```

### Multiplicity

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

1. OneToOne
2. ManyToOne
3. OneToMany

#### 4. ManyToMany

- By default the multiplicity will be considered as “OneToOne”.
- The multiplicity property in UML allows you to specify large number of elements or some collection of elements.
- The shape multiplicity’s source property is used to set the source label to connector and the target property is used to set the target label to connector.
- To set an optionality or cardinality for the connector source label, use optional property.
- The **LowerBounds** and **UpperBounds** could be natural constants or constant expressions evaluated to natural (non negative) number. Upper bound could be also specified as asterisk ‘\*’ which denotes unlimited number of elements. Upper bound should be greater than or equal to the lower bound.
- The following code example illustrates how to customize the multiplicity.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
Id = "Connector1",
Type = Segments.Straight,
Shape = new DiagramConnectorShape()
{
Type = ConnectionShapes.UmlClassifier,
//Set an relationship for connector
Relationship = ClassifierShape.Dependency,
Multiplicity = new DiagramClassifierMultiplicity()
{
//Set multiplicity type
Type = Multiplicity.OneToMany,
//Set source label to connector
Source = new SourceMultiplicityLabel()
{
Optional = true,
LowerBounds = "89",
UpperBounds = "67"
},
//Set target label to connector
Target = new TargetMultiplicityLabel()
{
Optional = true,
LowerBounds = "78",
UpperBounds = "90"
}
}
}
}
```

```

},
//Define connector start and end points
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 300 }
};
ConnectorCollection.Add(connector);
}
}

```

### UmlActivity diagram

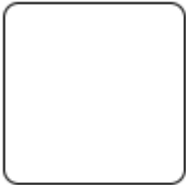
Activity diagram is basically a flowchart to represent the flow from one activity to another. The activity can be described as an operation of the system.

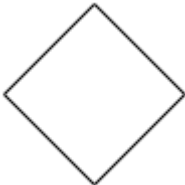
The purpose of an activity diagram can be described as follows.

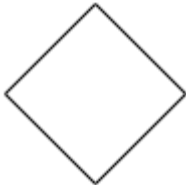
1. Draw the activity flow of a system.
2. Describe the sequence from one activity to another.
3. Describe the parallel, branched, and concurrent flow of the system.





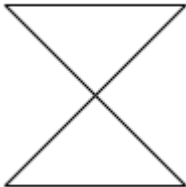
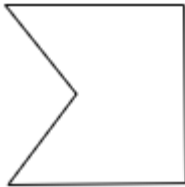
To create a UmlActivity, define type as "UmlActivity" and the list of built-in shapes as demonstrated as follows and it should be set in the "shape" property.

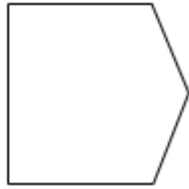
Shape	Image
-----	-----

	
Action	

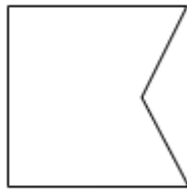
	
Decision	

	
MergeNode	

InitialNode	
FinalNode	
ForkNode	
JoinNode	
TimeEvent	
AcceptingEvent	



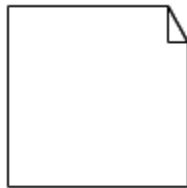
| SendSignal |



| ReceiveSignal |



| StructuredNode |



| Note |

The following code illustrates how to create a UMLActivity shapes.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initializes diagram control */
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Defines diagram's connector collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
public DiagramConstraints diagramConstraints = DiagramConstraints.Default;
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
```

```

Id = "UmlDiagram",
//Set node size
Width = 100,
Height = 100,
//position the node
OffsetX = 200,
OffsetY = 200,
Shape = new DiagramShape()
{
    Type = Syncfusion.Blazor.Diagrams.Shapes.UmlActivity,
    //Define UmlActivity shape
    UmlActivityShape = UmlActivityShapes.Action
}
};
NodeCollection.Add(node);
}
}

```

#### *UMLActivity connector*

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

The following code illustrates how to create a UMLActivity connector.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initializes diagram control */
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
    ConnectorCollection = new ObservableCollection<DiagramConnector>();
    DiagramConnector connector = new DiagramConnector()
    {
        Id = "Connector1",
        Type = Segments.Straight,
        Shape = new DiagramConnectorShape()
        {
            Type = ConnectionShapes.UmlActivity,
            UmlActivityFlow = UmlActivityFlows.Exception
        },
        //Define connector start and end points
        SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
        TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 }
    };
    ConnectorCollection.Add(connector);
}
}

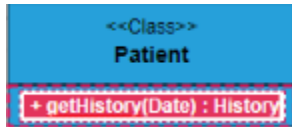
```



### Editing

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

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



### Connectors

#### Actions of Connectors in Blazor Diagram Component

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

#### Create connector

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

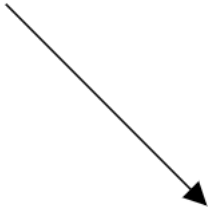
#### Add connectors through connectors collection

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

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

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    protected override void OnInitialized()
    {
        DiagramConnector diagramConnector = new DiagramConnector()
        {
            // Set the source and target point of the connector
            SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
            TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
            // Type of the connector segments
            Type = Segments.Straight,
        };
        //Add the connector into connectors's collection.
        ConnectorCollection.Add(diagramConnector);
    }
}
```



### *Add connectors at runtime*

You can add a connector at runtime by using the server-side method `AddConnector` in the `Diagram` component. The following code explains how to add connectors at runtime.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Add Connector" @onclick="@AddConnector">
<SfDiagram @ref="@Diagram" Height="600px">
</SfDiagram>
@code
{
    SfDiagram Diagram;
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    public void AddConnector()
    {
        DiagramConnector diagramConnector = new DiagramConnector()
        {
            Id = "Connector1",
            SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
            TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
            TargetDecorator = new ConnectorTargetDecorator()
            {
                Shape = DecoratorShapes.Arrow,
                Style = new DecoratorShapeStyle()
                {
                    Fill = "#6f409f",
                    StrokeColor = "#6f409f",
                    StrokeWidth = 1
                }
            },
            Style = new ConnectorShapeStyle()
            {
                StrokeColor = "#6f409f",
                StrokeWidth = 1
            },
            Type = Segments.Straight,
        };
        //Add the connector at the run time.
```

```
Diagram.AddConnector(diagramConnector);
}
}
```

Also, the connector can be added at runtime by using the **Add** method.

### CSHARP

```
// Add connector at runtime
public void AddConnector()
{
    DiagramConnector Connector2 = new DiagramConnector()
    {
        Id = "Connector2",
        // Set the source and target point of the connector
        SourcePoint = new ConnectorSourcePoint() { X = 200, Y = 100 },
        TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 200 },
    };
    ConnectorCollection.Add(Connector2);
}
```

### Connectors from the palette

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

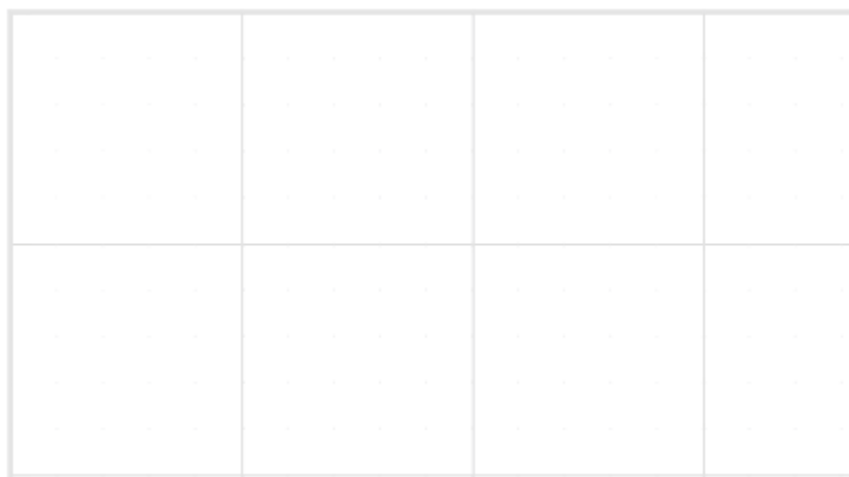
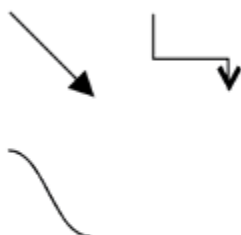
For more information about adding connectors from symbol palette, refer to the [Symbol Palette](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initializes the symbol palette */
<div style="width: 200px; float: left">
<SfSymbolPalette Height="600px" @ref="@SymbolPalette" Palettes="@Palettes">
</SfSymbolPalette>
</div>
<SfDiagram ID="diagram" @ref="@diagram" Width="500px" Height="600px">
</SfDiagram>
@code{
    SfDiagram diagram;
    SfSymbolPalette SymbolPalette;
    public ObservableCollection<Object> Connectors { get; set; }
    public ObservableCollection<SymbolPalettePalette> Palettes;
    protected override void OnInitialized()
    {
        Palettes = new ObservableCollection<SymbolPalettePalette>();
        //Initializes connector symbols for the symbol palette
        Connectors = new ObservableCollection<Object>()
        {
            new DiagramConnector()
            {
                Id = "Link1",
                // Sets the preview size
                PreviewSize = new SymbolSizeModel() { Width = 100, Height = 100 },
                // Type of the connector segments
                Type = Segments.Straight,
            }
        }
    }
}
```

```
// Set the source and target point of the connector
SourcePoint = new ConnectorSourcePoint() { X = 0, Y = 0 },
TargetPoint = new ConnectorTargetPoint() { X = 60, Y = 60 }
},
new DiagramConnector()
{
    Id = "Link2",
    // Sets the preview size
    PreviewSize = new SymbolSizeModel() { Width = 100, Height = 100 },
    // Type of the connector segments
    Type = Segments.Orthogonal,
    // Set the source and target point of the connector
    SourcePoint = new ConnectorSourcePoint() { X = 0, Y = 0 },
    TargetPoint = new ConnectorTargetPoint() { X = 60, Y = 60 },
    Style = new ConnectorShapeStyle() { StrokeWidth = 1 },
    // Sets the shape for target decorator
    TargetDecorator = new ConnectorTargetDecorator() { Shape =
DecoratorShapes.OpenArrow }
},
new DiagramConnector()
{
    Id = "Link3",
    // Sets the preview size
    PreviewSize = new SymbolSizeModel() { Width = 100, Height = 100 },
    // Type of the connector segments
    Type = Segments.Bezier,
    // Set the source and target point of the connector
    SourcePoint = new ConnectorSourcePoint() { X = 0, Y = 0 },
    TargetPoint = new ConnectorTargetPoint() { X = 60, Y = 60 },
    // Sets the shape for target decorator
    TargetDecorator = new ConnectorTargetDecorator() { Shape =
DecoratorShapes.None }
}
};
Palettes.Add(new SymbolPalettePalette() { Id = "Connectors", Expanded =
true, Symbols = Connectors, Title = "Connectors" });
}
}
```

### Connectors



*Draw connectors using drawing object*

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

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

*Remove connectors at runtime*

A connector can be removed from the diagram at runtime by using the [Remove](#) method.

The following code shows how to remove a connector at runtime.

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Remove Connector" @onclick="@RemoveConnector" />
<SfDiagram @ref="@Diagram" Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
// Reference of the diagram
SfDiagram Diagram;
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
DiagramConnector diagramConnector = new DiagramConnector()
{
Id = "Connector1",
// Set the source and target point of the connector
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
Fill = "#6f409f",
StrokeColor = "#6f409f",
StrokeWidth = 1
}
}
},
}
```

```
// Style of the connector segment
Style = new ConnectorShapeStyle()
{
    StrokeColor = "#6f409f",
    StrokeWidth = 1
},
// Type of the connector
Type = Segments.Straight,
};
//Add the connector at the run time.
ConnectorCollection.Add(diagramConnector);
}
// Remove Node at runtime
public void RemoveConnector()
{
    Diagram.Remove(ConnectorCollection[0]);
}
}
```

A connector can be removed from the diagram by using the native **RemoveAt** method. Refer to the following example that shows how to remove the connector at runtime.

### **CSHARP**

```
public void RemoveConnector()
{
    ConnectorCollection.RemoveAt(0);
}
```

### *Update connectors at runtime*

You can change any connector's properties at runtime.

The following code example explains how to change the connector properties.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" @ref="@Diagram" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    SfDiagram Diagram;
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    protected override void OnInitialized()
    {
        DiagramConnector diagramConnector = new DiagramConnector()
        {
            SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
            TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
            TargetDecorator = new ConnectorTargetDecorator()
            {
                Shape = DecoratorShapes.Arrow,
                Style = new DecoratorShapeStyle()
            }
        }
    }
}
```

```

{
    Fill = "#6f409f",
    StrokeColor = "#6f409f",
    StrokeWidth = 1
}
},
Style = new ConnectorShapeStyle()
{
    StrokeColor = "#6f409f",
    StrokeWidth = 1
},
Type = Segments.Straight,
};
//Add the connector into connectors's collection.
ConnectorCollection.Add(diagramConnector);
}
public void AddConnector()
{
    Diagram.BeginUpdate();
    Diagram.Connectors[0].SourcePoint.X = 50;
    Diagram.Connectors[0].SourcePoint.Y = 50;
    Diagram.EndUpdate();
}
}

```

### Connections

The connectors are used to create a link between two points, nodes or ports to represent the relationships between them.

### Connections with nodes

The [SourceID](#) and [TargetID](#) properties allow to define the nodes to be connected.

The following code example illustrates how to connect two nodes.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's nodes collection
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>();
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    public DiagramConstraints diagramConstraints = DiagramConstraints.Default;
    protected override void OnInitialized()
    {
        //Creates source node
        DiagramNode node1 = new DiagramNode()
        {
            OffsetX = 100,

```

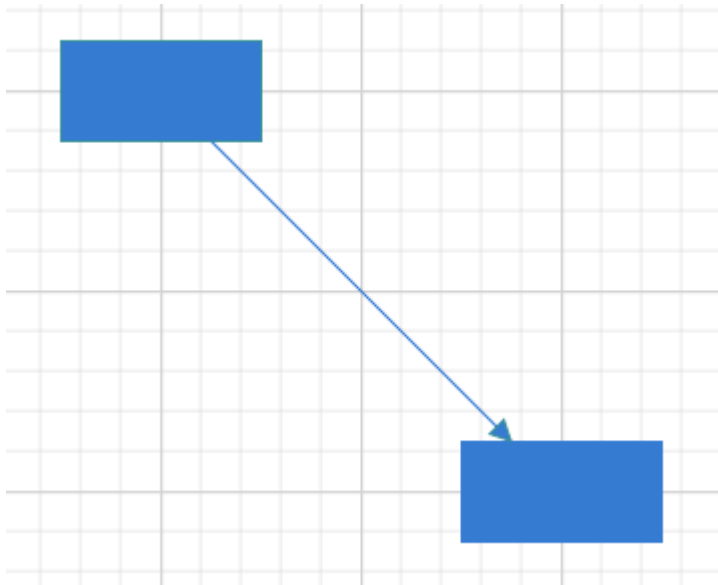
```
OffsetY = 100,
Height = 50,
Width = 100,
Id = "node1",
Shape = new DiagramShape()
{
    Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
    BasicShape = BasicShapes.Rectangle
},
Style = new NodeShapeStyle()
{
    Fill = "#37909A",
    StrokeColor = "#37909A",
},
};
//Add node into node's collection
NodeCollection.Add(node1);
//Create a target node
DiagramNode node2 = new DiagramNode()
{
    OffsetX = 300,
    OffsetY = 300,
    Height = 50,
    Width = 100,
    Id = "node2",
    Shape = new DiagramShape()
    {
        Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
        BasicShape = BasicShapes.Rectangle
    },
    Style = new NodeShapeStyle()
    {
        Fill = "#37909A",
        StrokeColor = "#37909A",
    },
};
//Add node into node's collection
NodeCollection.Add(node2);
//create the connector with source node and target node id.
DiagramConnector diagramConnector = new DiagramConnector()
{
    //Source node id of the connector.
    SourceID = "node1",
    //Target node id of the connector.
    TargetID = "node2",
    TargetDecorator = new ConnectorTargetDecorator()
    {
        Shape = DecoratorShapes.Arrow,
        Style = new DecoratorShapeStyle()
        {
            Fill = "#37909A",
            StrokeColor = "#37909A",
            StrokeWidth = 1
        }
    },
    Style = new ConnectorShapeStyle()
    {
```



```

StrokeColor = "#37909A",
StrokeWidth = 1
},
Type = Segments.Straight,
};
//Adding connector into connector's collection
ConnectorCollection.Add(diagramConnector);
}
}

```



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

When you remove both InConnect and OutConnect [NodeConstraints](#) from **Default**, the node restricts connector to establish connection in it. The following code illustrates how to disable InConnect constraints.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
//Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
protected override void OnInitialized()
{
//Creates source node
DiagramNode node1 = new DiagramNode()
{
OffsetX = 100,
OffsetY = 100,
}
}
}

```

```

Height = 50,
Width = 100,
Shape = new DiagramShape()
{
    Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
    BasicShape = BasicShapes.Ellipse
},
Style = new NodeShapeStyle()
{
    Fill = "#37909A",
    StrokeColor = "#37909A",
},
};
//Add node into node's collection
NodeCollection.Add(node1);
//To disable the inconnect constraints to node.
node1.Constraints = NodeConstraints.Default & ~NodeConstraints.InConnect;
}
}

```

### Connections with ports

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

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

### ASPX-CS

```

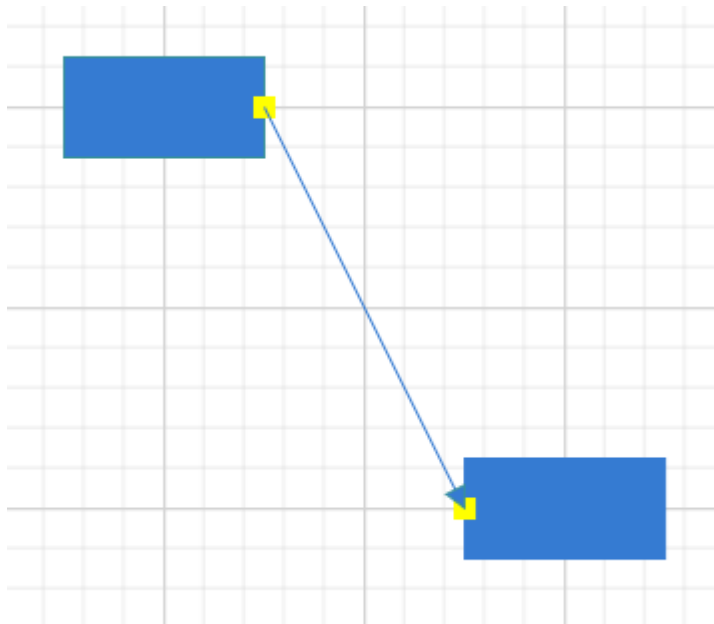
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram @ref="@diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's nodes collection
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>();
    public SfDiagram diagram;
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    public DiagramConstraints diagramConstraints = DiagramConstraints.Default;
    protected override void OnInitialized()
    {
        //Create a source node
        DiagramNode node1 = new DiagramNode()
        {
            OffsetX = 100,
            OffsetY = 100,
            Height = 50,
            Width = 100,
            Id = "node1",
            Shape = new DiagramShape()
            {
                Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,

```

```
BasicShape = BasicShapes.Rectangle
},
Style = new NodeShapeStyle()
{
Fill = "#37909A",
StrokeColor = "#37909A",
},
//Create the port for source node
Ports = new ObservableCollection<DiagramPort>()
{
new DiagramPort()
{
Id = "port1",
Offset = new Syncfusion.Blazor.Diagrams.NodePortOffset() { X = 1, Y = 0.5 },
Height = 10,
Width = 10,
Visibility = PortVisibility.Visible,
Style = new PortShapeStyle()
{
Fill = "yellow",
StrokeColor = "yellow"
}
},
},
};
//Add node into node's collection
NodeCollection.Add(node1);
//Create a target node
DiagramNode node2 = new DiagramNode()
{
OffsetX = 300,
OffsetY = 300,
Height = 50,
Width = 100,
Id = "node2",
Shape = new DiagramShape()
{
Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
BasicShape = BasicShapes.Rectangle
},
Style = new NodeShapeStyle()
{
Fill = "#37909A",
StrokeColor = "#37909A",
},
//Create the port for target node.
Ports = new ObservableCollection<DiagramPort>()
{
new DiagramPort()
{
Id = "port2",
Offset = new Syncfusion.Blazor.Diagrams.NodePortOffset() { X = 0, Y = 0.5 },
Height = 10,
Width = 10,
Visibility = PortVisibility.Visible,
Style = new PortShapeStyle()
{

```

```
Fill = "yellow",
StrokeColor = "yellow"
},
},
};
//Add node into node's collection
NodeCollection.Add(node2);
//create the connector from port to port
DiagramConnector diagramConnector = new DiagramConnector()
{
    SourceID = "node1",
    TargetID = "node2",
    //source node port id.
    SourcePortID = "port1",
    //Target node port id.
    TargetPortID = "port2",
    TargetDecorator = new ConnectorTargetDecorator()
    {
        Shape = DecoratorShapes.Arrow,
        Style = new DecoratorShapeStyle()
        {
            Fill = "#37909A",
            StrokeColor = "#37909A",
            StrokeWidth = 1
        }
    },
    Style = new ConnectorShapeStyle()
    {
        StrokeColor = "#37909A",
        StrokeWidth = 1
    },
    Type = Segments.Straight,
};
ConnectorCollection.Add(diagramConnector);
}
```



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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
    //Defines diagram's nodes collection
    public ObservableCollection<DiagramNode>
    NodeCollection = new ObservableCollection<DiagramNode>();
    protected override void OnInitialized()
    {
        //Creates source node
        DiagramNode node1 = new DiagramNode()
        {
            OffsetX = 100,
            OffsetY = 100,
            Height = 50,
            Width = 100,
            Shape = new DiagramShape()
            {
                Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
                BasicShape = BasicShapes.Ellipse
            },
            Style = new NodeShapeStyle()
            {
                Fill = "#37909A",
                StrokeColor = "#37909A",
            },
        },
    }
```

```

};
//Add node into node's collection
NodeCollection.Add(node1);
//To disable the inconnect constraints to node.
node1.Constraints = NodeConstraints.Default & ~NodeConstraints.InConnect;
DiagramPort Port = new DiagramPort()
{
    Offset = new Syncfusion.Blazor.Diagrams.NodePortOffset() { X = 0, Y = 0.5 },
    Height = 10,
    Width = 10,
    Visibility = PortVisibility.Visible,
    Style = new PortShapeStyle()
    {
        Fill = "yellow",
        StrokeColor = "yellow"
    }
};
//Port constraints to allow in connectors.
Port.Constraints = PortConstraints.InConnect;
NodeCollection[0].Ports.Add(Port);
}
}

```

See also

- [How to customize the connector properties](#)
- [How to interact the connector](#)
- [How to change the segments](#)
- [How to get the connector events](#)
- [How to add annotations to the connectors](#)

### Segments in Blazor Diagram Component

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

#### Straight

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

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    //Defines the diagram constraints.
    public DiagramConstraints diagramConstraints = DiagramConstraints.Default;
    protected override void OnInitialized()
    {

```

```

DiagramConnector diagramConnector = new DiagramConnector()
{
    SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
    TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
    TargetDecorator = new ConnectorTargetDecorator()
    {
        Shape = DecoratorShapes.Arrow,
        Style = new DecoratorShapeStyle()
        {
            Fill = "#6f409f",
            StrokeColor = "#6f409f",
            StrokeWidth = 1
        }
    },
    Style = new ConnectorShapeStyle()
    {
        StrokeColor = "#6f409f",
        StrokeWidth = 1
    },
    //Specify the segment type as straight.
    Type = Segments.Straight,
};
ConnectorCollection.Add(diagramConnector);
}

```



### Orthogonal

Orthogonal segments are used to create segments that are perpendicular to each other. Set the segment [Type](#) as orthogonal to create a default orthogonal segment and need to specify [Type](#). The following code example illustrates how to create a default orthogonal segment.

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

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>

```

```

@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
public DiagramConstraints diagramConstraints = DiagramConstraints.Default;
protected override void OnInitialized()
{
DiagramConnector diagramConnector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
Fill = "#6f409f",
StrokeColor = "#6f409f",
StrokeWidth = 1
}
},
Style = new ConnectorShapeStyle()
{
StrokeColor = "#6f409f",
StrokeWidth = 1
},
//Specify the segments type as Orthogonal.
Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector);
}
}

```

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

#### ASPX-CS

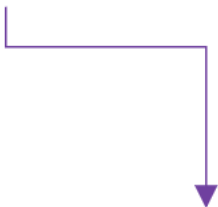
```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
public DiagramConstraints diagramConstraints = DiagramConstraints.Default;
protected override void OnInitialized()
{
DiagramConnector diagramConnector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
TargetDecorator = new ConnectorTargetDecorator()

```



```
{
    Shape = DecoratorShapes.Arrow,
    Style = new DecoratorShapeStyle()
    {
        Fill = "#6f409f",
        StrokeColor = "#6f409f",
        StrokeWidth = 1
    }
},
Style = new ConnectorShapeStyle()
{
    StrokeColor = "#6f409f",
    StrokeWidth = 1
},
//Specify the connector type as orthogonal.
Type = Segments.Orthogonal,
//Initialize the segments collection
Segments = new ObservableCollection<DiagramConnectorSegment>()
{
    //Create a new segment with length and direction
    new DiagramConnectorSegment()
    {
        Length = 100,
        Type = Segments.Orthogonal,
        Direction = Direction.Right,
    },
    //Create another new segment with length and direction
    new DiagramConnectorSegment()
    {
        Length = 100,
        Type = Segments.Orthogonal,
        Direction = Direction.Bottom,
    }
},
};
ConnectorCollection.Add(diagramConnector);
}
```



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

*Avoid overlapping*

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

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
public DiagramConstraints diagramConstraints = DiagramConstraints.Default;
protected override void OnInitialized()
{
//Create source node
DiagramNode node1 = new DiagramNode()
{
OffsetX = 100,
OffsetY = 100,
Height = 50,
Width = 100,
Id = "node1",
Shape = new DiagramShape()
{
Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
BasicShape = BasicShapes.Ellipse
},
Style = new NodeShapeStyle()
{
Fill = "#37909A",
StrokeColor = "#37909A",
},
Ports = new ObservableCollection<DiagramPort>()
{
new DiagramPort()
{
Id = "port1",
Offset = new Syncfusion.Blazor.Diagrams.NodePortOffset() { X = 1, Y = 0.5 },
Height = 10,
Width = 10,
Visibility = PortVisibility.Visible,
Style = new PortShapeStyle()
{
Fill = "yellow",
StrokeColor = "yellow"
}
},
},
};
//Add node into node's collection
NodeCollection.Add(node1);
```

```
//Create target node
DiagramNode node2 = new DiagramNode()
{
    OffsetX = 300,
    OffsetY = 300,
    Height = 50,
    Width = 100,
    Id = "node2",
    Shape = new DiagramShape()
    {
        Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
        BasicShape = BasicShapes.Ellipse
    },
    Style = new NodeShapeStyle()
    {
        Fill = "#37909A",
        StrokeColor = "#37909A",
    },
    Ports = new ObservableCollection<DiagramPort>()
    {
        new DiagramPort()
        {
            Id = "port2",
            Offset = new Syncfusion.Blazor.Diagrams.NodePortOffset() { X = 0, Y = 0.5 },
            Height = 10,
            Width = 10,
            Visibility = PortVisibility.Visible,
            Style = new PortShapeStyle()
            {
                Fill = "yellow",
                StrokeColor = "yellow"
            }
        },
    },
};
//Add node into node's collection
NodeCollection.Add(node2);
//Create a connector.
DiagramConnector diagramConnector = new DiagramConnector()
{
    SourceID = "node1",
    TargetID = "node2",
    SourcePortID = "port1",
    TargetPortID = "port2",
    TargetDecorator = new ConnectorTargetDecorator()
    {
        Shape = DecoratorShapes.Arrow,
        Style = new DecoratorShapeStyle()
        {
            Fill = "#37909A",
            StrokeColor = "#37909A",
            StrokeWidth = 1
        }
    },
    Style = new ConnectorShapeStyle()
    {
        StrokeColor = "#37909A",
```

```
StrokeWidth = 1
},
Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector);
}
}
```

### Bezier

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

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    protected override void OnInitialized()
    {
        DiagramConnector diagramConnector = new DiagramConnector()
        {
            SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
            TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
            TargetDecorator = new ConnectorTargetDecorator()
            {
                Shape = DecoratorShapes.Arrow,
                Style = new DecoratorShapeStyle()
                {
                    Fill = "#6f409f",
                    StrokeColor = "#6f409f",
                    StrokeWidth = 1
                }
            },
            Style = new ConnectorShapeStyle()
            {
                StrokeColor = "#6f409f",
                StrokeWidth = 1
            },
            Type = Segments.Bezier,
        };
        //Add the connector into connectors's collection.
        ConnectorCollection.Add(diagramConnector);
    }
}
```



*See also*

- [How to customize the connector properties](#)
- [How to interact the connector](#)
- [How to get the connector events](#)

## Customization in Blazor Diagram Component

### Decorator

Diagram allows you to customize the connector appearances. The following topics shows how to customize several properties of the connectors.

- Starting and ending points of a connector can be decorated with some customizable shapes like arrows, circles, diamond, or path. The connection end points can be decorated with the [SourceDecorator](#) and [TargetDecorator](#) properties of the connector.
- The [Shape](#) property of [SourceDecorator](#) allows to define the shape of the decorators. Similarly, the [Shape](#) property of [TargetDecorator](#) allows to define the shape of the decorators.
- To create custom shape for source decorator, use [PathData](#) property. Similarly, to create custom shape for target decorator, use [PathData](#) property.
- The following code example illustrates how to create decorators of various shapes.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    protected override void OnInitialized()
    {
        DiagramConnector diagramConnector = new DiagramConnector()
        {
            SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
            TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 100 },
            SourceDecorator = new ConnectorSourceDecorator()
```

```

{
    Shape = DecoratorShapes.Circle,
    Style = new DecoratorShapeStyle()
    {
        StrokeColor = "#37909A",
        Fill = "#37909A",
        StrokeWidth = 1
    },
},
},
TargetDecorator = new ConnectorTargetDecorator()
{
    Shape = DecoratorShapes.Custom,
    Style = new DecoratorShapeStyle()
    {
        StrokeColor = "#37909A",
        Fill = "#37909A",
        StrokeWidth = 1
    },
    PathData = "M80.5,12.5 C80.5,19.127417 62.59139,24.5 40.5,24.5
C18.40861,24.5 0.5,19.127417 0.5,12.5 C0.5,5.872583 18.40861,0.5 40.5,0.5
C62.59139,0.5 80.5,5.872583 80.5,12.5 z"
},
    Style = new ConnectorShapeStyle()
    {
        StrokeColor = "#37909A",
        StrokeWidth = 1
    },
    Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector);
}
}

```

### Padding

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

- The [SourcePadding](#) property of connector defines space between the source point and the source node of the connector.
- The [TargetPadding](#) property of connector defines space between the end point and the target node of the connector.
- The following code example illustrates how to leave space between the connection end points and source, target nodes.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's nodes collection

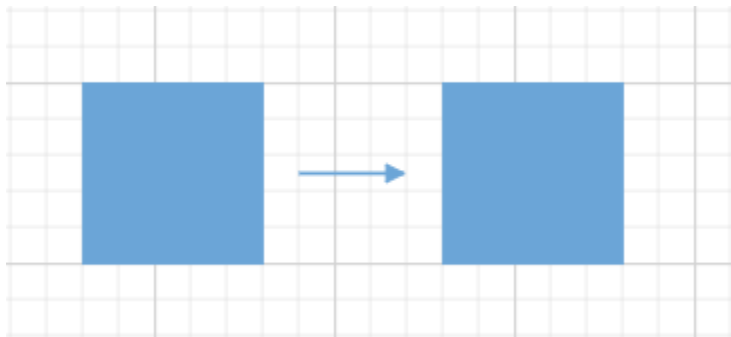
```

```
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
    //Create a node
    DiagramNode node1 = new DiagramNode()
    {
        OffsetX = 100,
        OffsetY = 100,
        Height = 50,
        Width = 100,
        Id = "node1",
        Shape = new DiagramShape()
        {
            Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
            BasicShape = BasicShapes.Rectangle
        },
        Style = new NodeShapeStyle()
        {
            Fill = "#6BA5D7",
            StrokeColor = "#6BA5D7",
        },
    };
    //Add node into node's collection
    NodeCollection.Add(node1);
    //Create a node
    DiagramNode node2 = new DiagramNode()
    {
        OffsetX = 300,
        OffsetY = 300,
        Height = 50,
        Width = 100,
        Id = "node2",
        Shape = new DiagramShape()
        {
            Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
            BasicShape = BasicShapes.Rectangle
        },
        Style = new NodeShapeStyle()
        {
            Fill = "#6BA5D7",
            StrokeColor = "#6BA5D7",
        },
    };
    //Add node into node's collection
    NodeCollection.Add(node2);
    DiagramConnector diagramConnector = new DiagramConnector()
    {
        SourceID = "node1",
        TargetID = "node2",
        //Specifies the source and target padding values.
        SourcePadding = 20,
        TargetPadding = 20,
        TargetDecorator = new ConnectorTargetDecorator()
    }
```

```

{
    Shape = DecoratorShapes.Arrow,
    Style = new DecoratorShapeStyle()
    {
        Fill = "#6BA5D7",
        StrokeColor = "#6BA5D7",
        StrokeWidth = 1
    },
    Style = new ConnectorShapeStyle()
    {
        StrokeColor = "#6BA5D7",
        StrokeWidth = 1
    },
    Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector);
}
}

```



### Flip

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

- The Flip is also applicable for nodes.

---

The flip is not applicable when the connectors connect in nodes

---

The flip types are as follows:

- **HorizontalFlip** is used to interchange the connector source and target x points.
- **VerticalFlip** is used to interchange the connector source and target y points.
- **Both** is used to interchange the source point as target point and target point as source point

### ASPX-CS

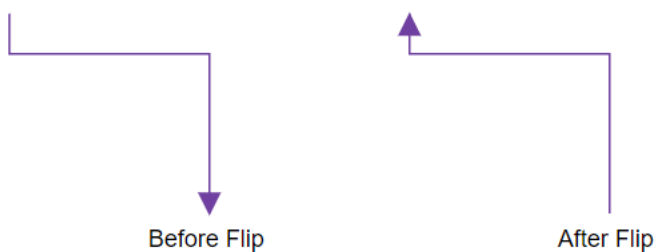
```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{

```



```
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
    DiagramConnector diagramConnector = new DiagramConnector()
    {
        SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
        TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
        //Specify the flip direction
        Flip = FlipDirection.Both,
        TargetDecorator = new ConnectorTargetDecorator()
        {
            Shape = DecoratorShapes.Arrow,
            Style = new DecoratorShapeStyle()
            {
                Fill = "#6f409f",
                StrokeColor = "#6f409f",
                StrokeWidth = 1
            }
        },
        Style = new ConnectorShapeStyle()
        {
            StrokeColor = "#6f409f",
            StrokeWidth = 1
        },
        Type = Segments.Orthogonal,
    };
    ConnectorCollection.Add(diagramConnector);
}
}
```



### Bridging

Line bridging creates a bridge for lines to smartly cross over the other lines at points of intersection. By default, [BridgeDirection](#) is set to [Top](#). Depending on the given direction, the bridging direction appears. Bridging can be enabled/disabled either with the [Connector.Constraints](#) or [Diagram.Constraints](#).

The following code example illustrates how to enable line bridging.

### ASPX-CS

```

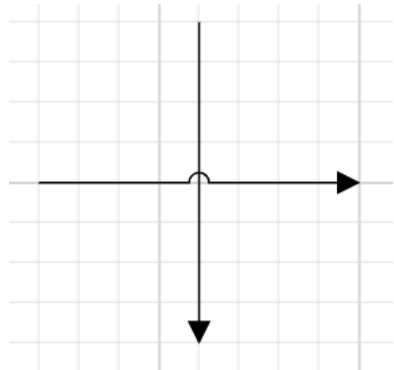
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection"
Constraints="@diagramConstraints">
</SfDiagram>
@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
//Enable the bridging constraint for diagram.
public DiagramConstraints diagramConstraints = DiagramConstraints.Default |
DiagramConstraints.Bridging;
protected override void OnInitialized()
{
DiagramConnector diagramConnector1 = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 200, Y = 200 },
TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
Fill = "#6f409f",
StrokeColor = "#6f409f",
StrokeWidth = 1
}
},
Style = new ConnectorShapeStyle()
{
StrokeColor = "#6f409f",
StrokeWidth = 1
},
Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector1);
DiagramConnector diagramConnector2 = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 300 },
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
StrokeColor = "#6BA5D7",
Fill = "#6BA5D7",
StrokeWidth = 1
}
},
Style = new ConnectorShapeStyle()
{
StrokeColor = "#6BA5D7",
StrokeWidth = 1
},
Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector2);
}
}

```

```
};
ConnectorCollection.Add(diagramConnector2);
}
}
```

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

Limitation: **Bezier** segments do not support bridging.



#### Corner radius

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

#### ASPX-CS

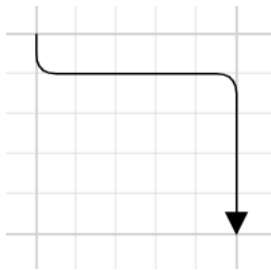
```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
DiagramConnector diagramConnector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
//specify the corner radius value.
CornerRadius = 10,
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
Fill = "#6f409f",
StrokeColor = "#6f409f",
StrokeWidth = 1
}
},
Style = new ConnectorShapeStyle()
{

```

```

StrokeColor = "#6f409f",
StrokeWidth = 1
},
Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector);
}
}

```



### Appearance

- The connector's [StrokeWidth](#), [StrokeColor](#), [StrokeDashArray](#), and [Opacity](#) properties are used to customize the appearance of the connector segments.
- The [Visible](#) property of the connector enables or disables the visibility of connector.
- Default values for all the [Connectors](#) can be set using the [ConnectorDefaults](#) properties. For example, if all connectors have the same type or having the same property then such properties can be moved into [ConnectorDefaults](#).

### Segment appearance

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

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
DiagramConnector diagramConnector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
Fill = "#6f409f",
StrokeColor = "#6f409f",

```

```

StrokeWidth = 1
},
//To customize appearance of the connector segments
Style = new ConnectorShapeStyle()
{
StrokeColor = "red",
StrokeWidth = 2,
StrokeDashArray = "2,2"
},
Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector);
}
}

```

### Decorator appearance

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

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

### ASPX-CS

```

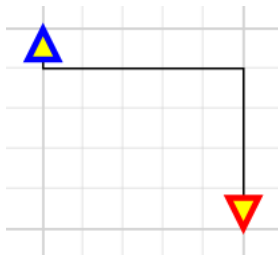
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
DiagramConnector diagramConnector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
SourceDecorator = new ConnectorSourceDecorator()
{
Shape = DecoratorShapes.Arrow,
Height = 15,
Width = 15,
Style = new DecoratorShapeStyle()
{
StrokeColor = "blue",
Fill = "yellow",
StrokeWidth = 3
},
},
},
}
}

```

```

TargetDecorator = new ConnectorTargetDecorator()
{
    Shape = DecoratorShapes.Arrow,
    Height = 15,
    Width = 15,
    Style = new DecoratorShapeStyle()
    {
        StrokeColor = "red",
        Fill = "yellow",
        StrokeWidth = 3
    },
},
Style = new ConnectorShapeStyle() { StrokeColor = "black", StrokeWidth = 1 },
Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector);
}
}

```



### Constraints

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

The following code illustrates how to disable selection.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    protected override void OnInitialized()
    {
        DiagramConnector diagramConnector = new DiagramConnector()
        {
            SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
            TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
            TargetDecorator = new ConnectorTargetDecorator()
            {
                Shape = DecoratorShapes.Arrow,

```

```

Style = new DecoratorShapeStyle()
{
    Fill = "black",
    StrokeColor = "black",
    StrokeWidth = 1
}
},
Style = new ConnectorShapeStyle()
{
    StrokeColor = "black",
    StrokeWidth = 1
},
Type = Segments.Orthogonal,
//Disable the select constraint
Constraints = ConnectorConstraints.Default & ~ConnectorConstraints.Select,
};
ConnectorCollection.Add(diagramConnector);
}
}

```

### Custom properties

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

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    protected override void OnInitialized()
    {
        DiagramConnector diagramConnector = new DiagramConnector()
        {
            SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
            TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
            TargetDecorator = new ConnectorTargetDecorator()
            {
                Shape = DecoratorShapes.Arrow,
                Style = new DecoratorShapeStyle()
                {
                    Fill = "black",
                    StrokeColor = "black",
                    StrokeWidth = 1
                }
            },
            Style = new ConnectorShapeStyle()
            {
                StrokeColor = "black",

```

```

StrokeWidth = 1
},
Type = Segments.Orthogonal,
//Define the add info value.
AddInfo = "Central Connector",
};
ConnectorCollection.Add(diagramConnector);
}
}

```

### Stack order

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

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

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
DiagramConnector diagramConnector1 = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 100 },
//Specify the z-index value
ZIndex = 2,
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
StrokeColor = "black",
Fill = "black",
StrokeWidth = 1
},
},
Style = new ConnectorShapeStyle()
{
StrokeColor = "black",
StrokeWidth = 1
},
Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector1);
DiagramConnector diagramConnector2 = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 200 },

```



```

TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
//Specify the z-index value
ZIndex = 1,
TargetDecorator = new ConnectorTargetDecorator()
{
    Shape = DecoratorShapes.Arrow,
    Style = new DecoratorShapeStyle()
    {
        StrokeColor = "black",
        Fill = "black",
        StrokeWidth = 1
    },
},
Style = new ConnectorShapeStyle()
{
    StrokeColor = "black",
    StrokeWidth = 1
},
Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector2);
}
}

```

*See also*

- [How to interact the connector](#)
- [How to change the segments](#)
- [How to get the connector events](#)

### Interaction in Blazor Diagram Component

Connectors can be selected, dragged, and routed over the diagram page.

#### Select

A connector can be selected at runtime by using the [Select](#) method and clear the selection in the diagram using the [ClearSelection](#). The following code explains how to select and clear selection in the diagram.

#### ASPX-CS

```

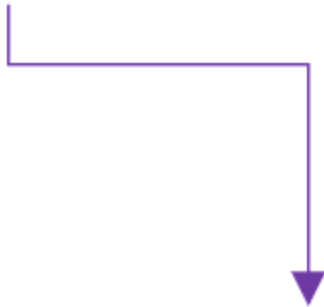
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Select" @onclick="OnSelect">
<input type="button" value="UnSelect" @onclick="@UnSelect" />
<SfDiagram @ref="@Diagram" Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code {
    // reference of the diagram
    SfDiagram Diagram;
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    protected override void OnInitialized()
    {
        DiagramConnector diagramConnector = new DiagramConnector()
    }
}

```

```
{
    Id = "Connector1",
    SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
    TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
    TargetDecorator = new ConnectorTargetDecorator()
    {
        Shape = DecoratorShapes.Arrow,
        Style = new DecoratorShapeStyle()
        {
            Fill = "#6f409f",
            StrokeColor = "#6f409f",
            StrokeWidth = 1
        }
    },
    Style = new ConnectorShapeStyle()
    {
        StrokeColor = "#6f409f",
        StrokeWidth = 1
    },
    Type = Segments.Orthogonal
};
ConnectorCollection.Add(diagramConnector);
}
public void OnSelect()
{
    // Select the Connector
    Diagram.Select(new ObservableCollection<DiagramConnector>() {
    Diagram.Connectors[0] }, null);
}
public void UnSelect()
{
    // clear selection in the diagram
    Diagram.ClearSelection();
}
}
```

And also the selection can be enabled during the interaction.

- An element can be selected by clicking that element.
- When you select the elements in the diagram, the [SelectionChanged](#) event gets triggered and do customization in this event.



### Drag

A connector can be dragged at runtime by using the [Drag](#) method. The following code explains how to drag the connector by using the drag method.

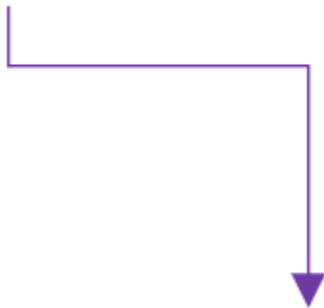
### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Drag" @onclick="OnDrag">
<SfDiagram @ref="@Diagram" Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code {
// reference of the diagram
SfDiagram Diagram;
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
DiagramConnector diagramConnector = new DiagramConnector()
{
Id = "Connector1",
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
Fill = "#6f409f",
StrokeColor = "#6f409f",
StrokeWidth = 1
}
},
},
Style = new ConnectorShapeStyle()
```

```
{
    StrokeColor = "#6f409f",
    StrokeWidth = 1
},
Type = Segments.Orthogonal
};
ConnectorCollection.Add(diagramConnector);
}
public void OnDrag()
{
    // Drag the connector
    Diagram.Drag(Diagram.Connectors[0], 10, 10);
}
}
```

And also drag the connector during the interaction.

- An object can be dragged by clicking and dragging it. When multiple elements are selected, dragging any one of the selected elements moves all the selected elements.
- When you drag the elements in the diagram, the [OnPositionChange](#) event gets triggered and to do customization in this event.



#### *End point dragging*

The connector can be selected by clicking it. When the connector is selected, circles will be added on the starting and ending of the connector that is represented by Thumbs. Clicking and dragging those handles helps you to adjust the source and target points.

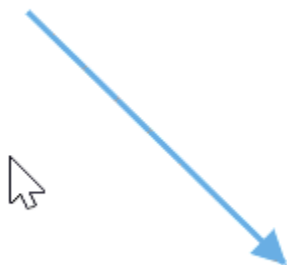
#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
```

```

@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
DiagramConnector diagramConnector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
Fill = "#6f409f",
StrokeColor = "#6f409f",
StrokeWidth = 1
}
},
Style = new ConnectorShapeStyle()
{
StrokeColor = "#6f409f",
StrokeWidth = 1
},
Type = Segments.Straight,
};
//Add the connector into connectors's collection.
ConnectorCollection.Add(diagramConnector);
}
}

```



#### [How to route connectors](#)

The connectors in the diagram can be overlapped with any neighboring nodes when the node is placed in contact with the connector. This will make less clarity about the connector path flow. This can be avoided using the Routing process. The routing is the process of updating the connector's geometry to avoid the overlapping with any neighboring nodes in their path.

This behavior can be enabled by adding the `DiagramConstraints.LineRouting` enum value to the `Constraints` property of the diagram.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Constraints="@Constraints" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's nodes collection
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>();
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    public DiagramConstraints Constraints { get; set; }
    protected override void OnInitialized()
    {
        Constraints = DiagramConstraints.Default | DiagramConstraints.LineRouting |
        DiagramConstraints.Bridging;
        //Creates source node
        DiagramNode node1 = new DiagramNode()
        {
            OffsetX = 100,
            OffsetY = 100,
            Height = 50,
            Width = 100,
            Id = "node1",
            Shape = new DiagramShape()
            {
                Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
                BasicShape = BasicShapes.Rectangle
            },
            Style = new NodeShapeStyle()
            {
                Fill = "#37909A",
                StrokeColor = "#37909A",
            },
        };
        //Add node into node's collection
        NodeCollection.Add(node1);
        //Create a target node
        DiagramNode node2 = new DiagramNode()
        {
            OffsetX = 100,
            OffsetY = 300,
            Height = 50,
            Width = 100,
            Id = "node2",
            Shape = new DiagramShape()
            {
                Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
                BasicShape = BasicShapes.Rectangle
            },
        },
```

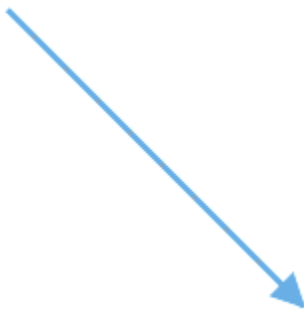
```
Style = new NodeShapeStyle()
{
    Fill = "#37909A",
    StrokeColor = "#37909A",
},
};
//Add node into node's collection
NodeCollection.Add(node2);
//Create a target node
DiagramNode node3 = new DiagramNode()
{
    OffsetX = 500,
    OffsetY = 300,
    Height = 50,
    Width = 100,
    Id = "node3",
    Shape = new DiagramShape()
    {
        Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
        BasicShape = BasicShapes.Rectangle
    },
    Style = new NodeShapeStyle()
    {
        Fill = "#37909A",
        StrokeColor = "#37909A",
    },
};
//Add node into node's collection
NodeCollection.Add(node3);
//create the connector with source node and target node id.
DiagramConnector diagramConnector = new DiagramConnector()
{
    //Source node id of the connector.
    SourceID = "node1",
    //Target node id of the connector.
    TargetID = "node2",
    TargetDecorator = new ConnectorTargetDecorator()
    {
        Shape = DecoratorShapes.Arrow,
        Style = new DecoratorShapeStyle()
        {
            Fill = "#37909A",
            StrokeColor = "#37909A",
            StrokeWidth = 1
        }
    },
    Style = new ConnectorShapeStyle()
    {
        StrokeColor = "#37909A",
        StrokeWidth = 1
    },
    Type = Segments.Orthogonal,
};
//Adding connector into connector's collection
ConnectorCollection.Add(diagramConnector);
}
```



### *Segment editing*

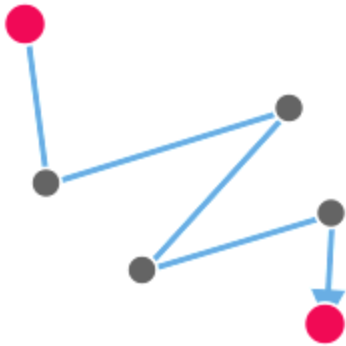
#### *Straight segment editing*

- End point of each straight segment is represented by a thumb that enables to edit the segment.
- Any number of new segments can be inserted into a straight line by clicking, when Shift and Ctrl keys are pressed (Ctrl+Shift+Click).



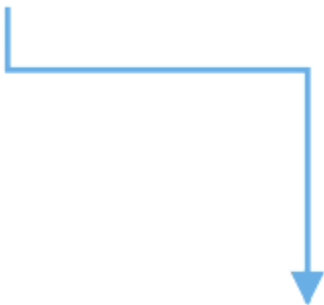
- Straight segments can be removed by clicking the segment end point, when Ctrl and Shift keys are pressed (Ctrl+Shift+Click).





### Orthogonal Segment Editing

- Orthogonal thumbs allow you to adjust the length of adjacent segments by clicking and dragging it.
- When necessary, some segments are added or removed automatically, when dragging the segment. This is to maintain proper routing of orthogonality between segments.



### Bezier segment editing

- A segment control point of the Bezier connector is used to change the bezier vectors, points of the connector.

### See also

- [How to customize the connector properties](#)
- [How to change the segments](#)
- [How to get the connector events](#)

## Events in Blazor Diagram Component

### *Selection change*

The [SelectionChanged](#) event will be triggered when you select or unselect the node or connector. The [IBlazorSelectionChangeEventArgs](#) interface is used to get selection change event arguments.

The following code example explains how to get the selection change event in the diagram.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
<DiagramEvents SelectionChanged="@SelectionChanged"></DiagramEvents>
</SfDiagram>
@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
DiagramConnector diagramConnector = new DiagramConnector()
{
Id = "Connector1",
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
Fill = "#6f409f",
StrokeColor = "#6f409f",
StrokeWidth = 1
}
},
Style = new ConnectorShapeStyle()
{
StrokeColor = "#6f409f",
StrokeWidth = 1
},
Type = Segments.Orthogonal
};
ConnectorCollection.Add(diagramConnector);
}
// SelectionChange event for diagram
public void SelectionChanged(IBlazorSelectionChangeEventArgs args)
{
Console.WriteLine(args.NewValue.Connectors[0].Id);
}
}
```

### *Position change*

The [OnPositionChange](#) events will be triggered when dragging the node or connector in interaction. The [IBlazorDraggingEventArgs](#) interface is used to get position change event arguments.

**ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
  <DiagramEvents OnPositionChange="@OnPositionChange"></DiagramEvents>
</SfDiagram>
@code
{
  //Defines diagram's connector collection
  public ObservableCollection<DiagramConnector> ConnectorCollection = new
  ObservableCollection<DiagramConnector>();
  protected override void OnInitialized()
  {
    DiagramConnector diagramConnector = new DiagramConnector()
    {
      Id = "Connector1",
      SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
      TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
      TargetDecorator = new ConnectorTargetDecorator()
      {
        Shape = DecoratorShapes.Arrow,
        Style = new DecoratorShapeStyle()
        {
          Fill = "#6f409f",
          StrokeColor = "#6f409f",
          StrokeWidth = 1
        }
      },
      Style = new ConnectorShapeStyle()
      {
        StrokeColor = "#6f409f",
        StrokeWidth = 1
      },
      Type = Segments.Orthogonal
    };
    ConnectorCollection.Add(diagramConnector);
  }
  // Position change event for diagram
  public void OnPositionChange(IBlazorDraggingEventArgs args)
  {
    Console.WriteLine(args.NewValue.Connectors[0].Id);
  }
}

```

*Connection change*

The [OnConnectionChange](#) event will notify when the connection is changed. The [IBlazorConnectionChangeEventArgs](#) interface is used to get event arguments.

**ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection"
  Constraints="@diagramConstraints">
  <DiagramEvents OnConnectionChange="@OnConnectionChange"></DiagramEvents>

```

```
</SfDiagram>
@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
//Defines diagram's constraints values.
public DiagramConstraints diagramConstraints = DiagramConstraints.Default;
private void OnConnectionChange(IBlazorConnectionChangeEventArgs args)
{
Console.WriteLine("Oldvalue", args.OldValue);
Console.WriteLine("NewValue", args.NewValue);
}
protected override void OnInitialized()
{
DiagramConnector diagramConnector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
Fill = "#6f409f",
StrokeColor = "#6f409f",
StrokeWidth = 1
}
},
Style = new ConnectorShapeStyle()
{
StrokeColor = "#6f409f",
StrokeWidth = 1
},
Type = Segments.Straight,
};
//Add the connector into connectors's collection.
ConnectorCollection.Add(diagramConnector);
}
}
```

*See also*

- [How to customize the connector properties](#)
- [How to interact the connector](#)
- [How to change the segments](#)

### Group in Blazor Diagram Component

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

### Create group

#### Add group when initializing diagram

A group can be added to the diagram model through `Nodes` collection. To define an object as group, add the child objects to the `Children` collection of the group. The following code illustrates how to create a group node.

- While creating group, its child node needs to be declared before the group declaration.
- Add a node to the existing group child by using the `diagram.Group` method.
- The group's `diagram.UnGroup` method is used to define whether the group can be ungrouped or not.
- A group can be added into a child of another group.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize the diagram with NodeCollection */
<SfDiagram Height="500px" @ref="diagram" Nodes="@NodeCollection">
</SfDiagram>
@code{
SfDiagram diagram;
ObservableCollection<DiagramNode> NodeCollection;
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = createNode("node1", 100, 100, "Node1");
DiagramNode node2 = createNode("node2", 300, 100, "Node2");
DiagramNode node3 = createNode("node3", 200, 250, "Node3");
DiagramNode groupnode = new DiagramNode();
// Grouping node 1 and node 2 into a single group
groupnode.Children = new string[] { "node1", "node2" };
NodeCollection.Add(node1);
NodeCollection.Add(node2);
NodeCollection.Add(node3);
NodeCollection.Add(groupnode);
}
public DiagramNode createNode(string id, double offx, double offy, string
content)
{
DiagramNode node = new DiagramNode()
{
Id = id,
OffsetX = offx,
OffsetY = offy,
Height = 100,
Width = 100,
Style = new NodeShapeStyle() { Fill = "darkcyan" }
};
DiagramNodeAnnotation annotation = new DiagramNodeAnnotation()
{
Id = "annotation1",
Content = content,
Style = new AnnotationStyle() { Color = "white", Fill = "transparent",
StrokeColor = "None" },
}
```

```

};
node.Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    annotation
};
return node;
}
protected override async Task OnAfterRenderAsync(bool firstRender)
{
    if (firstRender)
    {
        await Task.Delay(500);
        diagram.SelectAll();
        // Adding the third node into the existing group
        diagram.Group();
    }
}
}

```

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

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize the diagram with NodeCollection */
<SfDiagram Height="500px" @ref="diagram" Nodes="@NodeCollection">
</SfDiagram>
@code{
    SfDiagram diagram;
    ObservableCollection<DiagramNode> NodeCollection;
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node1 = createNode("node1", 100, 100, "Node1");
        DiagramNode node2 = createNode("node2", 300, 100, "Node2");
        DiagramNode groupnode = new DiagramNode();
        // Grouping node 1 and node 2 into a single group
        groupnode.Children = new string[] { "node1", "node2" };
        NodeCollection.Add(node1);
        NodeCollection.Add(node2);
        NodeCollection.Add(groupnode);
    }
    public DiagramNode createNode(string id, double offx, double offy, string content)
    {
        DiagramNode node = new DiagramNode()
        {
            Id = id,
            OffsetX = offx,
            OffsetY = offy,
            Height = 100,
            Width = 100,
            Style = new NodeShapeStyle() { Fill = "darkcyan" }
        };
        DiagramNodeAnnotation annotation = new DiagramNodeAnnotation()
    }
}

```

```

{
    Id = "annotation1",
    Content = content,
    Style = new AnnotationStyle() { Color = "white", Fill = "transparent",
    StrokeColor = "None" },
};
node.Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    annotation
};
return node;
}
protected override async Task OnAfterRenderAsync(bool firstRender)
{
    if (firstRender)
    {
        await Task.Delay(500);
        diagram.SelectAll();
        // Ungroup the selected group into nodes
        diagram.UnGroup();
    }
}
}

```

### Add group at runtime

A group node can be added at runtime by using Nodes collection of diagram. The following code illustrates how a group node is added at runtime

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="AddGroup" @onclick="@AddGroup" />
/* Initialize the diagram with NodeCollection */
<SfDiagram Height="500px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    ObservableCollection<DiagramNode> NodeCollection;
    DiagramNode groupnode = new DiagramNode();
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node1 = createNode("node1", 100, 100, "Node1");
        DiagramNode node2 = createNode("node2", 300, 100, "Node2");
        // Grouping node 1 and node 2 into a single group
        groupnode.Children = new string[] { "node1", "node2" };
        NodeCollection.Add(node1);
        NodeCollection.Add(node2);
    }
    public DiagramNode createNode(string id, double offx, double offy, string
    content)
    {
        DiagramNode node = new DiagramNode()
        {
            Id = id,
            OffsetX = offx,

```

```

OffsetY = offy,
Height = 100,
Width = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7" }
};
DiagramNodeAnnotation annotation = new DiagramNodeAnnotation()
{
    Id = "annotation1",
    Content = content,
    Style = new AnnotationStyle() { Color = "white", Fill = "transparent",
    StrokeColor = "None" },
};
node.Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    annotation
};
return node;
}
private void AddGroup()
{
    NodeCollection.Add(groupnode);
}
}

```

### Group from SymbolPalette

Group Nodes can be predefined and added to SymbolPalette. You can drop those Groups into Diagram, when required. The following code illustrates how to add group into SymbolPalette.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<div class="control-section">
    @*Hidden:Lines*@
    <style>
        .sb-mobile-palette {
            width: 140px;
            height: 100%;
            float: left;
        }
        .sb-mobile-diagram {
            width: calc(100% - 242px);
            height: 100%;
            float: left;
        }
    </style>
    @*End:Hidden*@
    <div style="width: 100%">
        <div class="sb-mobile-palette-bar">
            <div id="palette-icon" style="float: right;" role="button" class="e-ddb-
            icons1 e-toggle-palette"></div>
        </div>
        <div id="palette-space" class="sb-mobile-palette">
            @* Initialize Symbol palette with customize symbol size*@
            <SfSymbolPalette Width="100%" Height="700px" SymbolHeight="60"
            SymbolWidth="60" SymbolInfo="@symbolInfo">

```



```

<SymbolPalettePalettes>
@* Sets the palette header property for the symbols *@
<SymbolPalettePalette Id="basic" Expanded="true" Symbols="@BasicShapes"
IconCss="e-ddb-icons e-flow" Title="Group Shapes">
</SymbolPalettePalette>
</SymbolPalettePalettes>
</SfSymbolPalette>
</div>
<div id="diagram-space" class="sb-mobile-diagram">
<div class="content-wrapper" style="border: 1px solid #D7D7D7">
<SfDiagram ID="diagram" Width="500px" Height="400px">
</SfDiagram>
</div>
</div>
</div>
</div>
@code{
// Defines palette's basic-shape collection
public ObservableCollection<DiagramNode> BasicShapes { get; set; }
public SymbolInfo symbolInfo;
protected override void OnInitialized()
{
//Sets the size, appearance and description of a symbol
symbolInfo = new SymbolInfo()
{
Description = new SymbolDescription()
{
Text = "GroupCollection",
Overflow = TextOverflow.Wrap,
Wrap = TextWrap.Wrap
},
Height = 50,
Width = 50
};
//Initialize the basicshapes for the symbol palette
BasicShapes = new ObservableCollection<DiagramNode>();
DiagramNode groupnode = createNode("groupnode", 100, 100, "GroupNode",
Syncfusion.Blazor.Diagrams.BasicShapes.Rectangle);
DiagramNode node1 = createNode("node1", 100, 100, "Node1",
Syncfusion.Blazor.Diagrams.BasicShapes.Rectangle);
DiagramNode node2 = createNode("node2", 150, 150, "Node2",
Syncfusion.Blazor.Diagrams.BasicShapes.Ellipse);
//Grouping node 1 and node 2 into a single group
groupnode.Children = new string[] { "node1", "node2" };
BasicShapes.Add(node1);
BasicShapes.Add(node2);
BasicShapes.Add(groupnode);
}
public DiagramNode createNode(string id, double offx, double offy, string
content, Syncfusion.Blazor.Diagrams.BasicShapes shape)
{
DiagramNodeAnnotation Annotation = new DiagramNodeAnnotation()
{
Id = "annotation1",
Content = content,
Style = new AnnotationStyle()
{

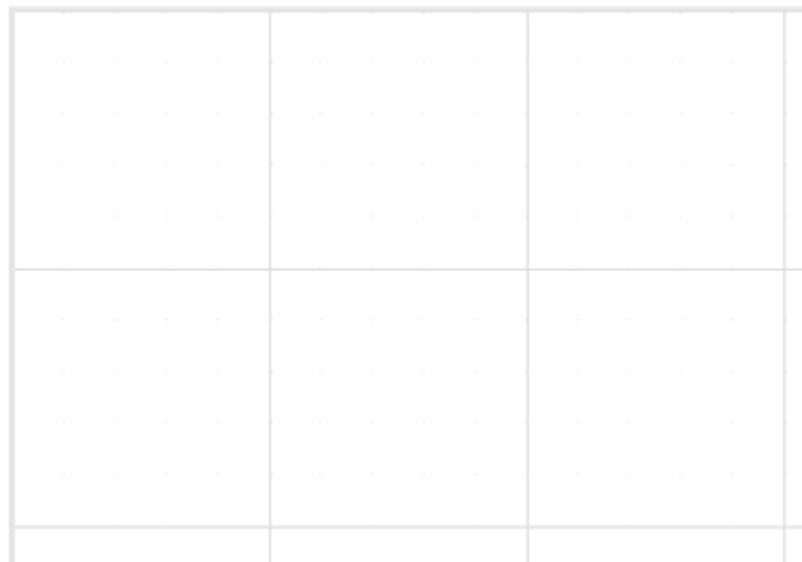
```

```

Color = "white",
Fill = "transparent",
StrokeColor = "None"
},
};
if (content == "GroupNode")
{
Annotation.Style.Color = "black";
}
DiagramNode Node = new DiagramNode()
{
Id = id,
OffsetX = offx,
OffsetY = offy,
Height = 50,
Width = 50,
Shape = new DiagramShape() { Type = Shapes.Basic, BasicShape = shape },
Annotations = new ObservableCollection<DiagramNodeAnnotation>() { Annotation
}
};
if (content != "GroupNode")
{
Node.Style = new NodeShapeStyle() { Fill = "#6BA5D7" };
}
return Node;
}
}

```

### Group Shapes



### Update position at runtime

You can change the position of the group similar to node. For more information about node positioning, refer to [Positioning](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
```

```
@using System.Collections.ObjectModel
<input type="button" value="UpdatePosition" @onclick="@UpdatePosition" />
/* Initialize the diagram with NodeCollection */
<SfDiagram Height="500px" Nodes="@NodeCollection">
</SfDiagram>
@code{
ObservableCollection<DiagramNode> NodeCollection;
DiagramNode groupnode = new DiagramNode();
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = createNode("node1", 100, 100, "Node1");
DiagramNode node2 = createNode("node2", 300, 100, "Node2");
// Grouping node 1 and node 2 into a single group
groupnode.Children = new string[] { "node1", "node2" };
NodeCollection.Add(node1);
NodeCollection.Add(node2);
NodeCollection.Add(groupnode);
}
public DiagramNode createNode(string id, double offx, double offy, string
content)
{
DiagramNode node = new DiagramNode()
{
Id = id,
OffsetX = offx,
OffsetY = offy,
Height = 100,
Width = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7" }
};
DiagramNodeAnnotation annotation = new DiagramNodeAnnotation()
{
Id = "annotation1",
Content = content,
Style = new AnnotationStyle()
{
Color = "white",
Fill = "transparent",
StrokeColor = "None"
},
};
node.Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
annotation
};
return node;
}
private void UpdatePosition()
{
NodeCollection[2].BeginUpdate();
NodeCollection[2].OffsetX = 500;
NodeCollection[2].OffsetY = 200;
NodeCollection[2].EndUpdate();
}
}
```

### Appearance

You can change the appearance of the group similar to node. For more information about node appearance, refer to [Appearance](#).

### Interaction

You can edit the group and its children at runtime. We able to interact the group as like the node interaction like resize, rotate and drag. For more information about node interaction, refer to [Interaction](#).

### See Also

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

### Annotations

#### Actions of annotation in Blazor Diagram Component

The [Annotation](#) is a block of text that can be displayed over a node or connector and it is used to textually represent an object with a string that can be edited at run time. Multiple annotations can be added to a node or connector.

<!-- markdownlint-disable MD033 -->

#### Create annotations

An annotation can be added to a node or connector by defining the annotation object and adding that to the annotation collection of the node or connector. The [Content](#) property of annotation defines the text to be displayed. The following code explains how to create an annotation.

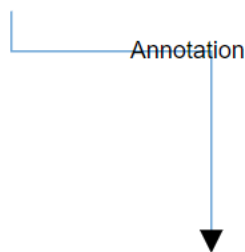
#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
Width = 100,
Height = 100,
```

```

OffsetX = 100,
OffsetY = 100,
Style = new NodeShapeStyle()
{
    Fill = "#6BA5D7",
    StrokeColor = "white"
},
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    // A annotation is created and stored in Annotations collection of Node.
    new DiagramNodeAnnotation() { Content = "Node" }
};
NodeCollection.Add(node);
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
    SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 40 },
    TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 160 },
    Type = Segments.Orthogonal,
    Style = new ConnectorShapeStyle()
    {
        StrokeColor = "#6BA5D7"
    },
    Annotations = new ObservableCollection<DiagramConnectorAnnotation>()
    {
        //A annotation is created and stored in Annotations collection of Connector.
        new DiagramConnectorAnnotation() { Content = "Connector" }
    };
ConnectorCollection.Add(connector);
}
}

```



\* [Id](#) for each annotation should be unique and so it is further used to find the annotation at runtime and do any customization.

\* By default, node's annotation positioned in center point of the shape.

\* By default, connector's path annotation positioned center point of its path.

#### *Add annotations at runtime*

Annotations can be added at runtime by using the [AddLabels](#) method. The following code explains how to add an annotation to a node.

**ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input value="Addlabel" type="button" @onclick="@AddLabel" name="Addlabel"
/>
<SfDiagram Height="600px" @ref="@diagram" Nodes="@NodeCollection">
</SfDiagram>
@code
{
    // Reference to diagram
    SfDiagram diagram;
    //Defines diagram's node collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node = new DiagramNode()
        {
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            Style = new NodeShapeStyle()
            {
                Fill = "#6BA5D7",
                StrokeColor = "white"
            },
        };
        NodeCollection.Add(node);
    }
    //Method to add labels at runtime
    public void AddLabel()
    {
        ObservableCollection<DiagramNodeAnnotation> annotations = new
        ObservableCollection<DiagramNodeAnnotation>()
        {
            new DiagramNodeAnnotation() { Content = "Annotation" },
        };
        // AddLabels method is used to add annotations at run time
        diagram.AddLabels(diagram.Nodes[0], annotations);
    }
}

```

Also, the annotations can be added at runtime by using the **Add** method.

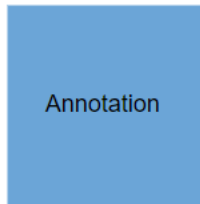
**CSHARP**

```

//Method to add labels at runtime
public void AddLabel()
{
    diagram.Nodes[0].Annotations = new
    ObservableCollection<DiagramNodeAnnotation>();
    DiagramNodeAnnotation annotation = new DiagramNodeAnnotation() { Content =
    "Annotation" };
    (diagram.Nodes[0].Annotations as
    ObservableCollection<DiagramNodeAnnotation>).Add(annotation);
}

```

}



You cannot reset the annotation collection directly to add or update the annotation collection.

#### *Remove annotations*

A collection of annotations can be removed from the node by using the [RemoveLabels](#) method. The following code explains how to remove an annotation to a node.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input value="Removelabel" type="button" @onclick="@RemoveLabel"
name="Removelabel" />
<SfDiagram Height="600px" @ref="@diagram" Nodes="@NodeCollection">
</SfDiagram>
@code
{
//Reference to diagram
SfDiagram diagram;
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
},
};
node.Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation() {Id="label", Content = "Annotation" },
};
NodeCollection.Add(node);
}
//Method to remove labels at runtime
```

```

public void RemoveLabel()
{
    ObservableCollection<DiagramNodeAnnotation> annotations = new
    ObservableCollection<DiagramNodeAnnotation>()
    {
        new DiagramNodeAnnotation() {Id="label", Content = "Annotation" }
    };
    // RemoveLabels method is used to remove label at run time.
    diagram.RemoveLabels(diagram.Nodes[0], annotations);
}
}

```

Also, A collection of annotations can be removed from the node by using the **Remove** and **RemoveAt** method.

### **CSHARP**

```

//Method to remove labels at runtime using RemoveAt method.
public void RemoveLabel()
{
    (diagram.Nodes[0].Annotations as
    ObservableCollection<DiagramNodeAnnotation>).RemoveAt(0);
}
//Method to remove labels at runtime using Remove method.
public void RemoveLabel()
{
    DiagramNodeAnnotation annotation = diagram.Nodes[0].Annotations[0] as
    DiagramNodeAnnotation;
    (diagram.Nodes[0].Annotations as
    ObservableCollection<DiagramNodeAnnotation>).Remove(annotation);
}

```

\* You can delete multiple annotations from node to pass the collection of annotation objects as argument.

\* Both the AddLabels and RemoveLabels API's are applicable to nodes and connectors.

\* The **Add**, **Remove**, and **RemoveAt** methods are applicable for connectors too.

### *Update annotations at runtime*

You can get the annotation directly from the node's annotations collection property and you can change any annotation properties at runtime.

The following code sample shows how the annotation of the node changed at runtime.

### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input value="Updatelabel" type="button" @onclick="@UpdateLabel"
name="Updatelabel" />
<SfDiagram Height="600px" @ref="@diagram" Nodes="@NodeCollection">
</SfDiagram>
@code
{
    //Reference to diagram

```



```
SfDiagram diagram;
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
    NodeCollection = new ObservableCollection<DiagramNode>();
    DiagramNode node = new DiagramNode()
    {
        Width = 100,
        Height = 100,
        OffsetX = 100,
        Annotations = new ObservableCollection<DiagramNodeAnnotation>()
        {
            new DiagramNodeAnnotation() { Content = "Node" }
        },
        OffsetY = 100,
        Style = new NodeShapeStyle()
        {
            Fill = "#6BA5D7",
            StrokeColor = "white"
        },
    };
    NodeCollection.Add(node);
}
public void UpdateLabel()
{
    diagram.Nodes[0].Annotations[0].Content = "Updated Annotation";
}
}
```

#### See also

- [How to add or remove annotation constraints](#)
- [How to customize the annotation](#)
- [How to interact the annotation at runtime](#)
- [How to localize the annotation Text](#)
- [Accessibility](#)

#### Annotation for node in Blazor Diagram Component

Diagram allows you to customize the position and appearance of the annotation efficiently. Annotation can be aligned relative to the node boundaries. It has Margin, Offset, Horizontal, and Vertical alignment settings. It is quite tricky when all four alignments are used together but gives more control over alignments properties of the [DiagramNodeAnnotation](#) class. Annotations of a node can be positioned using the following properties of [DiagramNodeAnnotation](#).

- [Offset](#)
- [HorizontalAlignment](#)
- [VerticalAlignment](#)
- [Margin](#)

### Offset

The [Offset](#) property of an annotation is used to align the annotations based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

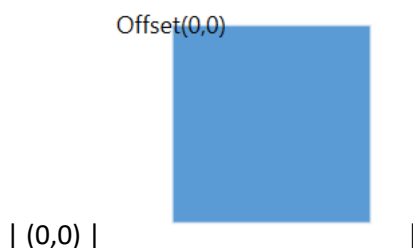
The following code shows the relationship between the shape annotation position and path annotation offset (fraction values).

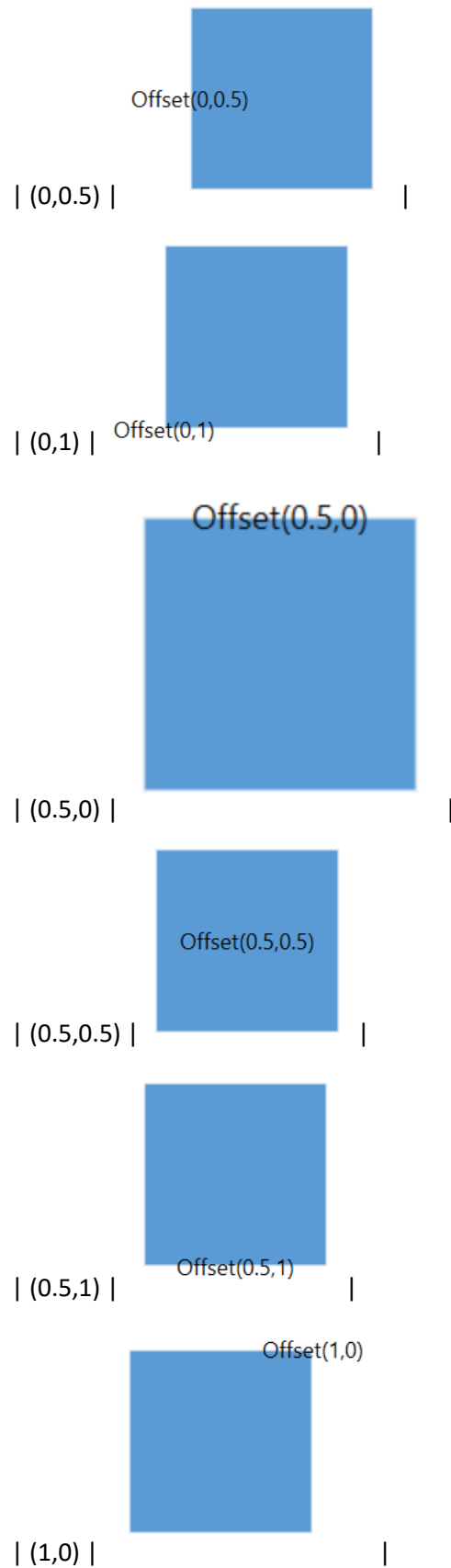
### ASPX-CS

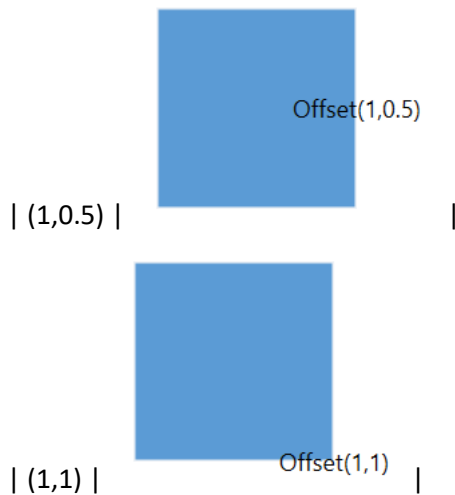
```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
Width = 100,
Height = 100,
OffsetX = 100,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Offset(0,0)",
Offset = new NodeAnnotationOffset() { X = 0, Y = 0 }
},
},
OffsetY = 100,
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
},
};
NodeCollection.Add(node);
}
}
```

| Offset values | Output |

|---|---|








---

\* Type of the offset property for node's shape annotation is `NodeAnnotationOffset`.

\* Type of the offset property for connector's path annotation is double.

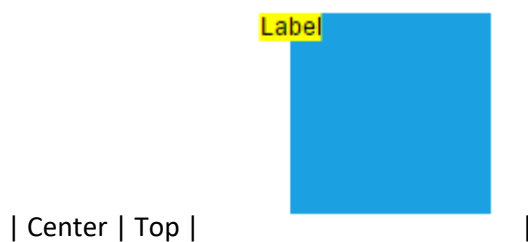
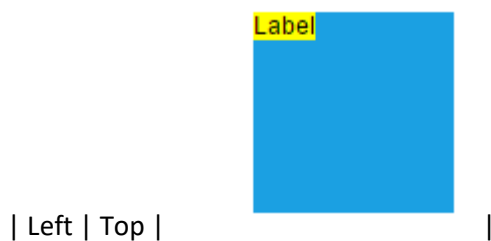
---

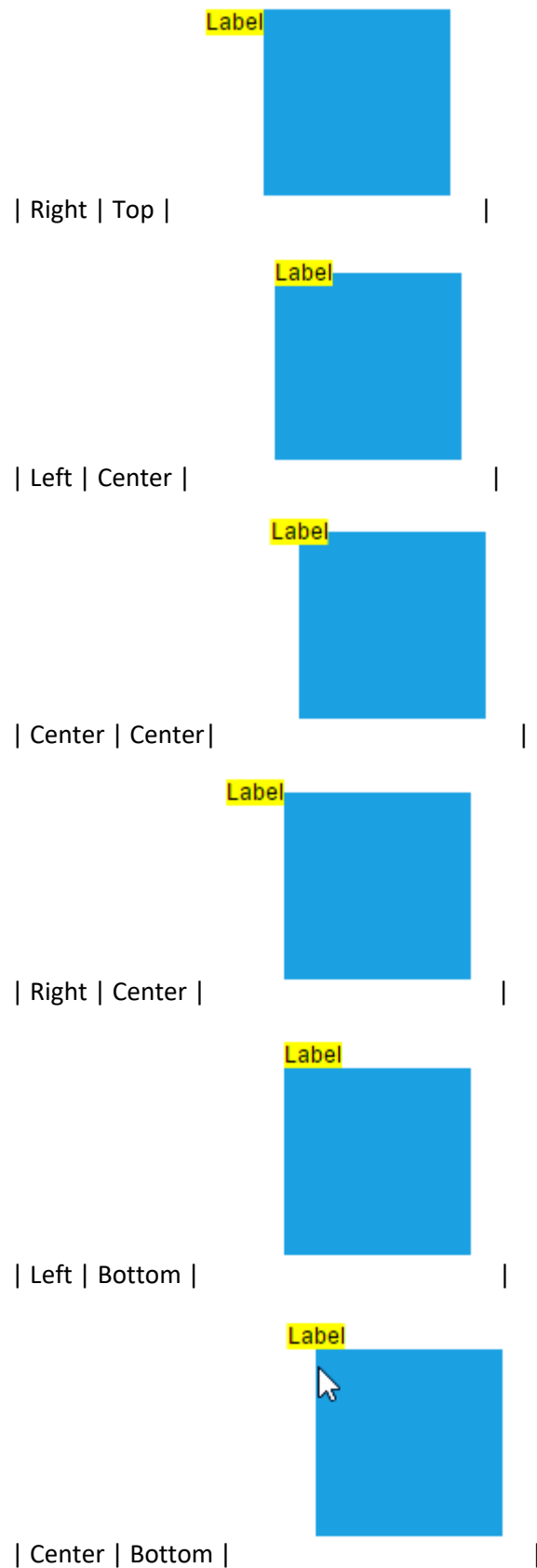
#### *Horizontal and vertical alignment*

- The [HorizontalAlignment](#) property of annotation is used to set how the annotation is horizontally aligned at the annotation position determined from the fraction values.
- The [VerticalAlignment](#) property is used to set how the annotation is vertically aligned at the annotation position.

The following table shows all the possible alignments visually with 'offset (0, 0)'.

Horizontal Alignment	Vertical Alignment	Output with Offset(0,0)
-----	-----	-----







| Right | Bottom |

The following code explains how to align annotations.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
    //Defines diagram's node collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node1 = new DiagramNode()
        {
            Id = "node1",
            Width = 100,
            Height = 100,
            OffsetX = 250,
            OffsetY = 250,
            Annotations = new ObservableCollection<DiagramNodeAnnotation>()
            {
                new DiagramNodeAnnotation()
                {
                    Content = "Annotation",
                    HorizontalAlignment = HorizontalAlignment.Left,
                    VerticalAlignment = VerticalAlignment.Center
                }
            },
            Style = new NodeShapeStyle()
            {
                Fill = "#6BA5D7",
                StrokeColor = "white"
            },
        };
        NodeCollection.Add(node1);
    }
}
```

\* The value of the `HorizontalAlignment` is `Center` by default.

\* The value of the `VerticalAlignment` is `Center` by default.

\* Alignment positioned based on the offset value.

### Margin

[Margin](#) is an absolute value used to add some blank space to any one of its four sides. The annotations can be displaced with the margin property. The following code example explains how to align an annotation based on its Offset, HorizontalAlignment, VerticalAlignment, and Margin values.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = new DiagramNode()
{
Id = "node1",
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
// Sets the margin for the content
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Task1",
Margin = new NodeAnnotationMargin(){ Top = 10},
HorizontalAlignment = HorizontalAlignment.Center,
VerticalAlignment = VerticalAlignment.Top,
Offset = new NodeAnnotationOffset(){ X = .5 ,Y = 1}
}
},
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
},
};
NodeCollection.Add(node1);
}
}
```

### Text align

The [TextAlign](#) property of annotation allows you to set how the text should be aligned (Left, Right, Center, or Justify) inside the text block. The following code explains how to set TextAlign for an annotation.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
```

```
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = new DiagramNode()
{
Id = "node1",
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
// Sets the textAlign as left for the content
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Text align is set as Left",
Style = new AnnotationStyle(){ TextAlign = TextAlign.Left}
}
},
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
},
};
NodeCollection.Add(node1);
}
}
```

*See also*

- [How to add annotation for Connector](#)
- [How to add or remove annotation constraints](#)
- [How to customize the annotation](#)
- [How to interact the annotation at runtime](#)

#### Annotation for Connector in Blazor Diagram Component

Annotations of a connector can be positioned using the following properties of AnnotationEditorViewModel class.

- Offset
- Alignment
- Displacement
- SegmentAngle



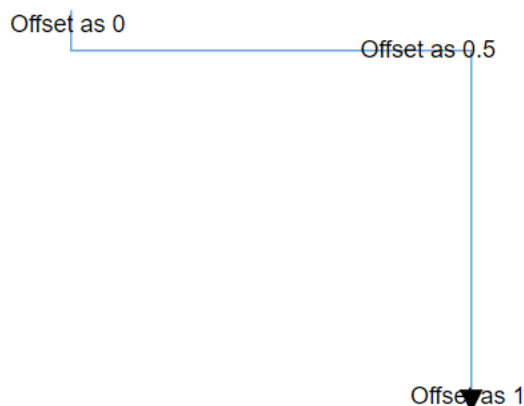
### Offset

The [Offset](#) property of annotation is used to align the annotations based on fractions. 0 represents Top-Left corner, 1 represents Bottom-Right corner, and 0.5 represents half of Width/Height.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 40 },
TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 160 },
Type = Segments.Orthogonal,
Style = new ConnectorShapeStyle()
{
StrokeColor = "#6BA5D7"
},
Annotations = new ObservableCollection<DiagramConnectorAnnotation>()
{
new DiagramConnectorAnnotation() { Content = "Offset as 0",Offset=0 },
new DiagramConnectorAnnotation() { Content = "Offset as 0.5",Offset=0.5 },
new DiagramConnectorAnnotation() { Content = "Offset as 1",Offset=1 },
}
};
ConnectorCollection.Add(connector);
}
```

The following image shows the relationship between the annotation position and offset (fraction values).



---

By default, offset value of the connector annotation is 0.5.

---

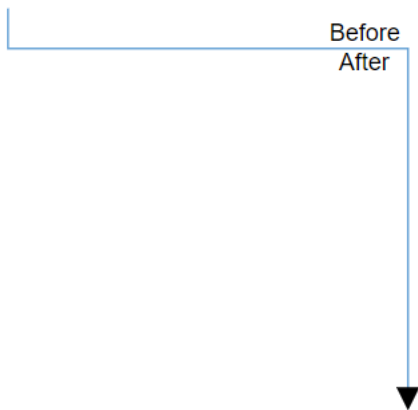
### [Alignment](#)

The connector's annotation can be aligned over its segment path using the [Alignment](#) property of annotation.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 40 },
TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 160 },
Type = Segments.Orthogonal,
Style = new ConnectorShapeStyle()
{
StrokeColor = "#6BA5D7"
},
Annotations = new ObservableCollection<DiagramConnectorAnnotation>()
{
new DiagramConnectorAnnotation()
{
Content = "Before",
Alignment=AnnotationAlignment.Before
},
new DiagramConnectorAnnotation()
{
Content = "After",
Alignment=AnnotationAlignment.After
},
},
};
ConnectorCollection.Add(connector);
}
```

The following screenshot shows how the annotation of the connector aligned over its path.




---

By default, Alignment value of the connector annotation is **Center**.

---

#### *Displacement*

The [Displacement](#) property is used to dislocate the annotation by the value given. By default, annotation will be in center of the connector path. When you assign value to the Displacement property, annotation will be displaced from its position by displacement value.

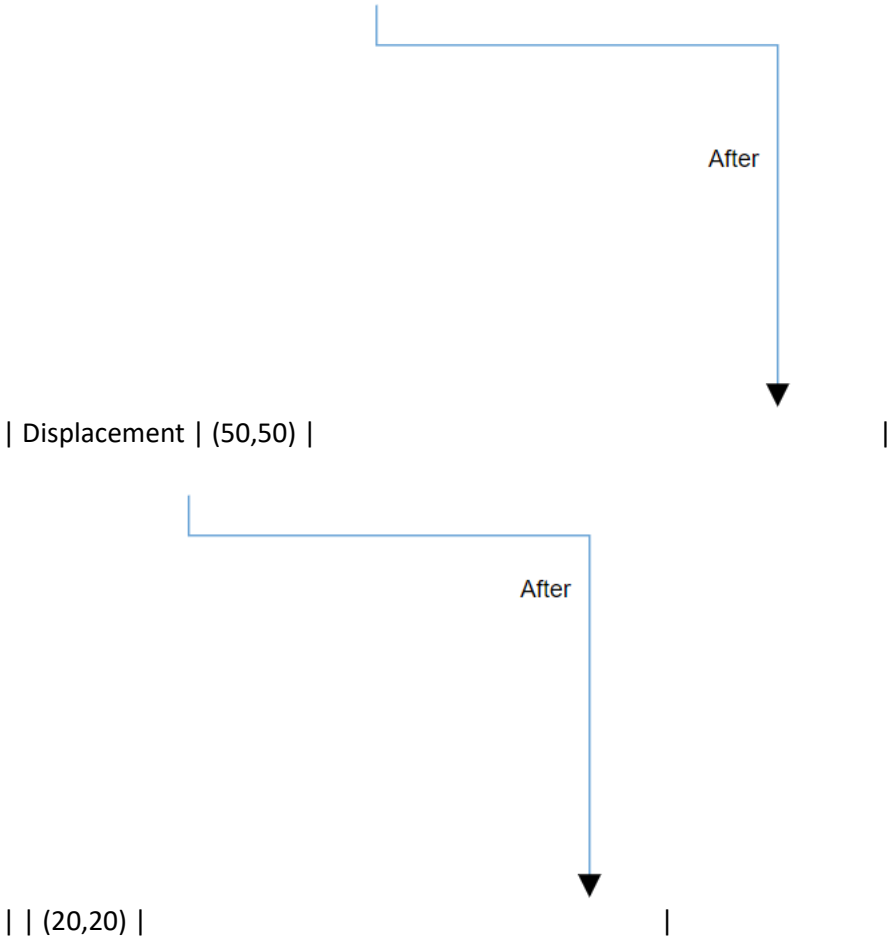
#### **ASPX-CS**

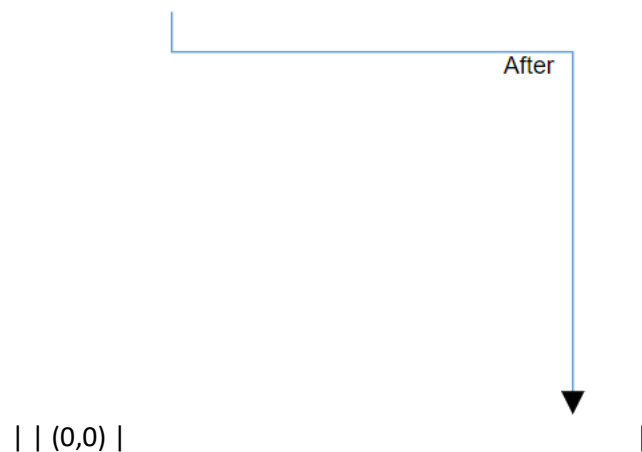
```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 40 },
TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 160 },
Type = Segments.Orthogonal,
Style = new ConnectorShapeStyle()
{
StrokeColor = "#6BA5D7"
},
Annotations = new ObservableCollection<DiagramConnectorAnnotation>()
{
new DiagramConnectorAnnotation()
{
Content = "After",
Displacement=new ConnectorDisplacementPoint() {X=50,Y=50},
Alignment=AnnotationAlignment.After
},
}
};
};
```

```
ConnectorCollection.Add(connector);  
}  
}
```

The following sample shows how the annotation of the displacement happens from its path.

Property	Value	Output
---	---	---





By default, Offset value of the connector annotation is { 0.5, 0.5}.

#### Segment angle

The [SegmentAngle](#) property is used to rotate the annotation based on the connectors segment direction. By default, annotation will be in rotated in the connector path. When you assign value to the SegmentPath property, annotation will be rotated from its position based on the annotation direction.

The following code example shows how the connector annotation rotated in its path direction.

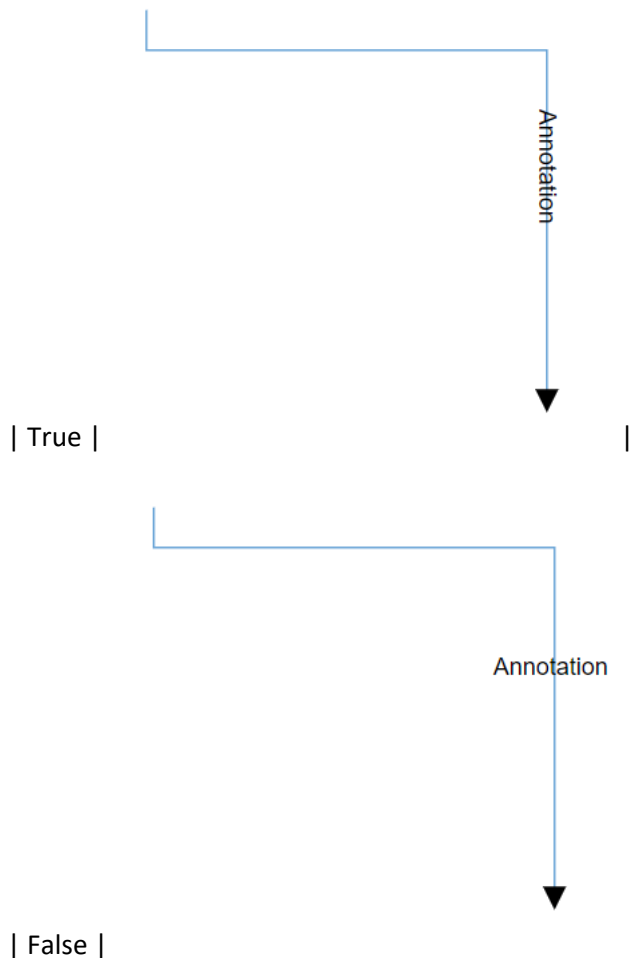
#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection { get;
    set; }
    protected override void OnInitialized()
    {
        ConnectorCollection = new ObservableCollection<DiagramConnector>();
        DiagramConnector connector = new DiagramConnector()
        {
            SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 40 },
            TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 160 },
            Type = Segments.Orthogonal,
            Style = new ConnectorShapeStyle()
            {
                StrokeColor = "#6BA5D7"
            },
            Annotations = new ObservableCollection<DiagramConnectorAnnotation>()
            {
                new DiagramConnectorAnnotation()
                {
                    Content = "Annotation",
                    SegmentAngle=true,
                    Offset=0.7
                },
            }
        }
    }
}
```

```
};  
ConnectorCollection.Add(connector);  
}  
}
```

| Segment Angle | Output |

|---|---|



By default, the SegmentAngle will be disabled.

*See also*

- [How to add annotation for Node](#)
- [How to add or remove annotation constraints](#)
- [How to customize the annotation](#)
- [How to interact the annotation at runtime](#)

## Appearance in Blazor Diagram Component

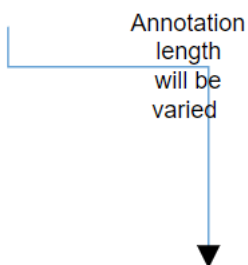
### *Customize the size for an annotation*

Diagram allows you to set size for annotations by using the Height and Width properties. The default value of the [Width](#), and [Height](#) properties are 0, and it takes the node or connector size as default.

The following code example shows how the annotation size is customized.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection { get;
    set; }
    protected override void OnInitialized()
    {
        ConnectorCollection = new ObservableCollection<DiagramConnector>();
        DiagramConnector connector = new DiagramConnector()
        {
            SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 40 },
            TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 160 },
            Type = Segments.Orthogonal,
            Style = new ConnectorShapeStyle() { StrokeColor = "#6BA5D7" },
            Annotations = new ObservableCollection<DiagramConnectorAnnotation>()
            {
                new DiagramConnectorAnnotation()
                {
                    Content = "Annotation length will be varied",
                    Width = 50,
                    Height = 50
                },
            }
        };
        ConnectorCollection.Add(connector);
    }
}
```



### *Hyperlink*

Diagram provides support to add a [Hyperlink](#) to the nodes or connectors annotation. It can also be customized.

**ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = new DiagramNode()
{
Id = "node1",
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
// Sets the annotation for the Node
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
// Add text as hyperlink.
new DiagramNodeAnnotation()
{
Hyperlink = new NodeHyperlink(){ Link = "https://www.syncfusion.com"}
},
},
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
};
NodeCollection.Add(node1);
}
}

```



## Hyperlink with content

**ASPX-CS**

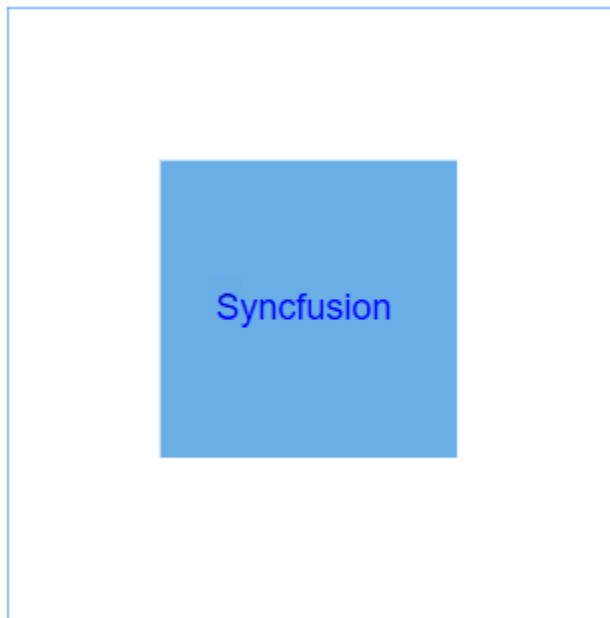
```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
//Defines diagram's node collection

```



```
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
    NodeCollection = new ObservableCollection<DiagramNode>();
    DiagramNode node1 = new DiagramNode()
    {
        Id = "node1",
        Width = 100,
        Height = 100,
        OffsetX = 100,
        OffsetY = 100,
        // Sets the annotation for the Node
        Annotations = new ObservableCollection<DiagramNodeAnnotation>()
        {
            // Add text as hyperlink.
            new DiagramNodeAnnotation()
            {
                Hyperlink = new NodeHyperlink()
                { Content = "Syncfusion", Link = "https://www.syncfusion.com" }
            },
            Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
        };
        NodeCollection.Add(node1);
    }
}
```



### Wrapping

When text overflows node boundaries, you can control it by using the [TextWrapping](#) which wraps text into multiple lines. The wrapping property of the annotation defines how the text should be wrapped.

The following code explains how to wrap a text in a node.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = new DiagramNode()
{
Id = "node1",
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
//Sets the annotation for the node
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Annotation Text Wrapping",
Style = new AnnotationStyle() { TextWrapping = TextWrap.Wrap }
},
},
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
};
NodeCollection.Add(node1);
}
}

```

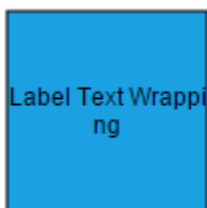
| Value | Description | Image |

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



| No Wrap | Text will not be wrapped. |

| Wrap | Text-wrapping occurs, when the text overflows beyond the available node width. |



| **WrapWithOverflow (Default)** | Text-wrapping occurs, when the text overflows beyond the available node width. However, the text may overflow beyond the node width in the case of a very long word. |



#### Text overflow

The [TextOverflow](#) property specifies how the overflowed content that is not displayed should be signaled to the user. The TextOverflow can have the following values.

- **Wrap:** Wraps the text to next line, when it exceeds its bounds.
- **Ellipsis:** It truncates the overflown text and render an ellipsis ("...") to represent the clipped text.
- **Clip:** The text is clipped and the overflow text will not be shown.

The following code sample shows how the different types of overflow property working for the different types of text wrapping.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = new DiagramNode()
{
Id = "node1",
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
// Sets the style for the text to be displayed
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "The text element with property of overflow as Wrap and wrapping as NoWrap",
Style = new AnnotationStyle()
{
TextOverflow = TextOverflow.Wrap,
TextWrapping=TextWrap.NoWrap
}
},
},
},
```

```

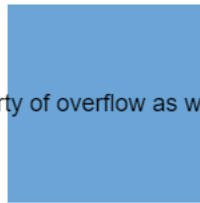
},
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
};
NodeCollection.Add(node1);
}
}

```

| TextOverflow | Wrapping | Image |

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

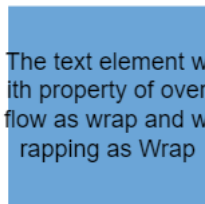
The text element with property of overflow as wrap and wrapping as NoWrap



| Wrap | No Wrap |

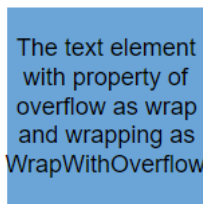


The text element with property of overflow as wrap and wrapping as Wrap



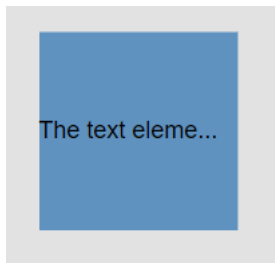
| Wrap | Wrap |

The text element with property of overflow as wrap and wrapping as WrapWithOverflow

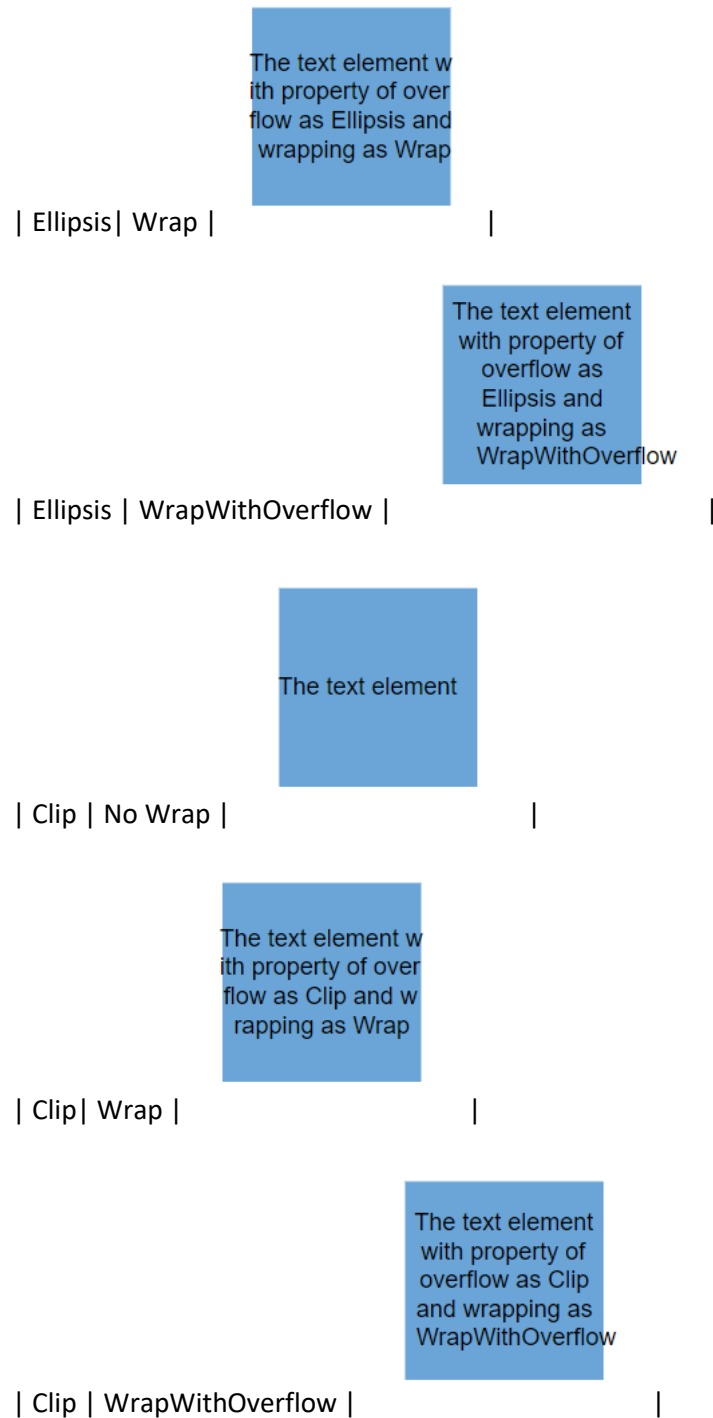


| Wrap | WrapWithOverflow |

The text eleme...



| Ellipsis | No Wrap |



---

All the customization over the overflow is also applicable to connector's annotation.

---

#### *Change the appearance of annotation*

You can change the font style of the annotations with the font specific properties (FontSize, FontFamily, Color). The following code explains how to customize the appearance of the annotation.

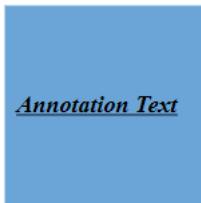
- The label's [Bold](#), [Italic](#), and [TextDecoration](#) properties are used to style the label's text.

- The label's [Fill](#), [StrokeColor](#), and [StrokeWidth](#) properties are used to define the background color and border color of the annotation and the [Opacity](#) property is used to define the transparency of the annotations.
- The [Visibility](#) property of the annotation enables or disables the visibility of annotation.

The Fill, Border, and Opacity appearances of the text can also be customized with appearance specific properties of annotation. The following code explains how to customize Background, Opacity, and Border of the annotation.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = new DiagramNode()
{
Id = "node1",
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
// Sets the annotation for the node
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Annotation Text",
Style = new AnnotationStyle()
{
Color="black",
Bold = true,
Italic = true,
TextDecoration=TextDecoration.Underline,
FontSize = 12,
FontFamily = "TimesNewRoman"
}
}
},
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
};
NodeCollection.Add(node1);
}
}
```



### Update the annotation style at runtime

You can change the font style of the annotations with the font specific properties (FontSize, FontFamily, and Color). The following code explains how to update the appearance of the annotation.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Update Style" @onclick="@UpdateStyle" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
    // Reference of the diagram
    SfDiagram Diagram;
    //Defines diagram's node collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node1 = new DiagramNode()
        {
            Id = "node1",
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            // Sets the annotation for the node
            Annotations = new ObservableCollection<DiagramNodeAnnotation>()
            {
                new DiagramNodeAnnotation()
                {
                    Content = "Annotation Text",
                    Style = new AnnotationStyle()
                    {
                        Color = "black",
                        Bold = true,
                        Italic = true,
                        TextDecoration = TextDecoration.Underline,
                        FontSize = 12,
                        FontFamily = "TimesNewRoman"
                    }
                }
            },
            Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
        };
        NodeCollection.Add(node1);
    }
}
```

```
public void UpdateStyle()
{
    // Change the style of the annotation
    Diagram.BeginUpdate();
    Diagram.Nodes[0].Annotations[0].Style.Bold = false;
    Diagram.Nodes[0].Annotations[0].Style.TextDecoration = TextDecoration.None;
    Diagram.Nodes[0].Annotations[0].Style.Color = "Red";
    Diagram.EndUpdate();
}
}
```

### *Change the editing mode*

Diagram provides support to edit an annotation at runtime, either programmatically or interactively. By default, the annotation is in view mode. But it can be brought to edit mode in two ways.

- You can edit the annotation programmatically by using the [StartTextEdit](#) method.
- Also, you can edit the annotation interactively.
- By double-clicking the annotation.
- By selecting the item and pressing the F2 key.

Double-clicking any annotation will enable the editing and the node enables first annotation editing. When the focus of editor is lost, the annotation for the node is updated. When you double-click the node/connector/diagram model, the [DoubleClick](#) event gets triggered.

### *Set annotation to read only*

Diagram allows to create read-only annotations. You have to set the read-only property of annotation to enable or disable the read-only [Constraints](#). The following code explains how to enable read-only mode.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
    //Defines diagram's node collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node1 = new DiagramNode()
        {
            Id = "node1",
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            //Sets the constraints as Read only
            Annotations = new ObservableCollection<DiagramNodeAnnotation>()
            {
                new DiagramNodeAnnotation()
                {
                    Content = "Annotation Text",
```



```

Constraints = AnnotationConstraints.ReadOnly
}
},
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
};
NodeCollection.Add(node1);
}
}

```

### Create multiple annotations

You can add any number of annotations to a node or connector. The following code example shows how to add multiple annotations to a node. Different labels by position is same or different point of the shapes of connector depends upon the offset values specified.

### ASPX-CS

```

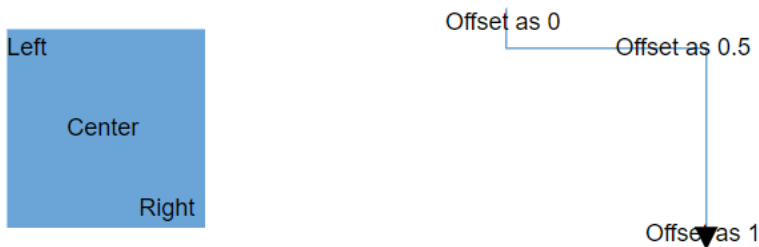
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = new DiagramNode()
{
Id = "node1",
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
},
// Sets the multiple annotation for the node
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Left", Offset = new NodeAnnotationOffset() { X = .12, Y = .1}
},
new DiagramNodeAnnotation()
{
Content = "Center", Offset = new NodeAnnotationOffset() { X = .5, Y = .5}
},
new DiagramNodeAnnotation()
{
Content = "Right", Offset = new NodeAnnotationOffset() { X = .82, Y = .9}
}
}
}
}

```

```

}
},
};
NodeCollection.Add(node1);
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
    SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 40 },
    TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 160 },
    Type = Segments.Orthogonal,
    Style = new ConnectorShapeStyle()
    {
        StrokeColor = "#6BA5D7"
    },
    Annotations = new ObservableCollection<DiagramConnectorAnnotation>()
    {
        new DiagramConnectorAnnotation() { Content = "Offset as 0", Offset=0 },
        new DiagramConnectorAnnotation() { Content = "Offset as 0.5", Offset=0.5 },
        new DiagramConnectorAnnotation() { Content = "Offset as 1", Offset=1 },
    }
};
ConnectorCollection.Add(connector);
}
}

```



- 
- \* Type of the annotation's property of the node or connector was ObservableCollection.
  - \* Default value of the annotation will be null.
  - \* All the same customization can be applicable for the annotations.
  - \* Text Editing can be stated only the first annotation of the annotation collection when you double click the node or connector.
- 

### Constraints

[AnnotationConstraints](#) are used to enable or disable certain behaviors of the annotation. Constraints are provided as flagged enumerations, so that multiple behaviors can be enabled or disabled with bitwise operators.

AnnotationConstraints may have multiple behaviors as follows:

| Constraints | Usages |

|---|---|

| ReadOnly | Enables or disables whether the annotation to be read only or not. |

- | Select | Enables or disables whether the annotation to be selectable. |
- | Drag | Enables or disables whether the annotation to be dragged. |
- | Resize | Enables or disables an Annotation to be Resized. |
- | Rotate | Enables or disables whether the annotation to be Rotated. |
- | Interaction | Enables or disables select, drag, resize and rotate behaviors. |
- | None | Disables all behaviors of Annotation. |

---

The default value is `AnnotationConstraints.InheritReadOnly` for constraints property of the annotation.

---

Refer to [Constraints](#) to learn about how to enable or disable the annotation constraints.

#### *Annotation template*

Diagram provides the template support for annotation. You should define an SVG/HTML content to the diagram's `AnnotationTemplate` property.

The following code explains how to define a template in node's annotation. similarly, you can define it in connectors.

#### **ASPX-CS**

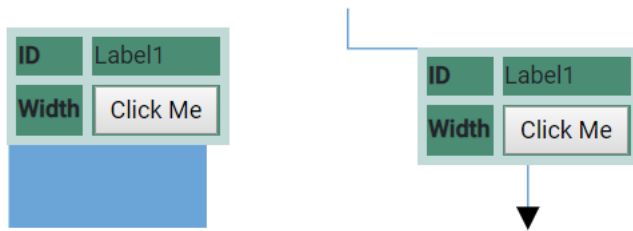
```
<style>
th {
border: 5px solid #c1dad7
}
td {
border: 5px solid #c1dad7
}
.c1 {
background: #4b8c74
}
.c2 {
background: #74c476
}
.c3 {
background: #a4e56d
}
.c4 {
background: #cffc83
}
</style>
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" ModelType="@model" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
<DiagramTemplates>
<AnnotationTemplate>
@{
<table style="width:100%;">
<tbody>
<tr>
<th class="c1">ID</th>
<td class="c1">Label1<b></b></td>
</tr>
<tr>
```

```

<th class="c1">Width</th>
<td class="c1">
<b> <input type="button" value="Click Me"> </b>
</td>
</tr>
</tbody>
</table>
}
</AnnotationTemplate>
</DiagramTemplates>
</SfDiagram>
@code
{
//Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
public Type model = typeof(Node);
public class Node
{
public string Id { get; set; }
public double Width { get; set; }
}
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramNode node = new DiagramNode()
{
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
},
};
node.Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation() {Id="label1",
AnnotationType=AnnotationType.Template }
};
NodeCollection.Add(node);
DiagramConnector connector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 220, Y = 40 },
TargetPoint = new ConnectorTargetPoint() { X = 310, Y = 150 },
Type = Segments.Orthogonal,
Style = new ConnectorShapeStyle()
{
StrokeColor = "#6BA5D7"
},
Annotations = new ObservableCollection<DiagramConnectorAnnotation>()
{

```

```
new DiagramConnectorAnnotation()
{
    Id="label2",
    AnnotationType=AnnotationType.Template
},
};
ConnectorCollection.Add(connector);
}
```



\* You need to specify the width value by default, if the annotation has template.

\* The AnnotationTemplate property accepts the template string.

*See also*

- [How to add or remove annotation constraints](#)
- [How to interact the annotation at runtime](#)
- [How to add annotation for Node](#)
- [How to add annotation for Connector](#)

### Interaction in Blazor Diagram Component

Basic interactions of selecting and resizing can be applied over an annotation. These interactions can be controlled by annotation and its parent node or connector. To learn about annotation constraints, refer to the [Annotation Constraints](#).

#### Selecting the annotation

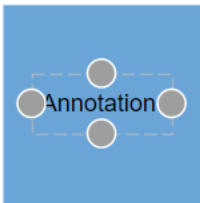
Selection of annotation can be enabled by using the [Constraints](#) property of `Annotation` and setting its value to `AnnotationConstraints.Select`.

The following code snippet explains how the select constraints are enabled for annotation.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
    //Defines diagram's node collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
```

```
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = new DiagramNode()
{
Id = "node1",
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
},
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Annotation",
Height=30,
Width=70,
Constraints=AnnotationConstraints.Select
}
},
};
NodeCollection.Add(node1);
}
}
```



#### *Dragging the annotation*

The dragging process can be applied over an annotation and dragging can be controlled by the annotation and its parent node or connector. Dragging of annotation can be enabled by using the Constraints property of annotation and setting its value to `AnnotationConstraints.Drag`.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
```

```
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node1 = new DiagramNode()
{
Id = "node1",
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
},
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Annotation",
Height=30,
Width=70,
Constraints=AnnotationConstraints.Select|AnnotationConstraints.Drag
}
},
};
NodeCollection.Add(node1);
}
}
```



### *Resizing the annotation*

Resizing of the annotation can be enabled by using the Constraints property of `Annotation` and setting its value to `AnnotationConstraints.Resize`. The following code snippet explains how the Resize constraints are enabled for annotation.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
    //Defines diagram's node collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection { get;
    set; }
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node1 = new DiagramNode()
        {
            Id = "node1",
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            Style = new NodeShapeStyle()
            {
                Fill = "#6BA5D7",
                StrokeColor = "white"
            },
            Annotations = new ObservableCollection<DiagramNodeAnnotation>()
            {
                new DiagramNodeAnnotation()
                {
                    Content = "Annotation",
                    Height=30,
                    Width=70,
                    Constraints=AnnotationConstraints.Select|AnnotationConstraints.Resize
                }
            },
        };
        NodeCollection.Add(node1);
    }
}
```





### *Rotate the annotation*

Resizing of the annotation can be enabled by using the Constraints property of `Annotation` and setting its value to `AnnotationConstraints.Resize`. The following code snippet explains how the Resize constraints are enabled for annotation.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code
{
    //Defines diagram's node collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection { get;
    set; }
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node1 = new DiagramNode()
        {
            Id = "node1",
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            Style = new NodeShapeStyle()
            {
                Fill = "#6BA5D7",
                StrokeColor = "white"
            },
            Annotations = new ObservableCollection<DiagramNodeAnnotation>()
            {
                new DiagramNodeAnnotation()
                {
                    Content = "Annotation",
                    Height=30,
                    Width=70,
                    Constraints=AnnotationConstraints.Select|AnnotationConstraints.Rotate
                }
            },
        };
    }
}
```

```
NodeCollection.Add(node1);
}
}
```



#### How to restrict the dragging area

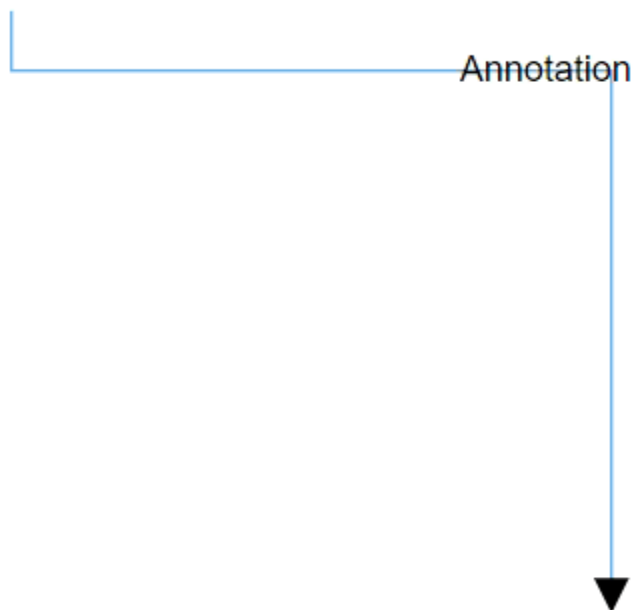
The diagram control now supports defining the [DragLimit](#) to the label when dragging from the connector and also update the position to the nearest segment offset. You can set the value to dragLimit left, right, top, and bottom properties that allows dragging of connector labels to a certain limit based on the user defined values.

The following code explains how to set a dragLimit for connector annotations.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection { get;
    set; }
    protected override void OnInitialized()
    {
        ConnectorCollection = new ObservableCollection<DiagramConnector>();
        DiagramConnector connector = new DiagramConnector()
        {
            SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
            TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 300 },
            Type = Segments.Orthogonal,
            Style = new ConnectorShapeStyle()
            {
                StrokeColor = "#6BA5D7"
            },
            Annotations = new ObservableCollection<DiagramConnectorAnnotation>()
            {
                new DiagramConnectorAnnotation()
                {
                    Content = "Annotation",
                    Constraints=AnnotationConstraints.Select|AnnotationConstraints.Drag,
```

```
DragLimit=new ConnectorAnnotationDragLimit() {  
Left=10,Top=10,Right=10,Bottom=10}  
},  
};  
ConnectorCollection.Add(connector);  
}  
}
```



---

\* By default, the drag limit will be disabled for the connector. It can be enabled by setting connector constraints to drag.

\* The drag limit is applicable only for the connector.

---

*See also*

- [How to add or remove annotation constraints](#)
- [How to customize the annotation](#)
- [How to add annotation for Node](#)
- [How to add annotation for Connector](#)

## Events in Blazor Diagram Component

### Text edit

The TextEdit event will notify the annotation content changes after editing. The [IBlazorTextEditEventArgs](#) interface is used to get event arguments.

The following code example shows how to register and get the notification from the TextEdit event.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
  <DiagramEvents TextEdited="@TextEdited"></DiagramEvents>
</SfDiagram>
@code
{
  //Defines diagram's nodes collection
  public ObservableCollection<DiagramNode> NodeCollection { get; set; }
  //Triggered this event when complete the editing for Annotation and update
  the old text and new text values.
  private void TextEdited(IBlazorTextEditEventArgs args)
  {
    Console.WriteLine("Oldvalue", args.OldValue);
    Console.WriteLine("NewValue", args.NewValue);
  }
  protected override void OnInitialized()
  {
    NodeCollection = new ObservableCollection<DiagramNode>();
    DiagramNode node = new DiagramNode()
    {
      Width = 100,
      Height = 100,
      OffsetX = 100,
      OffsetY = 100,
      Style = new NodeShapeStyle()
      {
        Fill = "#6BA5D7",
        StrokeColor = "white"
      },
    };
    node.Annotations = new ObservableCollection<DiagramNodeAnnotation>()
    {
      new DiagramNodeAnnotation() {Content = "Annotation" }
    };
    NodeCollection.Add(node);
  }
}
```

### Double click

The DoubleClick event will notify the annotation start editing. The [IDoubleClickEventArgs](#) interface is used to get the position actually clicked and clicked object.

The following code example shows how to register and get the notification from the [OnDoubleClick](#) event.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
<DiagramEvents OnDoubleClick="@OnDoubleClick" />
</SfDiagram>
@code
{
//Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection
{ get; set; }
//Triggered this event when double click on the Annotation and update the
position and source for clicked item.
private void OnDoubleClick(IBlazorDoubleClickEventArgs args)
{
Console.WriteLine("Position", args.Position);
Console.WriteLine("Source", args.Source);
}
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
},
};
node.Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation() {Content = "Annotation" }
};
NodeCollection.Add(node);
}
}

```

### Key down

The keydown event occurs when a keyboard key is pressed down and updated the respective keyboard key pressed.

### Key up

The keyup event occurs when a keyboard key is released and updated the respective keyboard key pressed.

The following code example shows how to register and get the notification from the onkeydown and onkeyup events.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">

```

```

<DiagramEvents OnKeyDown="@OnKeyDown" OnKeyUp="@OnKeyUp"></DiagramEvents>
</SfDiagram>
@code
{
//Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection
{ get; set; }
//Occurs when click the annotation and enter the character in key down
state
private void OnKeyDown(IKeyEventArgs args)
{
}
//Occurs when click the annotation and enter the character in key release
state
private void OnKeyUp(IKeyEventArgs args)
{
}
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
},
};
node.Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation() {Content = "Annotation" }
};
NodeCollection.Add(node);
}
}

```

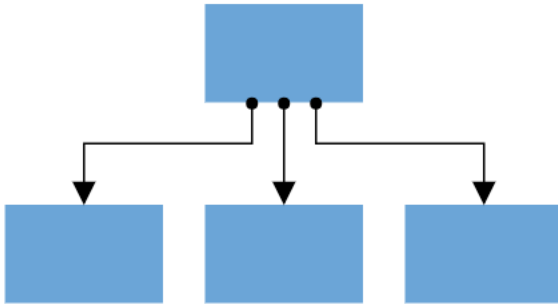
*See also*

- [How to add or remove annotation constraints](#)
- [How to customize the annotation](#)
- [How to interact the annotation at runtime](#)
- [How to add annotation for Node](#)
- [How to add annotation for Connector](#)

## Ports

### Actions of port in Blazor Diagram Component

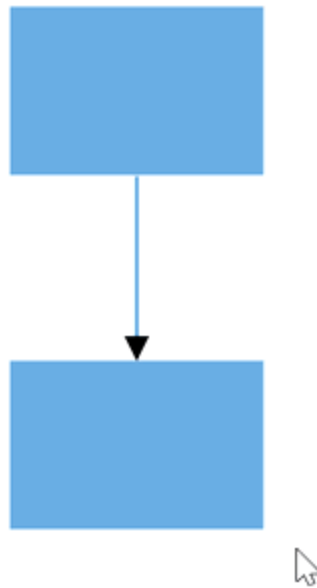
Port is a special connection point in a Node that you can glue the connectors. When you glue a connector to a node or port, they stay connected, even if one of the node is moved.



### *Connections*

There are two main types of connections, dynamic and port. The difference between these two connections is whether or not a connector remains glued to a specific connection point when you move the attached node or connector.

A dynamic connection is one where the connector will move around the node as you move the node. Diagram will always ensure the connector is the shortest, most direct line possible. You can create a dynamic connection by selecting the entire node (rather than the port) and connect it to another shape (rather than to a port).



Ports act as the connection points of the node and allows creating connections with only those specific points as shown in the following image.





Inbox - keerthivasan.ramamoorth

### Create ports

To add a connection port, define the port object and add it to node's ports collection. The [Offset](#) property of the port accepts an object of fractions and used to determine the position of ports. The following code explains how to add ports when initializing the node.

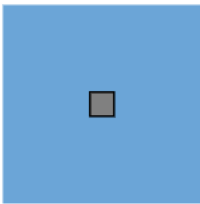
### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes collection.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
```

```

OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
// Initialize port collection
Ports = new ObservableCollection<DiagramPort>()
{
    // Sets the position for the port
    new DiagramPort()
    {
        Style=new PortShapeStyle(){ Fill="gray" },
        Offset= new NodePortOffset(){X=0.5,Y=0.5},
        Visibility = PortVisibility.Visible
    }
};
NodeCollection.Add(node1);
}
}

```



#### Add ports at runtime

Add ports at runtime by using the server-side method [Add] in the port collection. The following code explains how to add ports to node at runtime.

The port's [Id](#) property is used to define the unique ID for the port and it is further used to find the port at runtime. If **Id** is not set, then default **Id** is automatically set.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="AddPorts" @onclick="@AddPorts" />
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        //A node is created and stored in nodes array
        DiagramNode node1 = new DiagramNode()
        {
            //Position of the node
            OffsetX = 250,

```

```

OffsetY = 250,
//Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
};
NodeCollection.Add(node1);
}
public void AddPorts()
{
// Initialize port collection
NodeCollection[0].Ports.Add(new DiagramPort()
{
Id = "port1",
Offset = new NodePortOffset() { X = 0, Y = 0.5 },
Visibility = PortVisibility.Visible
});
NodeCollection[0].Ports.Add(new DiagramPort()
{
Id = "port2",
Offset = new NodePortOffset() { X = 1, Y = 0.5 },
Visibility = PortVisibility.Visible
});
NodeCollection[0].Ports.Add(new DiagramPort()
{
Id = "port3",
Offset = new NodePortOffset() { X = 0.5, Y = 0 },
Visibility = PortVisibility.Visible
});
NodeCollection[0].Ports.Add(new DiagramPort()
{
Id = "port4",
Offset = new NodePortOffset() { X = 0.5, Y = 1 },
Visibility = PortVisibility.Visible
});
}
}

```

### *Remove ports at runtime*

A collection of ports can be removed from the node by using the native[RemoveAt`] method. Refer to the following example that shows how to remove ports at runtime.

### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="RemovePorts" @onclick="@RemovePorts" />
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in nodes array.
DiagramNode node1 = new DiagramNode()

```

```

{
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
// Initialize port collection
Ports = new ObservableCollection<DiagramPort>()
{
new DiagramPort()
{
Id = "port1",
Offset = new NodePortOffset() { X = 0, Y = 0.5 },
Visibility = PortVisibility.Visible,
//Set the style for the port
Style= new PortShapeStyle(){ Fill="red", StrokeColor="black",
StrokeWidth=2},
// Sets the shape of the port as Circle
Width= 12,
Height=12,
Shape= PortShapes.Circle
}
},
};
NodeCollection.Add(node1);
}
public void RemovePorts()
{
(NodeCollection[0].Ports as ObservableCollection<DiagramPort>).RemoveAt(0);
}
}

```

### Update ports at runtime

You can change any port properties at runtime.

The following code example explains how to change the port properties at runtime.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Update Port" @onclick="@UpdatePort" />
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// Initialize port collection
ObservableCollection<DiagramPort> Ports1 = new
ObservableCollection<DiagramPort>() { };
Ports1.Add(new DiagramPort()
{

```

```
Id = "port1",
Offset = new NodePortOffset() { X = 0, Y = 0.5 },
Visibility = PortVisibility.Visible
});
// A node is created and stored in nodes array
DiagramNode node1 = new DiagramNode()
{
    // Position of the node
    OffsetX = 250,
    OffsetY = 250,
    // Size of the node
    Width = 100,
    Height = 100,
    Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
    Ports = Ports1
};
NodeCollection.Add(node1);
}
public void UpdatePort()
{
    //update ports at run time
    NodeCollection[0].BeginUpdate();
    NodeCollection[0].Ports[0].Offset.X = 1;
    NodeCollection[0].Ports[0].Offset.Y = 1;
    NodeCollection[0].EndUpdate();
}
}
```

*See also*

- [How to create a node](#)
- [How to customize the ports](#)
- [How to interact the ports](#)
- [How to set the position of the port](#)

### Positioning in Blazor Diagram Component

Diagram allows you to customize the position and appearance of the port efficiently. Port can be aligned relative to the node boundaries. It has Margin, Offset, Horizontal, and Vertical alignment settings. It is quite tricky when all four alignments are used together but gives more control over alignments properties of the [DiagramPort](#) class. Ports of a node can be positioned using the following properties of [DiagramPort](#).

- [Offset](#)
- [HorizontalAlignment](#)
- [VerticalAlignment](#)
- [Margin](#)

### Offset

The [Offset](#) property is used to align the Ports based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in nodes collection.
DiagramNode node1 = new DiagramNode()
{
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
// Initialize port collection
Ports = new ObservableCollection<DiagramPort>()
{
new DiagramPort()
{
Id = "port1",
Offset = new NodePortOffset() { X = 0, Y = 0.5 },
Visibility = PortVisibility.Visible,
//Set the style for the port
Style= new PortShapeStyle(){ Fill="gray", StrokeColor="black"},
// Sets the shape of the port as Square
Width= 12,
Height=12,
Shape= PortShapes.Square,
}
},
};
NodeCollection.Add(node1);
}
}

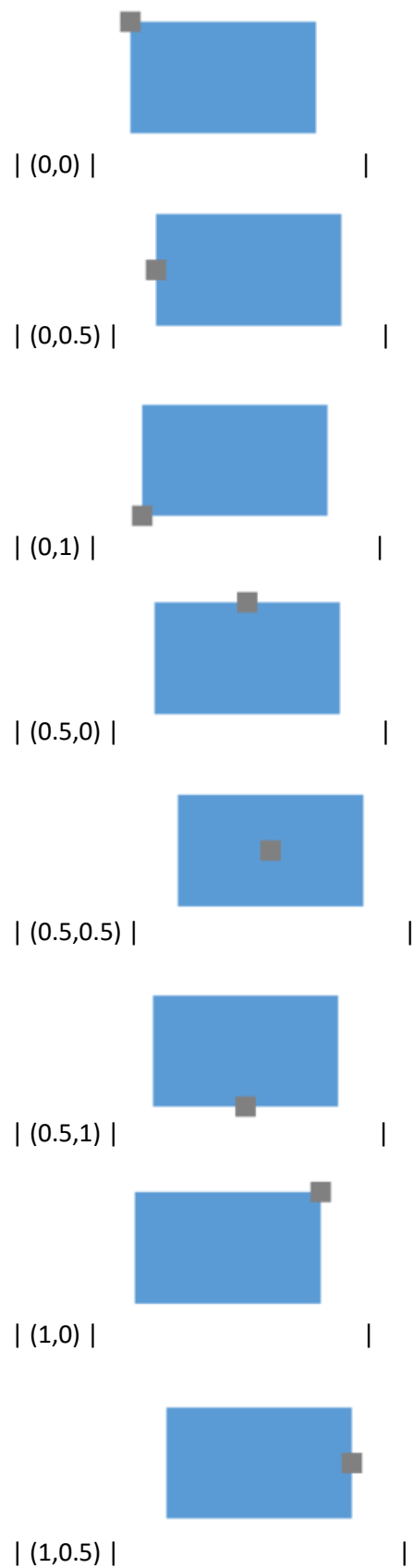
```



The following code shows the relationship between the shape port position and path port offset (fraction values).

| Offset values | Output |

|---|---|





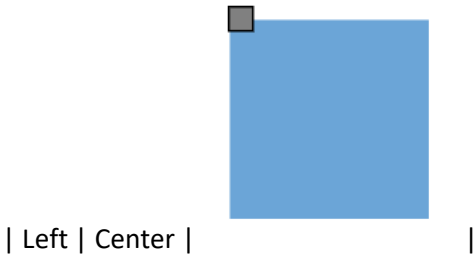
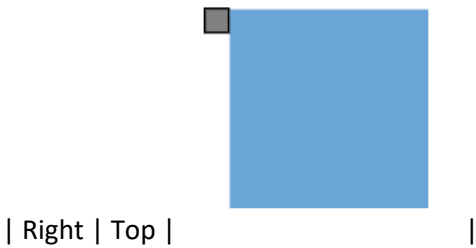
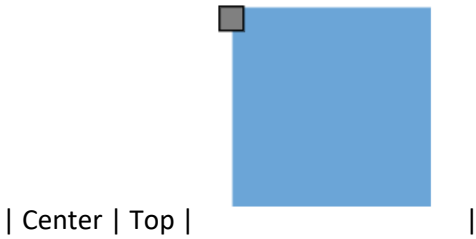
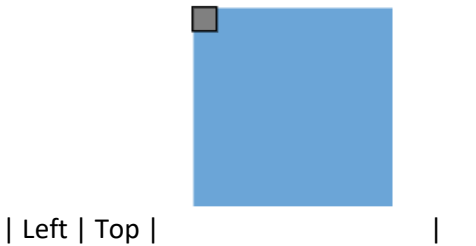
Type of the offset property for node’s shape port is NodePortOffset.

Horizontal and vertical alignment

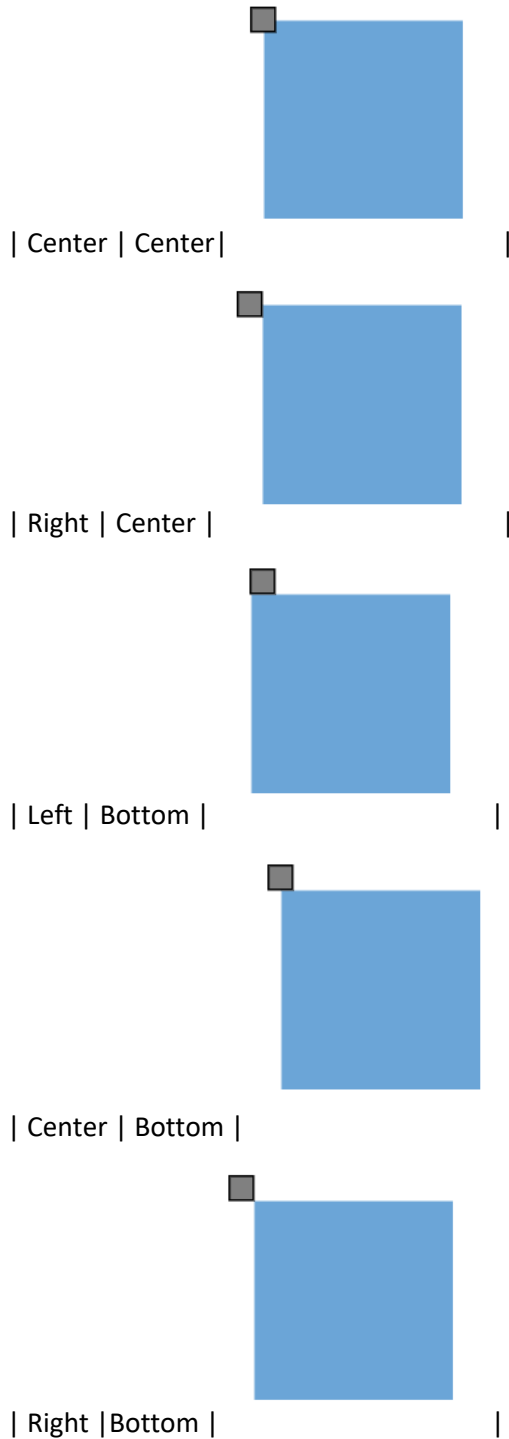
The [HorizontalAlignment](#) property of the port is used to set how the port is horizontally aligned at the port position determined from the fraction values. The [VerticalAlignment](#) property is used to set how the port is vertically aligned at the port position.

The following table shows all the possible alignments visually with offset (0, 0).

Horizontal Alignment	Vertical Alignment	Output with Offset(0,0)
-----	-----	-----







The following code explains how to align ports.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
```

```

public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
    // A node is created and stored in nodes array.
    DiagramNode node1 = new DiagramNode()
    {
        // Position of the node
        OffsetX = 250,
        OffsetY = 250,
        // Size of the node
        Width = 100,
        Height = 100,
        Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
        // Initialize port collection
        Ports = new ObservableCollection<DiagramPort>()
        {
            new DiagramPort()
            {
                Id = "port1",
                Offset = new NodePortOffset() { X = 0, Y = 0 },
                Visibility = PortVisibility.Visible,
                //Set the style for the port
                Style= new PortShapeStyle(){ Fill="gray", StrokeColor="black"},
                // Sets the shape of the port as Square
                Width= 12,
                Height=12,
                Shape= PortShapes.Square,
                HorizontalAlignment = HorizontalAlignment.Left,
                VerticalAlignment = VerticalAlignment.Top
            }
        },
    };
    NodeCollection.Add(node1);
}

```

---

\* The value of the `HorizontalAlignment` is `Center` by default.

\* The value of the `VerticalAlignment` is `Center` by default.

\* Alignment positioned based on the offset value.

---

### Margin

[Margin](#) is an absolute value used to add some blank space to any one of its four sides. The ports can be displaced with the margin property. The following code example explains how to align an port based on its Offset, HorizontalAlignment, VerticalAlignment, and Margin values.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{

```

```

public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
    // A node is created and stored in nodes array.
    DiagramNode node1 = new DiagramNode()
    {
        // Position of the node
        OffsetX = 250,
        OffsetY = 250,
        // Size of the node
        Width = 100,
        Height = 100,
        Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
        // Initialize port collection
        Ports = new ObservableCollection<DiagramPort>()
        {
            new DiagramPort()
            {
                Id = "port1",
                Offset = new NodePortOffset() { X = 0.5, Y = 1 },
                Visibility = PortVisibility.Visible,
                //Set the style for the port
                Style= new PortShapeStyle(){ Fill="gray", StrokeColor="black"},
                // Sets the shape of the port as Circle
                Width= 12,
                Height=12,
                Shape= PortShapes.Square,
                HorizontalAlignment = HorizontalAlignment.Left,
                VerticalAlignment = VerticalAlignment.Top,
                Margin=new PortMargin() {Top=10}
            }
        },
    };
    NodeCollection.Add(node1);
}

```



*See also*

- [How to create a node](#)
- [How to customize the ports](#)
- [How to interact the ports](#)

## Appearance in Blazor Diagram Component

### Appearance

- The shape of a port can be changed by using the [Shape](#) property. To explore the different types of port shapes, refer to Port Shapes. If you need to render a custom shape, then you can set shape to path and define path using the path data property.
- The appearance of the ports can be customized by using the [StrokeColor](#), [StrokeWidth](#), and [Fill](#) properties.
- Customize the port size by using the [Width](#) and [Height](#) properties of port.
- The ports [Visibility](#) property allows you to define when the port should be visible.

The following code explains how to change the appearance of the port.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in nodes array.
DiagramNode node1 = new DiagramNode()
{
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
// Initialize port collection
Ports = new ObservableCollection<DiagramPort>()
{
new DiagramPort()
{
Id = "port1",
Offset = new NodePortOffset() { X = 0, Y = 0.5 },
Visibility = PortVisibility.Visible,
//Set the style for the port
Style= new PortShapeStyle(){ Fill="red", StrokeColor="black",
StrokeWidth=2},
// Sets the shape of the port as Circle
Width= 12,
Height=12,
Shape= PortShapes.Circle
}
},
};
NodeCollection.Add(node1);
}
```



### Visibility

The visibility of the ports depends upon the properties of **Connect**, **Hidden**, **Hover**, and **Visible**. By default, **PortVisibility** is set to **Hidden**.

| Property | Definition |

|---|---|

| **Hover** | Port is visible when mouseover the DiagramElement. |

| **Hidden** | Port is not visible for the DiagramElement. |

| **Connect** | Specifies to visible the port when mouseover the DiagramElement and enable the **PortConstraints** as **InConnect** and **OutConnect**. |

| **Visible** | Port is always visible for the DiagramElement. |

| **Default** | By default, **PortVisibility** is set to **Hidden**. So, the port is not visible for the DiagramElement. |

### Types of port shapes

We have provided some basic built-in [PortShapes](#) for the port. Please find the shapes as follows.

- Circle
- Custom
- Square
- X

### Custom shape

We have provided custom shape support for port. you can able to add the custom path data instead of build-in shapes. Please find the code example that explains how to change the custom shape for port.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes array.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
```

```

OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
// Initialize port collection
Ports = new ObservableCollection<DiagramPort>()
{
    new DiagramPort()
    {
        Id = "port1",
        Offset = new NodePortOffset() { X = 0.5, Y = 0.5 },
        Visibility = PortVisibility.Visible,
        //Set the style for the port
        Style= new PortShapeStyle(){ Fill="gray", StrokeColor="black"},
        // Sets the shape of the port as Circle
        Width= 12,
        Height=12,
        Shape= PortShapes.Custom,
        PathData="M540.3643,137.9336L546.7973,159.7016L570.3633,159.7296L550.7723,171.9366L558.9053,194.9966L540.3643,179.4996L521.8223,194.9966L529.9553,171.9366L510.3633,159.7296L533.9313,159.7016L540.3643,137.9336z"
    }
},
};
NodeCollection.Add(node1);
}
}

```



### Constraints

The constraints property allows you to enable or disable certain behaviors of ports. For more information about port constraints, refer to [Port Constraints](#). You can verify the [Constraints](#) to learn how to enable or disable the port constraints.

The PortConstraints may have multiple behaviors listed as follows:

| Constraints | Usages |

|---|---|

| None | Disables all behaviors of Port. |

| Drag | Defines whether port to be dragged at boundaries of node. |

| Draw | Enables or disables to draw a connector. |

| InConnect | Enables or disables connecting to the incoming Connector. |

| OutConnect | Enables or disables connecting the outgoing Connector. |

#### *Custom properties*

The [AddInfo](#) property of the port allows you to maintain additional information to the port.

*See also*

- [How to create a node](#)
- [How to customize the ports](#)
- [How to set the position of the port](#)
- [How to interact the ports](#)

#### Interaction in Blazor Diagram Component

The port can be dragged in the diagram area and create the connector over the port by using the [port constraints](#).

#### *Drag*

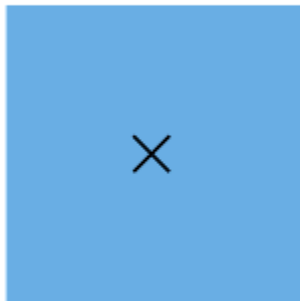
Diagram provides the support to drag the port interactively, it can be dragged out of the node bounds.

The following code explains how to drag the port.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in nodes array.
DiagramNode node1 = new DiagramNode()
{
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
// Initialize port collection
Ports = new ObservableCollection<DiagramPort>()
{
new DiagramPort()
{
Id = "port1",
Offset = new NodePortOffset() { X = 0.5, Y = 0.5 },
Visibility = PortVisibility.Visible,
//Set the style for the port
Style= new PortShapeStyle(){ Fill = "gray", StrokeColor = "black"},
// Sets the shape of the port as Circle
Width = 12,
Height = 12,
Shape = PortShapes.X,
```

```
// Enable drag operation for Port
Constraints = PortConstraints.Default|PortConstraints.Drag
}
},
};
NodeCollection.Add(node1);
}
}
```



### Draw

Diagram provides the support to draw the connector in the port.

The following code explains how to draw the connector by using the port constraints.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in nodes array.
DiagramNode node1 = new DiagramNode()
{
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
```



```
// Initialize port collection
Ports = new ObservableCollection<DiagramPort>()
{
    new DiagramPort()
    {
        Id = "port1",
        Offset = new NodePortOffset() { X = 1, Y = 0.5 },
        Visibility = PortVisibility.Visible,
        //Set the style for the port
        Style = new PortShapeStyle(){ Fill = "gray", StrokeColor = "black"},
        // Sets the shape of the port as Circle
        Width = 12,
        Height = 12,
        Shape = PortShapes.Circle,
        // Enable drag operation for Port
        Constraints = PortConstraints.Default | PortConstraints.Draw
    }
},
};
NodeCollection.Add(node1);
}
```



*See also*

- [How to create a node](#)
- [How to customize the ports](#)
- [How to set the position of the port](#)

## Constraints in Blazor Diagram Component

Constraints are used to enable or disable certain behaviors of the diagram, nodes, and connectors. Constraints are provided as flagged enumerations, so that multiple behaviors can be enabled or disabled using the Bitwise operators (&, |, ~, <<, etc.).

To know more about Bitwise operators, refer to the [Bitwise Operations](#).

### Diagram constraints

Diagram constraints allow you to enable or disable the following behaviors:

- Page editing
- Bridging
- Zoom and pan
- Undo or redo
- Tooltip

The following list of diagram constraints are used to Enables or Disables certain features of the diagram.

Constraints	Description
None	Disable all diagram functionalities
Bridging	Enables or Disable Bridging support for connector in diagram
Undo/redo	Enables or Disable the Undo/Redo support for the diagram
Tooltip	Enables or Disable Tooltip support support for the diagram
UserInteraction	Enables or Disable user interaction support for the diagram
ApiUpdate	Enables or Disable update API support for the diagram
PageEditable	Enables or Disable Page Editable support for the diagram
Zoom	Enables or Disable Zoom support for the diagram
PanX	Enables or Disable Paning X coordinate support for the diagram
PanY	Enables or Disable Paning Y coordinate support for the diagram
Pan	Enables or Disable panning both X and Y coordinates support for the diagram
ZoomTextEdit	Enables or Disables zooming the text box while editing the text
LineRouting	Enables or Disable the line routing for the diagram
Virtualization	Enables or Disable Virtualization support the diagram
Default	Enables or Disable all constraints in diagram

The following example shows how to disable page editing using the diagram constraints.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize the diagram with constraints *@
```

```
<SfDiagram Height="600px" Nodes="@NodeCollection"
Constraints="@DiagramConstraints">
</SfDiagram>
@code{
//sets the Diagram constraints...
DiagramConstraints DiagramConstraints = DiagramConstraints.Default &
~DiagramConstraints.PageEditable;
//Initialize the Nodes Collection.
ObservableCollection<DiagramNode> NodeCollection;
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
Id = "node1",
Height = 100,
Width = 100,
OffsetX = 100,
OffsetY = 100,
};
NodeCollection.Add(node);
}
}
```

The following another code example shows how the virtualization can be enabled for the diagram.

#### CSHARP

```
//enabled the Virtualization functionality for the diagram
Diagram.Constraints = DiagramConstraints.Default |
DiagramConstraints.Virtualization;
```

For more information about diagram constraints, refer to the [DiagramConstraints](#).

By default, the following constraints are enabled in the diagram,

- \* UndoRedo
- \* UserInteraction
- \* ApiUpdate
- \* PageEditable
- \* Zoom
- \* Pan

The diagram constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#) in the diagram.

#### Node constraints

Node constraints allows you to enable or disable the following behaviors of node. They are as follows:

- Selection
- Deletion
- Drag

- Resize
- Rotate
- Connect
- Shadow
- Tooltip

The following list of node constraints are used to Enables or Disables certain features of node.

Constraints	Description
-----	-----
None	Disable all node Constraints
Select	Enables or Disables node to be selected
Drag	Enables or Disables node to be Dragged
Rotate	Enables or Disables node to be rotating
Shadow	Enables or disables node to display shadow
PointerEvents	Enables or disables node to provide pointer option
Delete	Enables or Disables node to be deleting
InConnect	Enables or disables node to provide in connect option
OutConnect	Enables or disables node to provide out connect option
Individual	Enables node to provide individual resize option
Expandable	Enables node to provide Expandable option
AllowDrop	Enables node to provide allow to drop option
Inherit	Enables node to inherit the interaction option from the parent object
ResizeNorthEast	Enable or disable to Resizing NorthEast side of the node
ResizeEast	Enable or disable to Resizing East side of the node
ResizeSouthEast	Enable or disable to Resizing SouthEast side of the node
ResizeSouth	Enable or disable to Resizing South side of the node
ResizeSouthWest	Enable or disable to Resizing SouthWest side of the node
ResizeWest	Enable or disable to Resizing West side of the node
ResizeNorthWest	Enable or disable to Resizing NorthWest side of the node
ResizeNorth	Enable or disable to Resizing North side of the node
Resize	Enables or Disables to Resizing of the node
AspectRatio	Enables the Aspect ratio of the node
Tooltip	Enables or disables tool tip for the Nodes
InheritTooltip	Enables or disables inherit tool tip option from the parent object
ReadOnly	Enables the ReadOnly support for annotation in node

|HideThumbs| Enable to hide all resize thumbs for the node|

|AllowMovingOutsideLane| Enables or disables child in parent for the swimLane node|

|Default| Enables all default constraints for the node|

The following example shows how to disable rotation using the node constraints.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize the diagram with NodeCollection */
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    ObservableCollection<DiagramNode> NodeCollection;
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node = new DiagramNode()
        {
            Id = "node1",
            Height = 100,
            Width = 100,
            OffsetX = 100,
            OffsetY = 100,
            //sets the NodeConstraints constraints...
            Constraints = NodeConstraints.Default & ~NodeConstraints.Rotate
        };
        NodeCollection.Add(node);
    }
}
```

The following another code example shows how the tooltip can be enabled for the node.

#### CSHARP

```
//enabled the tooltip constraints for the node
node.Constraints = NodeConstraints.Default | NodeConstraints.Tooltip;
```

For more information about node constraints, refer to the [NodeConstraints](#).

**Note :** By default, the following constraints are enabled for the node,

- \* Select
- \* Drag
- \* Resize
- \* Rotate
- \* Delete
- \* InConnect
- \* OutConnect

\* Expandable

\* InheritTooltip

The node constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#).

### Connector constraints

Connector constraints allow you to enable or disable certain behaviors of connectors.

- Selection
- Deletion
- Drag
- Segment editing
- Tooltip
- Bridging

The following list of connector constraints are used to Enables or Disables certain features of connectors.

Constraints	Description
-----	-----
None	Disable all connector Constraints
Select	Enables or Disables node to be selected
Delete	Enables or Disables node to be deleting
Drag	Enables or Disables node to be Dragged
DragSourceEnd	Enables connectors source end to be selected
DragTargetEnd	Enables connectors target end to be selected
DragSegmentThumb	Enables control point and end point of every segment in a connector for editing
AllowDrop	Enables allow drop support to the connector
Bridging	Enables bridging to the connector
BridgeObstacle	Enables or Disables Bridge Obstacles with overlapping of connectors
InheritBridging	Enables to inherit bridging option from the parent object
PointerEvents	Enables to set the pointer events
Tooltip	Enables or disables tool tip for the connectors
InheritTooltip	Enables or disables to inheriting tool tip option from the parent object
Interaction	Enables or disables Interaction for the connector
ReadOnly	Enables or disables readonly for the connector
LineRouting	Enables or disables routing to the connector
InheritLineRouting	Enables or disables to inheriting routing option from the parent
Default	Enables all constraints for the connector

The following code shows how to disable selection by using the connector constraints.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize the diagram with ConnectorCollection */
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
ObservableCollection<DiagramConnector> ConnectorCollection;
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
Id = "connector1",
Type = Segments.Straight,
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
//sets the ConnectorConstraints...
Constraints = ConnectorConstraints.Default & ~ConnectorConstraints.Select
};
ConnectorCollection.Add(connector);
}
}
```

The following another code example shows how the tooltip can be enabled for the connector.

#### CSHARP

```
//enabled the tooltip constraints for the connector
connector.Constraints = ConnectorConstraints.Default |
ConnectorConstraints.Tooltip;
```

For more information about connector constraints, refer to the [ConnectorConstraints](#).

By default, the following constraints are enabled for the connector,

- \* Select
- \* Drag
- \* DragSourceEnd
- \* DragTargetEnd
- \* Delete
- \* BridgeObstacle
- \* InheritBridging
- \* PointerEvents
- \* InheritTooltip
- \* InheritLineRouting

**Note :** The connector constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#).

### Port constraints

You can enable or disable certain behaviors of port. They are as follows:

- Connect
- ConnectOnDrag

The following list of port constraints are used to Enable or Disable certain features of ports.

Constraints	Description
-----	-----
None	Disable all port Constraints
Drag	Enables or disables to drag the port
Draw	Enables to create the connection when mouse hover on the port
InConnect	Enables or disables to only connect the target end of connector
OutConnect	Enables or disables to only connect the source end of connector
Default	Enables all constraints for the port

The following code shows how to disable creating connections with a port.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize the diagram with NodeCollection */
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
ObservableCollection<DiagramNode> NodeCollection;
protected override void OnInitialized()
{
//Initialize the NodeCollection.
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
Id = "node1",
Height = 100,
Width = 100,
OffsetX = 100,
OffsetY = 100,
};
node.Ports = new ObservableCollection<DiagramPort>()
{
new DiagramPort()
{
Id="port1",
Offset=new NodePortOffset() {X=0,Y=0.5},
Shape=PortShapes.Circle,
Visibility=PortVisibility.Visible,
//sets the PortConstraints...
```



```

Constraints=PortConstraints.None
}
};
NodeCollection.Add(node);
}
}

```

The following another code example shows to modify the port constraints to accept target connection alone.

### CSHARP

```

//Enable to create target connection alone.
port.Constraints = PortConstraints.InConnect;

```

For more information about port constraints, refer to the [PortConstraints](#).

By default, the following constraints are enabled for the port,

\* InConnect

\* OutConnect

**Note :** The port constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#).

### Annotation constraints

You can enable or disable read-only mode for the annotations by using the annotation constraints.

The following list of annotation constraints are used to Enables or Disables certain features of annotations.

Constraints	Description
-----   -----	
ReadOnly	Enables or Disables the ReadOnly Constraints
InheritReadOnly	Enables or Disables to inherit the ReadOnly option from the parent object
Select	Enables or Disable select support for the annotation
Drag	Enables or Disable drag support for the annotation
Resize	Enables or Disable resize support for the annotation
Rotate	Enables or Disable rotate support for the annotation
Interaction	Enables or Disable annotation to inherit the interaction option
None	Disables all constraints for the annotation

The following code shows how to enable read-only mode for the annotations.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize the diagram with NodeCollection *@
<SfDiagram Height="600px" Nodes="@NodeCollection">

```

```
</SfDiagram>
@code{
ObservableCollection<DiagramNode> NodeCollection;
protected override void OnInitialized()
{
//Initialize the NodeCollection.
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
Id = "node1",
Height = 100,
Width = 100,
OffsetX = 100,
OffsetY = 100,
};
node.Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Id="annotation1",
Content="Annotation Text Wrapping",
//sets the Constraints for Annotation...
Constraints=AnnotationConstraints.ReadOnly,
Style=new AnnotationStyle(){Color="#000000", Fill="transparent",
FontFamily="TimesNewRoman", FontSize=12, Bold=true, Italic=true},
}
};
NodeCollection.Add(node);
}
}
```

The following another code example shows how to enable the selection and dragging functionality for the annotation.

#### CSHARP

```
//Enable select and drag constraints for the annotation.
annotation.Constraints = AnnotationConstraints.Select |
AnnotationConstraints.Drag;
```

For more details about annotation constraints, refer to the [AnnotationConstraints](#).

The annotation constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#).

#### Selector constraints

Selector visually represents the selected elements with certain editable thumbs. The visibility of the thumbs can be controlled with selector constraints. The part of selector is categorized as follows:

- Resizer
- Rotator
- User handles

The following list of selector constraints are used to Enables or Disables certain features of selected items.

Constraints	Description
-----	-----
None	Hides all the selector elements
ConnectorSourceThumb	Shows or hides the source thumb of the connector
ConnectorTargetThumb	Shows or hides the target thumb of the connector
ResizeSouthEast	Shows or hides the bottom right resize handle of the selector
ResizeSouthWest	Shows or hides the bottom left resize handle of the selector
ResizeNorthEast	Shows or hides the top right resize handle of the selector
ResizeNorthWest	Shows or hides the top left resize handle of the selector
ResizeEast	Shows or hides the middle right resize handle of the selector
ResizeWest	Shows or hides the middle left resize handle of the selector
ResizeSouth	Shows or hides the bottom center resize handle of the selector
ResizeNorth	Shows or hides the top center resize handle of the selector
Rotate	Shows or hides the rotate handle of the selector
UserHandle	Shows or hides the user handles of the selector
ToolTip	Shows or hides the default tooltip of selected items
ResizeAll	Shows or hides all resize handles of the selector
All	Shows all handles of the selector

The following code shows how to hide rotator.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection"
SelectedItems="@selectedItems">
</SfDiagram>
@code{
ObservableCollection<DiagramNode> NodeCollection;
DiagramSelectedItems selectedItems = new DiagramSelectedItems() {
Constraints = SelectorConstraints.All & ~SelectorConstraints.Rotate };
protected override void OnInitialized()
{
//Initialize the NodeCollection.
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
Id = "node1",
Height = 100,
Width = 100,
OffsetX = 100,
```

```
OffsetY = 100,
};
NodeCollection.Add(node);
}
}
```

The following another code example shows how to enable the userhandle functionality for the selected item.

### **C#**

```
//Enable userhandle constraint for the selected item.
selectedItems.Constraints = SelectorConstraints.All |
SelectorConstraints.UserHandle;
```

For more information about selector constraints, refer to the [SelectorConstraints](#).

By default, the following constraints are enabled for the selected items,

- \* ResizeAll
- \* UserHandle
- \* Rotate
- \* ToolTip

**Note :** The port constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#).

### **Snap constraints**

Snap constraints control the visibility of gridlines and enable or disable snapping. Snap constraints allow to set the following behaviors.

- Show only horizontal or vertical gridlines.
- Show both horizontal and vertical gridlines.
- Snap to either horizontal or vertical gridlines.
- Snap to both horizontal and vertical gridlines.

The following list of snapping constraints are used to Enables or Disables certain features of snapping.

Constraints	Description
None	Disable to snapping the nodes/connectors in diagram
ShowHorizontalLines	Displays only the horizontal gridlines in diagram
ShowVerticalLines	Displays only the Vertical gridlines in diagram
ShowLines	Display both Horizontal and Vertical gridlines
SnapToHorizontalLines	Enables the object to snap only with horizontal gridlines
SnapToVerticalLines	Enables the object to snap only with Vertical gridlines
SnapToLines	Enables the object to snap with both horizontal and Vertical gridlines

|SnapToObject| Enables the object to snap with the other objects in the diagram|

|All| Shows gridlines and enables snapping|

The following code shows how to show only horizontal gridlines.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
@* Initialize the snapsettings with constraints *@
<DiagramSnapSettings
Constraints="SnapConstraints.ShowHorizontalLines"></DiagramSnapSettings>
</SfDiagram>
@code{
ObservableCollection<DiagramNode> NodeCollection;
protected override void OnInitialized()
{
//Initialize the NodeCollection.
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
Id = "node1",
Height = 100,
Width = 100,
OffsetX = 100,
OffsetY = 100,
};
NodeCollection.Add(node);
}
}
```

For more information about snap constraints, refer to the [SnapConstraints](#).

By default, the following constraints are enabled for the snap functionality in the diagram,

\* ShowLines

\* SnapToLines

**Note :** The snap constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#).

#### Boundary constraints

Boundary constraints defines a boundary for the diagram inside that the interaction should be done. Boundary constraints allow to set the following behaviors.

- Infinite boundary
- Diagram sized boundary
- Page sized boundary

The following list of constraints are used to Enables or Disables certain features of boundary interactions of the diagram.

| Constraints | Description |

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

|Infinity|Allow the interactions to take place at the infinite height and width|

|Diagram|Allow the interactions to take place around the diagram height and width|

|Page|Allow the interactions to take place around the page height and width|

The following code shows how to limit the interaction done inside a diagram within a page.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
  @* Initialize the pagesettings with boundary constraints *@
  <DiagramPageSettings Width="600" Height="500"
    BoundaryConstraints="BoundaryConstraints.Page"></DiagramPageSettings>
</SfDiagram>
@code{
  ObservableCollection<DiagramNode> NodeCollection;
  protected override void OnInitialized()
  {
    //Initialize the NodeCollection.
    NodeCollection = new ObservableCollection<DiagramNode>();
    DiagramNode node = new DiagramNode()
    {
      Id = "node1",
      Height = 100,
      Width = 100,
      OffsetX = 100,
      OffsetY = 100,
    };
    NodeCollection.Add(node);
  }
}
```

For more information about selector constraints, refer to the [BoundaryConstraints](#).

By default, the following boundary constraints are enabled for the snap functionality in the diagram,

\* Diagram

**Note :** The boundary constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#).

#### Inherit behaviors

Some of the behaviors can be defined through both the specific object (node or connector) and diagram. When the behaviors are contradictorily defined through both, the actual behavior is set through inherit options.

The following code example shows how to inherit the line bridging behavior from the diagram model.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize the diagram with constraints *@
```

```

<SfDiagram Height="600px" Connectors="@ConnectorCollection"
Constraints="@DiagramConstraints">
</SfDiagram>
@code{
//Sets the diagram constraints
public DiagramConstraints DiagramConstraints = DiagramConstraints.Default |
DiagramConstraints.Bridging;
ObservableCollection<DiagramConnector> ConnectorCollection;
protected override void OnInitialized()
{
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
Id = "connector1",
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
//sets the ConnectorConstraints...
Constraints = ConnectorConstraints.Default |
ConnectorConstraints.InheritBridging
};
DiagramConnector connector1 = new DiagramConnector()
{
Id = "connector2",
SourcePoint = new ConnectorSourcePoint() { X = 200, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 100, Y = 200 },
};
ConnectorCollection.Add(connector);
ConnectorCollection.Add(connector1);
}
}

```

### Bitwise operations

Bitwise operations are used to manipulate the flagged enumerations `enum`. In this section, Bitwise operations are shown by using the node constraints. The same is applicable when working with node constraints, connector constraints, or port constraints.

### Add operation

You can add or enable multiple values at a time by using the Bitwise `|` (OR) operator.

The following code shows to add line routing constraints into the default diagram constraints to enable line routing functionality into the diagram.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Constraints="@DiagramConstraint">
</SfDiagram>
@code{
//To adding line routing constraint with default constraints.
DiagramConstraints DiagramConstraint = DiagramConstraints.Default |
DiagramConstraints.LineRouting;
}

```

### Remove Operation

You can remove or disable values by using the Bitwise '&~' (XOR) operator.

The following code shows to remove zoom and pan constraints from the default constraints to disable zoom and panning functionality in the diagram.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Constraints="@DiagramConstraint">
</SfDiagram>
@code{
//To removing zoom and panning constraints from the default constraints
//It has disabled zoom and panning functionality for the diagram.
DiagramConstraints DiagramConstraint = DiagramConstraints.Default &~
(DiagramConstraints.Zoom | DiagramConstraints.Pan);
}
```

### Check operation

You can check any value by using the Bitwise '&' (AND) operator.

#### CSHARP

```
if ((node.constraints & (NodeConstraints.Rotate)) ==
(NodeConstraints.Rotate));
```

In the previous example, check whether the rotate constraints are enabled in a node. When node constraints have rotated constraints, the expression returns a rotate constraint.

## Interaction in Blazor Diagram Component

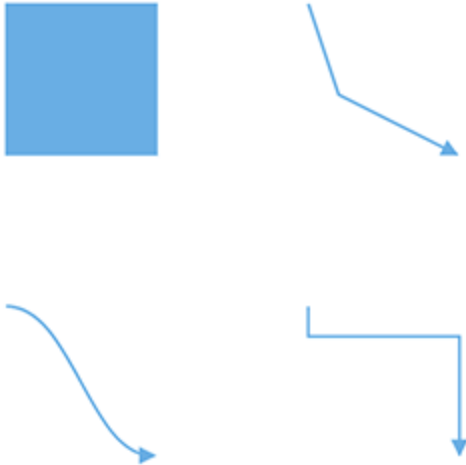
### Selection

Selector provides a visual representation of selected elements. It behaves like a container and allows to update the size, position, and rotation angle of the selected elements through interaction and by using program. Single or multiple elements can be selected at a time.

#### Single selection

An element can be selected by clicking that element. During single click, all previously selected items are cleared. The following image shows how the selected elements are visually represented.





- While selecting the diagram elements, the following events can be used to do your customization.
- When selecting/unselecting the diagram elements, the [SelectionChanged](#) event gets triggered.

#### Selecting a group

When a child element of any group is clicked, its contained group is selected instead of the child element. With consecutive clicks on the selected element, selection is changed from top to bottom in the hierarchy of parent group to its children.

#### Multiple selection

Multiple elements can be selected with the following ways:

- Ctrl+Click

During single click, any existing item in the selection list be cleared, and only the item clicked recently is there in the selection list. To avoid cleaning the old selected item, Ctrl key must be on hold when clicking.



- Selection rectangle/rubber band selection

Clicking and dragging the diagram area allows to create a rectangular region. The elements that are covered under the rectangular region are selected at the end.



#### Select/Unselect elements using program

The server-side methods [Select](#) and [ClearSelection](#) help to select or clear the selection of the elements at runtime.

Get the current selected items from the [Nodes](#) and [Connectors](#) collection of the [SelectedItems](#) property of the diagram model.

#### Select entire elements in diagram programmatically

The server-side method [SelectAll](#) used to select all the elements such as nodes/connectors in the diagram. Refer to the following link which shows how to use [SelectAll](#) method on the diagram.

#### Drag

- An object can be dragged by clicking and dragging it. When multiple elements are selected, dragging any one of the selected elements move every selected element.
- When you drag the elements in the diagram, the [OnPositionChange](#) event gets triggered and to do customization in this event.



## Resize

- Selector is surrounded by eight thumbs. When dragging these thumbs, selected items can be resized.
- When one corner of the selector is dragged, opposite corner is in a static position.
- When a node is resized, the [OnSizeChange](#) event gets triggered.



---

While dragging and resizing, the objects are snapped towards the nearest objects to make better alignments. For better alignments, refer to [Snapping](#).

---

## Rotate

- A rotate handler is placed above the selector. Clicking and dragging the handler in a circular direction lead to rotate the node.
- The node is rotated with reference to the static pivot point.
- Pivot thumb (thumb at the middle of the node) appears while rotating the node to represent the static point.

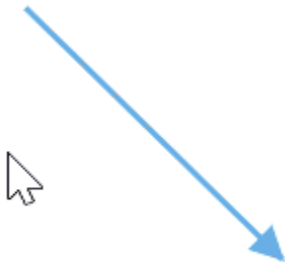


### Connection editing

- Each segment of a selected connector is editable with some specific handles/thumbs.

#### End point handles

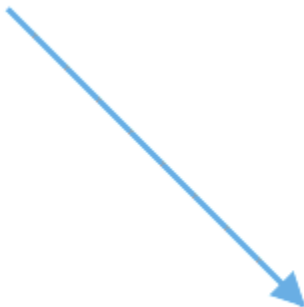
Source and target points of the selected connectors are represented with two handles. Clicking and dragging those handles help you to adjust the source and target points.



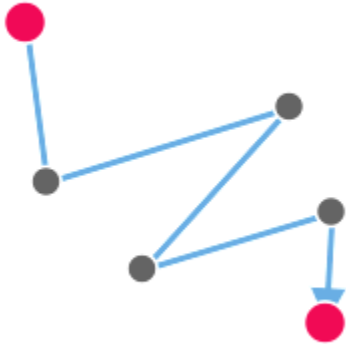
- If you drag the connector end points, then the following events can be used to do your customization.
- When you connect connector with ports/node or disconnect from it, the [ConnectionChange](#) event gets triggered.

#### Straight segment editing

- End point of each straight segment is represented by a thumb that enables to edit the segment.
- Any number of new segments can be inserted into a straight line by clicking, when Shift and Ctrl keys are pressed (Ctrl+Shift+Click).

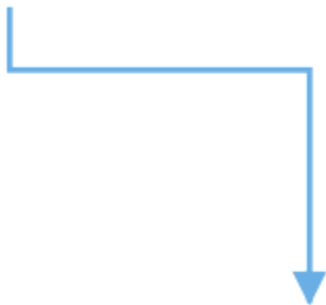


- Straight segments can be removed by clicking the segment end point, when Ctrl and Shift keys are pressed (Ctrl+Shift+Click).



### Orthogonal thumbs

- Orthogonal thumbs allow you to adjust the length of adjacent segments by clicking and dragging it.
- When necessary, some segments are added or removed automatically, when dragging the segment. This is to maintain proper routing of orthogonality between segments.



### Drag and drop nodes over other elements

Diagram provides support to drop a node/connector over another node/connector. The [OnDrop](#) event is raised to notify that an element is dropped over another one and it is disabled, by default. It can be enabled with the `constraints` property.

### User handles

- User handles are used to add some frequently used commands around the selector. To create user handles, define and add them to the [UserHandles](#) collection of the [SelectedItems](#) property.

- The name property of user handle is used to define the name of the user handle and its further used to find the user handle at runtime and do any customization.

### Alignment

User handles can be aligned relative to the node boundaries. It has [Margin](#), [Offset](#), [Side](#), [HorizontalAlignment](#), and [VerticalAlignment](#) settings. It is quite tricky when all four alignments are used together but gives more control over alignment.

### Offset

The [Offset](#) property of [UserHandles](#) is used to align the user handle based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

### Side

The [Side](#) property of [UserHandles](#) is used to align the user handle by using the [Top](#), [Bottom](#), [Left](#), and [Right](#) options.

### Horizontal and vertical alignments

The [HorizontalAlignment](#) property of [UserHandles](#) is used to set how the user handle is horizontally aligned at the position based on the [Offset](#). The [VerticalAlignment](#) property is used to set how user handle is vertically aligned at the position.

### Margin

Margin is an absolute value used to add some blank space in any one of its four sides. The [UserHandles](#) can be displaced with the [Margin](#) property.

### Notification for the mouse button clicked

The diagram component notifies the mouse button clicked. For example, whenever the right mouse button is clicked, the clicked button is notified as right. The mouse click is notified with,

Notification	Description
Left	When the left mouse button is clicked, left is notified
Middle	When the mouse wheel is clicked, middle is notified
Right	When the right mouse button is clicked, right is notified

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using Syncfusion.Blazor.Navigations
@using System.Collections.ObjectModel
<h2>Diagram</h2>
<br />
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
<link
href="https://ej2.syncfusion.com/javascript/demos/src/diagram/styles/diagram
-common.css" rel="stylesheet">
<SfDiagram Height="600px" Nodes="@NodeCollection" @onclick='@Click'>
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
```

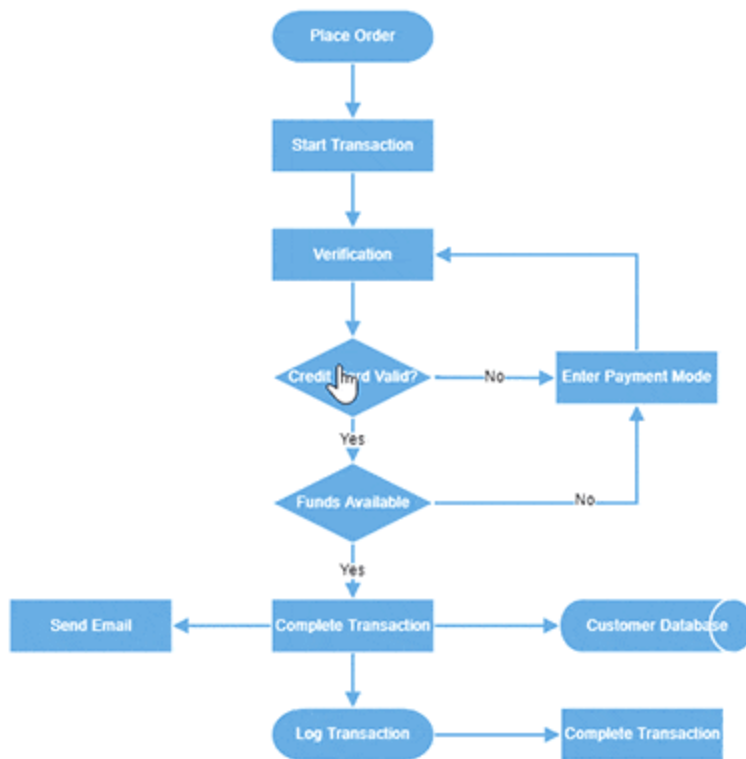
```
string greeting;
protected override void OnInitialized()
{
    // A node is created and stored in nodes array.
    DiagramNode node1 = new DiagramNode()
    {
        // Position of the node
        OffsetX = 250,
        OffsetY = 250,
        // Size of the node
        Width = 100,
        Height = 100,
        // Add node
        Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeDashArray = "5,5",
        StrokeColor = "red", StrokeWidth = 2 },
    };
    NodeCollection.Add(node1);
}
private async void Click()
{
    //Sets the export option for diagram
    IBlazorClickEventArgs options = new IBlazorClickEventArgs()
    {
        Console.WriteLine("Button", options.MouseButtons);
    };
}
}
```

### Appearance

The appearance of the user handle can be customized by using the [Size](#), [BorderColor](#), [BackgroundColor](#), [Visible](#), [PathData](#), and [PathColor](#) properties of the [UserHandles](#).

### Zoom pan

- When a large diagram is loaded, only certain portion of the diagram is visible. The remaining portions are clipped. Clipped portions can be explored by scrolling the scrollbars or panning the diagram.
- Diagram can be zoomed in or out by using Ctrl + mouse wheel.



### Keyboard

Diagram provides support to interact with the elements with key gestures. By default, some in-built commands are bound with a relevant set of key combinations.

The following table illustrates those commands with the associated key values.

Shortcut Key	Command	Description
-----	-----	-----
Ctrl + A	<b>SelectAll</b>	Select all nodes/connectors in the diagram.
Ctrl + C	Copy	Copy the diagram selected elements.
Ctrl + V	Paste	Pastes the copied elements.
Ctrl + X	Cut	Cuts the selected elements.
Ctrl + Z	Undo	Reverses the last editing action performed on the diagram.
Ctrl + Y	Redo	Restores the last editing action when no other actions have occurred since the last undo on the diagram.
Delete	Delete	Deletes the selected elements.
Ctrl/Shift + Click on object		Multiple selection (Selector binds all selected nodes/connectors).
Up Arrow	Nudge("up")	<b>NudgeUp</b> : Moves the selected elements towards up by one pixel.
Down Arrow	Nudge("down")	<b>NudgeDown</b> : Moves the selected elements towards down by one pixel.



- | Left Arrow | Nudge("left") | **NudgeLeft**: Moves the selected elements towards left by one pixel.
- | Right Arrow | Nudge("right") | **NudgeRight**: Moves the selected elements towards right by one pixel.
- | Ctrl + MouseWheel | Zoom | Zoom (Zoom in/Zoom out the diagram).
- | F2 | **StartLabelEditing** | Starts to edit the label of selected element.
- | Esc | **EndLabelEditing** | Sets the label mode as view and stops editing.

See Also

- [How to control the diagram history](#)
- [How to create overview control to the diagram](#)

## Tools in Blazor Diagram Component

### Drawing tools

Drawing tool allows you to draw any kind of node/connector during runtime by clicking and dragging on the diagram page.

### Shapes

To draw a shape, set the JSON of that shape to the drawType property of the diagram and activate the drawing tool by using the [Tool](#) property. The following code example illustrates how to draw a rectangle at runtime.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input Type="button" value="addNode" @onclick="AddNode" />
<SfDiagram @ref="diagram" Nodes="@NodeCollection" Height="600px">
</SfDiagram>
@code
{
    //Reference to diagram
    SfDiagram diagram;
    //Defines diagram's nodes collection
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        DiagramNode node = new DiagramNode()
        {
            Id = "group",
            OffsetX = 200,
            OffsetY = 200,
            Width = 100,
            Height = 100,
            Annotations = new ObservableCollection<DiagramNodeAnnotation>()
            {
                new DiagramNodeAnnotation()
                {
                    Content = "Node1",
                    Style = new AnnotationStyle()
                    {
                        Color = "white",
```

```

}
}
},
Style = new NodeShapeStyle() { Fill = "cornflowerblue", StrokeColor =
"white" }
};
NodeCollection.Add(node);
}
private void AddNode()
{
//To draw an object once, activate draw once
diagram.Tool = DiagramTools.DrawOnce;
//Initialize the drawing object to draw the shape
diagram.DrawingObject = new DiagramNode()
{
Shape = new DiagramShape()
{
Type = Shapes.Basic,
BasicShape = BasicShapes.Rectangle
},
};
}
}
}

```

### Connectors

To draw connectors, set the JSON of the connector to the drawType property. The drawing [Tool](#) can be activated by using the tool property. The following code example illustrates how to draw a straight line connector.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input Type="button" value="AddConnector" @onclick="AddConnector" />
<SfDiagram @ref="diagram" Nodes="@NodeCollection" Height="600px">
</SfDiagram>
@code
{
//Reference to diagram
SfDiagram diagram;
//Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
protected override void OnInitialized()
{
DiagramNode node = new DiagramNode()
{
Id = "group",
OffsetX = 200,
OffsetY = 200,
Width = 100,
Height = 100,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{

```

```

Content = "Node1",
Style = new AnnotationStyle()
{
    Color = "white",
}
},
Style = new NodeShapeStyle() { Fill = "cornflowerblue", StrokeColor =
"white" }
};
NodeCollection.Add(node);
}
private void AddConnector()
{
    //To draw an object once, activate draw once
    diagram.Tool = DiagramTools.DrawOnce;
    //Initialize the drawing object to draw the connectors
    diagram.DrawingObject = new DiagramConnector()
    {
        Id = "connector1",
        Type = Segments.Straight,
        Segments = new ObservableCollection<DiagramConnectorSegment>()
        {
            new DiagramConnectorSegment()
            {
                Type = Segments.Polyline,
            }
        },
    };
}
}
}

```

## Text

Diagram allows you to create a textNode, when you click on the diagram page. The following code illustrates how to draw a text.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input Type="button" value="TextNode" @onclick="TextNode" />
<SfDiagram @ref="diagram" Nodes="@NodeCollection" Height="600px">
</SfDiagram>
@code
{
    //Reference to diagram
    SfDiagram diagram;
    //Defines diagram's nodes collection
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>();
    protected override void OnInitialized()
    {
        DiagramNode node = new DiagramNode()
        {
            Id = "group",
            OffsetX = 200,

```

```

OffsetY = 200,
Width = 100,
Height = 100,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    new DiagramNodeAnnotation()
    {
        Content = "Node1",
        Style = new AnnotationStyle()
        {
            Color = "white",
        }
    },
    Style = new NodeShapeStyle() { Fill = "cornflowerblue", StrokeColor = "white" }
};
NodeCollection.Add(node);
}
private void TextNode()
{
    //To draw an object once, activate draw once
    diagram.Tool = DiagramTools.DrawOnce;
    //Initialize the drawing object to draw the text node
    diagram.DrawingObject = new DiagramNode()
    {
        Shape = new DiagramShape()
        {
            Type = Shapes.Text,
            TextContent = "Text",
        },
    };
}
}

```

Once you activate the TextTool, perform label editing of a node/connector.

### Polygon shape

Diagram allows to create the polygon shape by clicking and moving the mouse at runtime on the diagram page.

The following code illustrates how to draw a polygon shape.

### ASPX-CS

```

@using System.Collections.ObjectModel
@using Syncfusion.Blazor.Diagrams
<input Type="button" value="Polygon" @onclick="Polygon" />
<SfDiagram @ref="diagram" Nodes="@NodeCollection" Height="600px">
</SfDiagram>
@code
{
    //Reference to diagram
    SfDiagram diagram;
    //Defines diagram's nodes collection

```

```
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
protected override void OnInitialized()
{
    DiagramNode node = new DiagramNode()
    {
        Id = "group",
        OffsetX = 200,
        OffsetY = 200,
        Width = 100,
        Height = 100,
        Annotations = new ObservableCollection<DiagramNodeAnnotation>()
        {
            new DiagramNodeAnnotation()
            {
                Content = "Node1",
                Style = new AnnotationStyle()
                {
                    Color = "white",
                }
            },
        },
        Style = new NodeShapeStyle() { Fill = "cornflowerblue", StrokeColor =
"white" }
    };
    NodeCollection.Add(node);
}
private void Polygon()
{
    //To draw an object once, activate draw once
    diagram.Tool = DiagramTools.DrawOnce;
    //Initialize the drawing object to draw the polygon shape
    diagram.DrawingObject = new DiagramNode()
    {
        Id = "polygon",
        Shape = new DiagramShape()
        {
            Type = Shapes.Basic,
            BasicShape = BasicShapes.Polygon,
        },
    };
}
```

### Tool selection

There are some functionalities that can be achieved by clicking and dragging on the diagram surface. They are as follows,

- Draw selection rectangle: MultipleSelect tool
- Pan the diagram: Zoom pan
- Draw nodes/connectors: DrawOnce/DrawOnce

As all the three behaviors are completely different, you can achieve only one behavior at a time based on the tool that you choose.

When more than one of those tools are applied, a tool is activated based on the precedence given in the following table.

Precedence	Tools	Description
1st	ContinuesDraw	Allows you to draw the nodes or connectors continuously. Once it is activated, you cannot perform any other interaction in the diagram.
2nd	DrawOnce	Allows you to draw a single node or connector. Once you complete the DrawOnce action, SingleSelect, and MultipleSelect tools are automatically enabled.
3rd	ZoomPan	Allows you to pan the diagram. When you enable both the SingleSelect and ZoomPan tools, you can perform the basic interaction as the cursor hovers node/connector. Panning is enabled when cursor hovers the diagram.
4th	MultipleSelect	Allows you to select multiple nodes and connectors. When you enable both the MultipleSelect and ZoomPan tools, cursor hovers the diagram. When panning is enabled, you cannot select multiple nodes.
5th	SingleSelect	Allows you to select individual nodes or connectors.
6th	None	Disables all tools.

Set the desired [Tool](#) to the tool property of the diagram model.

The following code illustrates how to enable single tools,

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Connectors="@ConnectorCollection" Height="600px" Tool="@tool">
</SfDiagram>
@code
{
    //Enable the single tool
    public DiagramTools tool = DiagramTools.DrawOnce;
    //Defines diagram's connectors collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
}
```

The following code illustrates how to enable multiple tools,

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Connectors="@ConnectorCollection" @ref="diagram" Height="600px"
Tool="@tool">
</SfDiagram>
@code
{
    //Reference to diagram
    SfDiagram diagram;
    //Enable the multiple tools
    public DiagramTools tool = DiagramTools.DrawOnce | DiagramTools.ZoomPan;
```

```
//Defines diagram's connectors collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
}
```

## Gridlines in Blazor Diagram Component

Gridlines are the pattern of lines drawn behind the diagram elements. It provides a visual guidance while dragging or arranging the objects on the diagram surface.

The model's [SnapSettings](#) property is used to customize the gridlines and control the snapping behavior in the diagram.

### Customize the gridlines visibility

The [SnapConstraints](#) enables you to show/hide the gridlines. The following code example illustrates how to show or hide gridlines.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px">
  @* Shows both horizontal and vertical gridlines *@
  <DiagramSnapSettings Constraints="@snapConstraints">
    <HorizontalGridlines LineColor="blue" LineDashArray="2,2">
    </HorizontalGridlines>
    <VerticalGridlines LineColor="blue" LineDashArray="2,2">
    </VerticalGridlines>
  </DiagramSnapSettings>
</SfDiagram>
@code{
  SnapConstraints snapConstraints = SnapConstraints.ShowLines;
}
```

To show only horizontal/vertical gridlines or to hide gridlines, refer to [Constraints](#).

### Appearance

The appearance of the gridlines can be customized by using a set of predefined properties.

- The [HorizontalGridLines](#) and the [VerticalGridLines](#) properties allow to customize the appearance of the horizontal and vertical gridlines respectively.
- The horizontal gridlines [LineColor](#) and [LineDashArray](#) properties are used to customizes the line color and line style of the horizontal gridlines.
- The vertical gridlines [LineColor](#) and [LineDashArray](#) properties are used to customizes the line color and line style of the vertical gridlines.

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

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px">
  @* Shows both horizontal and vertical gridlines *@
  <DiagramSnapSettings Constraints="SnapConstraints.ShowLines">
  @* Customizes the line color and line style to the gridlines *@
  <HorizontalGridlines LineColor="blue" LineDashArray="2,2">
```

```
</HorizontalGridlines>
<VerticalGridlines LineColor="blue" LineDashArray="2,2">
</VerticalGridlines>
</DiagramSnapSettings>
</SfDiagram>
```

### Line intervals

Thickness and the space between gridlines can be customized by using horizontal gridline's [LinesInterval](#) and vertical gridline's [LinesInterval](#) properties. In the line intervals collections, values at the odd places are referred as the thickness of lines and values at the even places are referred as the space between gridlines.

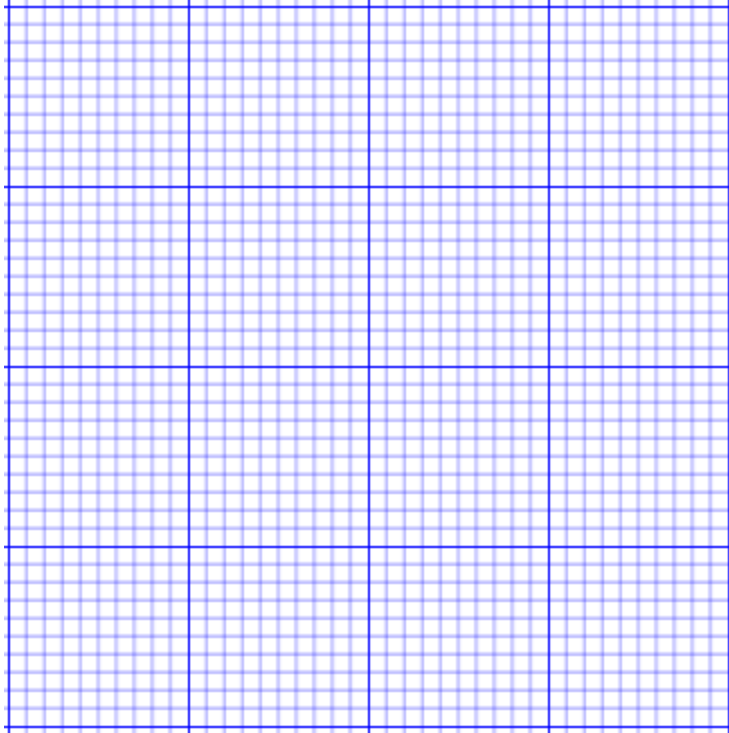
The following code example illustrates how to customize the thickness of lines and the line intervals.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px">
  @* Customize the appearance of the grid lines *@
  <DiagramSnapSettings Constraints="SnapConstraints.ShowLines">
    <HorizontalGridlines LineColor="blue" LineDashArray="2,2"
      LineIntervals="@LineIntervals">
    </HorizontalGridlines>
    <VerticalGridlines LineColor="blue" LineDashArray="2,2"
      LineIntervals="@LineIntervals">
    </VerticalGridlines>
  </DiagramSnapSettings>
</SfDiagram>

@code{
  //Sets the line intervals for the gridlines
  public double[] LineIntervals { get; set; } = new double[]
  {
    1.25, 14, 0.25, 15, 0.25, 15, 0.25, 15, 0.25, 15
  };
}
```





## Snapping

### *Snap to lines*

This feature allows the diagram objects to snap to the nearest intersection of gridlines while being dragged or resized. This feature enables easier alignment during layout or design.

Snapping to gridlines can be enabled/disabled with the [SnapConstraints](#). The following code example illustrates how to enable/disable the snapping to gridlines.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
  <DiagramSnapSettings Constraints="@snapConstraints"></DiagramSnapSettings>
</SfDiagram>
@code{
  //Sets the snap constraints
  public SnapConstraints snapConstraints = SnapConstraints.ShowLines |
  SnapConstraints.SnapToLines;
  public ObservableCollection<DiagramNode> NodeCollection { get; set; }
  protected override void OnInitialized()
  {
    NodeCollection = new ObservableCollection<DiagramNode>();
    DiagramNode diagramNode = new DiagramNode();
    diagramNode.OffsetX = 100;
    diagramNode.OffsetY = 100;
    diagramNode.Width = 100;
    diagramNode.Height = 100;
    diagramNode.Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor =
    "#6BA5D7" };
    diagramNode.Id = "node1";
    NodeCollection.Add(diagramNode);
  }
}
```

```
}
}
```

### Customization of snap intervals

By default, the objects are snapped towards the nearest gridline. The gridline or position towards where the diagram object snaps can be customized with the horizontal gridline's [SnapInterval](#) and the vertical gridline's [SnapInterval](#) properties.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
  <DiagramSnapSettings Constraints="@snapConstraints"></DiagramSnapSettings>
</SfDiagram>
@code{
  //Sets the snap constraints
  public SnapConstraints snapConstraints = SnapConstraints.ShowLines |
  SnapConstraints.SnapToLines;
  public ObservableCollection<DiagramNode> NodeCollection { get; set; }
  protected override void OnInitialized()
  {
    NodeCollection = new ObservableCollection<DiagramNode>();
    DiagramNode diagramNode = new DiagramNode();
    diagramNode.OffsetX = 100;
    diagramNode.OffsetY = 100;
    diagramNode.Width = 100;
    diagramNode.Height = 100;
    diagramNode.Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor =
    "#6BA5D7" };
    diagramNode.Id = "node1";
    NodeCollection.Add(diagramNode);
  }
}
```

### Snap to objects

The snap to object provides visual cues to assist with aligning and spacing diagram elements. A node can be snapped with its neighboring objects based on certain alignments. Such alignments are visually represented as smart guides.

The [SnapObjectDistance](#) property allows you to define minimum distance between the selected object and the nearest object.

The [SnapAngle](#) property allows you to define the snap angle by which the object needs to be rotated.

The [SnapConstraints](#) property allows you to enable or disable the certain features of the snapping, refer to [SnapConstraints](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
  <DiagramSnapSettings Constraints="@snapConstraints" SnapAngle="10"
  SnapObjectDistance="10">
</DiagramSnapSettings>
```

```

</SfDiagram>
@code{
//Sets the Snap to objects constraints...
public SnapConstraints snapConstraints = SnapConstraints.ShowLines |
SnapConstraints.SnapToObject;
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode diagramNode = new DiagramNode();
diagramNode.OffsetX = 100;
diagramNode.OffsetY = 100;
diagramNode.Width = 100;
diagramNode.Height = 100;
diagramNode.Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor =
"#6BA5D7" };
diagramNode.Id = "node1";
NodeCollection.Add(diagramNode);
diagramNode = new DiagramNode();
diagramNode.OffsetX = 300;
diagramNode.OffsetY = 100;
diagramNode.Width = 100;
diagramNode.Height = 100;
diagramNode.Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor =
"#6BA5D7" };
diagramNode.Id = "node2";
NodeCollection.Add(diagramNode);
}
}

```

## Page Settings in Blazor Diagram Component

Page settings enable to customize the appearance, width, and height of the diagram page.

### Page size and appearance

- The size and appearance of the diagram pages can be customized with the page settings property.
- The [Width](#) and [Height](#) properties of page settings define the size of the page and based on the size, the [Orientation](#) will be set for the page. In addition to that, the appearance of the page can be customized with [Source](#) and set of appearance specific properties.
- The [Color](#) property is used to customize the background color and border color of the page.
- The [Margin](#) property is used to define the page margin.
- To explore those properties, refer to [Page Settings](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" id="diagram">
@*Initialize the page settings with page orientation and break lines*@
<DiagramPageSettings Height="300" Width="300" Orientation="@orientation"
ShowPageBreaks="true">
@*Set the page background image*@

```

```

<DiagramBackground
Source="https://www.w3schools.com/images/w3schools_green.jpg" />
@*Set the page margin*@
<PageSettingsMargin Left="10" Top="10" Bottom="10" />
</DiagramPageSettings>
</SfDiagram>
@code
{
//Sets the page orientation as landscape.
public PageOrientation orientation = PageOrientation.Landscape;
//Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
protected override void OnInitialized()
{
DiagramNode node = new DiagramNode()
{
Id = "group",
OffsetX = 200,
OffsetY = 200,
Width = 100,
Height = 100,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Node1",
Style = new AnnotationStyle()
{
Color = "white",
}
},
},
Style = new NodeShapeStyle() { Fill = "cornflowerblue", StrokeColor =
"white" }
};
NodeCollection.Add(node);
}
}

```

### Set background image

Stretch and align the background image anywhere over the diagram area.

- The [Source](#) property of [Background](#) allows you to set the path of the image.
- The [Scale](#) and the [Align](#) properties help to stretch/align the background images.

The following code illustrates how to stretch and align the background image.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px" id="diagram">
@*Initialize the page settings with page orientation and break lines*@
<DiagramPageSettings Height="300" Width="300" Orientation="@orientation"
ShowPageBreaks="true">

```

```

<DiagramBackground
Source="https://www.w3schools.com/images/w3schools_green.jpg" />
<PageSettingsMargin Left="10" Top="10" Bottom="10" />
</DiagramPageSettings>
</SfDiagram>
@code
{
//Sets the page orientation as landscape.
public PageOrientation orientation = PageOrientation.Landscape;
}

```

### Multiple page and page breaks

When multiple page is enabled, the size of the page dynamically increases or decreases in multiples of page width and height and completely fits diagram within the page boundaries. Page breaks is used as a visual guide to see how pages are split into multiple pages.

The [MultiplePage](#) and [ShowPageBreak](#) properties of page settings allow you to enable/disable multiple pages and page breaks respectively. The following code illustrates how to enable multiple page and page break lines.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Nodes="@NodeCollection" Height="600px" id="diagram">
<DiagramSnapSettings>
<HorizontalGridlines LineColor="gray" />
<VerticalGridlines LineColor="gray" />
</DiagramSnapSettings>
@*Initialize the page settings with page orientation and multiple pages*
<DiagramPageSettings Height="300" Width="300" MultiplePage="true"
Orientation="@orientation" ShowPageBreaks="true">
<DiagramBackground Color="lightblue" />
<PageSettingsMargin Left="10" Top="10" Bottom="10" />
</DiagramPageSettings>
</SfDiagram>
@code
{
//Sets the page orientation as landscape.
public PageOrientation orientation = PageOrientation.Landscape;
//Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
protected override void OnInitialized()
{
DiagramNode node = new DiagramNode()
{
Id = "group",
OffsetX = 200,
OffsetY = 200,
Width = 100,
Height = 100,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{

```

```

Content = "Node1",
Style = new AnnotationStyle()
{
    Color = "white",
}
},
Style = new NodeShapeStyle() { Fill = "cornflowerblue", StrokeColor =
"white" }
};
NodeCollection.Add(node);
}
}

```

### Boundary constraints

The diagram provides support to restrict/customize the interactive region, out of which the elements cannot be dragged, resized, or rotated. The [BoundaryConstraints](#) property of page settings allows you to customize the interactive region. To explore the boundary constraints, refer to [Boundary Constraints](#).

The following code example illustrates how to define boundary constraints with respect to the page.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Nodes="@NodeCollection" Height="600px">
    <DiagramSnapSettings>
        <HorizontalGridlines LineColor="gray" />
        <VerticalGridlines LineColor="gray" />
    </DiagramSnapSettings>
    @*Initialize the page settings with page orientation and break lines*
    <DiagramPageSettings Height="300" Width="300" MultiplePage="true"
    Orientation="@orientation" BoundaryConstraints="@boundaryConstraints"
    ShowPageBreaks="true">
        <DiagramBackground Color="lightblue" />
        <PageSettingsMargin Left="10" Top="10" Bottom="10" />
    </DiagramPageSettings>
</SfDiagram>
@code
{
    //Reference to diagram
    SfDiagram diagram;
    public PageOrientation orientation = PageOrientation.Landscape;
    public BoundaryConstraints boundaryConstraints = BoundaryConstraints.Page;
    //Defines diagram's nodes collection
    public ObservableCollection<DiagramNode>
    NodeCollection = new ObservableCollection<DiagramNode>();
    protected override void OnInitialized()
    {
        DiagramNode node = new DiagramNode()
        {
            Id = "group",
            OffsetX = 200,
            OffsetY = 200,
            Width = 100,
            Height = 100,

```

```
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    new DiagramNodeAnnotation()
    {
        Content = "Node1",
        Style = new AnnotationStyle()
        {
            Color = "white",
        }
    },
    new NodeShapeStyle() { Fill = "cornflowerblue", StrokeColor = "white" }
};
NodeCollection.Add(node);
```

## Scroll Settings in Blazor Diagram Component

The diagram can be scrolled by using the vertical and horizontal ScrollBars. In addition to the ScrollBars, mouse wheel can be used to scroll the diagram. Diagram's [ScrollSettings](#) enable you to read the current scroll status, view port size, current zoom and zoom factor. It also allows you to scroll the diagram programmatically.

## Get current scroll status

Scroll settings allow you to read the scroll status, [ViewportWidth](#), [ViewportHeight](#), and [CurrentZoom](#) with a set of properties. To explore those properties, see [Scroll Settings](#).

## Define scroll status

Diagram allows you to pan the diagram before loading, so that any desired region of a large diagram is made to view. You can programmatically pan the diagram with the [HorizontalOffset](#) and [VerticalOffset](#) properties of scroll settings. The following code illustrates how to set pan the diagram programmatically.

In the following example, the vertical scroll bar is scrolled down by 50px and horizontal scroll bar is scrolled to right by 100px.

## ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px">
  @* Sets the ScrollSettings for the diagram *@
  <DiagramScrollSettings HorizontalOffset="100" VerticalOffset="50">
  </DiagramScrollSettings>
</SfDiagram>
```

## Update scroll status

You can programmatically change the scroll offsets at runtime by using the server-side method `update`. The following code illustrates how to change the scroll offsets and zoom factor at runtime.

## ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px">
@* Sets the ScrollSettings for the diagram *@
```

```

<DiagramScrollSettings HorizontalOffset="@horizontalOffset"
VerticalOffset="@verticalOffset">
</DiagramScrollSettings>
</SfDiagram>
@code{
public double horizontalOffset { get; set; } = 100;
public double verticalOffset { get; set; } = 100;
public void updateScrollValues()
{
//Update scroll settings
verticalOffset = 400;
horizontalOffset = 200;
}
}

```

### Autoscroll

Autoscroll feature automatically scrolls the diagram, whenever the node or connector is moved beyond the boundary of the diagram. So that, it is always visible during dragging, resizing, and multiple selection operations. Autoscroll is automatically triggered when any one of the following is done towards the edges of the diagram.

- Node dragging, resizing
- Connection editing
- Rubber band selection
- Label dragging

The Autoscroll behavior in your diagram can be enabled/disabled by using the [CanAutoscroll](#) property of the diagram.

### Autoscroll border

The Autoscroll border is used to specify the maximum distance between the object and diagram edge to trigger Autoscroll. The default value is set as 15 for all sides (left, right, top, and bottom) and it can be changed by using the [AutoScrollMargin](#) property of scroll settings. The following code example illustrates how to set Autoscroll margin.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px">
@* Sets the ScrollSettings for the diagram *@
<DiagramScrollSettings HorizontalOffset="100" VerticalOffset="50">
@* Sets the Auto Scroll border for the diagram *@
<AutoScrollMargin Left="15" Bottom="15" Right="15"
Top="15"></AutoScrollMargin>
</DiagramScrollSettings>
</SfDiagram>

```

### Scroll limit

The scroll limit allows you to define the scrollable region of the diagram. It includes the following options:

- Allows to scroll in all directions without any restriction.



- Allows to scroll within the diagram content.
- Allows to scroll within the specified scrollable area.
- The [ScrollLimit](#) property of scroll settings helps to limit the scrolling.

The ScrollSettings [ScrollableArea](#) allow to extend the scrollable region that is based on the scroll limit.

The following code example illustrates how to specify the scroll limit.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px">
  /* Sets the ScrollLimit of scroll settings */
  <DiagramScrollSettings HorizontalOffset="100" VerticalOffset="50"
    ScrollLimit="ScrollLimit.Infinity">
  </DiagramScrollSettings>
</SfDiagram>
```

#### Scroll Padding

The scroll padding allows you to extend the scrollable region that is based on the scroll limit. The following code example illustrates how to set scroll padding to diagram region.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px">
  /* Sets the ScrollSettings for the diagram */
  <DiagramScrollSettings HorizontalOffset="100" VerticalOffset="50">
  /* Sets the Padding for the diagram Scroll */
  <AutoScrollPadding Right="50" Bottom="50"></AutoScrollPadding>
  </DiagramScrollSettings>
</SfDiagram>
```

#### Scrollable Area

Scrolling beyond any particular rectangular area can be restricted by using the [ScrollableArea](#) property of scroll settings. To restrict scrolling beyond any custom region, set the [ScrollLimit](#) as "limited". The following code example illustrates how to customize scrollable area.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px">
  /* Sets the scrollable Area */
  <DiagramScrollSettings HorizontalOffset="100" VerticalOffset="50"
    ScrollLimit="ScrollLimit.Infinity" ScrollableArea="@scrollArea">
  </DiagramScrollSettings>
</SfDiagram>
@code{
  public class Area
  {
    public int X { get; set; }
    public int Y { get; set; }
    public int Width { get; set; }
    public int Height { get; set; }
  }
}
```

```
// Sets the values of scroll area
object scrollArea = new Area() { X = 0, Y = 0, Width = 500, Height = 500 };
}
```

### UpdateViewport

The [UpdateViewport](#) method is used to update the diagram page and view size at runtime.

## Data Binding in Blazor Diagram Component

- Diagram can be populated with the **Nodes** and **Connectors** based on the information provided from an external data source.
- Diagram exposes its specific data-related properties allowing you to specify the data source fields from where the node information has to be retrieved from.
- The [DataSource](#) property is used to define the data source either as a collection of objects or as an instance of [DataSource](#) that needs to be populated in the diagram.
- The **ID** property is used to define the unique field of each JSON data.
- The **ParentId** property is used to defines the parent field which builds the relationship between ID and parent field.
- The **Root** property is used to define the root node for the diagram populated from the data source.
- To explore those properties, see [DataSourceSettings](#).
- Diagram supports two types of data binding. They are:
  1. Local data
  2. Remote data

### Local data

Diagram can be populated based on the user defined JSON data (Local Data) by mapping the relevant data source fields.

To map the user defined JSON data with diagram, configure the fields of [DataSourceSettings](#). The following code example illustrates how to bind local data with the diagram.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Layout="@LayoutValue"
ConnectorDefaults="@ConnectorDefault" NodeDefaults="@NodeDefaults">
<DiagramDataSource Id="Name" ParentId="Category" DataSource="@DataSource"
DataMapSettings="@datamap">
<DiagramDataMapSettings>
<DiagramDataMapSetting Property="Annotations[0].Content"
Field="Name"></DiagramDataMapSetting>
</DiagramDataMapSettings>
</DiagramDataSource>
</SfDiagram>
@code
{
//Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
//Defines diagram's connector collection
```

```

public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
//Defines the node default values.
DiagramNode NodeDefaults = new DiagramNode()
{
    Width = 95,
    Height = 30,
    BackgroundColor = "#6BA5D7",
    Shape = new DiagramShape() { Type = Shapes.Basic, BasicShape =
    BasicShapes.Rectangle },
    Style = new NodeShapeStyle { Fill = "#ffeec7", StrokeColor = "#ffeec7",
    StrokeWidth = 1, },
    Annotations = new ObservableCollection<DiagramNodeAnnotation>()
    {
        new DiagramNodeAnnotation()
        {
            Id = "label1",
            Style = new AnnotationStyle()
            {
                Color = "black"
            },
        },
    };
//Defines the connector's default values.
DiagramConnector ConnectorDefault = new DiagramConnector
{
    Type = Segments.Orthogonal,
    Style = new ConnectorShapeStyle() { StrokeColor = "#4d4d4d", StrokeWidth = 2
    },
    TargetDecorator = new ConnectorTargetDecorator()
    {
        Shape = DecoratorShapes.None,
    };
//Create the layout info
TreeInfo LayoutInfo = new TreeInfo()
{
    //Enable the sub-tree.
    CanEnableSubTree = true,
    //Specify the sub-tree orientation
    Orientation = SubTreeOrientation.Horizontal,
};
//Create the data map settings.
List<DiagramDataMapSetting> datamap { get; set; } = new
List<DiagramDataMapSetting>()
{
    new DiagramDataMapSetting() { Property = "Shape.TextContent", Field = "Name"
    };
};
DiagramLayout LayoutValue = new DiagramLayout() { };
protected override void OnInitialized()
{
    LayoutValue = new DiagramLayout()
    {
        Type = LayoutType.HierarchicalTree,
        VerticalSpacing = 30,
    };
}

```

```

HorizontalSpacing = 30,
EnableAnimation = true,
LayoutInfo = this.LayoutInfo
};
}
//Create the hierarchical details with needed properties.
public class HierarchicalDetails
{
    public string Name { get; set; }
    public string FillColor { get; set; }
    public string Category { get; set; }
}
//Create the data source with node name and fill color values.
public List<object> DataSource = new List<object>()
{
    new HierarchicalDetails() { Name ="Diagram",
    Category="",FillColor="#916DAF"},
    new HierarchicalDetails() { Name ="Layout", Category="Diagram",FillColor=""},
    new HierarchicalDetails() { Name ="Tree Layout",
    Category="Layout",FillColor=""},
    new HierarchicalDetails() { Name ="Organizational Chart",
    Category="Layout",FillColor=""},
    new HierarchicalDetails() { Name ="Hierarchical Tree", Category="Tree
    Layout",FillColor=""},
    new HierarchicalDetails() { Name ="Radial Tree", Category="Tree
    Layout",FillColor=""},
    new HierarchicalDetails() { Name ="Mind Map", Category="Hierarchical
    Tree",FillColor=""},
    new HierarchicalDetails() { Name ="Family Tree", Category="Hierarchical
    Tree",FillColor=""},
    new HierarchicalDetails() { Name ="Management", Category="Organizational
    Chart",FillColor=""},
    new HierarchicalDetails() { Name ="Human Resources",
    Category="Management",FillColor=""},
    new HierarchicalDetails() { Name ="University",
    Category="Management",FillColor=""},
    new HierarchicalDetails() { Name ="Business",
    Category="#Management",FillColor=""}
};
}

```

## See Also

- [How to arrange the diagram nodes and connectors using varies layout](#)

## Layout

### Automatic Layout in Blazor Diagram Component

Diagram provides support to auto-arrange the nodes in the diagram area that is referred as [Layout](#). It includes the following layout modes:

#### Layout modes

- Hierarchical layout

- Organization chart
- Mind Map layout
- Radial tree
- Symmetric layout
- Complex hierarchical tree layout

*See also*

- [How to create a node](#)
- [How to create a connector](#)
- [How to generate the organization chart](#)
- [How to generate the hierarchical layout](#)

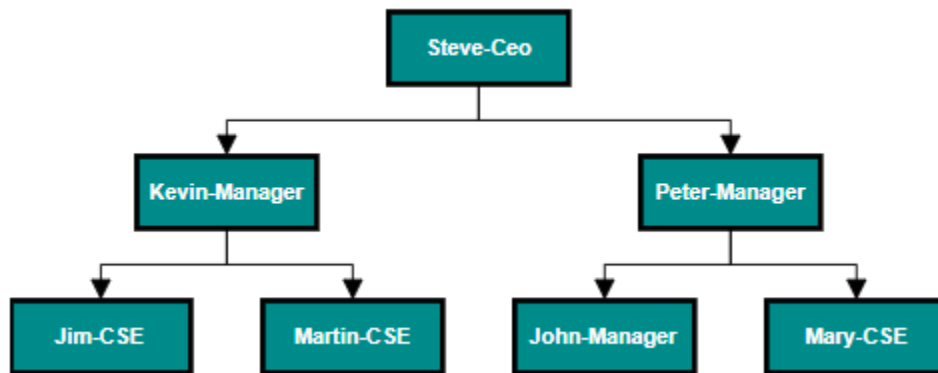
### Hierarchical Layout in Blazor Diagram Component

The hierarchical tree layout arranges nodes in a tree-like structure, where the nodes in the hierarchical layout may have multiple parents. There is no need to specify the layout root. To arrange the nodes in a hierarchical structure, specify the layout [Type](#) as `HierarchicalTree`. The following example shows how to arrange the nodes in a hierarchical structure.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram ID="diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" NodeDefaults="@NodeDefaults"
ConnectorDefaults="@ConnectorDefaults" Layout="@LayoutValue">
</SfDiagram>
@code {
    ObservableCollection<DiagramNode> NodeCollection;
    ObservableCollection<DiagramConnector> ConnectorCollection;
    //Uses layout to auto-arrange nodes on the diagram page
    DiagramLayout LayoutValue = new DiagramLayout()
    {
        //Sets layout type as HierarchicalTree...
        Type = LayoutType.HierarchicalTree,
        VerticalSpacing = 40,
        HorizontalSpacing = 40,
    };
    //Sets the default properties for nodes
    DiagramNode NodeDefaults = new DiagramNode()
    {
        Height = 40,
        Width = 100,
        //Initializing the default node's shape style
        Style = new NodeShapeStyle() { Fill = "darkcyan", StrokeWidth = 3,
        StrokeColor = "Black" },
        Annotations = new ObservableCollection<DiagramNodeAnnotation>()
        {
            new DiagramNodeAnnotation() { Style = new AnnotationStyle() { Color =
            "white", Bold = true }, }
        }
    };
    //Sets the default properties for the connectors
    DiagramConnector ConnectorDefaults = new DiagramConnector()
```

```
{
    Type = Syncfusion.Blazor.Diagrams.Segments.Orthogonal,
};
protected override void OnInitialized()
{
    //Initializing node and connectors
    NodeCollection = new ObservableCollection<DiagramNode>()
    {
        new DiagramNode() {Id="node1",Annotations = new
        ObservableCollection<DiagramNodeAnnotation>() {new
        DiagramNodeAnnotation() {Content="Steve-Ceo"}}},
        new DiagramNode() {Id="node2",Annotations = new
        ObservableCollection<DiagramNodeAnnotation>() {new
        DiagramNodeAnnotation() {Content="Kevin-Manager"}}},
        new DiagramNode() {Id="node3",Annotations = new
        ObservableCollection<DiagramNodeAnnotation>() {new
        DiagramNodeAnnotation() {Content="Peter-Manager"}}},
        new DiagramNode() {Id="node4",Annotations = new
        ObservableCollection<DiagramNodeAnnotation>() {new
        DiagramNodeAnnotation() {Content="Jim-CSE"}}},
        new DiagramNode() {Id="node5",Annotations = new
        ObservableCollection<DiagramNodeAnnotation>() {new
        DiagramNodeAnnotation() {Content="Martin-CSE"}}},
        new DiagramNode() {Id="node6",Annotations = new
        ObservableCollection<DiagramNodeAnnotation>() {new
        DiagramNodeAnnotation() {Content="John-Manager"}}},
        new DiagramNode() {Id="node7",Annotations = new
        ObservableCollection<DiagramNodeAnnotation>() {new
        DiagramNodeAnnotation() {Content="Mary-CSE"}}},
    };
    ConnectorCollection = new ObservableCollection<DiagramConnector>()
    {
        new DiagramConnector() {Id="connector1",SourceID="node1",TargetID="node2"},
        new DiagramConnector() {Id="connector2",SourceID="node1",TargetID="node3"},
        new DiagramConnector() {Id="connector3",SourceID="node2",TargetID="node4"},
        new DiagramConnector() {Id="connector4",SourceID="node2",TargetID="node5"},
        new DiagramConnector() {Id="connector5",SourceID="node3",TargetID="node6"},
        new DiagramConnector() {Id="connector6",SourceID="node3",TargetID="node7"},
    };
    }
}
```



### Customizing the properties

#### Orientation

You can change the orientation at runtime. The following code shows how to change the layout.

#### CSHARP

```
// Change the orientation at runtime
public void UpdateOrientation()
{
    Diagram.Layout.Orientation = LayoutOrientation.BottomToTop;
}
```

#### Spacing

You can change the horizontal and vertical spacing for the diagram layout.

#### CSHARP

```
// Update the spacing
public void UpdateSpacing()
{
    Diagram.BeginUpdate();
    Diagram.Layout.HorizontalSpacing += 10;
    Diagram.Layout.VerticalSpacing += 10;
    Diagram.EndUpdate();
}
```

#### Margin

You can change the margin values for the diagram layout.

#### CSHARP

```
// Update the margin values
public void UpdateMargin()
{
    Diagram.BeginUpdate();
    Diagram.Layout.Margin.Left += 10;
    Diagram.Layout.Margin.Top += 10;
    Diagram.EndUpdate();
}
```

### Expand and collapse the layout

Diagram allows to expand or collapse the subtrees of a layout. The node's `isExpanded` property allows you to expand or collapse its children. The following code example shows how to expand or collapse the children of a node.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram ID="diagram" Height="600px" Layout="@LayoutValue"
ConnectorDefaults="@ConnectorDefault" NodeDefaults="@NodeDefaults"
SelectedItems="@selectedItems">
<DiagramDataSource Id="Name" ParentId="Category" DataSource="@DataSource">
<DiagramDataMapSettings>
<DiagramDataMapSetting Property="Shape.TextContent"
Field="Name"></DiagramDataMapSetting>
<DiagramDataMapSetting Property="Style.StrokeColor"
Field="FillColor"></DiagramDataMapSetting>
<DiagramDataMapSetting Property="Style.Fill"
Field="FillColor"></DiagramDataMapSetting>
</DiagramDataMapSettings>
</DiagramDataSource>
<DiagramPageSettings>
<DiagramFitOptions CanFit="@CanFit" Mode="@Mode"></DiagramFitOptions>
</DiagramPageSettings>
</SfDiagram>
@code{
DiagramSelectedItems selectedItems = new DiagramSelectedItems()
{
Constraints = SelectorConstraints.All & ~SelectorConstraints.ResizeAll &
~SelectorConstraints.Rotate
};
bool CanFit = true;
FitModes Mode = FitModes.Width;
TreeInfo LayoutInfo = new TreeInfo()
{
CanEnableSubTree = true,
Orientation = SubTreeOrientation.Horizontal
};
DiagramLayout LayoutValue = new DiagramLayout() { };
DiagramConnector ConnectorDefault = new DiagramConnector()
{
TargetDecorator = new ConnectorTargetDecorator() { Shape =
DecoratorShapes.None },
Type = Segments.Orthogonal,
Style = new ConnectorShapeStyle() { StrokeColor = "#6d6d6d" },
Constraints = 0,
CornerRadius = 5
};
DiagramNode NodeDefaults = new DiagramNode
{
Style = new NodeShapeStyle() { Fill = "#659be5", StrokeColor = "none", Color =
"white", StrokeWidth = 2, },
BackgroundColor = "#659be5",
Shape = new DiagramShape() { Type = Syncfusion.Blazor.Diagrams.Shapes.Text,
Margin = new BasicShapeMargin() { Left = 10, Right = 10, Bottom = 10, Top =
10 } },
}
```

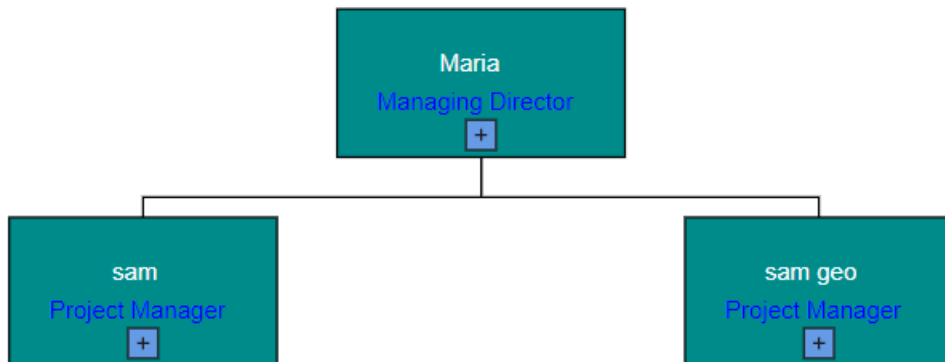


```

ExpandIcon = new NodeExpandIcon()
{
    Height = 10,
    Width = 10,
    Shape = IconShapes.Minus,
    Fill = "lightgray",
    Offset = new IconOffset() { X = 0.5, Y = 1 },
    VerticalAlignment = VerticalAlignment.Auto,
    Margin = new ExpandIconMargin() { Bottom = 0, Left = 0, Right = 0, Top = 0 },
},
CollapseIcon = new NodeCollapseIcon()
{
    Offset = new IconOffset() { X = 0.5, Y = 1 },
    VerticalAlignment = VerticalAlignment.Auto,
    Height = 10,
    Width = 10,
    Shape = IconShapes.Plus,
    Fill = "lightgray",
    Padding = new IconPadding() { Top = 5 }
};
};
public class HierarchicalDetails
{
    public string Name { get; set; }
    public string FillColor { get; set; }
    public string Category { get; set; }
}
public List<object> DataSource = new List<object>()
{
    new HierarchicalDetails() { Name = "Diagram",
    Category="", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Layout",
    Category="Diagram", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Tree layout",
    Category="Layout", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Organizational chart",
    Category="Layout", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Hierarchical tree", Category="Tree
    layout", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Radial tree", Category="Tree
    layout", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Mind map", Category="Hierarchical
    tree", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Family tree", Category="Hierarchical
    tree", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Management", Category="Organizational
    chart", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Human resources",
    Category="Management", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "University",
    Category="Management", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Business",
    Category="#Management", FillColor="#659be5"}
};
protected override void OnInitialized()
{
    LayoutValue = new DiagramLayout()

```

```
{
    Type = LayoutType.HierarchicalTree,
    VerticalSpacing = 30,
    HorizontalSpacing = 30,
    EnableAnimation = true,
    LayoutInfo = this.LayoutInfo
};
}
```



You can use the **EnableAnimation** property to enable or disable animation option when a node is expanded or collapsed.

#### Complex hierarchical tree

Complex hierarchical tree layout is the extended version of the hierarchical tree layout. The child had been two or more parents. To create a complex hierarchical tree, the **Type** of layout should be set as **ComplexHierarchicalTree**.

The following code example shows how to create a complex hierarchical tree.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram @ref="@diagram" id="diagram" Height="600px"
    tool="@DiagramTools.ZoomPan" NodeDefaults="@NodeDefaults"
    ConnectorDefaults="@ConnectorDefault" Layout="@LayoutValue">
    <DiagramDataSource Id="Name" ParentId="ReportingPerson"
    DataMapSettings="@datamap" DataSource="@dataSource"></DiagramDataSource>
</SfDiagram>
@code {
    SfDiagram diagram;
    List<DiagramDataMapSetting> datamap = new List<DiagramDataMapSetting>() {
        new DiagramDataMapSetting() { Property = "Style.fill", Field = "fillColor"
    },
        new DiagramDataMapSetting() { Property = "Style.strokeColor", Field =
        "border" }
    };
    DiagramLayout LayoutValue = new DiagramLayout()
    {
        Type = LayoutType.ComplexHierarchicalTree,
        HorizontalSpacing = 40,
        VerticalSpacing = 40,
```

```

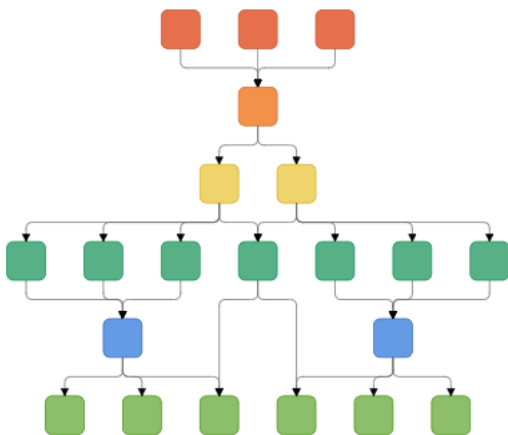
Orientation = LayoutOrientation.TopToBottom,
Margin = new LayoutMargin() { Left = 20, Top = 20 },
};
DiagramNode NodeDefaults = new DiagramNode
{
    Height = 40,
    Width = 40,
    Shape = new DiagramShape() { BasicShape = BasicShapes.Rectangle, Type =
    Syncfusion.Blazor.Diagrams.Shapes.Basic, CornerRadius = 7 },
};
DiagramConnector ConnectorDefault = new DiagramConnector
{
    Type = Syncfusion.Blazor.Diagrams.Segments.Orthogonal,
    CornerRadius = 7,
    TargetDecorator = new ConnectorTargetDecorator() { Width = 7, Height = 7 },
    Style = new ConnectorShapeStyle() { StrokeColor = "#6d6d6d" }
};
public class DataModel
{
    public string Name;
    public string fillColor;
    public string border;
    public string[] ReportingPerson;
}
public object dataSource = new List<object>()
{
    new DataModel { Name = "node11", fillColor = "#e7704c", border = "#c15433"
    },
    new DataModel { Name = "node12", ReportingPerson = new string[]{ "node114"
    }, fillColor = "#efd46e", border = "#d6b123" },
    new DataModel { Name = "node13", ReportingPerson = new string[] { "node12"
    }, fillColor = "#58b087", border = "#16955e" },
    new DataModel { Name = "node14", ReportingPerson = new string[] { "node12"
    }, fillColor = "#58b087", border = "#16955e" },
    new DataModel { Name = "node15", ReportingPerson = new string[] { "node12"
    }, fillColor = "#58b087", border = "#16955e" },
    new DataModel { Name = "node16", ReportingPerson = new string[] {},
    fillColor = "#14ad85" },
    new DataModel { Name = "node17", ReportingPerson = new string[] { "node13",
    "node14", "node15" }, fillColor = "#659be5", border = "#3a6eb5" },
    new DataModel { Name = "node18", ReportingPerson = new string[] {},
    fillColor = "#14ad85" },
    new DataModel { Name = "node19", ReportingPerson = new string[] { "node16",
    "node17", "node18" }, fillColor = "#8dbe6c", border = "#489911" },
    new DataModel { Name = "node110", ReportingPerson = new string[] {
    "node16", "node17", "node18" }, fillColor = "#8dbe6c", border = "#489911" },
    new DataModel { Name = "node111", ReportingPerson = new string[] {
    "node16", "node17", "node18", "node116" }, fillColor = "#8dbe6c", border =
    "#489911" },
    new DataModel { Name = "node21", fillColor = "#e7704c", border = "#c15433"
    },
    new DataModel { Name = "node22", ReportingPerson = new string[] { "node114"
    }, fillColor = "#efd46e", border = "#d6b123" },
    new DataModel { Name = "node23", ReportingPerson = new string[] { "node22"
    }, fillColor = "#58b087", border = "#16955e" },
    new DataModel { Name = "node24", ReportingPerson = new string[] { "node22"
    }, fillColor = "#58b087", border = "#16955e" },

```

```

new DataModel { Name = "node25", ReportingPerson = new string[] { "node22"
}, fillColor = "#58b087", border = "#16955e" },
new DataModel { Name = "node26", ReportingPerson = new string[] {},
fillColor = "#14ad85" },
new DataModel { Name = "node27", ReportingPerson = new string[] { "node23",
"node24", "node25" }, fillColor = "#659be5", border = "#3a6eb5" },
new DataModel { Name = "node28", ReportingPerson = new string[] {},
fillColor = "#14ad85" },
new DataModel { Name = "node29", ReportingPerson = new string[] { "node26",
"node27", "node28", "node116" }, fillColor = "#8dbe6c", border = "#489911"
},
new DataModel { Name = "node210", ReportingPerson = new string[] {
"node26", "node27", "node28" }, fillColor = "#8dbe6c", border = "#489911" },
new DataModel { Name = "node211", ReportingPerson = new string[] {
"node26", "node27", "node28" }, fillColor = "#8dbe6c", border = "#489911" },
new DataModel { Name = "node31", fillColor = "#e7704c", border = "#c15433"
},
new DataModel { Name = "node114", ReportingPerson = new string[] {
"node11", "node21", "node31" }, fillColor = "#f3904a", border = "#d3722e" },
new DataModel { Name = "node116", ReportingPerson = new string[] {
"node12", "node22" }, fillColor = "#58b087", border = "#16955e" }
};
}

```



See also

- [How to create a node](#)
- [How to create a connector](#)

### Organizational Chart in Blazor Diagram Component

An organizational chart is a diagram that displays the structure of an organization and relationships. To create an organizational chart, the [Type](#) of layout should be set as an `OrganizationalChart`.

The following code example illustrates how to create an organizational chart.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram ID="diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" NodeDefaults="@NodeDefaults"
ConnectorDefaults="@ConnectorDefaults" Layout="@LayoutValue">
</SfDiagram>
@code {
    ObservableCollection<DiagramNode> NodeCollection;
    ObservableCollection<DiagramConnector> ConnectorCollection;
    //Initializing layout
    DiagramLayout LayoutValue = new DiagramLayout()
    {
        //Sets layout type as OrganizationalChart...
        Type = LayoutType.OrganizationalChart,
        VerticalSpacing = 40,
        HorizontalSpacing = 40,
    };
    //Initializing node defaults
    DiagramNode NodeDefaults = new DiagramNode()
    {
        Height = 40,
        Width = 100,
        //sets the node shape style
        Style = new NodeShapeStyle() { Fill = "darkcyan", StrokeWidth = 3,
        StrokeColor = "Black" },
        Annotations = new ObservableCollection<DiagramNodeAnnotation>()
        {
            new DiagramNodeAnnotation()
            {
                Style= new AnnotationStyle()
                {
                    Color="white",
                    Bold=true
                }
            }
        }
    };
    //Initializing connector defaults
    DiagramConnector ConnectorDefaults = new DiagramConnector()
    {
        Type = Syncfusion.Blazor.Diagrams.Segments.Orthogonal,
    };
    protected override void OnInitialized()
    {
        //Initializing node and connectors
        NodeCollection = new ObservableCollection<DiagramNode>()
        {
            new DiagramNode() {Id="node1",Annotations = new
            ObservableCollection<DiagramNodeAnnotation>() {new
            DiagramNodeAnnotation() {Content="Project Management"}}},
            new DiagramNode() {Id="node2",Annotations = new
            ObservableCollection<DiagramNodeAnnotation>() {new
            DiagramNodeAnnotation() {Content="R&D Team"}}},
```

```

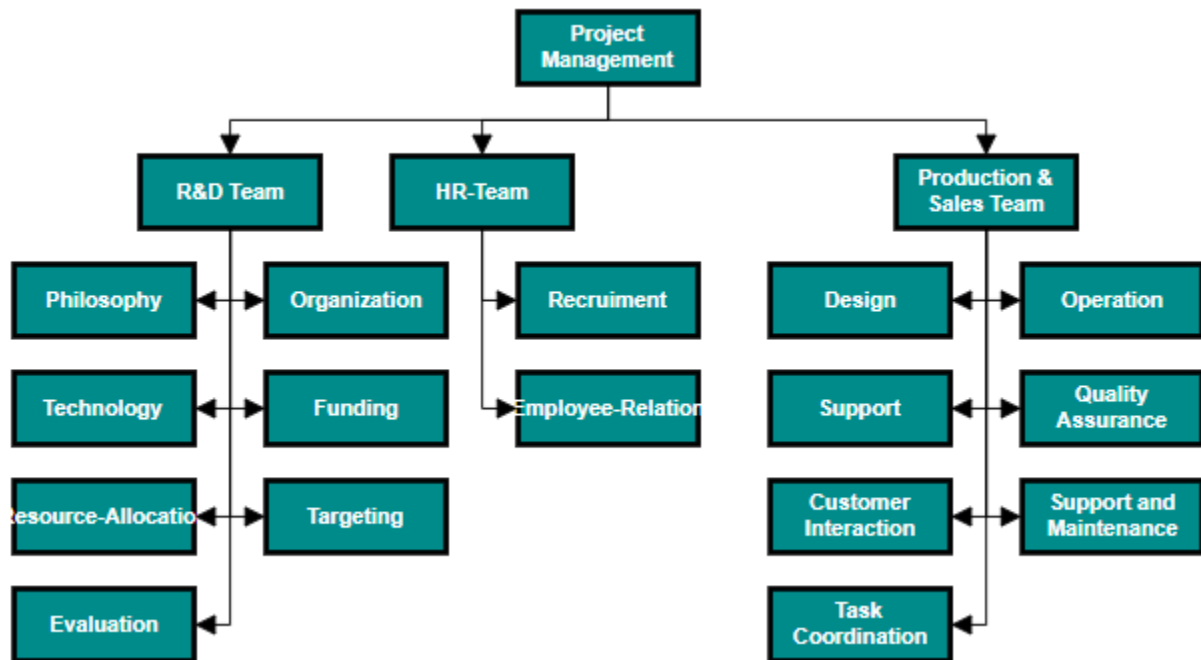
new DiagramNode() {Id="node3",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Philosophy"}}},
new DiagramNode() {Id="node4",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Organization"}}},
new DiagramNode() {Id="node5",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Technology"}}},
new DiagramNode() {Id="node6",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Funding"}}},
new DiagramNode() {Id="node7",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Resource-Allocation"}}},
new DiagramNode() {Id="node8",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Targeting"}}},
new DiagramNode() {Id="node9",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Evaluation"}}},
new DiagramNode() {Id="node10",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="HR-Team"}}},
new DiagramNode() {Id="node11",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Recruitment"}}},
new DiagramNode() {Id="node12",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Employee-Relation"}}},
new DiagramNode() {Id="node13",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Production & Sales Team"}}},
new DiagramNode() {Id="node14",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Design"}}},
new DiagramNode() {Id="node15",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Operation"}}},
new DiagramNode() {Id="node16",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Support"}}},
new DiagramNode() {Id="node17",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Quality Assurance"}}},
new DiagramNode() {Id="node18",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Customer Interaction"}}},
new DiagramNode() {Id="node19",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Support and Maintenance"}}},
new DiagramNode() {Id="node20",Annotations = new
ObservableCollection<DiagramNodeAnnotation>() {new
DiagramNodeAnnotation() {Content="Task Coordination"}}},
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{

```

```

new DiagramConnector() {Id="connector1",SourceID="node1",TargetID="node2"},
new DiagramConnector() {Id="connector2",SourceID="node1",TargetID="node10"},
new DiagramConnector() {Id="connector3",SourceID="node1",TargetID="node13"},
new DiagramConnector() {Id="connector4",SourceID="node2",TargetID="node3"},
new DiagramConnector() {Id="connector5",SourceID="node2",TargetID="node4"},
new DiagramConnector() {Id="connector6",SourceID="node2",TargetID="node5"},
new DiagramConnector() {Id="connector7",SourceID="node2",TargetID="node6"},
new DiagramConnector() {Id="connector8",SourceID="node2",TargetID="node7"},
new DiagramConnector() {Id="connector9",SourceID="node2",TargetID="node8"},
new DiagramConnector() {Id="connector10",SourceID="node2",TargetID="node9"},
new DiagramConnector() {Id="connector11",SourceID="node2",TargetID="node11"},
new
DiagramConnector() {Id="connector12",SourceID="node10",TargetID="node12"},
new
DiagramConnector() {Id="connector13",SourceID="node10",TargetID="node14"},
new
DiagramConnector() {Id="connector14",SourceID="node13",TargetID="node15"},
new
DiagramConnector() {Id="connector15",SourceID="node13",TargetID="node16"},
new
DiagramConnector() {Id="connector16",SourceID="node13",TargetID="node17"},
new
DiagramConnector() {Id="connector17",SourceID="node13",TargetID="node18"},
new
DiagramConnector() {Id="connector18",SourceID="node13",TargetID="node19"},
new
DiagramConnector() {Id="connector19",SourceID="node13",TargetID="node20"},
};
}
}

```



Organizational chart layout starts parsing from root and iterate through all its child elements. The [LayoutInfo](#) property provides necessary information of a node's children and the way to arrange (direction, orientation, offsets, etc.) them. The arrangements can be customized by overriding this function as explained.

Node **LayoutInfo** property to set chart orientations, chart types, and offset to be left between parent and child nodes. The [LayoutInfo](#) property is called to configure every subtree of the organizational chart. It takes the following arguments.

1. **Node:** Parent node to that options are to be customized.
2. **Options:** Object to set the customizable properties.

#### Assistant

Assistants are child item that have a different relationship with the parent node. They are laid out in a dedicated part of the tree. A node can be specified as an assistant of its parent by adding it to the assistants property of the argument [Assistants](#).

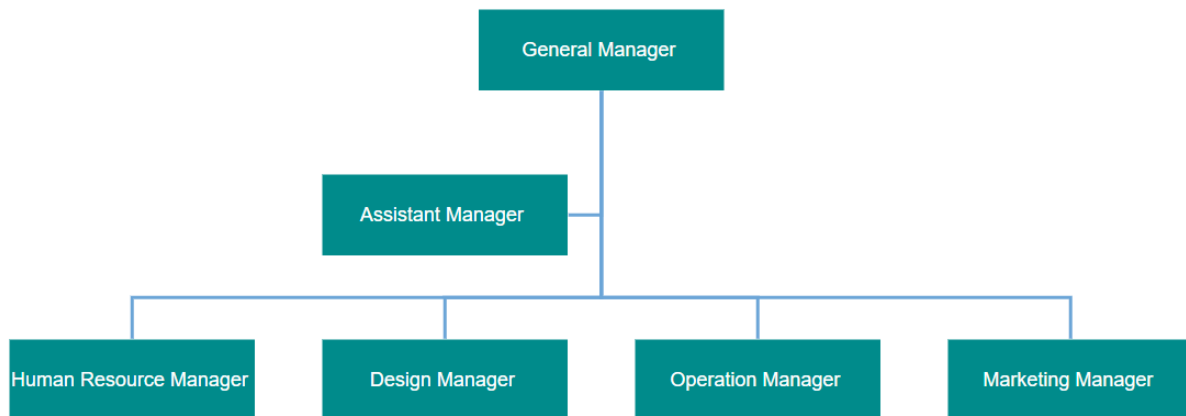
The following code example illustrates how to add assistants to layout.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" NodeDefaults="@NodeDefaults"
ConnectorDefaults="@ConnectorDefaults" Layout="@LayoutValue">
<DiagramDataSource Id="Id" ParentId="Team" DataSource="@DataSource"
DataMapSettings="@DataMap"></DiagramDataSource>
</SfDiagram>
@code {
//Initializing layout
DiagramLayout LayoutValue;
protected override void OnInitialized()
{
LayoutValue = new DiagramLayout()
{
//Sets layout type as OrganizationalChart...
Type = LayoutType.OrganizationalChart,
VerticalSpacing = 50,
HorizontalSpacing = 40,
//Initialize the layout info for the org chart layout
LayoutInfo = new TreeInfo
{
Orientation = SubTreeOrientation.Vertical,
Offset = -20,
CanEnableSubTree = true,
//provides an assistant details for the root node
GetAssistantDetails = new AssistantsDetails()
{
Root = "General Manager",
Assistants = new string[] { "Assistant Manager" }
}
}
};
}
//Initializing DataMap Setting
```



```
List<DiagramDataMapSetting> DataMap = new List<DiagramDataMapSetting>()
{
    new DiagramDataMapSetting() { Property = "Annotations[0].Content",
    Field = "Role" },
};
//Initializing node defaults
DiagramNode NodeDefaults = new DiagramNode()
{
    Width = 150,
    Height = 50,
    Annotations = new ObservableCollection<DiagramNodeAnnotation>() { new
    DiagramNodeAnnotation() { Id = "label1", Style = new AnnotationStyle() {
    Color = "white" } }, },
    Style = new NodeShapeStyle { Fill = "darkcyan", StrokeColor = "white", }
};
//Initializing connector defaults
DiagramConnector ConnectorDefaults = new DiagramConnector()
{
    Type = Syncfusion.Blazor.Diagrams.Segments.Orthogonal,
    Style = new ConnectorShapeStyle() { StrokeColor = "#6BA5D7", StrokeWidth = 2
    },
    TargetDecorator = new ConnectorTargetDecorator()
    {
        Shape = DecoratorShapes.None,
        Style = new DecoratorShapeStyle() { Fill = "#6BA5D7", StrokeColor =
        "#6BA5D7", },
    }
};
public class OrgChartDataModel
{
    public string Id { get; set; }
    public string Team { get; set; }
    public string Role { get; set; }
}
public object DataSource = new List<object>()
{
    new OrgChartDataModel() { Id= "1", Role= "General Manager" },
    new OrgChartDataModel() { Id= "2", Role= "Assistant Manager", Team= "1" },
    new OrgChartDataModel() { Id= "3", Role= "Human Resource Manager", Team= "1"
    },
    new OrgChartDataModel() { Id= "4", Role= "Design Manager", Team= "1" },
    new OrgChartDataModel() { Id= "5", Role= "Operation Manager", Team= "1" },
    new OrgChartDataModel() { Id= "6", Role= "Marketing Manager", Team= "1" }
};
}
```



### Customize layout

Orientation, spacings, and position of the layout can be customized with a set of properties.

To explore layout properties, refer to [Layout Properties](#).

### Layout bounds

Diagram provides support to align the layout within any custom rectangular area. For more information about bounds, refer to [Bounds](#).

### Layout alignment

The layout can be aligned anywhere over the layout bounds/viewport using the [HorizontalAlignment](#) and [VerticalAlignment](#) properties of the layout.

The following code illustrates how to align the layout at the top-left of the layout bounds.

### CSHARP

```

//Initialize the layout with VerticalAlignment alignment as bottom in page
DiagramLayout LayoutValue = new DiagramLayout()
{
    Type = LayoutType.OrganizationalChart,
    VerticalSpacing = 40,
    HorizontalSpacing = 40,
    VerticalAlignment = VerticalAlignment.Bottom
};
  
```

The following table illustrates the different chart orientations and chart types.

Orientation	Type	Description	Example
-----	-----	-----	-----

| Horizontal | Left | Arranges the child nodes horizontally at the left side of the parent. |



| | Right | Arranges the child nodes horizontally at the right side of the parent. |



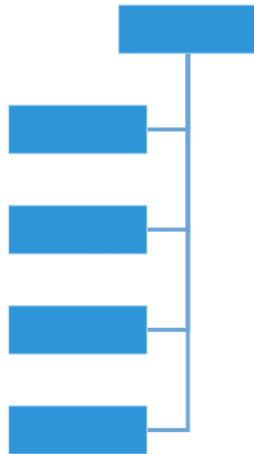
| |Center|Arranges the children like standard tree layout orientation. |



| |Balanced|Arranges the leaf level child nodes in multiple rows. |



|Vertical|Left|Arranges the children vertically at the left side of the parent. |



| |Right|Arranges the children vertically at the right side of the parent. |



| |Alternate|Arranges the children vertically at both left and right sides of the parent. |



|

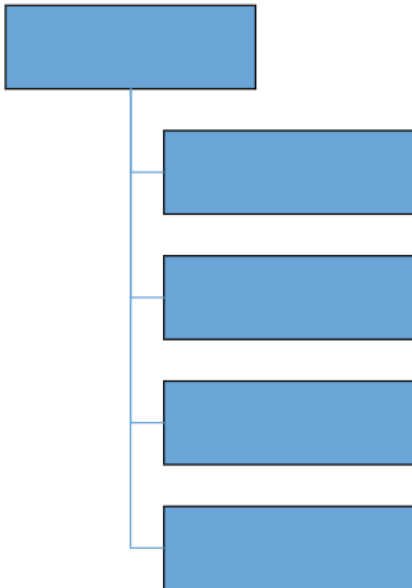
The following code example illustrates how to set the vertical right arrangement to the leaf level trees.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" NodeDefaults="@NodeDefaults"
ConnectorDefaults="@ConnectorDefaults" Layout="@LayoutValue">
<DiagramDataSource Id="Id" ParentId="Team" DataSource="@DataSource"
></DiagramDataSource>
</SfDiagram>
@code {
//Initializing layout
DiagramLayout LayoutValue;
protected override void OnInitialized()
{
LayoutValue = new DiagramLayout()
{
//Sets layout type as OrganizationalChart...
Type = LayoutType.OrganizationalChart,
VerticalSpacing = 50,
HorizontalSpacing = 40,
//Initialize the layout info for the org chart layout
LayoutInfo = new TreeInfo
{
Orientation = SubTreeOrientation.Vertical,
CanEnableSubTree = true,
}
};
}
//Initializing node defaults
DiagramNode NodeDefaults = new DiagramNode()
{
```

```
Width = 150,  
Height = 50,  
Annotations = new ObservableCollection<DiagramNodeAnnotation>()  
{  
    new DiagramNodeAnnotation()  
    {  
        Id = "label1",  
        Style = new AnnotationStyle()  
        {  
            Color = "white"  
        }  
    },  
},  
Style = new NodeShapeStyle  
{  
    Fill = "#6BA5D7",  
    StrokeColor = "black",  
},  
LayoutInfo = new DiagramNodeLayoutInfo()  
{  
    Orientation = Orientation.Vertical,  
    Type = SubTreeAlignments.Right  
}  
};  
  
//Initializing connector defaults  
DiagramConnector ConnectorDefaults = new DiagramConnector()  
{  
    Type = Syncfusion.Blazor.Diagrams.Segments.Orthogonal,  
    Style = new ConnectorShapeStyle()  
    {  
        StrokeColor = "#6BA5D7",  
        StrokeWidth = 1  
    },  
    TargetDecorator = new ConnectorTargetDecorator()  
    {  
        Shape = DecoratorShapes.None,  
        Style = new DecoratorShapeStyle()  
        {  
            Fill = "#6BA5D7",  
            StrokeColor = "#6BA5D7",  
        },  
    },  
};  
  
public class OrgChartDataModel  
{  
    public string Id { get; set; }  
    public string Team { get; set; }  
    public string Role { get; set; }  
}  
  
public object DataSource = new List<object>()  
{  
    new OrgChartDataModel() { Id= "1", Role= "General Manager" },  
    new OrgChartDataModel() { Id= "2", Role= "Human Resource Manager", Team= "1" },  
    new OrgChartDataModel() { Id= "3", Role= "Design Manager", Team= "1" },  
    new OrgChartDataModel() { Id= "4", Role= "Operation Manager", Team= "1" },  
    new OrgChartDataModel() { Id= "5", Role= "Marketing Manager", Team= "1" }
```

```
};
}
```



#### Layout spacing

Layout provides support to add space horizontally and vertically between the nodes. The [HorizontalSpacing](#) and [VerticalSpacing](#) properties of the layout allows you to set the space between the nodes horizontally and vertically.

#### Layout margin

Layout provides support to add some blank space between the layout bounds/viewport and the layout. The [Margin](#) property of the layout allows you to set the blank space.

#### CSHARP

```
//Initialize the organizational chart layout with Margin
DiagramLayout LayoutValue = new DiagramLayout()
{
    Type = LayoutType.OrganizationalChart,
    HorizontalSpacing = 40,
    VerticalSpacing = 40,
    Orientation = LayoutOrientation.TopToBottom,
    Margin = new LayoutMargin() { Left = 20, Top = 20 },
};
```

#### Layout orientation

Diagram provides support to customize the [Orientation](#) of layout. You can set the desired orientation using `Layout.Orientation`.

The following code illustrates how to arrange the nodes in a BottomToTop orientation.

#### CSHARP

```
//Initialize the layout with layout orientation as BottomToTop in page
```



```
DiagramLayout LayoutValue = new DiagramLayout()  
{  
    Type = LayoutType.OrganizationalChart,  
    HorizontalSpacing = 40,  
    VerticalSpacing = 40,  
    Orientation = LayoutOrientation.BottomToTop,  
};
```

### Fixed node

Layout provides support to arrange the nodes with reference to the position of a fixed node and set it to the [FixedNode](#) of the layout property. This is helpful when you try to expand/collapse a node. It might be expected that the position of the double-clicked node should not be changed.

### CSHARP

```
//Initialize the organizational chart layout with FixedNode  
DiagramLayout LayoutValue = new DiagramLayout()  
{  
    Type = LayoutType.OrganizationalChart,  
    FixedNode = "node1",  
    HorizontalSpacing = 40,  
    VerticalSpacing = 40,  
};
```

### Expand and collapse

Diagram allows to expand/collapse the subtrees of a layout. The node's `isExpanded` property allows you to expand/collapse its children. The following code example shows how to expand/collapse the children of a node.

### ASPX-CS

```
@code{  
    DiagramNode NodeDefaults = new DiagramNode()  
    {  
        //Initialize a expand icon for the node  
        ExpandIcon = new NodeExpandIcon()  
        {  
            Height = 15,  
            Width = 15,  
            Shape = IconShapes.Plus,  
            Fill = "lightgray",  
            Offset = new IconOffset() { X = .5, Y = .85 },  
        },  
        //Initialize a collapse icon for the node  
        CollapseIcon = new NodeCollapseIcon()  
        {  
            Height = 15,  
            Width = 15,  
            Shape = IconShapes.Minus,  
            Offset = new IconOffset() { X = .5, Y = .85 },  
        }  
    }  
}
```

In the previous example, while expanding/collapsing a node, it is set as fixed node in order to prevent it from repositioning.

#### Refresh layout

Diagram allows to refresh the layout at runtime. Use the below code example to refresh the layout.

#### CSHARP

```
//update the layout at runtime.
diagram.DoLayout();
//Here, diagram is instance of SfDiagram.
```

#### See also

- [How to create a node](#)
- [How to create a connector](#)

#### Mind Map layout in Blazor Diagram Component

A mind map is a diagram that displays the nodes as a spider diagram organizes information around a central concept. To create mind map, the [Type](#) of layout should be set as **MindMap**.

The following code example illustrates how to create an organizational chart.

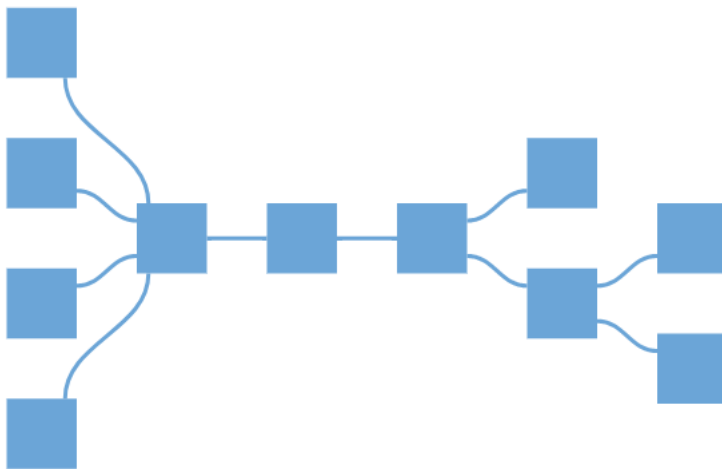
#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram ID="diagram" Height="600px" NodeDefaults="@NodeDefaults"
ConnectorDefaults="@ConnectorDefault" Layout="@LayoutValue">
<DiagramDataSource Id="Id" ParentId="ParentId" DataSource="@DataSource"
DataMapSettings="@DataMap"></DiagramDataSource>
</SfDiagram>
@code {
//Initializing MindMap layout
DiagramLayout LayoutValue = new DiagramLayout()
{
//Sets layout type as MindMap...
Type = LayoutType.MindMap,
Margin = new LayoutMargin() { Top = 20, Left = 20 }
};
List<DiagramDataMapSetting> DataMap = new List<DiagramDataMapSetting>()
{
new DiagramDataMapSetting() { Property = "Shape.Content",Field = "Label" },
};
DiagramNode NodeDefaults = new DiagramNode()
{
Width = 25,
Height = 25,
BackgroundColor = "#6BA5D7",
Annotations = new ObservableCollection<DiagramNodeAnnotation>() { new
DiagramNodeAnnotation() { Id = "label1", Style = new AnnotationStyle() {
Color = "black" } }, },
Style = new NodeShapeStyle { Fill = "#6BA5D7", StrokeWidth = 1, StrokeColor
= "white" },
Shape = new DiagramShape()
{
```

```

Type = Syncfusion.Blazor.Diagrams.Shapes.Text,
Content = "",
Margin = new BasicShapeMargin() { Left = 5, Right = 5, Bottom = 5, Top = 5 }
};
DiagramConnector ConnectorDefault = new DiagramConnector
{
    Type = Syncfusion.Blazor.Diagrams.Segments.Bezier,
    Style = new ConnectorShapeStyle() { StrokeColor = "#6BA5D7", StrokeWidth = 2
},
    TargetDecorator = new ConnectorTargetDecorator()
    {
        Shape = DecoratorShapes.None,
    }
};
public class MindMapDetails
{
    public string Id { get; set; }
    public string Label { get; set; }
    public string ParentId { get; set; }
    public string Branch { get; set; }
    public string Fill { get; set; }
}
public object DataSource = new List<object>()
{
    new MindMapDetails() { Id= "1",Label="Creativity", ParentId="", Branch =
    "Root"},
    new MindMapDetails() { Id= "2",   Label="Brainstorming", ParentId ="1",
    Branch = "Right" },
    new MindMapDetails() { Id= "3",   Label="Complementing", ParentId ="1",
    Branch = "Left" },
    new MindMapDetails() { Id= "4",   Label="Sessions", ParentId ="2", Branch =
    "subRight" },
    new MindMapDetails() { Id= "5",   Label="Complementing", ParentId ="2",
    Branch = "subRight" },
    new MindMapDetails() { Id= "6", Label= "Local", ParentId ="3", Branch =
    "subRight" },
    new MindMapDetails() { Id= "7", Label= "Remote", ParentId ="3", Branch =
    "subRight" },
    new MindMapDetails() { Id= "8", Label= "Individual", ParentId ="3", Branch =
    "subRight" },
    new MindMapDetails() { Id= "9", Label= "Teams", ParentId ="3", Branch =
    "subRight" },
    new MindMapDetails() { Id= "10", Label= "Ideas", ParentId ="5", Branch =
    "subRight" },
    new MindMapDetails() { Id= "11", Label= "Engagement", ParentId ="5", Branch
    = "subRight" },
};
}

```



See also

- [How to create a node](#)
- [How to create a connector](#)

### Radial Tree Layout in Blazor Diagram Component

The radial tree layout arranges nodes on a virtual concentric circle around a root node. Sub-trees formed by the branching of child nodes are located radially around the child nodes. This arrangement results in an ever-expanding concentric arrangement with radial proximity to the root node indicating the node level in the hierarchy. The layout [Root](#) property can be used to define the root node of the layout. When no root node is set, the algorithm automatically considers one of the diagram nodes as the root node.

To arrange nodes in a radial tree structure, set the [Type](#) of the layout as `RadialTree`. The following code illustrates how to arrange the nodes in a radial tree structure.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" NodeDefaults="@NodeDefaults"
ConnectorDefaults="@ConnectorDefault" Layout="@LayoutSetting">
<DiagramDataSource Id="Id" ParentId="ReportingPerson"
DataSource="@DataSource" DataMapSettings="@DataMap"></DiagramDataSource>
</SfDiagram>
@code{
//Uses layout to auto-arrange nodes on the diagram page
DiagramLayout LayoutSetting = new DiagramLayout()
{
//Sets layout type as RadialTree...
Type = LayoutType.RadialTree,
VerticalSpacing = 20,
HorizontalSpacing = 20
};
//sets Data map setting
```

```
List<DiagramDataMapSetting> DataMap { get; set; } = new
List<DiagramDataMapSetting>() {
//Data mapping for the label
new DiagramDataMapSetting() { Property = "Annotations[0].Content",
Field = "Name" },
};
//Sets the default properties for nodes and connectors
DiagramNode NodeDefaults = new DiagramNode()
{
Height = 100,
Width = 100,
//sets the default node's shape
Shape = new DiagramShape()
{
Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
BasicShape = BasicShapes.Ellipse
},
//sets the default node's annotation with style
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Id = "label1",
Style = new AnnotationStyle()
{
Color = "white"
}
},
},
//sets the default node's shape style
Style = new NodeShapeStyle
{
Fill = "#6BA5D7",
StrokeColor = "white"
}
};
DiagramConnector ConnectorDefault = new DiagramConnector
{
Type = Syncfusion.Blazor.Diagrams.Segments.Straight,
//sets the default connector's style
Style = new ConnectorShapeStyle()
{
StrokeColor = "#6BA5D7",
StrokeWidth = 2
},
//sets the connector's target decorator
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.None,
//sets decorator shape style
Style = new DecoratorShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "#6BA5D7",
},
}
};
```

```
//Initialize radial tree data
public class RadialTreeDetails
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Designation { get; set; }
    public string ReportingPerson { get; set; }
}
//Configures data source for diagram
public object DataSource = new List<object>()
{
    new RadialTreeDetails {Id = "parent", Name = "Maria Anders",Designation =
    "Managing Director", ReportingPerson = "" },
    new RadialTreeDetails {Id = "1", Name= "Ana Trujillo", Designation= "Project
    Manager", ReportingPerson= "parent" },
    new RadialTreeDetails {Id ="2",Name= "Lino Rodri", Designation="Project
    Manager",ReportingPerson= "parent" },
    new RadialTreeDetails {Id="3",Name = "Philip Cramer",Designation = "Project
    Manager",ReportingPerson= "parent" },
    new RadialTreeDetails {Id="4",Name= "Pedro Afonso",Designation= "Project
    Manager",ReportingPerson= "parent" },
    new RadialTreeDetails {Id ="5", Name="Anto Moreno",Designation= "Project
    Lead",ReportingPerson= "parent" },
    new RadialTreeDetails {Id="6",Name = "Elizabeth Roel",Designation= "Project
    Lead",ReportingPerson= "parent" },
    new RadialTreeDetails {Id="8",Name= "Eduardo Roel",Designation= "Project
    Lead",ReportingPerson= "parent" },
    new RadialTreeDetails {Id="9",Name= "Howard Snyd",Designation= "Project
    Lead",ReportingPerson= "parent" },
    new RadialTreeDetails {Id="10",Name= "Daniel Tonini",Designation= "Project
    Lead",ReportingPerson= "parent" },
    new RadialTreeDetails {Id="11",Name= "Nardo Batista",Designation= "Project
    Lead",ReportingPerson= "parent" },
};
}
```



See also

- [How to create a node](#)
- [How to create a connector](#)

### Symmetric layout in Blazor Diagram Component

The symmetric layout has been formed using nodes position by closer together or pushing them further apart. This is repeated iteratively until the system comes to an equilibrium state.

The layout's [SpringLength](#), defined as how long edges should be, ideally. This will be the resting length for the springs. Edge attraction and vertex repulsion forces to be defined by using layout's [SpringFactor](#), the more sibling nodes repel each other. The relative positions do not change any more from one iteration to the next. The number of iterations can be specified by using layout's [MaxIteration](#).

The following code illustrates how to arrange the nodes in a radial tree structure.

### ASPX-CS

```

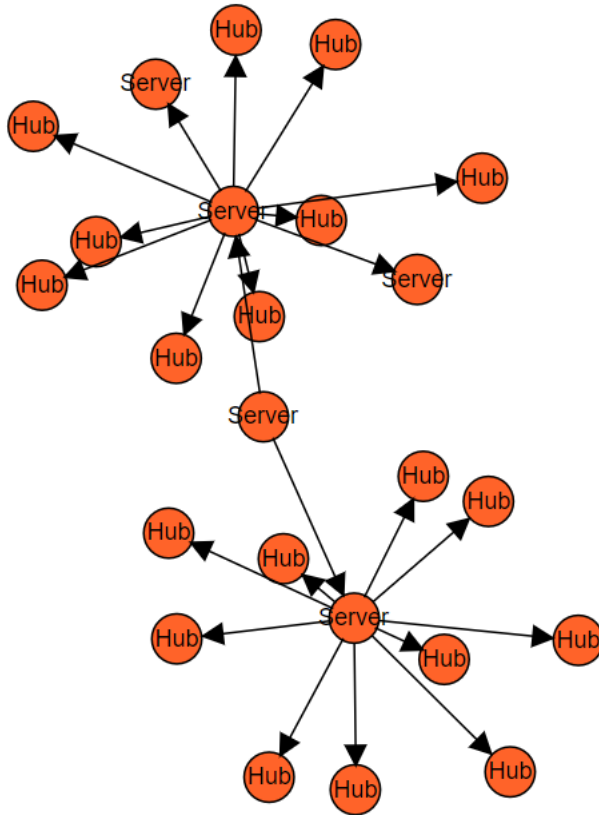
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram ID="diagram" Height="600px" NodeDefaults="@NodeDefaults"
ConnectorDefaults="@ConnectorDefaults" Layout="@LayoutValue">
<DiagramDataSource Id="Id" ParentId="Source" DataSource="@DataSource"
DataMapSettings="@DataMap"></DiagramDataSource>
</SfDiagram>
@code {
//Initializing SymmetricalLayout layout
DiagramLayout LayoutValue = new DiagramLayout()
{
//Sets layout type as SymmetricalLayout...
Type = LayoutType.SymmetricalLayout,
SpringFactor = 0.8,
SpringLength = 80,
MaxIteration = 500,

```

```
Margin = new LayoutMargin() { Top = 20, Left = 20 }
};
//Initializing DataMapSetting
List<DiagramDataMapSetting> DataMap = new List<DiagramDataMapSetting>()
{
    new DiagramDataMapSetting()
    {
        Property = "Annotations[0].Content",
        Field = "Type"
    },
};
DiagramNode NodeDefaults = new DiagramNode()
{
    Width = 25,
    Height = 25,
    Annotations = new ObservableCollection<DiagramNodeAnnotation>()
    {
        new DiagramNodeAnnotation()
        {
            Id = "label1",
            Style = new AnnotationStyle()
            {
                Color = "black"
            }
        },
        Style = new NodeShapeStyle
        {
            Fill = "#ff6329",
            StrokeColor = "black"
        },
        Shape = new DiagramShape()
        {
            Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
            BasicShape = BasicShapes.Ellipse
        }
    };
TreeInfo treeLayoutInfo = new TreeInfo()
{
    Orientation = SubTreeOrientation.Vertical,
    Offset = -20,
    GetAssistantDetails = new AssistantsDetails()
    {
        Root = "General Manager",
        Assistants = new string[] { "Assistant Manager" }
    };
DiagramConnector ConnectorDefaults = new DiagramConnector
{
    Type = Syncfusion.Blazor.Diagrams.Segments.Straight,
};
public class SymmetricalDetails
{
    public string Id { get; set; }
    public string Source { get; set; }
    public string Type { get; set; }
}
```



```
public object DataSource = new List<object>()
{
    new SymmetricalDetails() { Id= "1",Source="", Type = "Server" },
    new SymmetricalDetails() { Id= "2", Source="1", Type= "Server" },
    new SymmetricalDetails() { Id= "3", Source="1", Type= "Server" },
    new SymmetricalDetails() { Id= "4", Source="2", Type= "Server" },
    new SymmetricalDetails() { Id= "5", Source="2", Type= "Server" },
    new SymmetricalDetails() { Id= "6", Source= "2", Type= "Hub" },
    new SymmetricalDetails() { Id= "7", Source= "2", Type= "Hub" },
    new SymmetricalDetails() { Id= "8", Source= "2", Type= "Hub" },
    new SymmetricalDetails() { Id= "9", Source= "2", Type= "Hub" },
    new SymmetricalDetails() { Id= "10", Source= "2", Type= "Hub" },
    new SymmetricalDetails() { Id= "11", Source= "2", Type= "Hub" },
    new SymmetricalDetails() { Id= "12", Source= "2", Type= "Hub" },
    new SymmetricalDetails() { Id= "13", Source= "2", Type= "Hub" },
    new SymmetricalDetails() { Id= "14",Source= "2", Type= "Hub" },
    new SymmetricalDetails() { Id= "15", Source= "3", Type= "Hub" },
    new SymmetricalDetails() { Id= "16", Source= "3", Type= "Hub" },
    new SymmetricalDetails() { Id= "17", Source= "3", Type= "Hub" },
    new SymmetricalDetails() { Id= "18", Source= "3", Type= "Hub" },
    new SymmetricalDetails() { Id= "19", Source= "3", Type= "Hub" },
    new SymmetricalDetails() { Id= "20", Source= "3", Type= "Hub" },
    new SymmetricalDetails() { Id= "21", Source= "3", Type= "Hub" },
    new SymmetricalDetails() { Id= "22", Source= "3", Type= "Hub" },
    new SymmetricalDetails() { Id= "23",Source= "3", Type= "Hub" },
    new SymmetricalDetails() { Id= "24", Source="3", Type= "Hub" },
};
}
```



### *Customize the layout*

You can change the following properties of the symmetric layout.

- Spring length
- Spring factor
- Max iteration

To explore layout properties, refer to the [Layout Properties](#).

The following code is used to change the properties at runtime.

### **CSHARP**

```
public void UpdateProperties() {  
    Diagram.Layout.SpringLength += 10;  
    Diagram.Layout.SpringFactor += 10;  
    Diagram.Layout.MaxIteration += 20;  
}
```

### *See also*

- [How to create a node](#)
- [How to create a connector](#)
- [How to generate the organization chart](#)

### Line Distribution in Blazor Diagram Component

Line distribution is used to arrange the connectors without overlapping in automatic layout. In some cases, the automatic layout connectors connecting to the nodes will be overlapped with one another. So user can decide whether the segment of each connector from a single parent node should be same point or different point. The [ConnectionPointOrigin](#) property of layout is used to enable or disable the line distribution in layout. By default ConnectionPointOrigin will be `SamePoint`.

The following code example illustrates how to create a complex hierarchical tree with line distribution.

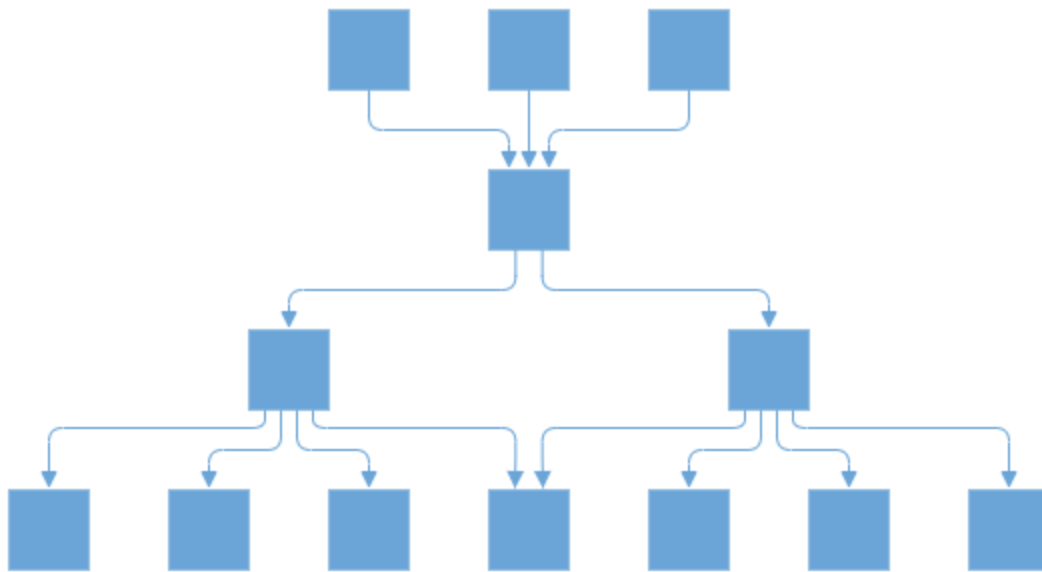
#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram ID="diagram" Height="600px" NodeDefaults="@NodeDefaults"
ConnectorDefaults="@ConnectorDefaults" Layout="@LayoutValue">
<DiagramDataSource Id="Name" ParentId="ReportingPerson"
DataSource="@DataSource"></DiagramDataSource>
<DiagramPageSettings>
<DiagramFitOptions CanFit="true" Mode="FitModes.Width"></DiagramFitOptions>
</DiagramPageSettings>
<DiagramSnapSettings
Constraints="SnapConstraints.None"></DiagramSnapSettings>
</SfDiagram>
@code{
DiagramLayout LayoutValue = new DiagramLayout() { };
DiagramConnector ConnectorDefaults = new DiagramConnector()
{
Type = Segments.Orthogonal,
CornerRadius = 7,
TargetDecorator = new ConnectorTargetDecorator() { Width = 7, Height = 7,
Style = new DecoratorShapeStyle() { Fill = "#6BA5D7", StrokeColor =
"#6BA5D7" } },
Style = new ConnectorShapeStyle() { StrokeColor = "#6BA5D7", StrokeWidth = 1
}
};
DiagramNode NodeDefaults = new DiagramNode
{
Height = 40,
Width = 40,
Style = new NodeShapeStyle() { StrokeWidth = 2, Fill = "#6BA5D7",
StrokeColor = "None" },
Shape = new DiagramShape() { BasicShape = BasicShapes.Rectangle, Type =
Syncfusion.Blazor.Diagrams.Shapes.Basic },
BackgroundColor = "#6BA5D7"
};
public class DataModel
{
public string Name;
public string FillColor;
public string[] ReportingPerson;
}
public object DataSource = new List<object>()
{
new DataModel { Name = "node11", FillColor = "#6BA5D7" },
new DataModel { Name = "node12", ReportingPerson = new string[]{ "node114"
}, FillColor = "#6BA5D7" },
```

```

new DataModel { Name = "node13", ReportingPerson = new string[] { "node12"
}, FillColor = "#6BA5D7" },
new DataModel { Name = "node14", ReportingPerson = new string[] { "node12"
}, FillColor = "#6BA5D7" },
new DataModel { Name = "node15", ReportingPerson = new string[] { "node12"
}, FillColor = "#6BA5D7" },
new DataModel { Name = "node18", ReportingPerson = new string[] {},
FillColor = "#6BA5D7" },
new DataModel { Name = "node21", FillColor = "#6BA5D7" },
new DataModel { Name = "node22", ReportingPerson = new string[] { "node114"
}, FillColor = "#6BA5D7" },
new DataModel { Name = "node23", ReportingPerson = new string[] { "node22"
}, FillColor = "#6BA5D7" },
new DataModel { Name = "node24", ReportingPerson = new string[] { "node22"
}, FillColor = "#6BA5D7" },
new DataModel { Name = "node25", ReportingPerson = new string[] { "node22"
}, FillColor = "#6BA5D7" },
new DataModel { Name = "node26", ReportingPerson = new string[] {},
FillColor = "#6BA5D7" },
new DataModel { Name = "node28", ReportingPerson = new string[] {},
FillColor = "#14ad85" },
new DataModel { Name = "node31", FillColor = "#6BA5D7" },
new DataModel { Name = "node114", ReportingPerson = new string[] {
"node11", "node21", "node31" }, FillColor = "#6BA5D7" }
};
protected override void OnInitialized()
{
LayoutValue = new DiagramLayout()
{
Type = LayoutType.ComplexHierarchicalTree,
VerticalAlignment = VerticalAlignment.Top,
HorizontalAlignment = HorizontalAlignment.Left,
HorizontalSpacing = 40,
VerticalSpacing = 40,
Orientation = LayoutOrientation.TopToBottom,
Margin = new LayoutMargin() { Left = 0, Top = 0, Bottom = 0, Right = 0 },
ConnectionPointOrigin = ConnectionPointOrigin.SamePoint
};
}
}

```



#### *Prevent connectors overlay*

The below constraints prevents the connector segments overlapping nodes with a complex hierarchical layout.

The following code illustrates how to prevent the connector segments overlapping in diagram layout.

#### **CSHARP**

```
protected override void OnInitialized()  
{  
    LayoutValue = new DiagramLayout()  
    {  
        //this prevents connector segments overlapping  
        enableRouting: true,  
    };  
}
```

#### *See also*

- [How to create a node](#)
- [How to create a connector](#)

#### Linear Arrangement in Blazor Diagram Component

Linear arrangement is used to linearly arrange the child nodes in layout, which means the parent node is placed in the center corresponding to its children. When line distribution is enabled, linear arrangement is also activated by default. The [Arrangement](#) property of layout is used to enable or disable the linear arrangement in layout. By default Arrangement will be **Nonlinear**.

---

Linear arrangement is applicable only for complex hierarchical tree layout.

---

The following code illustrates how to allow a linear arrangement in diagram layout.

#### **CSHARP**

```
protected override void OnInitialized()
{
    LayoutValue = new DiagramLayout()
    {
        Type = LayoutType.ComplexHierarchicalTree,
        HorizontalSpacing = 40,
        VerticalSpacing = 40,
        Orientation = LayoutOrientation.TopToBottom,
        //To arrange a child nodes in a linear manner
        Arrangement = ChildArrangement.Linear
    };
}
```

*See also*

- [How to create a node](#)
- [How to create a connector](#)

#### Accessibility in Blazor Diagram Component

Diagram provides built-in compliance with the [WAI-ARIA](#) specifications. WAI-ARIA Accessibility supports are achieved through the attributes like `aria-label`. It helps to provides information about elements in a document for assistive technology.

**Aria-label:** Attribute provides the text label with some default description for below elements in diagram.

<!-- markdownlint-disable MD033 -->

Element	Default description
ResizeNorthWest	Thumb to resize the selected object on the top-left corner.
ResizeNorthEast	Thumb to resize the selected object on the top-right side direction.
ResizeSouthWest	Thumb to resize the selected object on the bottom-left side direction.
ResizeSouthEast	Thumb to resize the selected object on the bottom-right side direction.
ResizeNorth	Thumb to resize the selected object on the top side direction.
ResizeSouth	Thumb to resize the selected object on the bottom side direction.
ResizeWest	Thumb to resize the selected object on the left side direction.
ResizeEast	Thumb to resize the selected object on the right side direction.
ConnectorSourceThumb	Thumb to move the source point of the connector.

ConnectorTargetThumb	Thumb to move the target point of the connector.
RotateThumb	Thumb to rotate the selected object.

## Commands in Blazor Diagram Component

<!-- markdownlint-disable MD010 -->

There are several commands available in the diagram as follows.

- Alignment commands
- Distribute commands
- Sizing commands
- Clipboard commands
- Grouping commands
- Z-order commands
- Zoom commands
- Nudge commands
- FitToPage commands
- Undo/Redo commands

### Alignment commands

Alignment commands enable you to align the selected or defined objects such as nodes and connectors with respect to the selection boundary. Refer to [Align](#) commands which shows how to use align methods in the diagram.

| Parameters | Description |

|:-----| :-----|

| [Alignment Options](#) | Defines the specific direction, with respect to which the objects to be aligned. The accepted values of the argument "alignment options" are as follows. <br> Left - Aligns all the selected objects at the left of the selection boundary. <br> Right - Aligns all the selected objects at the right of the selection boundary. <br> Center - Aligns all the selected objects at the center of the selection boundary. <br> Top - Aligns all the selected objects at the top of the selection boundary. <br> Bottom - Aligns all the selected objects at the bottom of the selection boundary. <br> Middle - Aligns all the selected objects at the middle of the selection boundary. |

| Objects | Defines the objects to be aligned. This is an optional parameter. By default, all the nodes and connectors in the selected region of the diagram gets aligned. |

| [Alignment Mode](#) | Defines the specific mode, with respect to which the objects to be aligned. This is an optional parameter. The default alignment mode is **Object**. <br> The accepted values of the argument "alignment mode" are as follows. <br> Object - Aligns the objects based on the first object in the selected list. <br> Selector - Aligns the objects based on the selection boundary. |

The following code example illustrates how to align all the selected objects at the left side of the selection boundary.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px" />
```

```
@code
{
//Reference to diagram
SfDiagram diagram;
private void Align()
{
//Aligns the selected items to left
diagram.Align(AlignmentOptions.Left, null, AlignmentMode.Selector);
}
}
```

### Distribute

The [Distribute](#) commands enable to place the selected objects on the page at equal intervals from each other. The selected objects are equally spaced within the selection boundary.

The factor to distribute the shapes **DistributeOptions** are listed as follows:

- **RightToLeft**: Distributes the objects based on the distance between the right and left sides of the adjacent objects.
- **Left**: Distributes the objects based on the distance between the left sides of the adjacent objects.
- **Right**: Distributes the objects based on the distance between the right sides of the adjacent objects.
- **Center**: Distributes the objects based on the distance between the center of the adjacent objects.
- **BottomToTop**: Distributes the objects based on the distance between the bottom and top sides of the adjacent objects.
- **Top**: Distributes the objects based on the distance between the top sides of the adjacent objects.
- **Bottom**: Distributes the objects based on the distance between the bottom sides of the adjacent objects.
- **Middle**: Distributes the objects based on the distance between the vertical center of the adjacent objects.

The following code example illustrates how to execute the space commands.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px" />
@code
{
//Reference to diagram
SfDiagram diagram;
private void Distribute()
{
//Distribute the elements in equal spacing
diagram.Distribute(DistributeOptions.RightToLeft);
}
}
```



### Sizing Commands

Sizing [SameSize](#) commands enable to equally size the selected nodes with respect to the first selected object.

**SizingOptions** are as follows:

- Width: Scales the width of the selected objects.
- Height: Scales the height of the selected objects.
- Size: Scales the selected objects both vertically and horizontally.

The following code example illustrates how to execute the size commands.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px" />
@code
{
    //Reference to diagram
    SfDiagram diagram;
    private void SameSize()
    {
        //Changing the selected nodes to same size
        SizingOptions sizingOptions = SizingOptions.Size;
        diagram.SameSize(sizingOptions);
    }
}
```

### Clipboard

Clipboard commands are used to cut, copy, or paste the selected elements. Refer to the following link which shows how to use clipboard methods in the diagram.

- Cuts the selected elements from the diagram to the diagram's clipboard, [Cut](#).
- Copies the selected elements from the diagram to the diagram's clipboard, [Copy](#).
- Pastes the diagram's clipboard data (nodes/connectors) into the diagram, [Paste](#).

The following code illustrates how to execute the clipboard commands.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px">
</SfDiagram>
@code
{
    //Reference to diagram
    SfDiagram diagram;
    private void Cut()
    {
        //copies the selected nodes
        this.diagram.Cut();
    }
    private void Copy()
    {

```

```
//copies the selected nodes
this.diagram.Copy();
}
private void Paste()
{
//pastes the copied objects
this.diagram.Paste();
}
}
```

### Grouping

**Grouping commands** are used to group/ungroup the selected elements on the diagram. Refer to the following link which shows how to use grouping commands in the diagram.

[Group](#) the selected nodes and connectors in the diagram.

[Ungroup](#) the selected nodes and connectors in the diagram.

The following code illustrates how to execute the grouping commands.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px">
</SfDiagram>
@code
{
//Reference to diagram
SfDiagram diagram;
private void SelectAll()
{
//Selects the nodes
this.diagram.SelectAll();
}
private void Group()
{
//Groups the selected elements.
this.diagram.Group();
}
}
```

### Z-Order command

**Z-Order commands** enable you to visually arrange the selected objects such as nodes and connectors on the page.

#### BringToFront command

The [BringToFront](#) command visually brings the selected element to front over all the other overlapped elements. The following code illustrates how to execute the **BringToFront** command.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px">
</SfDiagram>
@code
{
```

```
//Reference to diagram
SfDiagram diagram;
private void BringToFront()
{
    //Brings to front the selected node.
    this.diagram.BringToFront();
}
```

### SendToBack command

The [SendToBack](#) command visually moves the selected element behind all the other overlapped elements. The following code illustrates how to execute the `SendToBack` command.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px">
</SfDiagram>
@code
{
    //Reference to diagram
    SfDiagram diagram;
    private void SendToBack()
    {
        //Sends to front the selected node.
        this.diagram.SendToBack();
    }
}
```

### MoveForward command

The [MoveForward](#) command visually moves the selected element over the nearest overlapping element. The following code illustrates how to execute the `MoveForward` command.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px">
</SfDiagram>
@code
{
    //Reference to diagram
    SfDiagram diagram;
    private void MoveForward()
    {
        //move to Forward the selected node.
        this.diagram.MoveForward();
    }
}
```

### SendBackward command

The [SendBackward](#) command visually moves the selected element behind the underlying element. The following code illustrates how to execute the `SendBackward` command.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px">
</SfDiagram>
@code
{
//Reference to diagram
SfDiagram diagram;
private void SendBackward()
{
//send to Forward the selected node.
this.diagram.SendBackward();
}
}
```

### Zoom

The [Zoom](#) command is used to zoom-in and zoom-out the diagram view.

The following code illustrates how to zoom-in/zoom out the diagram.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px">
</SfDiagram>
@code
{
//Reference to diagram
SfDiagram diagram;
private void Zoom()
{
// Sets the ZoomFactor
//Defines the FocusPoint to zoom the Diagram with respect to any point
//When you do not set focus point, zooming is performed with reference to
the center of current Diagram view.
this.diagram.Zoom(1.2, new PointModel() { X = 100, Y = 100 });
}
}
```

### Nudge command

The [Nudge](#) commands move the selected elements towards up, down, left, or right by 1 pixel.

**NudgeDirection** nudge command moves the selected elements towards the specified direction by 1 pixel, by default.

The accepted values of the argument "direction" are as follows:

- Up: Moves the selected elements towards up by the specified delta value.
- Down: Moves the selected elements towards down by the specified delta value.
- Left: Moves the selected elements towards left by the specified delta value.
- Right: Moves the selected elements towards right by the specified delta value.

The following code illustrates how to execute nudge command.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px">
</SfDiagram>
@code
{
//Reference to diagram
SfDiagram diagram;
private void NudgeRight()
{
//Nudges to right
this.diagram.Nudge(NudgeDirection.Right);
}
}
```

### Nudge by using arrow keys

The corresponding arrow keys are used to move the selected elements towards up, down, left, or right direction by 1 pixel.



Nudge commands are particularly useful for accurate placement of elements.

### BringIntoView

The [BringIntoView](#) command brings the specified rectangular region into the viewport of the diagram.

The following code illustrates how to execute the `BringIntoView` command.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px">
</SfDiagram>
@code
{
//Reference to diagram
SfDiagram diagram;
private void BringIntoView()
{
var bound = new System.Drawing.Rectangle(600, 600, 500, 400);
//Brings the specified rectangular region of the Diagram content to the
viewport of the page.
this.diagram.BringIntoView(bound);
}
}
```

### BringToCenter

The [BringToCenter](#) command brings the specified rectangular region of the diagram content to the center of the viewport.

The following code illustrates how to execute the `BringToCenter` command.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
```

```
<SfDiagram @ref="@diagram" Height="600px">
</SfDiagram>
@code
{
//Reference to diagram
SfDiagram diagram;
private void BringToCenter()
{
var bound = new System.Drawing.Rectangle(600, 600, 500, 400);
//Brings the specified rectangular region of the Diagram content to the
viewport of the page.
this.diagram.BringToCenter(bound);
}
}
```

### FitToPage command

The [FitToPage](#) command helps to fit the diagram content into the view with respect to either width, height, or at the whole.

The [Mode](#) parameter defines whether the diagram must be horizontally/vertically fit into the viewport with respect to width, height, or entire bounds of the diagram.

The [Region](#) parameter defines the region that must be drawn as an image.

The [Margin](#) parameter defines the region/bounds of the diagram content that is to be fit into the view.

The [CanZoomIn](#) parameter enables/disables zooming to fit the smaller content into a larger viewport.

The [CustomBounds](#) parameter the custom region that must be fit into the viewport.

The following code illustrates how to execute `FitToPage` command.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px">
</SfDiagram>
@code
{
//Reference to diagram
SfDiagram diagram;
private void FitToPage()
{
//fit the diagram to the page
diagram.FitToPage();
}
}
```

### Undo and Redo command

The [Undo](#) and [Redo](#) commands help you to revert/restore the changes.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram @ref="@diagram" Height="600px">
</SfDiagram>
@code
```

```
{
//Reference to diagram
SfDiagram diagram;
private void Undo()
{
//Revert the changes
diagram.Undo();
}
private void Redo()
{
//Restore the changes
diagram.Redo();
}
}
```

### Command manager

Diagram provides support to map or bind command execution with desired combination of key gestures. Diagram provides some built-in commands.

The [CommandManager](#) provides support to define custom commands. The custom commands are executed when the specified key gesture is recognized.

### Command Execution Event

You can use the [OnCommandExecuted](#) event to trigger when execute the custom command in diagram.

### Custom command

To define a custom command, specify the following properties:

- [Gesture](#): A combination of [Keys](#) and [KeyModifiers](#).
- [Parameter](#): Defines any additional parameters that are required at runtime.
- [Name](#): Defines the name of the command.

To explore the properties of custom commands, refer to the [Commands](#).

The following code example shows how to define a custom command.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram @ref="@Diagram" Height="600px"
Nodes="@NodeCollection">
@* Initializing the custom commands *@
<DiagramCommandManager>
<DiagramCommands>
<DiagramCommand Name="customGroup">
<DiagramKeyGesture Key="Keys.G"
KeyModifiers="KeyModifiers.Control"></DiagramKeyGesture>
</DiagramCommand>
<DiagramCommand Name="customUnGroup">
<DiagramKeyGesture Key="Keys.U"
KeyModifiers="KeyModifiers.Control"></DiagramKeyGesture>
</DiagramCommand>
</DiagramCommands>
</DiagramCommandManager>
```

```

@* To define the custom commands execution event *@
<DiagramEvents OnCommandExecuted="@CommandExecute"></DiagramEvents>
</SfDiagram>
@code {
// Reference to diagram
SfDiagram Diagram;
// Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
//Initializing the nodes collection
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode diagramNode = new DiagramNode()
{
Id = "node1",
OffsetX = 100,
OffsetY = 100,
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#659be5", StrokeColor = "none" },
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation() { Content = "Node" }
}
};
NodeCollection.Add(diagramNode);
DiagramNode diagramNode1 = new DiagramNode()
{
Id = "node2",
OffsetX = 300,
OffsetY = 100,
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#659be5", StrokeColor = "none" },
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation() { Content = "Node" }
}
};
NodeCollection.Add(diagramNode1);
}
/// <summary>
/// Custom command execution event
/// </summary>
public void CommandExecute(ICommandExecuteEventArgs args)
{
if (args.Gesture.KeyModifiers == KeyModifiers.Control && args.Gesture.Key ==
Keys.G)
{
//Custom command to group the selected nodes
Diagram.Group();
}
if (args.Gesture.KeyModifiers == KeyModifiers.Control && args.Gesture.Key ==
Keys.U)
{
//Custom command to ungroup the selected items
}
}

```



```

if (Diagram.SelectedItems.Nodes.Count > 0 &&
Diagram.SelectedItems.Nodes[0].Children != null &&
Diagram.SelectedItems.Nodes[0].Children.Length > 0)
{
    Diagram.UnGroup();
}
}
}
}
}

```

### *Modify the existing command*

When any one of the default commands is not desired, they can be disabled. To change the functionality of a specific command, the command can be completely modified.

The following code example shows how to disable a command and how to modify the built-in commands.

### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram @ref="@Diagram" Height="600px"
Nodes="@NodeCollection">
    /* Initializing the custom commands */
    <DiagramCommandManager>
        <DiagramCommands>
            <DiagramCommand Name="navigationDown">
                <DiagramKeyGesture Key="Keys.Down"></DiagramKeyGesture>
            </DiagramCommand>
            <DiagramCommand Name="navigationUp">
                <DiagramKeyGesture Key="Keys.Up"></DiagramKeyGesture>
            </DiagramCommand>
            <DiagramCommand Name="navigationLeft">
                <DiagramKeyGesture Key="Keys.Left"></DiagramKeyGesture>
            </DiagramCommand>
            <DiagramCommand Name="navigationRight">
                <DiagramKeyGesture Key="Keys.Right"></DiagramKeyGesture>
            </DiagramCommand>
        </DiagramCommands>
    </DiagramCommandManager>
    /* To define the custom commands execution event */
    <DiagramEvents OnCommandExecuted="@CommandExecute"></DiagramEvents>
</SfDiagram>
@code {
    // Reference to diagram
    SfDiagram Diagram;
    // Defines diagram's nodes collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    protected override void OnInitialized()
    {
        //Initializing the nodes collection
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode diagramNode = new DiagramNode()
        {
            Id = "node1",
            OffsetX = 100,

```

```

OffsetY = 100,
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#659be5", StrokeColor = "none" },
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    new DiagramNodeAnnotation() { Content = "Node" }
};
NodeCollection.Add(diagramNode);
}
/// <summary>
/// Custom command execute event
/// </summary>
public void CommandExecute(ICommandExecuteEventArgs args)
{
    if (args.Gesture.Key == Keys.Left)
    {
        //Allow left arrow key to nudge the selected node in left
        if (Diagram.SelectedItems.Nodes.Count > 0)
            Diagram.Nudge(NudgeDirection.Left);
    }
    if (args.Gesture.Key == Keys.Down || args.Gesture.Key == Keys.Up ||
        args.Gesture.Key == Keys.Right)
    {
        //to disable a built-in command and none of action execute
    }
}
}

```

### See Also

- [How to create the custom context menu items](#)

## Undo Redo in Blazor Diagram Component

Diagram tracks the history of actions that are performed after initializing the diagram and provides support to reverse and restore those changes.

### Undo and redo

Diagram provides built-in support to track the changes that are made through interaction and through public APIs. The changes can be reverted or restored either through shortcut keys or through commands.

### Undo/redo through shortcut keys

Undo/redo commands can be executed through shortcut keys. Shortcut key for undo is Ctrl+Z and shortcut key for redo is Ctrl+Y.

### Undo/redo through public APIs

The server-side methods [Undo](#) and [Redo](#) help you to revert/restore the changes. The following code example illustrates how to undo/redo the changes through script.

### CSHARP

```
SfDiagram Diagram;
```

```
// Reverts the last action performed
this.Diagram.Undo();
// Restores the last undone action
this.Diagram.Redo();
```

When a change in the diagram is reverted or restored (undo/redo), the [HistoryChange](#) event gets triggered.

#### Group multiple changes

History list allows to revert or restore multiple changes through a single undo/redo command. For example, revert/restore the fill color change of multiple elements at a time.

The server-side method [StartGroupAction](#) is used to notify the diagram to start grouping the changes. The server-side method [EndGroupAction](#) is used to notify to stop grouping the changes. The following code illustrates how to undo/redo to change of multiple elements at a time.

#### CSHARP

```
SfDiagram Diagram;
//Starts grouping the changes
this.Diagram.StartGroupAction();
//Ends grouping the changes
this.Diagram.EndGroupAction();
```

#### Track custom changes

Diagram provides options to track the changes that are made to custom properties. For example, in case of an employee relationship diagram, track the changes in the employee information. The [HistoryList](#) of the diagram enables you to track such changes.

The following example illustrates how to track such custom property changes.

Before changing the employee information, save the existing information to history list by using the server-side method push of [HistoryList](#). The history list [CanLog](#) method can be used which takes a history entry as argument and returns whether the specific entry can be added or not.

The following code example illustrates how to save the existing property values.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600" @ref="@diagram" Nodes="@NodeCollection">
</SfDiagram>
@code{
//Reference of diagram
SfDiagram diagram;
public string Height { get; set; } = "500px";
//Initialize node collection with node
ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
//Unique id of the node
Id="NewIdea",
```

```
//Size of the node
Height=100,
Width=100,
//Position of the node
OffsetX=100,
OffsetY=100,
//Customize the appearance of the node
Style= new NodeShapeStyle() {Fill="#6BA5D7",StrokeColor="White"},
//Enable shadow constraint of the node
Constraints=NodeConstraints.Default|NodeConstraints.Shadow,
//Initialize annotation collection with annotation
Annotations=new ObservableCollection<DiagramNodeAnnotation>()
{
    new DiagramNodeAnnotation()
    {
        Content="node1",
        //Customize the appearance of the annotations
        Style=new AnnotationStyle()
        {
            Color="White",
            StrokeColor="None"
        },
    },
};
//Customizes the appearance of the node shadow style
Shadow=new DiagramShadow()
{
    Angle=50,
    Opacity=0.8,
    Distance=9
};
};
public void TrackCustomActions()
{
    //Creates a custom entry
    HistoryEntry historyEntry = new HistoryEntry() { UndoObject =
    diagram.Nodes[0] };
    // adds that to history list
    diagram.AddHistoryEntry(historyEntry);
    diagram.DataBind();
}
}
```

#### Track undo/redo actions

The [GetHistoryStack](#) method is used to get the collection of undo and redo actions which should be performed in the diagram.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize Diagram */
<SfDiagram Height="1000" @ref="@diagram" Nodes="@NodeCollection">
</SfDiagram>
@code
```

```
{
//Get the collection of undostack objects when passing true to
GetHistoryStack() method.
List<HistoryEntry> undostack = await diagram.GetHistoryStack(true);
//Get the collection of redo stack objects when passing true to
GetHistoryStack() method.
List<HistoryEntry> redostack = await diagram.GetHistoryStack(false);
}
```

### History change event

The [HistoryChange](#) event triggers, whenever the interaction of the node and connector is take place.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagram Height="600" >
</SfDiagram>
@code
{
public void Onhistorychange(IBlazorHistoryChangeArgs args)
{
//causes of history change
var cause = args.Cause;
}
}
```

### Events

#### Created

Triggered when the diagram is rendered completely.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
<DiagramEvents Created="Created"></DiagramEvents>
</SfDiagram>
@code
{
public void Created()
{
//Action to be performed.
}
}
```

#### Clicked

Triggers when a node, connector, or diagram is clicked.

Argument Name	Description
---------------	-------------

-----	-----
-------	-------

Element	Returns the object that is clicked or id of the diagram.
---------	--

- | Position | Returns the object position that is actually clicked. |
- | Count | Returns the number of times clicked. |
- | ActualObject | Returns the actual object that is clicked or id of the diagram. |
- | Button | Returns the button clicked. |

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
  <DiagramEvents Clicked="Clicked"></DiagramEvents>
</SfDiagram>
@code
{
  public void Clicked(IBlazorClickEventArgs args)
  {
    //Action to be performed.
  }
}
```

**ContextMenuItemClicked**

Triggers when a context menu item is clicked.

- | Argument Name | Description                                     |
|---------------|---|
| -----         | -----   |
| Element       | Returns the object that is clicked.             |
| Item          | Returns the actual object that is clicked.      |
| Name          | Returns the name of the object that is clicked. |
| Cancel        | Returns whether to cancel the change or not.    |

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
  <DiagramEvents
    ContextMenuItemClicked="ContextMenuItemClicked"></DiagramEvents>
</SfDiagram>
@code
{
  public void ContextMenuItemClicked(DiagramMenuEventArgs args)
  {
    //Action to be performed.
  }
}
```

**OnContextMenuOpen**

Triggers before opening the context menu.

- | Argument Name | Description |
|---------------|-------------|
| -----         | -----       |

- | Cancel | Returns whether to cancel the change or not. |
- | Element | Returns the object that is clicked. |
- | HiddenItems | Defines the hidden items of the diagram context menu. |
- | Items | Defines the items of the diagram context menu. |

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
  <DiagramEvents OnContextMenuOpen="OnContextMenuOpen"></DiagramEvents>
</SfDiagram>
@code
{
  public void OnContextMenuOpen(DiagramBeforeMenuOpenEventArgs args)
  {
    //Action to be performed.
  }
}
```

**DataLoaded**

Triggers after the diagram is are populated from the external data source

- | Argument Name | Description |
- | ----- | ----- |
- | Diagram | Returns the diagram. |

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
  <DiagramEvents DataLoaded="DataLoaded"></DiagramEvents>
</SfDiagram>
@code
{
  public void DataLoaded(IDataLoadedEventArgs args)
  {
    //Action to be performed.
  }
}
```

**OnDoubleClick**

Triggers when a node, connector, or diagram is clicked.

- | Argument Name | Description |
- | ----- | ----- |
- | Count | Returns the number of times clicked. |
- | Position | Returns the object position that is actually clicked. |
- | Element | Returns the object that is clicked or id of the diagram. |

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
  <DiagramEvents OnDoubleClick="OnDoubleClick"></DiagramEvents>
</SfDiagram>
@code
{
  public void OnDoubleClick(IBlazorDoubleClickEventArgs args)
  {
    //Action to be performed.
  }
}
```

### DragEnter

Triggers when a symbol is dragged into a diagram from the symbol palette.

Argument Name	Description
Cancel	Returns whether to add or remove the symbol from the diagram.
DiagramId	Returns the id of the diagram.
Element	Returns the node or connector that is dragged into a diagram.
Source	Returns the node or connector that is to be dragged into a diagram.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
  <DiagramEvents DragEnter="DragEnter"></DiagramEvents>
</SfDiagram>
@code
{
  public void DragEnter(IBlazorDragEnterEventArgs args)
  {
    //Action to be performed.
  }
}
```

### DragLeave

Triggers when a symbol is dragged outside of the diagram.

Argument Name	Description
DiagramId	Returns the id of the diagram.
Element	Returns the node or connector that is dragged outside of the diagram.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
  <DiagramEvents DragLeave="DragLeave"></DiagramEvents>
</SfDiagram>
```



```
@code
{
public void DragLeave(IBlazorDragLeaveEventArgs args)
{
//Action to be performed.
}
}
```

### OnDrop

Triggers when a symbol is dragged and dropped from the symbol palette to the drawing area.

Argument Name	Description
Cancel	Returns node or connector that is being dropped.
Element	Returns the node or connector that is dragged into a diagram.
Position	Returns the position of the object.
Source	Returns the object from where the element is dragged.
Target	Returns the object over which the object will be dropped.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
<DiagramEvents OnDrop="OnDrop"></DiagramEvents>
</SfDiagram>
@code
{
public void OnDrop(IBlazorDropEventArgs args)
{
//Action to be performed.
}
}
```

### HistoryChanged

Triggers when a change is reverted or restored(undo/redo).

Argument Name	Description
Cause	Returns the cause of the event.
Change	Returns an array of objects, where each object represents the changes made in the last undo/redo.
Source	Returns a collection of objects that are changed in the last undo/redo.
Action	Returns the event action.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
```

```
<DiagramEvents HistoryChanged="HistoryChanged"></DiagramEvents>
</SfDiagram>
@code
{
public void HistoryChanged(IBlazorHistoryChangeArgs args)
{
//Action to be performed.
}
}
```

### MouseEnter

Triggered when the mouse enters a node/connector.

Argument Name	Description
-----	-----
ActualObject	Returns when the mouse hovers to the target node or connector.
Element	Returns a parent node of the target node or connector.
Target	Returns the target object over which the selected object is dragged.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
<DiagramEvents MouseEnter="MouseEnter"></DiagramEvents>
</SfDiagram>
@code
{
public void MouseEnter(IBlazorMouseEventArgs args)
{
//Action to be performed.
}
}
```

### MouseLeave

Triggered when the mouse leaves node/connector.

Argument Name	Description
-----	-----
ActualObject	Returns when the mouse hovers to the target node or connector.
Element	Returns a parent node of the target node or connector.
Target	Returns the target object over which the selected object is dragged.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
<DiagramEvents MouseLeave="MouseLeave"></DiagramEvents>
</SfDiagram>
@code
{
```

```
public void MouseLeave (IBlazorMouseEventArgs args)
{
    //Action to be performed.
}
}
```

### OnPositionChange

Triggers while dragging the elements in a diagram.

Argument Name	Description
-----	-----
Source	Returns the node or connector that is being dragged.
State	Returns the state of drag event (Starting, dragging, and completed).
NewValue	Returns the current node or connector that is being dragged.
OldValue	Returns the previous node or connector that is dragged.
Target	Returns the target node or connector that is dragged.
TargetPosition	Returns the offset of the selected items.
AllowDrop	Returns whether the object that can be dropped over the element.
Cancel	Returns whether to cancel the change or not.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
  <DiagramEvents OnPositionChange="OnPositionChange"></DiagramEvents>
</SfDiagram>
@code
{
    public void OnPositionChange (IBlazorDraggingEventArgs args)
    {
        //Action to be performed.
    }
}
```

### PropertyChanged

Triggers once the node or connector property changed.

Argument Name	Description
-----	-----
Element	Returns the selected element.
Cause	Returns the action is nudged or not.
NewValue	Returns the new value of the property that is being changed.
OldValue	Returns the old value of the node property that is being changed.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
<DiagramEvents PropertyChanged="PropertyChanged"></DiagramEvents>
</SfDiagram>
@code
{
public void PropertyChanged(IBlazorPropertyChangeEventArgs args)
{
//Action to be performed.
}
}

```

### OnRotateChange

Triggers when the diagram elements are rotated.

Argument Name	Description
-----	-----
Source	Returns the node that is selected for rotation.
State	Returns the state of an event.
NewValue	Returns the current rotation angle.
OldValue	Returns the previous rotation angle.
Cancel	Returns whether to cancel the change or not.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
<DiagramEvents OnRotateChange="OnRotateChange"></DiagramEvents>
</SfDiagram>
@code
{
public void OnRotateChange(IRotationEventArgs args)
{
//Action to be performed.
}
}

```

### SelectionChanged

Triggers when the selection is changed in the diagram.

Argument Name	Description
-----	-----
Cause	Returns the actual cause of the event.
State	Returns the state of an event.
NewValue	Returns the collection of nodes and connectors that have to be added to the selection list.

| OldValue | Returns the collection of nodes and connectors that have to be removed from the selection list. |

| Type | Returns whether the item is added or removed from the selection list. |

| Cancel | Returns whether or not to cancel the selection change event or not. |

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
  <DiagramEvents SelectionChanged="SelectionChanged"></DiagramEvents>
</SfDiagram>
@code
{
  public void SelectionChanged(IBlazorSelectionChangeEventArgs args)
  {
    //Action to be performed.
  }
}
```

### OnSizeChange

Triggers when a node is resized.

| Argument Name | Description |

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

| Source | Returns the node that is selected for resizing. |

| State | Returns the state of an event. |

| NewValue | Returns the new width, height, offsetX, and offsetY values of the element that is being resized. |

| OldValue | Returns the previous width, height, offsetX, and offsetY values of the element that is being resized. |

| Cancel | Returns whether or not to cancel the size change event or not. |

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
  <DiagramEvents OnSizeChange="OnSizeChange"></DiagramEvents>
</SfDiagram>
@code
{
  public void OnSizeChange(ISizeChangeEventArgs args)
  {
    //Action to be performed.
  }
}
```

### TextEdited

Triggers when editor got focus at the time of node's label or text node editing.

| Argument Name | Description |

	-----		-----	
	Element		Returns a node or connector in which annotation is being edited.	
	Annotation		Returns an annotation which that is being edited.	
	NewValue		Returns the new text value of the element that is being changed.	
	OldValue		Returns the old text value of the element.	
	Cancel		Returns whether or not to cancel the event or not.	

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="1000px" Height="600px">
  <DiagramEvents TextEdited="TextEdited"></DiagramEvents>
</SfDiagram>
@code
{
  public void TextEdited(IBlazorTextEditEventArgs args)
  {
    //Action to be performed.
  }
}
```

**Native events**

The Diagram control provides event support, which triggers while interacting with the diagram. Also, Syncfusion provides native event support in blazor for the following events

	Event Name		Event Type	
	-----		-----	
	onfocus		FocusEventArgs	
	onclick		MouseEventArgs	
	onmousemove		MouseEventArgs	
	onmouseover		MouseEventArgs	
	onmouseout		MouseEventArgs	
	onmousedown		MouseEventArgs	
	onmouseup		MouseEventArgs	
	ondblclick		MouseEventArgs	
	onkeydown		KeyboardEventArgs	
	onkeyup		KeyboardEventArgs	
	ondrop		DragEventArgs	

The native events can be defined as mentioned below. For example, the onmousedown event in diagram.

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
```

```
<SfDiagram @onmousedown="@OnMouseDown" Height="600px"/>
@code
{
    public void OnMouseDown(MouseEventArgs args)
    {
        //Action to be performed
    }
}
```

## Virtualization in Blazor Diagram Component

### Virtualization in Diagram

Virtualization is the process of loading the diagramming objects available in the visible area of the Diagram control, that is, only the diagramming objects that lie within the **ViewPort** of the Scroll Viewer are loaded (remaining objects are loaded only when they come into view).

This feature gives an optimized performance while loading and dragging items to the Diagram that consists of many Nodes and Connectors.

The following code illustrates how to enable [Virtualization](#) mode in the diagram.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@* Initialize Diagram *@
<SfDiagram Height="600px" Constraints="@Constraints">
</SfDiagram>
@code
{
    public DiagramConstraints Constraints = DiagramConstraints.Default |
    DiagramConstraints.Virtualization;
}
```

## Serialization in Blazor Diagram Component

**Serialization** is the process of saving and loading for state persistence of the diagram.

### Save

The diagram is serialized as string while saving. The server-side method, [SaveDiagram](#) helps to serialize the diagram as a string. The following code illustrates how to save the diagram.

### CSHARP

```
SfDiagram Diagram;
//returns serialized string of the Diagram
string Data = await this.Diagram.SaveDiagram();
```

This string can be converted to JSON data and stored for future use. The following snippet illustrates how to save the serialized string into local storage.

### CSHARP

```
localStorage.setItem('fileName', saveData);
saveData = localStorage.getItem('fileName');
```

Diagram can also be saved as raster or vector image files. For more information about saving the diagram as images, refer to [Print and Export](#).

### Load

Diagram is loaded from the serialized string data by server-side method, [LoadDiagram](#). The following code illustrates how to load the diagram from serialized string data.

#### CSHARP

```
SfDiagram Diagram;
//returns serialized string of the Diagram
string Data = await this.Diagram.SaveDiagram();
//Loads the Diagram from saved json data
this.Diagram.LoadDiagram(this.Data);
```

---

Before loading a new diagram, existing diagram is cleared.

---

### Prevent Default Values

The diagram provides supports to simplifying the saved JSON object without adding the default properties that are presented in the diagram. The following code illustrates how to simplify the JSON object.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@* Initialize diagram *@
<SfDiagram Height="600px">
<DiagramSerializationSettings
PreventDefaults="true"></DiagramSerializationSettings>
</SfDiagram>
```

### Printing and Exporting in Blazor Diagram Component

Diagram provides support to export its content as image/svg files. The server-side method [ExportDiagram](#) helps to export the diagram. The following code illustrates how to export the diagram as image.

<!-- markdownlint-disable MD033 -->

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<input type="button" value="Export" @onclick="@OnExport" />
<SfDiagram ID="diagram" Height="600px" @ref="@diagram">
</SfDiagram>
@code{
SfDiagram diagram;
private void OnExport()
{
//Sets the export option for diagram
IExportOptions options = new IExportOptions()
{
//Sets the Mode for diagram export
Mode = ExportModes.Data,
};
diagram.ExportDiagram(options);
```



```
}  
}
```

### Exporting options

Diagram provides support to export the desired region of the diagram to desired formats.

#### File Name

[FileName](#) is the name of the file to be downloaded. By default, the file name is set to **Diagram**.

#### Format

[Format](#) is to specify the type/format of the exported file. By default, the diagram is exported as .jpg format. You can export diagram to the following formats:

- JPG
- PNG
- BMP
- SVG

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams  
<input type="button" value="Export" @onclick="@OnExport" />  
<SfDiagram ID="diagram" Height="600px" @ref="@diagram">  
</SfDiagram>  
@code{  
    SfDiagram diagram;  
    private void OnExport()  
    {  
        //Sets the export option for diagram  
        IExportOptions options = new IExportOptions()  
        {  
            Mode = ExportModes.Data,  
            //Sets the format for diagram export  
            Format = FileFormats.SVG  
        };  
        diagram.ExportDiagram(options);  
    }  
}
```

### Margin

[Margin](#) specifies the amount of space that has to be left around the diagram.

<!-- markdownlint-disable MD033 -->

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams  
<input type="button" value="Export" @onclick="@OnExport" />  
<SfDiagram ID="diagram" Height="600px" @ref="@diagram">  
</SfDiagram>  
@code{  
    SfDiagram diagram;  
    private void OnExport()  
    {
```

```
//Sets the export option for diagram
IExportOptions options = new IExportOptions()
{
    Mode = ExportModes.Data,
    FileName = "diagram",
    Stretch = Stretch.None,
    //Sets the margin for diagram export
    Margin = new DiagramMargin { Left = 10, Right = 10, Bottom = 10, Top = 10 },
    Format = FileFormats.SVG
};
diagram.ExportDiagram(options);
}
```

### Mode

[Mode](#) specifies whether the diagram is to be exported as files or as data (ImageURL/SVG). The exporting options are as follows:

- Data: Exports and downloads the diagram as image.
- Download: Exports the diagram as data of formats ImageURL/SVG.

For more information about the exporting modes, refer to [Exporting Modes](#).

The following code example illustrates how to export the diagram as raw data.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<input type="button" value="Export" @onclick="@OnExport" />
<SfDiagram ID="diagram" Height="600px" @ref="@diagram">
</SfDiagram>
@code{
    SfDiagram diagram;
    private void OnExport()
    {
        //Sets the export option for diagram
        IExportOptions options = new IExportOptions()
        {
            //Sets the mode for diagram export
            Mode = ExportModes.Data,
            FileName = "diagram",
            Stretch = Stretch.None,
            Margin = new Syncfusion.Blazor.Diagrams.MarginModel() { Left = 10, Right = 10, Bottom = 10, Top = 10 },
            Format = FileFormats.SVG
        };
        diagram.ExportDiagram(options);
    }
}
```

### Region

You can export any particular [Region](#) of the diagram and the region is categorized as follows.

- Region that fits all nodes and connectors that are added to model.

- Region that fits all pages (single or multiple pages based on page settings).

For more information about region, refer to [Regions](#).

The following code example illustrates how to export the region occupied by the diagram elements.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<input type="button" value="Export" @onclick="@OnExport" />
<SfDiagram ID="diagram" Height="600px" @ref="@diagram">
</SfDiagram>
@code{
SfDiagram diagram;
private void OnExport()
{
//Sets the export option for diagram
IExportOptions options = new IExportOptions()
{
Mode = ExportModes.Data,
FileName = "format",
Stretch = Stretch.None,
//Sets the region for diagram export
Region = DiagramRegions.Content,
Margin = new Syncfusion.Blazor.Diagrams.MarginModel() { Left = 10, Right = 10, Bottom = 10, Top = 10 },
Format = FileFormats.SVG
};
diagram.ExportDiagram(options);
}
}
```

#### Custom bounds

Diagram provides support to export any specific region of the diagram by using [Bounds](#).

The following code example illustrates how to export the region occupied by the diagram elements.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<input type="button" value="Export" @onclick="@OnExport" />
<SfDiagram ID="diagram" Height="600px" @ref="@diagram">
</SfDiagram>
@code{
SfDiagram diagram;
//Define bounds class for export the diagram
public class exportBounds
{
public double x { get; set; }
public double y { get; set; }
public double width { get; set; }
public double height { get; set; }
}
private void OnExport()
{
//Sets the export option for diagram
IExportOptions options = new IExportOptions()
```

```
{
Mode = ExportModes.Download,
FileName = "format",
PageHeight = 400,
PageWidth = 400,
Stretch = Stretch.None,
Region = DiagramRegions.CustomBounds,
//Sets the custom bound for diagram export
Bounds = new exportBounds() { x = 10, y = 10, width = 100, height = 100 },
Margin = new Syncfusion.Blazor.Diagrams.MarginModel() { Left = 10, Right = 10, Bottom = 10, Top = 10 },
Format = FileFormats.SVG
};
diagram.ExportDiagram(options);
}
}
```

### Export diagram with stretch option

Diagram provides support to export the diagram as image for [Stretch](#) option. The exported images will be clearer but larger in file size.

The following code example illustrates how to export the region occupied by the diagram elements.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<input type="button" value="Export" @onclick="@OnExport" />
<SfDiagram ID="diagram" Height="600px" @ref="@diagram">
</SfDiagram>
@code{
SfDiagram diagram;
private void OnExport()
{
//Sets the export option for diagram
IExportOptions options = new IExportOptions()
{
Mode = ExportModes.Data,
FileName = "region",
Stretch = Stretch.Stretch,
Region = DiagramRegions.Content,
Margin = new Syncfusion.Blazor.Diagrams.MarginModel() { Left = 10, Right = 10, Bottom = 10, Top = 10 },
Format = FileFormats.SVG
};
diagram.ExportDiagram(options);
}
}
```

### Print

The server-side method [Print](#) helps to print the diagram as image.

Name	Type	Description
region	enum	Sets the region of the diagram to be printed.

- | bounds | object | Prints any custom region of diagram. |
- | stretch | enum | Resizes the diagram content to fill its allocated space and printed. |
- | multiplePage | boolean | Prints the diagram into multiple pages. |
- | pageWidth | number | Sets the page width of the diagram while printing the diagram into multiple pages. |
- | pageHeight | number | Sets the page height of the diagram while printing the diagram into multiple pages. |
- | pageOrientation | enum | Sets the orientation of the page. |

The following code example illustrates how to export the region occupied by the diagram elements.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<input type="button" value="Print" @onclick="@OnPrint" />
<SfDiagram ID="diagram" Height="600px" @ref="@diagram">
</SfDiagram>
@code{
SfDiagram diagram;
private void OnPrint()
{
//Sets the print option for diagram
IPrintOptions options = new IPrintOptions()
{
MultiplePage = true,
PageHeight = 400,
PageWidth = 400,
Region = DiagramRegions.PageSettings,
};
diagram.Print(options);
}
}
```

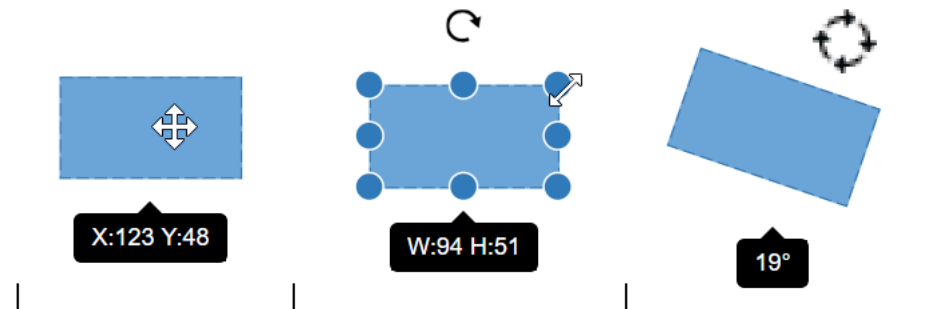
### Tooltip in Blazor Diagram Component

In Graphical User Interface (GUI), the tooltip is a message that is displayed when mouse hovers over an element. The diagram provides tooltip support while dragging, resizing, rotating a node, and when the mouse hovers any diagram element.

#### Default tooltip

By default, diagram displays a tooltip to provide the size, position, and angle related information while dragging, resizing, and rotating. The following images illustrate how the diagram displays the node information during an interaction.

- | Drag | Resize | Rotate |
- |-----|-----|-----|



### Common tooltip for all nodes and connectors

The diagram provides support to show tooltip when the mouse hovers over any node/connector. To show tooltip on mouse over, the [Tooltip](#) property of diagram model needs to be set with the tooltip **Content** and **Position** as shown in the following example.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initializes the diagram component */
<SfDiagram Height="600px" Nodes="@NodeCollection" Tooltip="@tooltip">
</SfDiagram>
@code{
//Defines diagram's node collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
// Defines position of the tooltip
public DiagramTooltip tooltip = new DiagramTooltip()
{
    Content = "Nodes",
    Position = Syncfusion.Blazor.Popups.Position.TopLeft
};
protected override void OnInitialized()
{
    NodeCollection = new ObservableCollection<DiagramNode>();
    //Defines nodes
    DiagramNode node = new DiagramNode()
    {
        Id = "node1",
        Width = 100,
        Height = 100,
        OffsetX = 200,
        OffsetY = 200,
        Style = new NodeShapeStyle() { StrokeColor = "#6BA5D7", Fill = "#6BA5D7" },
        Constraints = NodeConstraints.Default | NodeConstraints.Tooltip,
        Annotations = new ObservableCollection<DiagramNodeAnnotation>()
        {
            new DiagramNodeAnnotation()
            {
                Id = "label", Content = "Rectangle",
                Offset = new NodeAnnotationOffset()
                {
                    X = 0.5,
                    Y = 0.5
                },
                Style = new AnnotationStyle() { Color = "white" }
            }
        }
    }
}
```

```

}
}
};
NodeCollection.Add(node);
}
}

```

### Disable tooltip at runtime

The tooltip on mouse over can be disabled by assigning the [Tooltip](#) property as `null`. The following code example illustrates how to disable the mouse over tooltip at runtime.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@* Initializes the diagram component *@
<SfDiagram Height="600px" Tooltip="@tooltip">
</SfDiagram>
@code{
//Disables mouse over tooltip
public DiagramTooltip tooltip = null;
}

```

### Tooltip for a specific node/connector

The tooltip can be customized for each node and connector. Remove the `InheritTooltip` option from the [Constraints](#) of that node/connector. The following code example illustrates how to customize the tooltip for individual elements.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@* Initializes the diagram component *@
<SfDiagram Height="600px" Constraints="DiagramConstraints.Default |
DiagramConstraints.Tooltip" Nodes="@NodeCollection">
</SfDiagram>
@code{
// Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
//Defines nodes
DiagramNode node = new DiagramNode()
{
Id = "node1",
Width = 100,
Height = 100,
OffsetX = 200,
OffsetY = 200,
Style = new NodeShapeStyle() { StrokeColor = "#6BA5D7", Fill = "#6BA5D7" },
Constraints = NodeConstraints.Default | NodeConstraints.Tooltip,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{

```

```

Id = "label", Content = "Rectangle",
Offset = new NodeAnnotationOffset()
{
X = 0.5,
Y = 0.5
},
Style = new AnnotationStyle(){ Color = "white" }
},
//Defines mouse over tooltip for a node
Tooltip = new NodeTooltip()
{
//Sets the content of the tooltip
Content = "Node1",
//Sets the position of the tooltip
Position = Syncfusion.Blazor.Popups.Position.BottomRight,
//Sets the tooltip position relative to the node
RelativeMode = TooltipRelativeMode.Object
};
NodeCollection.Add(node);
}
}

```

### Tooltip alignments

#### *Tooltip relative to object*

The diagram provides support to show tooltip around the node/connector that is hovered by the mouse. The tooltip can be aligned by using the **Position** property of the tooltip. The **RelativeMode** property of the tooltip defines whether the tooltip has to be displayed around the object or at the mouse position.

The following code example illustrates how to position the tooltip around object.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initializes the diagram component */
<SfDiagram Height="600px"
Constraints="DiagramConstraints.Default | DiagramConstraints.Tooltip"
Nodes="@NodeCollection">
</SfDiagram>
@code{
// Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
//Defines nodes
DiagramNode node = new DiagramNode()
{
Id = "node1",
Width = 100,
Height = 100,
OffsetX = 200,
OffsetY = 200,
Style = new NodeShapeStyle() { StrokeColor = "#6BA5D7", Fill = "#6BA5D7" },
}
}
}

```



```

Constraints = NodeConstraints.Default | NodeConstraints.Tooltip,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    new DiagramNodeAnnotation()
    {
        Id = "label",
        Content = "Rectangle",
        Offset = new NodeAnnotationOffset()
        {
            X = 0.5,
            Y = 0.5
        },
        Style = new AnnotationStyle() { Color = "white" }
    }
},
//Defines mouse over tooltip for a node
Tooltip = new NodeTooltip()
{
    Content = "Node1",
    //Sets the position properties
    Position = Syncfusion.Blazor.Popups.Position.BottomRight,
    //Sets to show tooltip around the element
    RelativeMode = TooltipRelativeMode.Object
}
};
NodeCollection.Add(node);
}
}

```

#### *Tooltip relative to mouse position*

To display the tooltip at mouse position, need to set mouse option to the [RelativeMode](#) property of the tooltip. The following code example illustrates how to show tooltip at mouse position.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initializes the diagram component */
<SfDiagram Height="600px" Constraints="DiagramConstraints.Default |
DiagramConstraints.Tooltip" Nodes="@NodeCollection">
</SfDiagram>
@code{
// Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
    NodeCollection = new ObservableCollection<DiagramNode>();
    //Defines nodes
    DiagramNode node = new DiagramNode()
    {
        Id = "node1",
        Width = 100,
        Height = 100,
        OffsetX = 200,
        OffsetY = 200,
        Style = new NodeShapeStyle() { StrokeColor = "#6BA5D7", Fill = "#6BA5D7" },
    }
}
}

```

```

Constraints = NodeConstraints.Default | NodeConstraints.Tooltip,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    new DiagramNodeAnnotation()
    {
        Id = "label",
        Content = "Rectangle",
        Offset = new NodeAnnotationOffset()
        {
            X = 0.5,
            Y = 0.5
        },
        Style = new AnnotationStyle() { Color = "white" }
    }
},
//Defines mouse over tooltip for a node
Tooltip = new NodeTooltip()
{
    Content = "Node1",
    //Sets to show tooltip at mouse position
    RelativeMode = TooltipRelativeMode.Mouse
};
NodeCollection.Add(node);
}
}

```

### Tooltip animation

To animate the tooltip, a set of specific animation effects are available, and it can be controlled by using the [Animation](#) property. The animation property also allows you to set delay, duration, and various other effects of your choice.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initializes the Diagram component */
<SfDiagram Height="600px"
Constraints="DiagramConstraints.Default | DiagramConstraints.Tooltip"
Nodes="@NodeCollection">
</SfDiagram>
@code{
// Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
    NodeCollection = new ObservableCollection<DiagramNode>();
    DiagramNode node = new DiagramNode()
    {
        Id = "node1",
        Width = 100,
        Height = 100,
        OffsetX = 200,
        OffsetY = 200,
        Style = new NodeShapeStyle() { StrokeColor = "#6BA5D7", Fill = "#6BA5D7" },
        Constraints = NodeConstraints.Default | NodeConstraints.Tooltip,
    }
}
}

```

```

Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    new DiagramNodeAnnotation()
    {
        Id = "label", Content = "Rectangle",
        Offset = new NodeAnnotationOffset()
        {
            X = 0.5,
            Y = 0.5
        },
        Style = new AnnotationStyle(){ Color = "white" }
    },
    //Defines mouse over tooltip for a node
    Tooltip = new NodeTooltip()
    {
        Content = "Node1",
        Position = Syncfusion.Blazor.Popups.Position.BottomRight,
        RelativeMode = TooltipRelativeMode.Mouse,
        Animation = new Syncfusion.Blazor.Popups.AnimationModel()
        {
            //Animation settings to be applied on the tooltip, while it is being shown
            //over the target.
            Open = new Syncfusion.Blazor.Popups.TooltipAnimationSettings()
            {
                //Animation effect on the tooltip is applied during open and close actions.
                Effect = Syncfusion.Blazor.Popups.Effect.ZoomIn,
                //Duration of the animation that is completed per animation cycle.
                Duration = 1000,
                //Indicating the waiting time before animation begins.
                Delay = 0
            },
            //Animation settings to be applied on the tooltip, when it is closed.
            Close = new Syncfusion.Blazor.Popups.TooltipAnimationSettings()
            {
                Effect = Syncfusion.Blazor.Popups.Effect.ZoomOut,
                Duration = 500,
                Delay = 0
            }
        }
    };
    NodeCollection.Add(node);
}
}

```

### Layers in Blazor Diagram Component

**Layer** is used to organize related shapes on a diagram control. A layer is a named category of shapes. By assigning shapes to different layers, you can selectively view, remove, and lock different categories of shapes.

In diagram, [Layers](#) provide a way to change the properties of all shapes that have been assigned to that layer. The following properties can be set.

- Visible

- Lock
- Objects
- AddInfo

### Visible

The layer's [Visible](#) property is used to control the visibility of the elements assigned to the layer.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" Layers="@Layers">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>() { };
public ObservableCollection<DiagramLayer> Layers = new
ObservableCollection<DiagramLayer>() { };
protected override void OnInitialized()
{
// A node is created and stored in nodes array.
DiagramNode node1 = new DiagramNode()
{
Id = "node1",
OffsetX = 100,
OffsetY = 100,
Width = 100,
Height = 100,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation(){ Content = "Default Shape"}
}
};
NodeCollection.Add(node1);
DiagramNode node2 = new DiagramNode()
{
Id = "node2",
OffsetX = 300,
OffsetY = 100,
Width = 100,
Height = 100,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation(){ Content = "Path Element"}
},
Shape = new DiagramShape()
{
Type = Syncfusion.Blazor.Diagrams.Shapes.Path,
Data =
"M540.3643,137.9336L546.7973,159.7016L570.3633,159.7296L550.7723,171.9366L55
8.9053,194.9966L540.3643," +
"179.4996L521.8223,194.9966L529.9553,171.9366L510.3633,159.7296L533.9313,159
.7016L540.3643,137.9336z"
}
}
}
```

```

}
};
NodeCollection.Add(node2);
DiagramConnector connector1 = new DiagramConnector()
{
    Id = "connector1",
    SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 300 },
    TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 400 },
    Type = Segments.Straight
};
ConnectorCollection.Add(connector1);
string[] objects = new string[] { "node1" };
// initialize Layers
Layers.Add(new DiagramLayer() { Id = "Layer1", Visible = true, Objects =
objects });
}
}

```

### Lock

The layer's [Lock](#) property is used to prevent or allow changes to the elements dimension and position.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" Layers="@Layers">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>() { };
public ObservableCollection<DiagramLayer> Layers = new
ObservableCollection<DiagramLayer>() { };
protected override void OnInitialized()
{
    // A node is created and stored in nodes array.
    DiagramNode node1 = new DiagramNode()
    {
        Id = "node1",
        OffsetX = 100,
        OffsetY = 100,
        Width = 100,
        Height = 100,
        Annotations = new ObservableCollection<DiagramNodeAnnotation>()
        {
            new DiagramNodeAnnotation() { Content = "Default Shape" }
        }
    };
    NodeCollection.Add(node1);
    DiagramNode node2 = new DiagramNode()
    {
        Id = "node2",
        OffsetX = 300,
        OffsetY = 100,

```

```

Width = 100,
Height = 100,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    new DiagramNodeAnnotation() { Content = "Path Element" }
},
Shape = new DiagramShape()
{
    Type = Syncfusion.Blazor.Diagrams.Shapes.Path,
    Data =
        "M540.3643,137.9336L546.7973,159.7016L570.3633,159.7296L550.7723,171.9366L558.9053,194.9966L540.3643," +
        "179.4996L521.8223,194.9966L529.9553,171.9366L510.3633,159.7296L533.9313,159.7016L540.3643,137.9336z"
    }
};
NodeCollection.Add(node2);
DiagramConnector connector1 = new DiagramConnector()
{
    Id = "connector1",
    SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 300 },
    TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 400 },
    Type = Segments.Straight
};
ConnectorCollection.Add(connector1);
string[] objects = new string[] { "node1" };
string[] objects1 = new string[] { "node2" };
Layers.Add(new DiagramLayer() { Id = "Layer1", Visible = true, Objects = objects, Lock = true });
Layers.Add(new DiagramLayer() { Id = "layer2", Visible = true, Objects = objects1, Lock = false });
}
}

```

## Objects

The layer's [Objects](#) property defines the diagram elements to the layer.

## ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" Layers="@Layers">
</SfDiagram>
@code{
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>() { };
    public ObservableCollection<DiagramLayer> Layers = new
    ObservableCollection<DiagramLayer>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes array.
        DiagramNode node1 = new DiagramNode()
        {

```

```

Id = "node1",
OffsetX = 100,
OffsetY = 100,
Width = 100,
Height = 100,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    new DiagramNodeAnnotation() { Content = "Default Shape" }
};
NodeCollection.Add(node1);
DiagramNode node2 = new DiagramNode()
{
    Id = "node2",
    OffsetX = 300,
    OffsetY = 100,
    Width = 100,
    Height = 100,
    Annotations = new ObservableCollection<DiagramNodeAnnotation>()
    {
        new DiagramNodeAnnotation() { Content = "Path Element" }
    },
    Shape = new DiagramShape()
    {
        Type = Syncfusion.Blazor.Diagrams.Shapes.Path,
        Data =
            "M540.3643,137.9336L546.7973,159.7016L570.3633,159.7296L550.7723,171.9366L558.9053,194.9966L540.3643," +
            "179.4996L521.8223,194.9966L529.9553,171.9366L510.3633,159.7296L533.9313,159.7016L540.3643,137.9336z"
    }
};
NodeCollection.Add(node2);
DiagramConnector connector1 = new DiagramConnector()
{
    Id = "connector1",
    SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 300 },
    TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 400 },
    Type = Segments.Straight
};
ConnectorCollection.Add(connector1);
string[] objects = new string[] { "node1", "node2" };
string[] objects1 = new string[] { "node2" };
Layers.Add(new DiagramLayer() { Id = "Layer1", Visible = true, Objects = objects });
Layers.Add(new DiagramLayer() { Id = "layer2", Visible = true, Objects = objects1 });
}
}

```

### AddInfo

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

The following code illustrates how to add additional information to the layers.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" Layers="@Layers">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>() { };
public ObservableCollection<DiagramLayer> Layers = new
ObservableCollection<DiagramLayer>() { };
public class AdditionalInfo
{
public string Description { get; set; }
};
protected override void OnInitialized()
{
// A node is created and stored in nodes array.
DiagramNode node1 = new DiagramNode()
{
Id = "node1",
OffsetX = 100,
OffsetY = 100,
Width = 100,
Height = 100,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation(){ Content = "Default Shape"}
}
};
NodeCollection.Add(node1);
DiagramNode node2 = new DiagramNode()
{
Id = "node2",
OffsetX = 300,
OffsetY = 100,
Width = 100,
Height = 100,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation(){ Content = "Path Element"}
},
Shape = new DiagramShape()
{
Type = Syncfusion.Blazor.Diagrams.Shapes.Path,
Data =
"M540.3643,137.9336L546.7973,159.7016L570.3633,159.7296L550.7723,171.9366L55
8.9053,194.9966L540.3643," +
"179.4996L521.8223,194.9966L529.9553,171.9366L510.3633,159.7296L533.9313,159
.7016L540.3643,137.9336z"
}
};
NodeCollection.Add(node2);
DiagramConnector connector1 = new DiagramConnector()
{
Id = "connector1",

```



```

SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 300 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 400 },
Type = Segments.Straight
};
ConnectorCollection.Add(connector1);
string[] objects = new string[] { "node1", "node2" };
string[] objects1 = new string[] { "node2" };
Layers.Add(new DiagramLayer() { Id = "Layer1", Visible = true, Objects =
objects, AddInfo = new AdditionalInfo() { Description="Layer1" } });
Layers.Add(new DiagramLayer() { Id = "layer2", Visible = true, Objects =
objects1});
}
}

```

#### Add layer at runtime

Layers can be added at runtime by using the [AddLayer](#) public method. The layer's [ID](#) property defines the ID of the layer, and its further used to find the layer at runtime and do any customization.

The following code illustrates how to add a layer.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
<input type="button" value="Addlayer" @onclick="@addLayer" />
<SfDiagram Height="600px" @ref="@diagram">
</SfDiagram>
@code{
SfDiagram diagram;
DiagramLayer layer = new DiagramLayer()
{
    Id = "newlayer",
    Visible = true,
    Lock = false,
    Objects = new string[] { },
    ZIndex = -1,
    AddInfo = { }
};
List<DiagramConnector> connectors = new List<DiagramConnector>()
{
    new DiagramConnector()
    {
        Id = "connector2",
        SourcePoint = new ConnectorSourcePoint() { X = 200, Y = 100 },
        TargetPoint = new ConnectorTargetPoint() { X = 500, Y = 100 },
        Type = Segments.Straight
    }
};
// add the layers to the existing diagram layer collection
public void addLayer()
{
    diagram.AddLayer(layer, connectors);
}
}

```

### *Remove layer at runtime*

Layers can be removed at runtime by using the [RemoveLayer](#) public method.

The following code illustrates how to remove a layer.

#### **CSHARP**

```
SfDiagram diagram;  
// remove the diagram layers  
diagram.RemoveLayer("Layer1");
```

### *MoveObjects*

Objects of the layers can be moved by using the [MoveObjects](#) public method.

The following code illustrates how to move objects from one layer to another layer from the diagram.

#### **CSHARP**

```
SfDiagram diagram;  
// move the objects of diagram layers  
string[] objects = new string[] { "node2" };  
diagram.MoveObjects(objects, "Layer1");
```

### *BringLayerForward*

Layers can be moved forward at runtime by using the [BringLayerForward](#) public method.

The following code illustrates how to bring forward to layer.

#### **CSHARP**

```
SfDiagram diagram;  
// move the layer forward  
diagram.BringLayerForward("Layer1");
```

### *SendLayerBackward*

Layers can be moved backward at runtime by using the [SendLayerBackward](#) public method.

The following code illustrates how to send backward to layer.

#### **CSHARP**

```
SfDiagram diagram;  
// move the layer backward  
diagram.SendLayerBackward("Layer1");
```

### *CloneLayer*

Layers can be cloned with its object by using the [CloneLayer](#) public method.

The following code illustrates how to bring forward to layer.

#### **CSHARP**

```
SfDiagram diagram;  
// clone a layer with its objec  
diagram.CloneLayer("Layer1");
```

### [GetActiveLayer](#)

To get the active layers back in diagram, use the [GetActiveLayer](#) public method.

The following code illustrates how to bring forward to layer.

#### **CSHARP**

```
SfDiagram diagram;  
// gets the active layer back  
diagram.GetActiveLayer();
```

### [SetActiveLayer](#)

Set the active layer by using the [SetActiveLayer](#) public method.

The following code illustrates how to bring forward to layer.

#### **CSHARP**

```
SfDiagram diagram;  
// set the active layer  
//@param layerName defines the name of the layer which is to be active layer  
diagram.SetActiveLayer("Layer1");
```

## Context Menu in Blazor Diagram Component

<!-- markdownlint-disable MD010 -->

In graphical user interface (GUI), a context menu is a type of menu that appears when you perform right-click operation. You can create a nested level of context menu items.

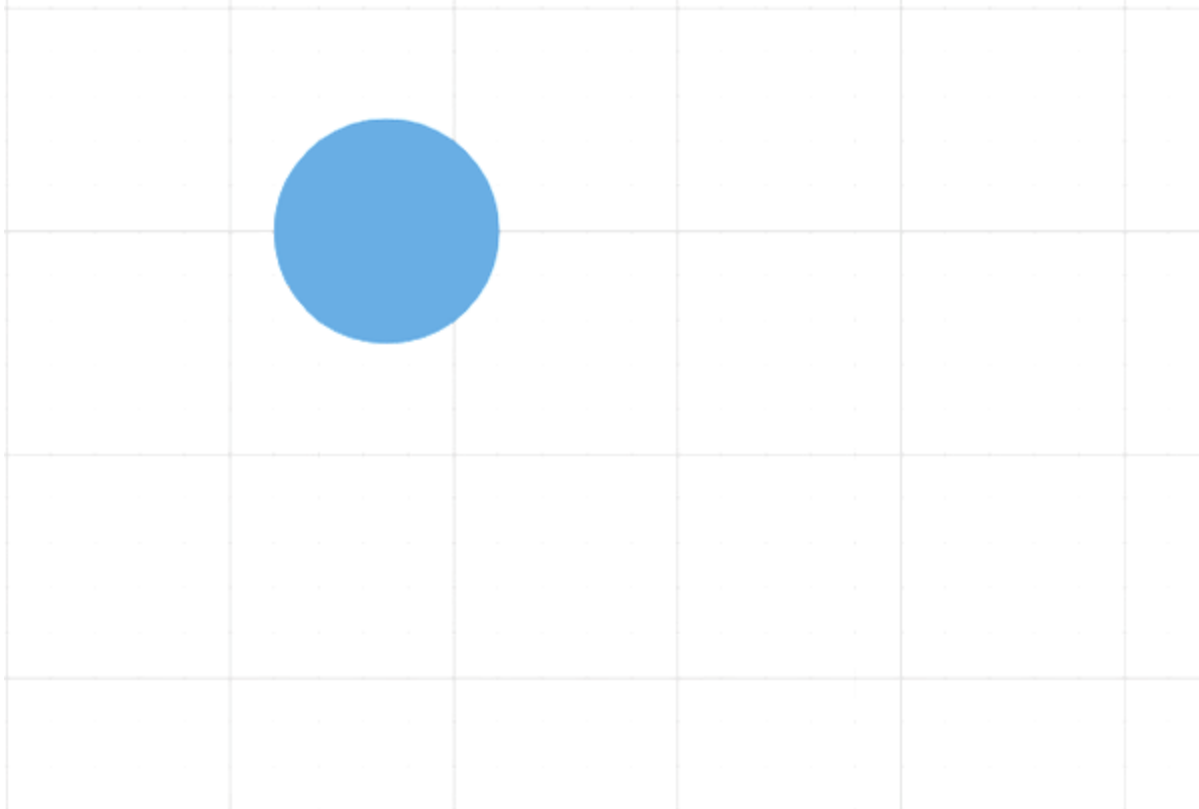
Diagram provides some in-built context menu items and allows to define custom menu items through the [ContextMenuSettings](#) property.

### Default context menu

The [Show](#) property helps you to enable or disable the context menu. Diagram provides some default context menu items such as copy, cut, select all, order, cut undo, redo, and group options. The following code shows how to enable the default context menu items.

#### **ASPX-CS**

```
<SfDiagram id="diagram" Height="600px">  
// Define context menu  
<DiagramContextMenuSettings Show="true">  
</DiagramContextMenuSettings>  
</SfDiagram>
```



The following code shows how to disable the default context menu items.

#### ASPX-CS

```
<SfDiagram id="diagram" Height="600px">  
  // Define context menu  
  <DiagramContextMenuSettings Show="false">  
  </DiagramContextMenuSettings>  
</SfDiagram>
```

#### Custom context menu

Custom context menu provides an option to add the new custom items to the context menu.

- Apart from the default context menu items, define some additional context menu items. Those additional items have to be defined and added to the [Items](#) property of the context menu.
- You can set the text and ID for the context menu item using the context menu **Text** and **Id** properties respectively.
- You can set an image for the context menu item using the context menu url property.
- The **IconCss** property defines the class or multiple classes separated by a space for the menu item that is used to include an icon. Menu item can include the font icon and sprite image.
- The **Target** property used to set the target to show the menu item.
- The **Separator** property defines the horizontal lines that are used to separate the menu items. You cannot select the separators. You can enable separators to group the menu items using the separator property.

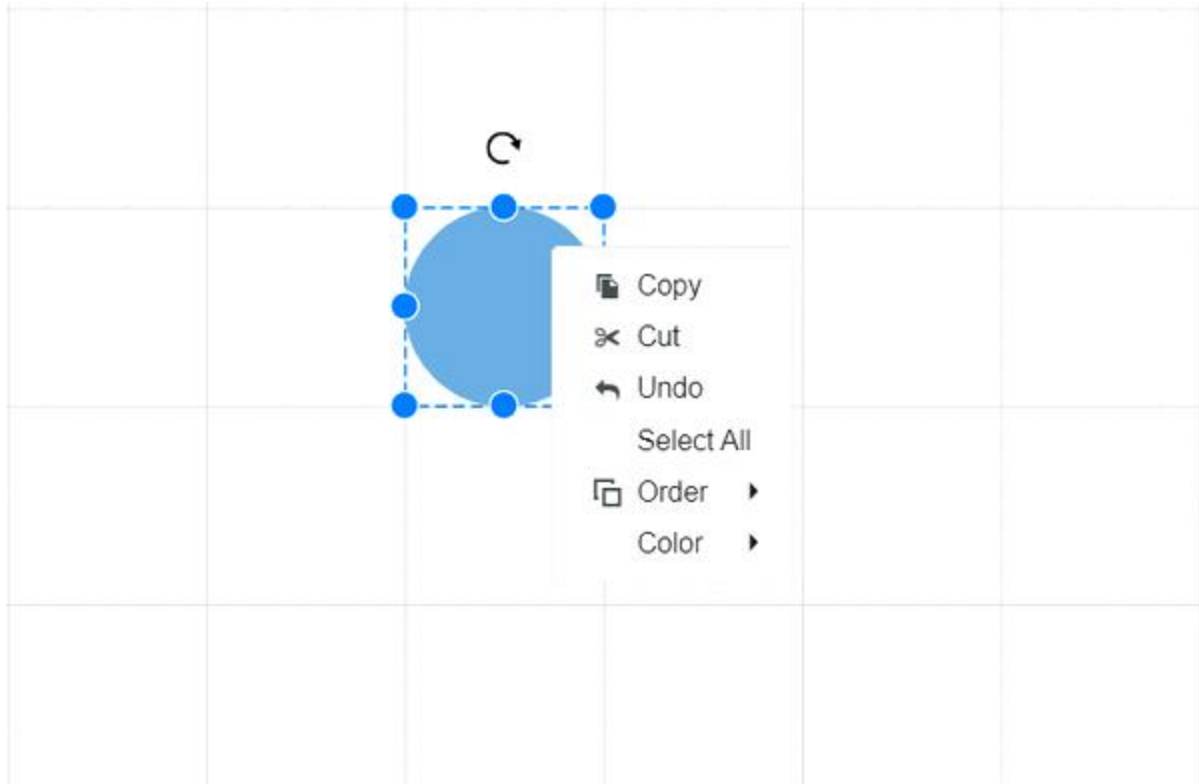
*Custom context menu along with default context menu*

The following code example shows how to add custom context menu items along with the default context menu. set the [ShowCustomMenuOnly](#) property to false to render both custom context menu and default context menu.

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px"
Nodes="@NodeCollection"
Connectors="@ConnectorCollection"
Constraints="@diagramConstraints">
// Defines context menu and set the ShowCustomMenuOnly to false to render
the custom context menu along with the default context menu
<DiagramContextMenuSettings Show="true" Items="@contextMenuItemModels"
ShowCustomMenuOnly="false">
</DiagramContextMenuSettings>
</SfDiagram>
@code
{
//Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
public DiagramConstraints diagramConstraints = DiagramConstraints.Default;
protected override void OnInitialized()
{
//Create a node
DiagramNode node1 = new DiagramNode()
{
OffsetX = 100,
OffsetY = 100,
Height = 100,
Width = 100,
Id = "node1",
Shape = new DiagramShape()
{
Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
BasicShape = BasicShapes.Ellipse
},
Style = new NodeShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "#6BA5D7",
},
};
//Add node into node's collection
NodeCollection.Add(node1);
//Create a node
DiagramNode node2 = new DiagramNode()
{
OffsetX = 300,
OffsetY = 100,
Height = 100,
```

```
Width = 100,
Id = "node2",
Shape = new DiagramShape()
{
    Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
    BasicShape = BasicShapes.Ellipse
},
Style = new NodeShapeStyle()
{
    Fill = "#6BA5D7",
    StrokeColor = "#6BA5D7",
},
};
//Add node into node's collection
NodeCollection.Add(node2);
DiagramConnector diagramConnector1 = new DiagramConnector()
{
    SourceID = "node1",
    TargetID = "node2",
    TargetDecorator = new ConnectorTargetDecorator()
    {
        Shape = DecoratorShapes.Arrow,
        Style = new DecoratorShapeStyle()
        {
            StrokeColor = "black",
            Fill = "black",
            StrokeWidth = 1
        },
    },
    Style = new ConnectorShapeStyle() { StrokeColor = "black", StrokeWidth = 1 },
    Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector1);
}
// Add the custom context menu items
List<ContextMenuItemModel> contextMenuItemModels = new
List<ContextMenuItemModel>()
{
    new ContextMenuItemModel()
    {
        Text = "color",
        Id = "Save",
        Target = ".e-elementcontent",
        // Add the nested custom context menu items
        Items = new List<ContextMenuItemModel>()
        {
            new ContextMenuItemModel() { Text = "Red", Id = "Red", },
            new ContextMenuItemModel() { Text = "Yellow", Id = "Yellow", },
            new ContextMenuItemModel() { Text = "Green", Id = "Green", }
        },
    },
};
}
```



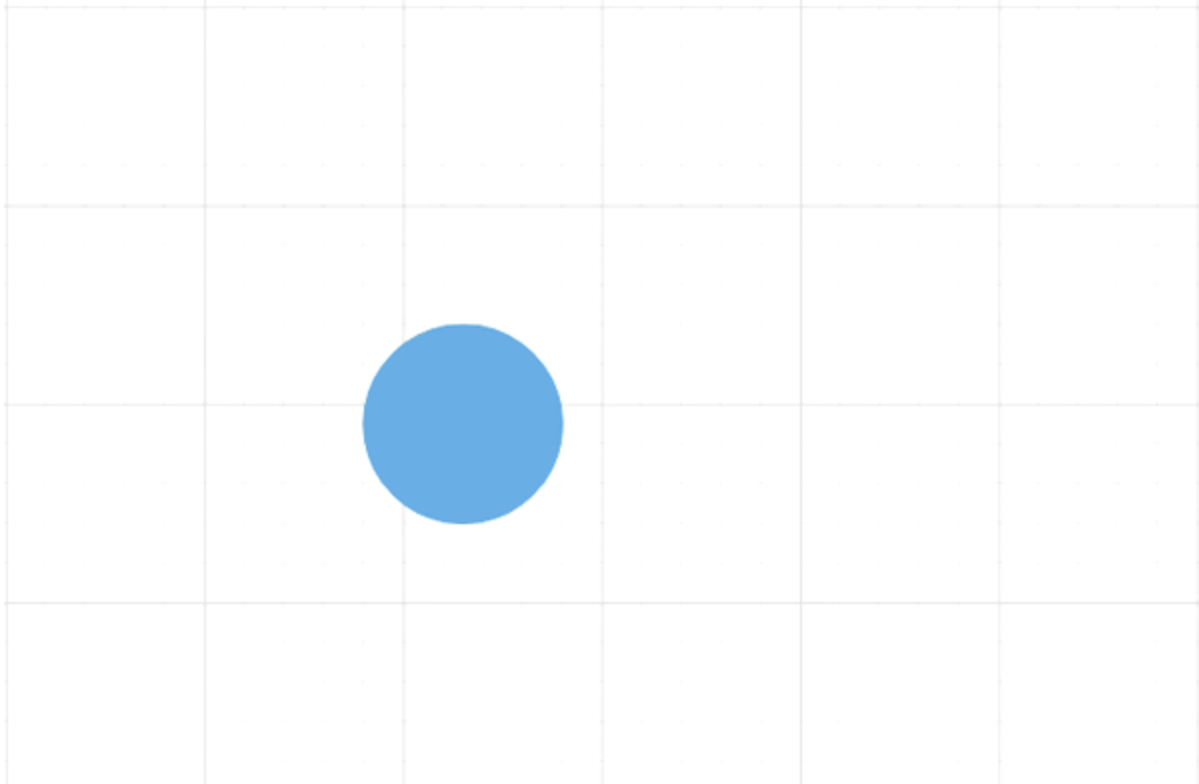
#### *Custom context menu alone*

To display the custom context menu items alone, set the [ShowCustomMenuOnly](#) property to true.

The following code example shows how to add custom context menu items alone.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px">
// Defines context menu and set the ShowCustomMenuOnly to true to render the
custom context menu alone
<DiagramContextMenuSettings Show="true" ShowCustomMenuOnly="true">
</DiagramContextMenuSettings>
</SfDiagram>
```



### Events

The Diagram control provides event support for context menu that triggers when rendering the context menu and triggers when clicking the items of the context menu.

#### *OnContextMenuOpen*

The Diagram control triggers the event [OnContextMenuOpen](#) when performing right click on the diagram or the diagram object.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px">
// Defines OnContextMenuOpen event
<DiagramEvents OnContextMenuOpen="@OnContextMenuOpen"></DiagramEvents>
// Defines context menu
<DiagramContextMenuSettings Show="true" ShowCustomMenuOnly="false">
</DiagramContextMenuSettings>
</SfDiagram>
@code
{
public void OnContextMenuOpen(DiagramBeforeMenuOpenEventArgs arg)
{
//Action to be performed
}
}
```

#### *ContextMenuItemClick*

The Diagram control triggers the event [ContextMenuItemClick](#) when clicking the context menu item.



**ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
<SfDiagram Height="600px">
// Defines the ContextMenuItemClicked event
<DiagramEvents
ContextMenuItemClicked="@ContextMenuItemClicked"></DiagramEvents>
// Defines the context menu
<DiagramContextMenuSettings Show="true" ShowCustomMenuOnly="false">
</DiagramContextMenuSettings>
</SfDiagram>
@code
{
public void ContextMenuItemClicked(DiagramMenuEventArgs arg)
{
//Action to be performed
}
}

```

The following code example shows how to add separate custom context menu items for node and connector. In the following code, the node color context menu item only render for node and the connector color context menu item only render for connector.

**ASPX-CS**

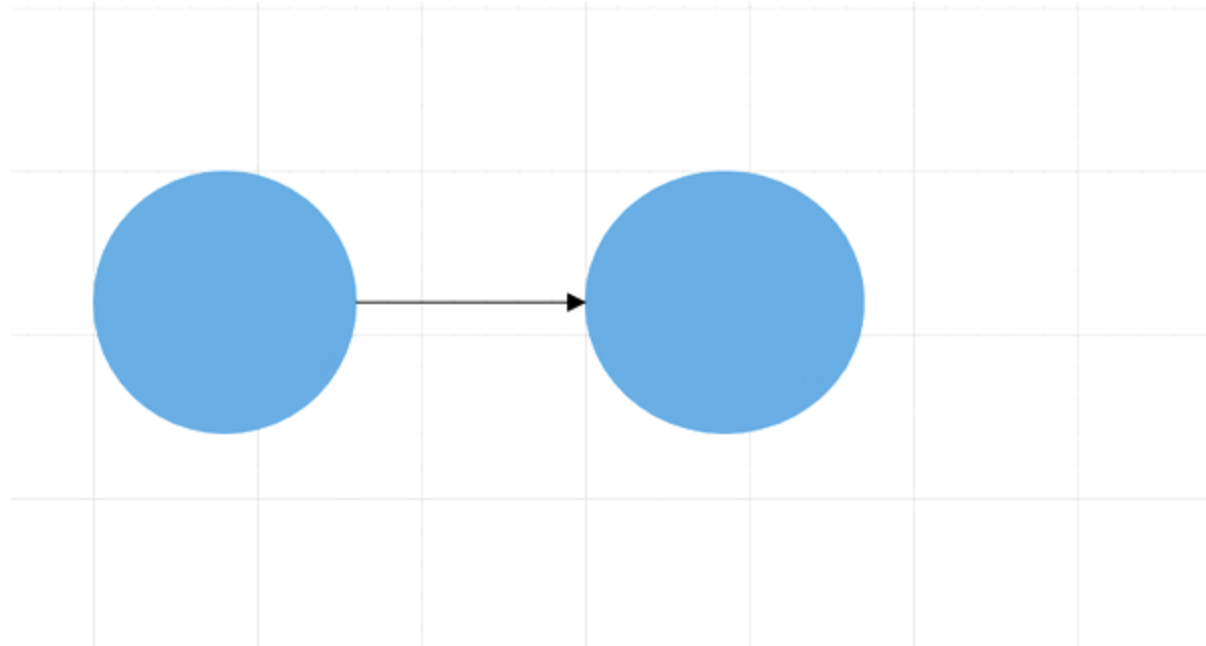
```

<SfDiagram @ref="@diagram" Height="600px"
Nodes="@NodeCollection"
Connectors="@ConnectorCollection"
Constraints="@diagramConstraints">
// Defines the ContextMenuItemClicked event
<DiagramEvents OnContextMenuOpen="@OnContextMenuOpen"></DiagramEvents>
// Defines the context menu
<DiagramContextMenuSettings Show="true" Items="@contextMenuItemModels"
ShowCustomMenuOnly="true">
</DiagramContextMenuSettings>
</SfDiagram>
@code
{
//Reference to diagram
SfDiagram diagram;
//Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
public DiagramConstraints diagramConstraints = DiagramConstraints.Default;
protected override void OnInitialized()
{
//Create a node
DiagramNode node1 = new DiagramNode()
{
OffsetX = 100,
OffsetY = 100,
Height = 100,
Width = 100,
}
}
}

```

```
Id = "node1",
Shape = new DiagramShape()
{
    Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
    BasicShape = BasicShapes.Ellipse
},
Style = new NodeShapeStyle()
{
    Fill = "#6BA5D7",
    StrokeColor = "#6BA5D7",
},
};
//Add node into node's collection
NodeCollection.Add(node1);
//Create a node
DiagramNode node2 = new DiagramNode()
{
    OffsetX = 200,
    OffsetY = 100,
    Height = 100,
    Width = 100,
    Id = "node2",
    Shape = new DiagramShape()
    {
        Type = Syncfusion.Blazor.Diagrams.Shapes.Basic,
        BasicShape = BasicShapes.Ellipse
    },
    Style = new NodeShapeStyle()
    {
        Fill = "#6BA5D7",
        StrokeColor = "#6BA5D7",
    },
};
//Add node into node's collection
NodeCollection.Add(node2);
DiagramConnector diagramConnector1 = new DiagramConnector()
{
    SourceID = "node1",
    TargetID = "node2",
    TargetDecorator = new ConnectorTargetDecorator()
    {
        Shape = DecoratorShapes.Arrow,
        Style = new DecoratorShapeStyle()
        {
            StrokeColor = "black",
            Fill = "black",
            StrokeWidth = 1
        },
    },
    Style = new ConnectorShapeStyle() { StrokeColor = "black", StrokeWidth = 1 },
    Type = Segments.Orthogonal,
};
ConnectorCollection.Add(diagramConnector1);
List<ContextMenuItemModel> contextMenuItemModels = new
List<ContextMenuItemModel>()
```

```
{
new ContextMenuItemModel()
{
Text ="Node Color",
Id="Node",
Target = ".e-elementcontent",
Items = new List<ContextMenuItemModel>()
{
new ContextMenuItemModel() { Text ="Red", Id="Red", },
new ContextMenuItemModel() { Text ="Yellow", Id="Yellow", },
new ContextMenuItemModel() { Text ="Green", Id="Green", }
}
},
new ContextMenuItemModel()
{
Text ="Connector Color",
Id="Connector",
Target = ".e-elementcontent",
Items = new List<ContextMenuItemModel>()
{
new ContextMenuItemModel() { Text ="Red", Id="black", },
new ContextMenuItemModel() { Text ="Yellow", Id="blue", },
new ContextMenuItemModel() { Text ="Green", Id="brown", }
}
},
};
public void OnContextMenuOpen(DiagramBeforeMenuOpenEventArgs arg)
{
if (diagram.SelectedItems.Nodes.Count > 0)
{
arg.HiddenItems.Add("Connector");
}
if (diagram.SelectedItems.Connectors.Count > 0)
{
arg.HiddenItems.Add("Node");
}
}
}
```



### Ruler in Blazor Diagram Component

The Ruler provides a horizontal and vertical guide for measuring in the Diagram control. The Ruler can be used to measure the diagram objects, indicate positions, and align diagram elements. This is especially useful in creating scale models.

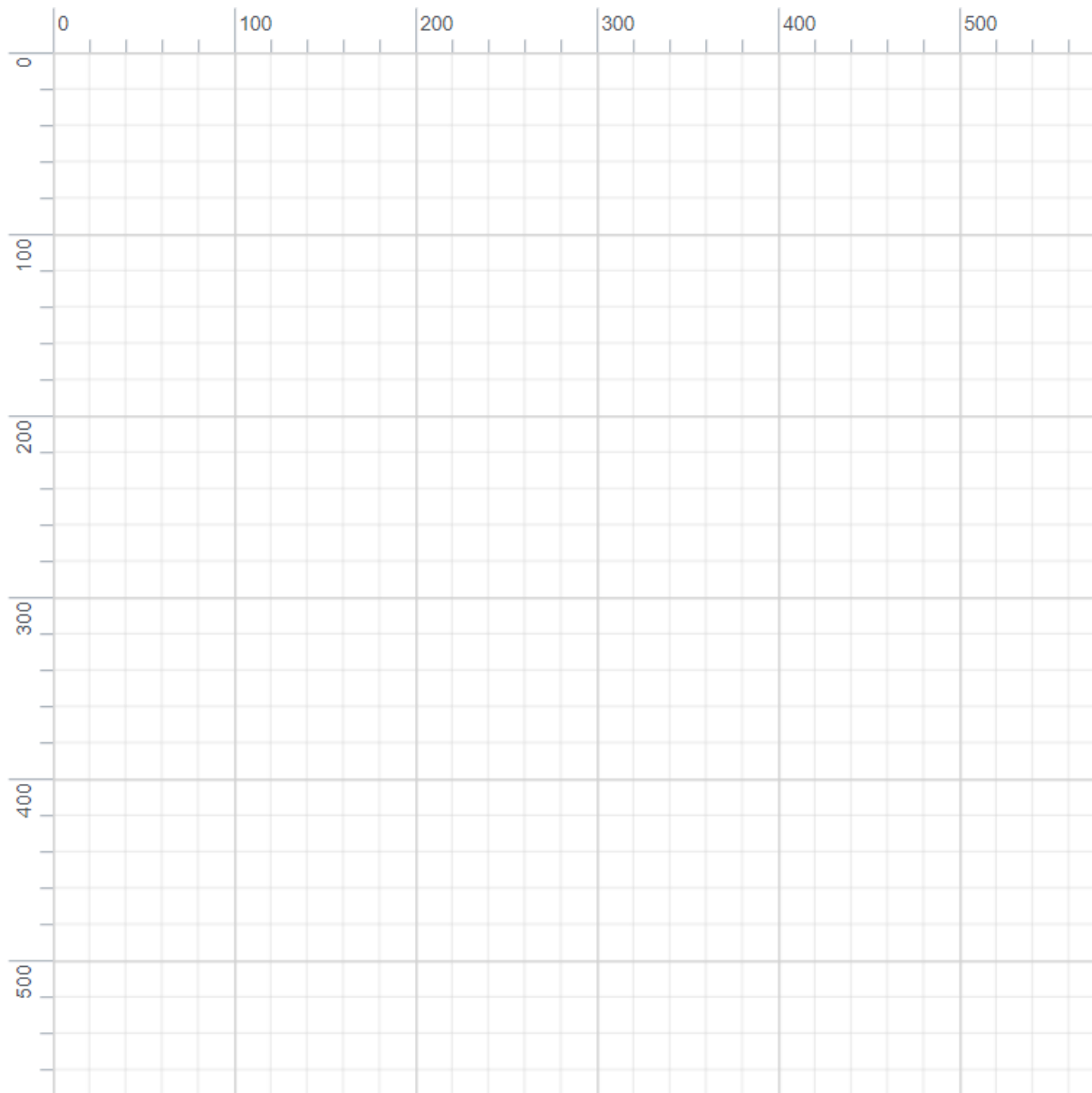
#### Adding Rulers to the Diagram

- The [DiagramRulerSettings](#) property is used to control the visibility and appearance of the ruler in the diagram.
- The RulerSettings [ShowRulers](#) property is used to show or hide the rulers in the diagram.
- The RulerSettings [HorizontalRuler](#) and [VerticalRuler](#) properties are used to customize the rulers appearance in the diagram.

The following code shows how to add a ruler to the diagram.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="600px" Height="600px">
  <DiagramRulerSettings ShowRulers="true">
  </DiagramRulerSettings>
</SfDiagram>
```



### Customizing the Ruler

By default, the ruler segments are arranged based on pixel values.

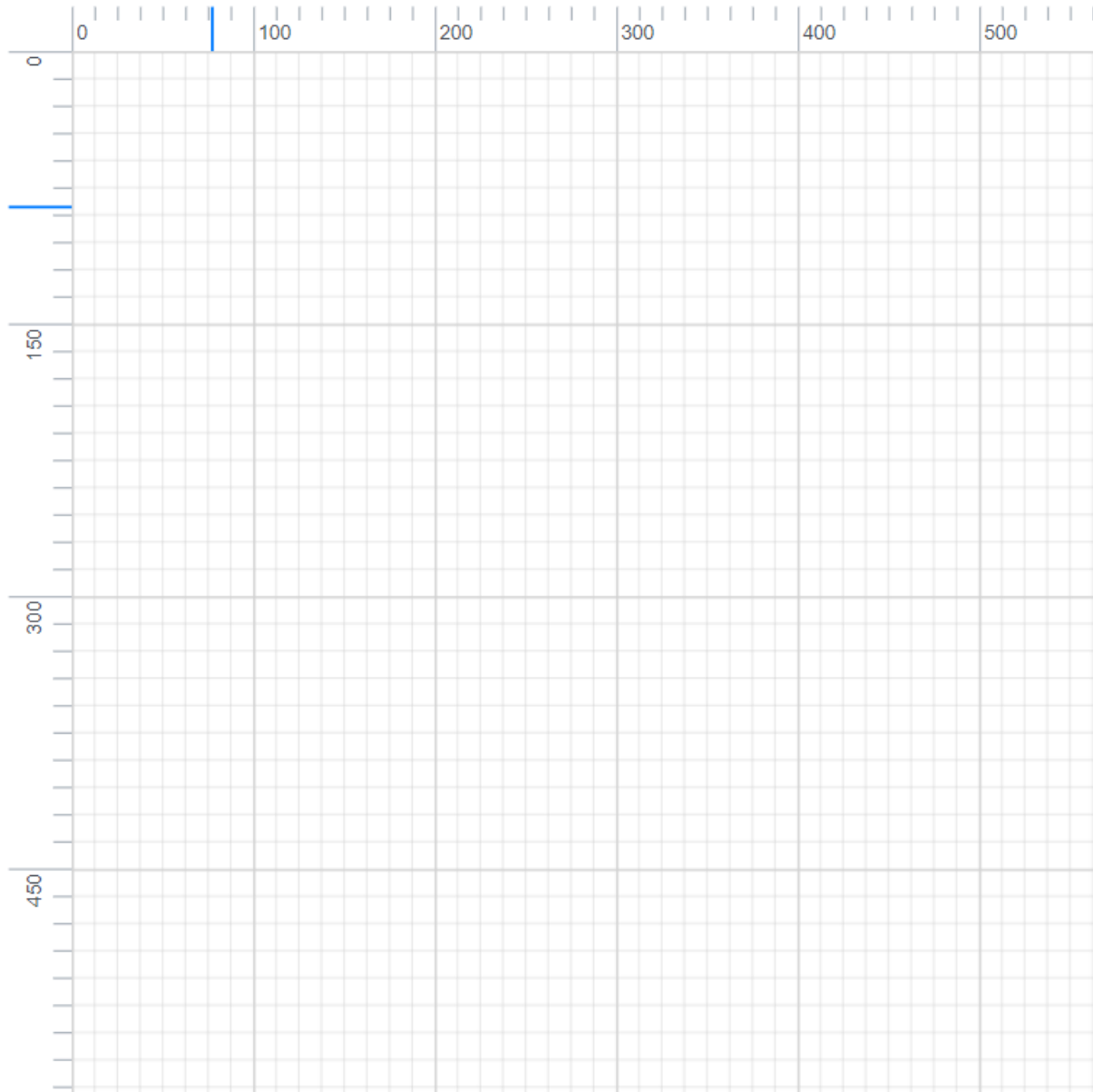
- The HorizontalRuler's [Interval](#) property allows you to define the interval between ruler segments and the [SegmentWidth](#) property allows you to define the segment width of the ruler. Similarly, you can use the VerticalRuler's [Interval](#) and [SegmentWidth](#) properties are used to define the interval and segment width of the vertical ruler.
- The HorizontalRuler's [TickAlignment](#) property is used to align the ruler tick either left or right side of the ruler. The VerticalRuler's [TickAlignment](#) property is used to align the ruler tick either top or bottom side of the ruler.
- The HorizontalRuler's [MarkerColor](#) and VerticalRuler's [MarkerColor](#) properties are used to define the ruler marker color and marker will be shown when performing the interaction in the diagram.

The following code shows how the diagram ruler can be customized.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<SfDiagram Width="600px" Height="600px">
  <DiagramRulerSettings ShowRulers="true">
    <HorizontalRuler Interval="8" SegmentWidth="100" Thickness="25"
      TickAlignment="@verticalTickAlignment"></HorizontalRuler>
    <VerticalRuler Interval="10" SegmentWidth="150" Thickness="35"
      TickAlignment="@horizontalTickAlignment"></VerticalRuler>
  </DiagramRulerSettings>
</SfDiagram>
@code{
  TickAlignment verticalTickAlignment = TickAlignment.LeftOrTop;
  TickAlignment horizontalTickAlignment = TickAlignment.RightOrBottom;
}
```

The MarkerColor property can be customized using the [Marker](#) CSS style.



### User Handles in Blazor Diagram Component

User handles are customizable handles that can be used to perform custom actions and default clipboard actions.

#### Initialization an userhandle

The user handle can enables for the selected nodes/connectors by setting a [SelectorConstraints](#) as **UserHandle** and then use the [DiagramUserHandle](#) class to create an object for the user handles. The following code example used to enable and create an user handles for the diagram nodes/connectors.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection"
SelectedItems="@SelectedModel">
```

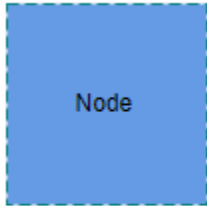
```

</SfDiagram>
@code{
// Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
// Defines diagram's SelectedItems
public Syncfusion.Blazor.Diagrams.DiagramSelectedItems SelectedModel { get;
set; }
ObservableCollection<DiagramUserHandle> UserHandles { get; set; }
protected override void OnInitialized()
{
//Creating the userhandle for cloning the objects
DiagramUserHandle cloneHandle = new DiagramUserHandle()
{
//Name of the user handle
Name = "clone",
//Set pathdata for userhandle
PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z",
//Set visibility for the user handle
Visible = true,
//Set the position for the user handle
Offset = 0,
//Set side based on the given offset
Side = Side.Bottom,
//set margin for the user handle
Margin = new DiagramUserHandleMargin() { Top = 0, Bottom = 0, Left = 0,
Right = 0 }
};
//Add user handle to the collection...
UserHandles = new ObservableCollection<DiagramUserHandle>()
{
cloneHandle
};
SelectedModel = new Syncfusion.Blazor.Diagrams.DiagramSelectedItems()
{
//Enable userhandle for selected items...
Constraints = SelectorConstraints.UserHandle,
UserHandles = this.UserHandles
};
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode diagramNode = new DiagramNode()
{
Id = "node1",
OffsetX = 100,
OffsetY = 100,
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#659be5", StrokeColor = "none" },
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Node"
}
}
}
}

```



```
};
NodeCollection.Add(diagramNode);
}
}
```



### Customization

If set `false` to the [DisableConnectors](#) property in userhandle, the userhandle prevents to rendering for the connectors. The following code example is used to show userhandle for the nodes alone.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram @ref="@Diagram" Height="600px"
Nodes="@NodeCollection"
Connectors="@ConnectorCollection"
SelectedItems="@SelectedItem">
</SfDiagram>
@code {
// Reference to diagram
SfDiagram Diagram;
// Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
// Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
// Defines diagram's SelectedItems
public Syncfusion.Blazor.Diagrams.DiagramSelectedItem SelectedModel { get;
set; }
ObservableCollection<DiagramUserHandle> UserHandles { get; set; }
protected override void OnInitialized()
{
//Creating the userhandle for cloning the objects
DiagramUserHandle cloneHandle = new DiagramUserHandle()
{
//Name of the user handle
Name = "clone",
//Set pathdata for userhandle
PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z",
//Set visibility for the user handle
Visible = true,
//Set the position for the user handle
```

```

Offset = 0,
//Set side based on the given offset
Side = Side.Bottom,
//Disable to render this userhandle for connectors
DisableConnectors = true,
//set margin for the user handle
Margin = new DiagramUserHandleMargin() { Top = 0, Bottom = 0, Left = 0,
Right = 0 }
};
//Add user handle to the collection...
UserHandles = new ObservableCollection<DiagramUserHandle>()
{
cloneHandle
};
SelectedModel = new Syncfusion.Blazor.Diagrams.DiagramSelectedItems()
{
//Enable userhandle for selected items...
Constraints = SelectorConstraints.UserHandle,
UserHandles = this.UserHandles
};
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode diagramNode = new DiagramNode()
{
Id = "node1",
OffsetX = 100,
OffsetY = 100,
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#659be5", StrokeColor = "none" },
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Node"
}
}
};
NodeCollection.Add(diagramNode);
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector diagramConnector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 300 }
};
ConnectorCollection.Add(diagramConnector);
}
}

```

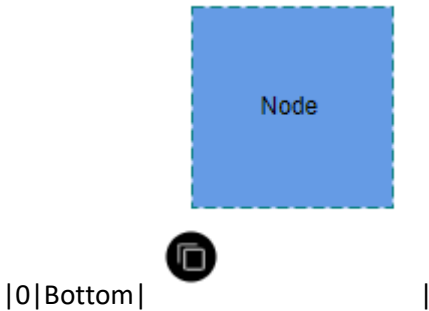
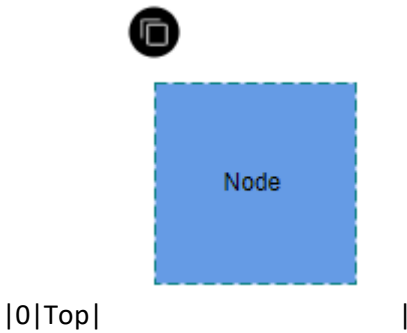
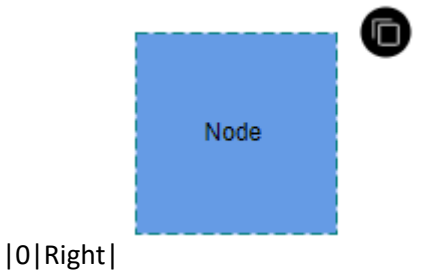
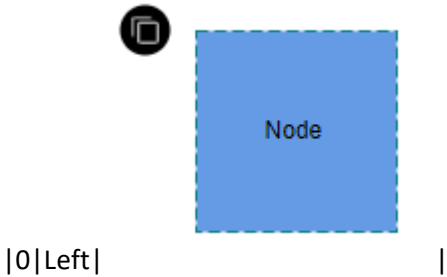
Also you can disable the [DisableNodes](#) property in userhandle, the userhandle prevent to rendering for the nodes.

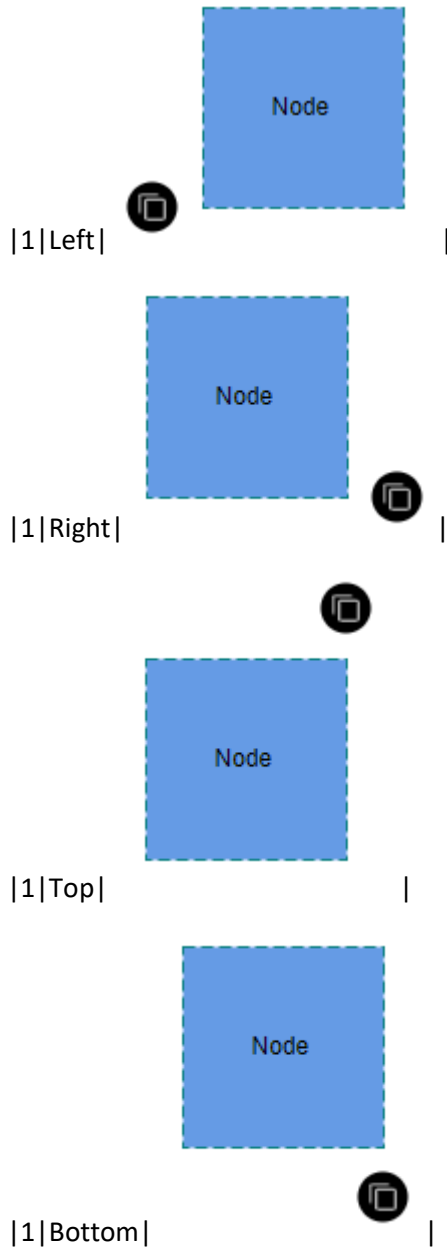
#### Position

The [Offset](#) property of userhandle is used to align the userhandles based on fractions. 0 represents Top-Left corner, 1 represents Bottom-Right corner, and 0.5 represents half of Width or Height. The [Side](#) property is used to set how the userhandle is aligned based on the given [Offset](#).

The following table shows all the possible alignments visually shows the userhandle positions.

Offset	Side	Output
-----	-----	-----





### Size

Diagram allows you set size for userhandles by using the [Size](#) property. The default value of the [Size](#) property is 25.

### Style

You can change the style of the userhandles with the specific properties of [PathColor](#), [BorderColor](#), [BackgroundColor](#) and [BorderWidth](#). The following code explains how to customize the appearance of the userhandles.

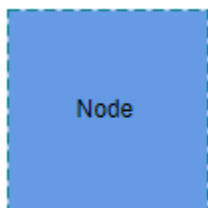
- The userhandle's [PathColor](#) property used to change the color of the given [PathData](#) of the userhandle.

- The userhandle [BorderColor](#), [BackgroundColor](#) properties are used to define the background color and border color of the userhandle and the [BorderWidth](#) property is used to define the border width of the userhandles.
- The [Visible](#) property of the userhandle enables or disables the visibility of userhandle.

The following code explains how to customize the appearance of the userhandle.

### CSHARP

```
//Creating the userhandle for cloning the objects
DiagramUserHandle cloneHandle = new DiagramUserHandle()
{
    //Name of the user handle
    Name = "clone",
    //Set pathdata for userhandle
    PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z",
    //Set visibility for the user handle
    Visible = true,
    //Set the position for the user handle
    Offset = 1,
    //Set side based on the given offset
    Side = Side.Bottom,
    //Disable to render this userhandle for connectors
    DisableConnectors = true,
    //set margin for the user handle
    Margin = new DiagramUserHandleMargin() { Top = 0, Bottom = 0, Left = 0,
    Right = 0 },
    //Set size of the user handle
    Size = 50,
    //Set pathcolor for given pathdata
    PathColor = "yellow",
    //Set Border color of the user handle
    BorderColor = "red",
    //Set Background Color of the user handle
    BackgroundColor = "green",
    //Set Border Width Color of the user handle
    BorderWidth = 3,
};
```



## Events

The Diagram control provides following list of events for the userhandle.

Event Name	Event Type	Description
-----	-----	-----
<a href="#">OnUserHandleMouseDown</a>	<a href="#">UserHandleEventArgs</a>	Triggered when the mouse pointer is over the userhandle and mouse button is down.
<a href="#">OnUserHandleMouseUp</a>	<a href="#">UserHandleEventArgs</a>	Triggered when the mouse pointer is over the userhandle and mouse button is released.
<a href="#">OnUserHandleMouseEnter</a>	<a href="#">UserHandleEventArgs</a>	Triggered when mouse enter into the userhandle.
<a href="#">OnUserHandleMouseLeave</a>	<a href="#">UserHandleEventArgs</a>	Triggered when mouse leaves the userhandle.

The following code explains how to use the `OnUserHandleMouseUp` event for an userhandle.

## ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram @ref="@Diagram" Height="600px"
Nodes="@NodeCollection"
Connectors="@ConnectorCollection"
SelectedItems="@SelectedItem">
<DiagramEvents OnUserHandleMouseUp="@OnUserHandleMouseUp"></DiagramEvents>
</SfDiagram>
@code {
// Reference to diagram
SfDiagram Diagram;
// Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
// Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
// Defines diagram's SelectedItems
public Syncfusion.Blazor.Diagrams.DiagramSelectedItem SelectedModel { get;
set; }
ObservableCollection<DiagramUserHandle> UserHandles { get; set; }
protected override void OnInitialized()
{
//Creating the userhandle for cloning the objects
DiagramUserHandle cloneHandle = new DiagramUserHandle()
{
//Name of the user handle
Name = "clone",
//Set pathdata for userhandle
PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z",
//Set visibility for the user handle
Visible = true,
//Set the position for the user handle
Offset = 0,
```

```
//Set side based on the given offset
Side = Side.Bottom,
//Disable to render this userhandle for connectors
DisableConnectors = true,
//set margin for the user handle
Margin = new DiagramUserHandleMargin() { Top = 0, Bottom = 0, Left = 0,
Right = 0 }
};
//Add user handle to the collection...
UserHandles = new ObservableCollection<DiagramUserHandle>()
{
cloneHandle
};
SelectedModel = new Syncfusion.Blazor.Diagrams.DiagramSelectedItems()
{
//Enable userhandle for selected items...
Constraints = SelectorConstraints.UserHandle,
UserHandles = this.UserHandles
};
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode diagramNode = new DiagramNode()
{
Id = "node1",
OffsetX = 100,
OffsetY = 100,
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#659be5", StrokeColor = "none" },
Annotations = new ObservableCollection<DiagramNodeAnnotation>() { new
DiagramNodeAnnotation() { Content = "Node" } }
};
NodeCollection.Add(diagramNode);
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector diagramConnector = new DiagramConnector()
{
SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 300, Y = 300 }
};
ConnectorCollection.Add(diagramConnector);
}
/// <summary>
/// mouse up event for the userhandles...
/// </summary>
public async Task OnUserHandleMouseUp(UserHandleEventArgs args)
{
if (Diagram.SelectedItems.Nodes.Count > 0)
{
await Diagram.Copy();
await Diagram.Paste();
}
}
}
```

### Fixed user handles

The fixed user handles are used to add some frequently used commands around the node and connector even without selecting it.

### Initialization an fixed user handles

To create the fixed user handles, define and add them to the collection of nodes and connectors property. The following code example used to create an fixed user handles for the nodes and connectors.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
DiagramNode node1 = new DiagramNode()
{
OffsetX = 250,
OffsetY = 250,
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
// A fixed user handle is created and stored in fixed user handle collection
of Node.
FixedUserHandles = new ObservableCollection<DiagramNodeFixedUserHandle>()
{
new DiagramNodeFixedUserHandle()
{
Id = "user1",
Height = 20,
Width = 20,
Visibility = true,
Padding = new IconPadding{Bottom=1,Left=1,Right=1,Top=1 },
Margin = new UserHandleMargin(){ Right = 20},
Offset = new UserHandleOffset() { X =0 , Y = 0 },
PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z"
},
},
};
NodeCollection.Add(node1);
}
```

### Customization the fixed user handle

- The id property of fixed user handle is used to define the unique identification of the fixed user handle and it is further used to add custom events to the fixed user handle.



- The fixed user handle can be positioned relative to the node and connector boundaries. It has offset, padding and cornerRadius settings. It is used to position and customize the fixed user handle.
- The **Padding** is used to leave the space that is inside the fixed user handle between the icon and border.
- The corner radius allows to create fixed user handles with rounded corners. The radius of the rounded corner is set with the **cornerRadius** property.

---

The PathData needs to be provided to render fixed user handle.

---

### Size

Diagram allows you set size for the fixed user handles by using the **width** and **height** property. The default value of the width and height property is 10.

### Style

- You can change the style of the fixed user handles with the specific properties of **borderColor**, **borderWidth**, and background color using the **handleStrokeColor**, **handleStrokeWidth**, and **fill** properties, and the icon **borderColor**, and **borderWidth** using the **iconStrokeColor** and **iconStrokeWidth**.
- The fixed user handle's **iconStrokeColor** and **iconStrokeWidth** property used to change the stroke color and stroke width of the given **pathData**.
- The fixed user handle **handleStrokeColor** and **fill** properties are used to define the background color and border color of the userhandle and the **handleStrokeWidth** property is used to define the border width of the fixed user handle.
- The **visible** property of the fixed user handle enables or disables the visibility of fixed user handle.

The following code explains how to customize the appearance of the fixed user handles.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
public ObservableCollection<DiagramConnector> ConnectorCollection { get;
set; }
public ObservableCollection<DiagramNode> NodeCollection { get; set; }
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>();
DiagramNode node = new DiagramNode()
{
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
```

```
// A fixed user handle is created and stored in fixed user handle collection
of Node.
FixedUserHandles = new ObservableCollection<DiagramNodeFixedUserHandle>()
{
    new DiagramNodeFixedUserHandle()
    {
        Id = "user1",
        Height = 20,
        Width = 20,
        Visibility = true,
        Padding = new IconPadding{Bottom=1,Left=1,Right=1,Top=1 },
        Margin = new UserHandleMargin(){ Right = 20},
        Offset = new UserHandleOffset() { X=0 , Y = 0 },
        PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z"
    },
};
NodeCollection.Add(node);
ConnectorCollection = new ObservableCollection<DiagramConnector>();
DiagramConnector connector = new DiagramConnector()
{
    SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 40 },
    TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 160 },
    Type = Segments.Orthogonal,
    Style = new ConnectorShapeStyle() { StrokeColor = "#6BA5D7" },
    // A fixed user handle is created and stored in fixed user handle collection
    of Connector.
    FixedUserHandles = new
    ObservableCollection<DiagramConnectorFixedUserHandle>()
    {
        new DiagramConnectorFixedUserHandle()
        {
            Id = "user1",
            Height = 25,
            Width = 25,
            Offset = 0.5,
            Visibility = true,
            Padding = new IconPadding{Bottom=1,Left=1,Right=1,Top=1 },
            Alignment = FixedUserHandleAlignment.After,
            Displacement = new ConnectorDisplacementPoint{X = 5, Y = 5 },
            PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z"
        }
    },
};
ConnectorCollection.Add(connector);
}
```

---

The fixed user handle id need to be unique.

---

Customizing the node fixed user handle

The node fixed user handle can be aligned relative to the node boundaries. It has `margin` and `offset` settings. It is quite useful to position the node fixed userhandle and used together and gives you more control over the node fixed user handle positioning.

Margin for the node fixed user handle

Margin is an absolute value used to add some blank space in any one of its four sides. The fixed user handle can be displaced with the `margin` property.

Offset for the node fixed user handle

The `offset` property of fixed user handle is used to align the user handle based on the `x` and `y` points. (0,0) represents the top or left corner and (1,1) represents the bottom or right corner.

The following table shows all the possible alignments visually shows the fixed user handle positions.

Offset	Margin	Output
-----	-----	-----
<div><div></div><div></div></div>		
(0,0)	Right = 20	
<div><div></div><div></div></div>		
(0.5,0)	Bottom = 20	
<div><div></div><div></div></div>		
(1,0)	Left = 20	



| (0,0.5) | Right = 20 |



| (0,1) | Left = 20 |



| (0,1) | Right = 20 |



| (0.5,1) | Top = 20 |



| (1,1) | Left = 20 |

The following code explains how to customize the node fixed user handle.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
DiagramNode node1 = new DiagramNode()
{
OffsetX = 250,
OffsetY = 250,
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
// A fixed user handle is created and stored in fixed user handle collection
of Node.
FixedUserHandles = new ObservableCollection<DiagramNodeFixedUserHandle>()
{
new DiagramNodeFixedUserHandle()
{
Id = "user1",
Height = 20,
Width = 20,
Visibility = true,
Padding = new IconPadding{Bottom=1,Left=1,Right=1,Top=1 },
Margin = new UserHandleMargin() { Right = 20},
Offset = new UserHandleOffset() { X = 0 , Y = 0 },
PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z"
},
},
};
NodeCollection.Add(node1);
}
}

```

### Customizing the connector fixed user handle

- The connector fixed user handle can be aligned relative to the connector boundaries. It has alignment, displacement and offset settings. It is useful to position the connector fixed user handle and used together and gives you more control over the connector fixed user handle positioning.
- The **offset** and **alignment** properties of fixed user handle allows you to align the connector fixed user handles to the segments.

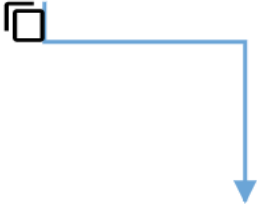
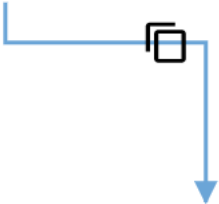

#### *Offset for the connector fixed user handle*

The **offset** property of connector fixed user handle is used to align the user handle based on fractions. 0 represents the connector source point, 1 represents the connector target point, and 0.5 represents the center point of the connector segment.

Alignment

The connector’s fixed user handle can be aligned over its segment path using the `alignment` property of fixed user handle.

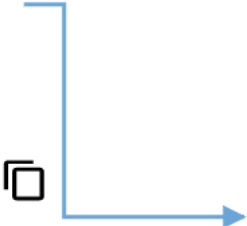
The following table shows all the possible alignments visually shows the fixed user handle positions.

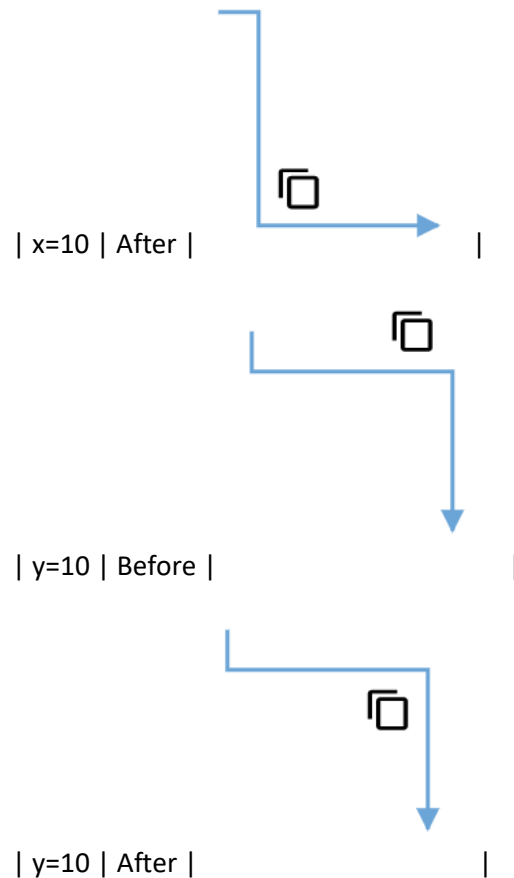
Offset	Alignment	Output
-----	-----	-----
0	Before	
0.5	Center	
1	After	

Displacement

The `displacement` property allows you to specify the space to be left from the connector segment based on the x and y value provided.

The following table shows all the possible alignments visually shows the fixed user handle positions.

Displacement	Alignment	Output
-----	-----	-----
x=10	Before	



Displacement will not be done if the alignment is set to be center.

The following code explains how to customize the connector fixed user handle.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    protected override void OnInitialized()
    {
        DiagramConnector diagramConnector = new DiagramConnector()
        {
            SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
            TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
            Type = Segments.Orthogonal,
            // A fixed user handle is created and stored in fixed user handle collection
            // of Connector.
            FixedUserHandles = new
            ObservableCollection<DiagramConnectorFixedUserHandle>()
            {
                new DiagramConnectorFixedUserHandle()
```

```

{
    Id = "user1",
    Height = 25,
    Width = 25,
    Offset = 0.5,
    Visibility = true,
    Padding = new IconPadding{Bottom=1,Left=1,Right=1,Top=1 },
    Alignment = FixedUserHandleAlignment.After,
    Displacement = new ConnectorDisplacementPoint{X = 5, Y = 5 },
    PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z"
}
},
};
ConnectorCollection.Add(diagramConnector);
}
}

```

### FixedUserHandle Event

The Diagram control provides following event for the fixed user handle.

Event Name	Event Type	Description
------------	------------	-------------

-----	-----	-----
-------	-------	-------

FixedUserHandleClick	FixedUserHandleClickEventArgs	Triggered when the mouse pointer is over the userhandle and mouse button is up.
----------------------	-------------------------------	---

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Height="600px" Nodes="@NodeCollection" @ref="diagram">
<DiagramEvents FixedUserHandleClick="Changed"></DiagramEvents>
</SfDiagram>
@code{
    SfDiagram diagram;
    public async void Changed(FixedUserHandleClickEventArgs args)
    {
        if (args.Element.Node.Id == "node1" &&
            args.FixedUserHandle.NodeFixedUserHandle.Id == "user1")
        {
            await diagram.Copy();
            await diagram.Paste();
        }
    }
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        DiagramNode node1 = new DiagramNode()
        {
            OffsetX = 250,
            OffsetY = 250,
            Id = "node1",

```



```

Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
// A fixed user handle is created and stored in fixed user handle collection
of Node.
FixedUserHandles = new ObservableCollection<DiagramNodeFixedUserHandle>()
{
    new DiagramNodeFixedUserHandle()
    {
        Id = "user1",
        Height = 20,
        Width = 20,
        Visibility = true,
        Padding = new IconPadding{Bottom=1,Left=1,Right=1,Top=1 },
        Margin = new UserHandleMargin(){ Right = 20},
        Offset = new UserHandleOffset() { X = 0 , Y = 0 },
        PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z"
    },
}
};
NodeCollection.Add(node1);
}
}

```

## CSS Structure in Blazor Diagram Component

### Customizing the connector end point handle

Use the following CSS to customize the connector end point handle.

#### SCSS

```

.e-diagram-endpoint-handle {
    fill: red;
    stroke: green;
}

```

### Customizing the connector end point handle when connected

Use the following CSS to customize the connector end point handle when connected.

#### SCSS

```

.e-diagram-endpoint-handle.e-connected {
    fill: red;
    stroke: green;
}

```

### Customizing the connector end point handle when disabled

Use the following CSS to customize the connector end point handle when disabled.

#### SCSS

```

.e-diagram-endpoint-handle.e-disabled {

```

```
fill: red;
opacity: 1;
stroke: green;
}
```

### Customizing the bezier connector handle

Use the following CSS to customize the bezier handle properties.

#### SCSS

```
.e-diagram-bezier-handle {
fill: red;
stroke: green;
}
```

### Customizing the bezier connector line

Use the following CSS to customize the bezier line properties.

#### SCSS

```
.e-diagram-bezier-line {
stroke: black;
}
```

### Customizing the resize handle

Use the following CSS to customize the resize handle.

#### SCSS

```
.e-diagram-resize-handle {
fill: white;
opacity: 1;
stroke: white;
}
```

### Customizing the selector pivot line

Use the following CSS to customize the line between the selector and rotate handle.

#### SCSS

```
.e-diagram-pivot-line {
stroke: red;
}
```

### Customizing the selector border

Use the following CSS to customize the selector border.

#### SCSS

```
.e-diagram-border {
stroke: red;
}
```

### Customizing the rotate handle

Use the following CSS to customize the rotate handle properties.

#### SCSS

```
.e-diagram-rotate-handle {  
  fill: red;  
  stroke: green;  
}
```

### Customizing the symbolpalette while hovering

Use the following CSS to customize the symbolpalette while hovering.

#### SCSS

```
.e-symbolpalette .e-symbol-hover:hover {  
  background: red;  
}
```

### Customizing the symbolpalette when selected

Use the following CSS to customize the symbolpalette when selected.

#### SCSS

```
.e-symbolpalette .e-symbol-selected {  
  background: white;  
}
```

### Customizing the ruler

Use the following CSS to customize the ruler properties.

#### SCSS

```
.e-diagram .e-ruler {  
  background-color: red;  
  font-size: 13px;  
}
```

### Customizing the ruler overlap

Use the following CSS to ruler overlap properties.

#### SCSS

```
.e-diagram .e-ruler-overlap {  
  background-color: red;  
}
```

### Customizing the text edit

Use the following CSS to customize the text edit properties.

#### SCSS

```
.e-diagram .e-diagram-text-edit {  
  background: white;  
}
```

```
border-color: red;
border-style: dashed;
border-width: 1px;
box-sizing: content-box;
color: black;
min-width: 50px;
}
```

### Customizing the text edit on selection

Use the following CSS to customize the text edit on selection properties.

#### SCSS

```
.e-diagram-text-edit::selection {
background: red;
color: green;
}
```

## Symbol Palette in Blazor Diagram Component

The **SymbolPalette** displays a collection of palettes. The palette shows a set of nodes and connectors. It allows to drag and drop the nodes and connectors into the diagram.

### Create symbol palette

The **Width** and **Height** properties of the symbol palette allows to define the size of the symbol palette.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@* Initializes the symbol palette *@
<SfSymbolPalette id="palettes" Height="600px">
</SfSymbolPalette>
```

### Add palettes to SymbolPalette

A palette allows to display a group of related symbols and it textually annotates the group with its header. A [Palettes](#) can be added as a collection of symbol groups.

The collection of predefined symbols can be added in palettes using the [Symbols](#) property. To initialize a palette, define a JSON object with the property [Id](#) that is unique ID is set to the palettes.

The following code example illustrates how to define a palette and how its added to symbol palette.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using Syncfusion.Blazor.Navigations
@using System.Collections.ObjectModel
@* Initializes the symbol palette *@
@* Defines how many palettes can be at expanded mode at a time *@
<SfSymbolPalette Height="600px" SymbolHeight="80" SymbolWidth="80"
EnableAnimation="false" ExpandMode="ExpandMode.Multiple"
@ref="@SymbolPalette"
Palettes="@Palettes">
</SfSymbolPalette>
@code{
SfSymbolPalette SymbolPalette;
```

```
public ObservableCollection<SymbolPalettePalette> Palettes;
// Defines palette's basic-shape collection
public ObservableCollection<Object> BasicShapes { get; set; }
// Defines palette's flow-shape collection
public ObservableCollection<Object> FlowShapes { get; set; }
// Defines palette's connector collection
public ObservableCollection<Object> Connectors { get; set; }
protected override void OnInitialized()
{
    Palettes = new ObservableCollection<SymbolPalettePalette>();
    //Initialize the basicshapes for the symbol palette
    BasicShapes = new ObservableCollection<Object>()
    {
        new DiagramNode()
        {
            Id = "Rectangle",
            Shape = new DiagramShape()
            {
                Type = Shapes.Basic,
                BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Rectangle
            }
        },
        new DiagramNode()
        {
            Id = "Ellipse",
            Shape = new DiagramShape()
            {
                Type = Shapes.Basic,
                BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Ellipse
            }
        },
        new DiagramNode()
        {
            Id = "Hexagon",
            Shape = new DiagramShape()
            {
                Type = Shapes.Basic,
                BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Hexagon
            }
        }
    };
    Palettes.Add(new SymbolPalettePalette()
    {
        Id = "BasicShapes",
        Expanded = true,
        Symbols = BasicShapes,
        Title = "Basicshapes"
    });
    //Initialize the flowshapes for the symbol palette
    FlowShapes = new ObservableCollection<Object>()
    {
        new DiagramNode()
        {
            Id = "process",
            Shape = new DiagramShape()
            {
                Type = Shapes.Flow,
```

```
FlowShape = Syncfusion.Blazor.Diagrams.FlowShapes.Process
},
new DiagramNode()
{
    Id = "document",
    Shape = new DiagramShape()
    {
        Type = Shapes.Flow,
        FlowShape = Syncfusion.Blazor.Diagrams.FlowShapes.Document
    }
},
new DiagramNode()
{
    Id = "predefinedprocess",
    Shape = new DiagramShape()
    {
        Type = Shapes.Flow,
        FlowShape = Syncfusion.Blazor.Diagrams.FlowShapes.PreDefinedProcess
    }
};
Palettes.Add(new SymbolPalettePalette()
{
    Id = "Flowshapes",
    Expanded = true,
    Symbols = FlowShapes,
    Title = "Flowshapes"
});
//Initializes connector symbols for the symbol palette
Connectors = new ObservableCollection<Object>()
{
    new DiagramConnector()
    {
        Id = "Link1",
        Type = Segments.Orthogonal,
        SourcePoint = new ConnectorSourcePoint() { X = 0, Y = 0 },
        TargetPoint = new ConnectorTargetPoint() { X = 40, Y = 40 },
        Style = new ConnectorShapeStyle() { StrokeWidth = 1 },
        TargetDecorator = new ConnectorTargetDecorator() { Shape =
DecoratorShapes.Arrow }
    },
    new DiagramConnector()
    {
        Id = "Link2",
        Type = Segments.Straight,
        SourcePoint = new ConnectorSourcePoint() { X = 0, Y = 0 },
        TargetPoint = new ConnectorTargetPoint() { X = 40, Y = 40 },
        Style = new ConnectorShapeStyle() { StrokeWidth = 1 },
        TargetDecorator = new ConnectorTargetDecorator() { Shape =
DecoratorShapes.Arrow }
    },
    new DiagramConnector()
    {
        Id = "Link3",
        Type = Segments.Bezier,
        SourcePoint = new ConnectorSourcePoint() { X = 0, Y = 0 },
```

```

TargetPoint = new ConnectorTargetPoint() { X = 40, Y = 40 },
Style = new ConnectorShapeStyle() { StrokeWidth = 1 },
TargetDecorator = new ConnectorTargetDecorator() { Shape =
DecoratorShapes.None }
}
};
Palettes.Add(new SymbolPalettePalette()
{
Id = "Connectors",
Expanded = true,
Symbols = Connectors,
Title = "Connectors"
});
}
}

```

### Customize the palette header

Palettes can be annotated with its header texts.

The [Title](#) displayed as the header text of palette.

The [Expanded](#) property of palette allows to expand/collapse its palette items.

The [Height](#) property of palette sets the height of the symbol group.

The [IconCss](#) property sets the content of the symbol group.

### Stretch the symbols into the palette

The [Fit](#) property defines whether the symbol has to be fit inside the size, that is defined by the symbol palette. For example, when you resize the rectangle in the symbol, ratio of the rectangle size has to be maintained rather changing into square shape. The following code example illustrates how to customize the symbol size.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize Symbol palette with customize symbol size*
<SfSymbolPalette Height="600px" SymbolHeight="60" SymbolWidth="60"
SymbolInfo="@symbolInfo" Palettes="@Palettes">
</SfSymbolPalette>
@code{
public ObservableCollection<SymbolPalettePalette> Palettes;
// Defines palette's basic-shape collection
public ObservableCollection<Object> BasicShapes { get; set; }
public SymbolInfo symbolInfo;
protected override void OnInitialized()
{
Palettes = new ObservableCollection<SymbolPalettePalette>();
// Enables to fit the content into the specified palette item size. When it
is set as false, the element is rendered with actual node size
symbolInfo = new SymbolInfo() { Fit = true };
//Initialize the basicshapes for the symbol palette
BasicShapes = new ObservableCollection<Object>()
{
new DiagramNode()
{

```

```

Id = "Rectangle",
Shape = new DiagramShape()
{
    Type = Shapes.Basic,
    BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Rectangle
}
};
Palettes.Add(new SymbolPalettePalette()
{
    Id = "BasicShapes",
    Expanded = true,
    Symbols = BasicShapes,
    Title = "Basicshapes"
});
}
}

```

### Add/Remove symbols to palette at runtime

- Symbols can be added to palette at runtime by using public method, [AddPaletteltem](#).
- Symbols can be removed from palette at runtime by using public method, [RemovePaletteltem](#).

### Customize the size of symbols

The size of the individual symbol can be customized. The [SymbolWidth](#) and [SymbolHeight](#) properties of node enables you to define the size of the symbols. The following code example illustrates how to change the size of a symbol.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize Symbol palette with customize symbol size */
<SfSymbolPalette Height="600px" SymbolHeight="60" SymbolWidth="60"
Palettes="@Palettes">
    @* Sets the margin for the symbols *
    <SymbolMargin Left="15" Bottom="15" Right="15" Top="15"></SymbolMargin>
</SfSymbolPalette>
@code{
    public ObservableCollection<SymbolPalettePalette> Palettes;
    // Defines palette's basic-shape collection
    public ObservableCollection<Object> BasicShapes { get; set; }
    protected override void OnInitialized()
    {
        Palettes = new ObservableCollection<SymbolPalettePalette>();
        BasicShapes = new ObservableCollection<Object>()
        {
            new DiagramNode()
            {
                Id = "Rectangle",
                Shape = new DiagramShape()
                {
                    Type = Shapes.Basic,
                    BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Rectangle
                }
            }
        }
    }
}

```



```

    },
    new DiagramNode()
    {
        Id="Ellipse" ,
        Shape = new DiagramShape()
        {
            Type = Shapes.Basic,
            BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Ellipse
        }
    },
    new DiagramNode()
    {
        Id="Hexagon" ,
        Shape = new DiagramShape()
        {
            Type = Shapes.Basic,
            BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Hexagon
        }
    }
    };
    Palettes.Add(new SymbolPalettePalette()
    {
        Id = "BasicShapes",
        Expanded = true,
        Symbols = BasicShapes,
        Title = "Basicshapes"
    });
}
}

```

The [SymbolMargin](#) property is used to create the space around elements, outside of any defined borders.

### Symbol preview

The symbol preview size of the palette items can be customized using [SymbolPreview](#). The [Width](#) and [Height](#) properties of SymbolPalette enables you to define the preview size to all the symbol palette items. The [Offset](#) of the dragging helper relative to the mouse cursor.

The following code example illustrates how to change the preview size of a palette item.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
/* Initialize Symbol palette with customize symbol size */
<SfSymbolPalette ID="palettes" Height="600px" SymbolHeight=60 SymbolWidth=60
Palettes="@Palettes">
/* Sets the margin for the symbols */
<SymbolMargin Left="15" Bottom="15" Right="15" Top="15"></SymbolMargin>
/* Specifies the preview size and position to symbol palette items */
<SymbolPaletteSymbolPreview Height="100" Width="100">
<SymbolPreviewOffset X="0.5" Y="0.5"></SymbolPreviewOffset>
</SymbolPaletteSymbolPreview>
</SfSymbolPalette>
@code{
    public ObservableCollection<SymbolPalettePalette> Palettes;

```

```
// Defines palette's basic-shape collection
public ObservableCollection<Object> BasicShapes { get; set; }
protected override void OnInitialized()
{
    Palettes = new ObservableCollection<SymbolPalettePalette>();
    BasicShapes = new ObservableCollection<Object>()
    {
        new DiagramNode()
        {
            Id = "Rectangle",
            Shape = new DiagramShape()
            {
                Type = Shapes.Basic,
                BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Rectangle
            }
        },
        new DiagramNode()
        {
            Id="Ellipse",
            Shape = new DiagramShape()
            {
                Type = Shapes.Basic,
                BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Ellipse
            }
        },
        new DiagramNode()
        {
            Id="Hexagon",
            Shape = new DiagramShape()
            {
                Type = Shapes.Basic,
                BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Hexagon
            }
        }
    };
    Palettes.Add(new SymbolPalettePalette()
    {
        Id = "BasicShapes",
        Expanded = true,
        Symbols = BasicShapes,
        Title = "Basicshapes"
    });
}
```

### Default settings

While adding more number of symbols such as nodes and connectors to the palette, define the default settings for those objects through the [NodeDefaults](#) and the [ConnectorDefaults](#) properties of diagram allows to define the default settings for nodes and connectors.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@using Syncfusion.Blazor.Navigations;
@* Initialize Symbol palette with customize symbol size*@
```

```

<SfSymbolPalette Height="600px" ExpandMode="ExpandMode.Multiple"
NodeDefaults="@PaletteNodeDefaults" Palettes="@Palettes">
<SymbolPaletteSymbolPreview Height="100" Width="100">
<SymbolPreviewOffset X="0.5" Y="0.5"></SymbolPreviewOffset>
</SymbolPaletteSymbolPreview>
<SymbolMargin Left="15" Bottom="15" Right="15" Top="15"></SymbolMargin>
</SfSymbolPalette>
@code{
public ObservableCollection<SymbolPalettePalette> Palettes;
// Defines palette's basic shape collection
public ObservableCollection<Object> BasicShapes { get; set; }
// Defines the default values for Nodes
public DiagramNode PaletteNodeDefaults;
protected override void OnInitialized()
{
Palettes = new ObservableCollection<SymbolPalettePalette>();
BasicShapes = new ObservableCollection<Object>()
{
new DiagramNode()
{
Id = "Rectangle",
Shape = new DiagramShape()
{
Type = Shapes.Basic,
BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Rectangle
}
},
new DiagramNode()
{
Id = "Ellipse",
Shape = new DiagramShape()
{
Type = Shapes.Basic,
BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Ellipse
}
},
new DiagramNode()
{
Id = "Hexagon",
Shape = new DiagramShape()
{
Type = Shapes.Basic,
BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Hexagon
}
}
};
// Sets the default values for Nodes
PaletteNodeDefaults = new DiagramNode()
{
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { StrokeColor = "#3A3A3A" }
};
Palettes.Add(new SymbolPalettePalette()
{
Id = "BasicShapes",
Expanded = true,

```

```

Symbols = BasicShapes,
Title = "Basicshapes"
});
}
}

```

### Adding symbol description for symbols in the palette

The diagram provides support to add symbol description below each symbol of a palette. This descriptive representation of each symbol will enhance the details of the symbol visually. The height and width of the symbol description can also be set individually.

- The property `getSymbolInfo`, can be used to add the symbol description at runtime.

The following code is an example to set a symbol description for symbols in the palette.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using Syncfusion.Blazor.Navigations
@using System.Collections.ObjectModel
/* Initializes the symbol palette */
/* Defines how many palettes can be at expanded mode at a time */
<SfSymbolPalette Height="600px" SymbolHeight="80" SymbolWidth="80"
EnableAnimation="false" ExpandMode="ExpandMode.Multiple"
@ref="@SymbolPalette" Palettes="@Palettes">
</SfSymbolPalette>
@code{
SfSymbolPalette SymbolPalette;
public ObservableCollection<SymbolPalettePalette> Palettes;
// Defines palette's basic-shape collection
public ObservableCollection<Object> BasicShapes { get; set; }
// Defines palette's flow-shape collection
public ObservableCollection<Object> FlowShapes { get; set; }
protected override void OnInitialized()
{
Palettes = new ObservableCollection<SymbolPalettePalette>();
//Initialize the basicshapes for the symbol palette
BasicShapes = new ObservableCollection<Object>()
{
new DiagramNode()
{
Id = "Rectangle",
Shape = new DiagramShape()
{
Type = Shapes.Basic,
BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Rectangle
},
// Sets symbol description for rectangle shape.
SymbolInfo= new SymbolInfo()
{
Description= new SymbolDescription
{
Text="Rectangle"
}
}
}
}
}

```

```
    },
    new DiagramNode()
    {
        Id = "Ellipse",
        Shape = new DiagramShape()
        {
            Type = Shapes.Basic,
            BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Ellipse
        },
        // Sets symbol description for ellipse shape.
        SymbolInfo= new SymbolInfo(){Description=new
        SymbolDescription{Text="Ellipse"}}
    },
    new DiagramNode()
    {
        Id = "Hexagon",
        Shape = new DiagramShape()
        {
            Type = Shapes.Basic,
            BasicShape = Syncfusion.Blazor.Diagrams.BasicShapes.Hexagon
        },
        // Sets symbol description for hexagon shape.
        SymbolInfo= new SymbolInfo(){Description=new
        SymbolDescription{Text="Hexagon"} }
    }
};
Palettes.Add(new SymbolPalettePalette()
{
    Id = "BasicShapes",
    Expanded = true,
    Symbols = BasicShapes,
    Title = "Basicshapes"
});
//Initialize the flowshapes for the symbol palette
FlowShapes = new ObservableCollection<Object>()
{
    new DiagramNode()
    {
        Id = "process",
        Shape = new DiagramShape()
        {
            Type = Shapes.Flow,
            FlowShape = Syncfusion.Blazor.Diagrams.FlowShapes.Process
        },
        // Sets symbol description for process shape.
        SymbolInfo= new SymbolInfo()
        {
            Description= new SymbolDescription
            {
                Text="Process"
            }
        }
    },
    new DiagramNode()
    {
        Id = "document",
        Shape = new DiagramShape()
```

```
{
    Type = Shapes.Flow,
    FlowShape = Syncfusion.Blazor.Diagrams.FlowShapes.Document
},
// Sets symbol description for document shape.
SymbolInfo= new SymbolInfo()
{
    Description= new SymbolDescription
    {
        Text="Document"
    }
},
};
Palettes.Add(new SymbolPalettePalette()
{
    Id = "Flowshapes",
    Expanded = true,
    Symbols = FlowShapes,
    Title = "Flowshapes"
});
}
```

### Palette interaction

Palette interaction notifies the element enter, leave, and dragging of the symbols into the diagram.

### Escape Key function

The diagram provides support to cancel the node from symbol palette when the ESC key is pressed.

### See Also

- [How to add the symbol to the diagram](#)

## Overview Control in Blazor Diagram Component

Overview control allows you to see a preview or an overall view of the entire content of a diagram. This helps you to look at the overall picture of a large diagram and to navigate, pan, or zoom, on a particular position of the page.

When you work on a very large diagram, you may not know the part you are working on, or navigation from one part to another might be difficult. One solution for navigation is to zoom out the entire diagram and find where you are. Then, you can zoom in an area you want to. This solution is not suitable when you need some frequent navigation.

Overview control solves these problems by showing a preview, that is, an overall view of the entire diagram. A rectangle indicates viewport of the diagram. Navigation becomes easy by dragging this rectangle.

### Create overview

The [SourceID](#) property of overview should be set with the corresponding diagram ID for the overall view.

The [Width](#) and [Height](#) properties of the overview allow you to define the size of the overview.

The following code illustrates how to create overview.

**ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram id="diagram" Height="600px" Layout="@LayoutValue"
ConnectorDefaults="@ConnectorDefault" NodeDefaults="@NodeDefaults">
<DiagramDataSource Id="Name" ParentId="Category" DataSource="@DataSource"
DataMapSettings="@datamap">
<DiagramDataMapSettings>
<DiagramDataMapSetting Property="Annotations[0].Content"
Field="Name"></DiagramDataMapSetting>
</DiagramDataMapSettings>
</DiagramDataSource>
</SfDiagram>
<SfOverview Height="150px" SourceID="diagram" />
@code
{
//Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
//Defines the node default values.
DiagramNode NodeDefaults = new DiagramNode()
{
Width = 95,
Height = 30,
BackgroundColor = "#6BA5D7",
Shape = new DiagramShape() { Type = Shapes.Basic, BasicShape =
BasicShapes.Rectangle },
Style = new NodeShapeStyle { Fill = "#ffeec7", StrokeColor = "#ffeec7",
StrokeWidth = 1, },
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Id = "label1",
Style = new AnnotationStyle()
{
Color = "black"
}
}
}
};
//Defines the connector's default values.
DiagramConnector ConnectorDefault = new DiagramConnector
{
Type = Segments.Orthogonal,
Style = new ConnectorShapeStyle() { StrokeColor = "#4d4d4d", StrokeWidth = 2
},
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.None,
}
};
//Create the layout info

```

```
TreeInfo LayoutInfo = new TreeInfo()
{
    //Create the layout info
    CanEnableSubTree = true,
    //Specify the sub-tree orientation
    Orientation = SubTreeOrientation.Horizontal
};
List<DiagramDataMapSetting> datamap { get; set; } = new
List<DiagramDataMapSetting>()
{
    new DiagramDataMapSetting() { Property = "Shape.TextContent", Field = "Name"
    }
};
DiagramLayout LayoutValue = new DiagramLayout() { };
protected override void OnInitialized()
{
    LayoutValue = new DiagramLayout()
    {
        Type = LayoutType.HierarchicalTree,
        VerticalSpacing = 30,
        HorizontalSpacing = 30,
        EnableAnimation = true,
        LayoutInfo = this.LayoutInfo
    };
}
//Create the hierarchical details with needed properties.
public class HierarchicalDetails
{
    public string Name { get; set; }
    public string FillColor { get; set; }
    public string Category { get; set; }
}
//Create the data source with node name and fill color values.
public List<object> DataSource = new List<object>()
{
    new HierarchicalDetails(){ Name ="Diagram",
    Category="",FillColor="#916DAF"},
    new HierarchicalDetails(){ Name ="Layout", Category="Diagram",FillColor=""},
    new HierarchicalDetails(){ Name ="Tree Layout",
    Category="Layout",FillColor=""},
    new HierarchicalDetails(){ Name ="Organizational Chart",
    Category="Layout",FillColor=""},
    new HierarchicalDetails(){ Name ="Hierarchical Tree", Category="Tree
    Layout",FillColor=""},
    new HierarchicalDetails(){ Name ="Radial Tree", Category="Tree
    Layout",FillColor=""},
    new HierarchicalDetails(){ Name ="Mind Map", Category="Hierarchical
    Tree",FillColor=""},
    new HierarchicalDetails(){ Name ="Family Tree", Category="Hierarchical
    Tree",FillColor=""},
    new HierarchicalDetails(){ Name ="Management", Category="Organizational
    Chart",FillColor=""},
    new HierarchicalDetails(){ Name ="Human Resources",
    Category="Management",FillColor=""},
    new HierarchicalDetails(){ Name ="University",
    Category="Management",FillColor=""},
```



```
new HierarchicalDetails() { Name = "Business",
Category = "#Management", FillColor = "" }
};
}
```

### Zoom and Pan

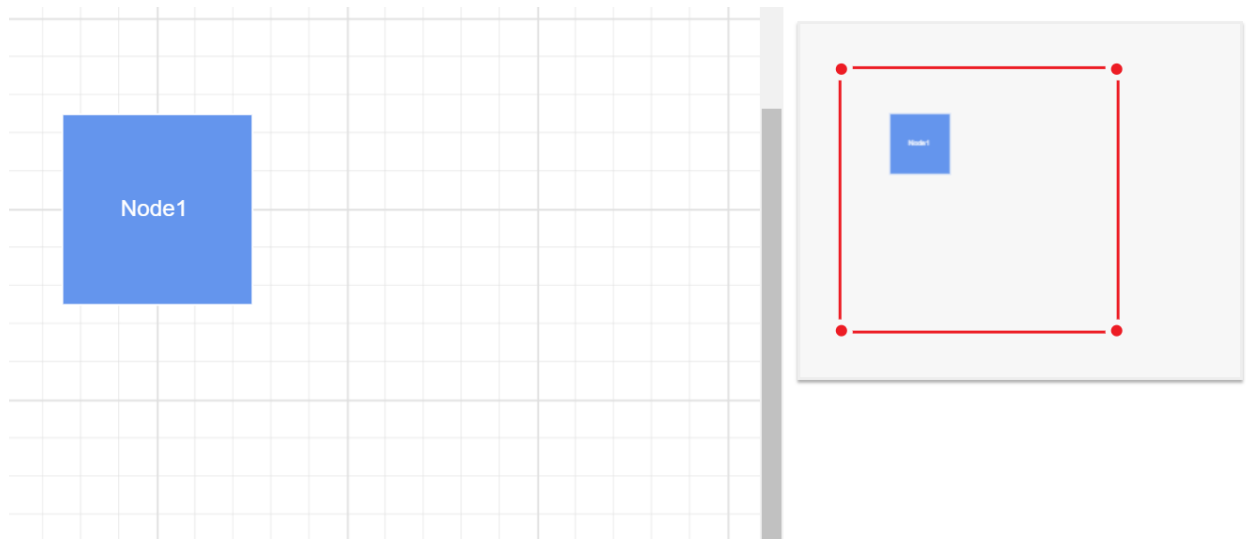
In overview, the view port of the diagram is highlighted with a red color rectangle. Diagram can be zoomed/panned by interacting with that. You can interact with overview as follows:

- Resize the rectangle: Zooms in/out the diagram.
- Drag the rectangle: Pans the diagram.
- Click at a position: Navigates to the clicked region.
- Choose a particular region by clicking and dragging: Navigates to the specified region.

The following image shows how the diagram is zoomed/panned with overview.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<SfDiagram Nodes="@NodeCollection" id="diagram" Height="600px">
<DiagramScrollSettings ScrollLimit="@ScrollLimit.Infinity" />
</SfDiagram>
<SfOverview Height="150px" SourceID="diagram" />
@code
{
//Defines diagram's nodes collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
protected override void OnInitialized()
{
DiagramNode node = new DiagramNode()
{
Id = "group",
OffsetX = 200,
OffsetY = 200,
Width = 100,
Height = 100,
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Content = "Node1",
Style = new AnnotationStyle()
{
Color = "white",
}
},
},
Style = new NodeShapeStyle() { Fill = "cornflowerblue", StrokeColor =
"white" }
};
NodeCollection.Add(node);
}
}
```



### Diagram Methods in Blazor Diagram Component

The following methods are used to perform the diagram functionalities.

#### Nodes

##### Add the node

You can add the node at runtime by using the [AddNode](#) method. The following code snippet shows how to add the node.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Add Node" @onclick="@AddNodeInCollection" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in the nodes collection.
DiagramNode node1 = new DiagramNode()
{
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
NodeCollection.Add(node1);
}
// Add node at runtime
public void AddNodeInCollection()
```

```
{
DiagramNode Node1 = new DiagramNode()
{
Id = "New Node1",
OffsetX = 100,
OffsetY = 100,
Width = 100,
Height = 100
};
Diagram.AddNode(Node1);
}
}
```

Add Node

#### *Remove the node and connector*

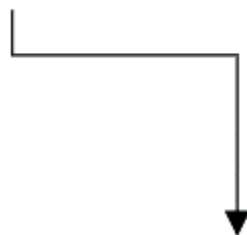
You can remove the node at runtime by using the [Remove](#) method.

The following code shows how to remove a node at runtime.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Remove Node" @onclick="@RemoveNode" />
<input type="button" value="Remove Connector" @onclick="@RemoveConnector" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
SnapConstraints Constraints;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
protected override void OnInitialized()
{
Constraints = SnapConstraints.None;
```

```
// A node is created and stored in node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
    new DiagramNode()
    {
        // Position of the node
        OffsetX = 250, OffsetY = 250,
        // Size of the node
        Width = 100, Height = 100,
        Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
    }
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
    new DiagramConnector()
    {
        Id = "Connector1",
        SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
        TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
        Type = Segments.Orthogonal
    }
};
// Remove Node at runtime
public void RemoveNode()
{
    Diagram.Remove(NodeCollection[0]);
}
// Remove connector at runtime
public void RemoveConnector()
{
    Diagram.Remove(ConnectorCollection[0]);
}
}
```

**Remove Node****Remove Connector**

### Add child to group node

You can add the child to group node at runtime by using the [AddChildToGroup](#) method. The following code shows how to add child in a group.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Add Child Node" @onclick="@AddChildNode" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
protected override void OnInitialized()
{
DiagramNode node1 = CreateNode("node1", 100, 100, "Node1");
DiagramNode node2 = CreateNode("node2", 300, 100, "Node2");
DiagramNode groupnode = new DiagramNode();
// Grouping node 1 and node 2 into a single group
groupnode.Children = new string[] { "node1", "node2" };
NodeCollection.Add(node1);
NodeCollection.Add(node2);
NodeCollection.Add(groupnode);
}
public DiagramNode CreateNode(string Id, double OffsetX, double OffsetY,
string Content)
{
DiagramNode Node = new DiagramNode()
{
Id = Id,
OffsetX = OffsetX,
OffsetY = OffsetY,
Height = 100,
Width = 100,
Style = new NodeShapeStyle() { Fill = "darkcyan" },
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Id="annotation1",
Content = Content,
Style=new AnnotationStyle() {Color="white",
Fill="transparent",StrokeColor="None"},
}
}
};
return Node;
}
public void AddChildNode()
{
DiagramNode Child = CreateNode("node2", 300, 100, "Node2");
// Add child node to group
Diagram.AddChildToGroup(NodeCollection[2], Child);
}
}
```

### Get the node

You can find and get the node from the id property itself. The following code shows how to find the node in the diagram.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Get Node and Update style" @onclick="@GetNode"
/>
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
// A node is created and stored in the nodes collection.
DiagramNode node1 = new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
NodeCollection.Add(node1);
}
// Add node at runtime
public void GetNode()
{
// Find the node
DiagramNode Node = Diagram.GetNode("Node1");
Node.Style.StrokeColor = "black";
}
}
```

Get Node and Update style



#### *Get the edges*

You can find what are all the in and out connectors that are connected to the node by using the [GetEdges](#) method. The following code snippet shows how to get the in and out connector list what are the connectors are connected to the node.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="UpdateEdges" @onclick="@UpdateEdges" />
<SfDiagram Height="600px" @ref="@Diagram"
Nodes="@NodeCollection" Connectors="@ConnectorCollection"
NodeDefaults="@NodeDefaults">
</SfDiagram>
@code{
// Reference of the diagram
SfDiagram Diagram;
// Define node and connector collection
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
// Set the default value of the node
DiagramNode NodeDefaults = new DiagramNode()
{
// Size of the node
Width = 80,
Height = 50,
// Style of the node
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
};
protected override void OnInitialized()
{
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{

```

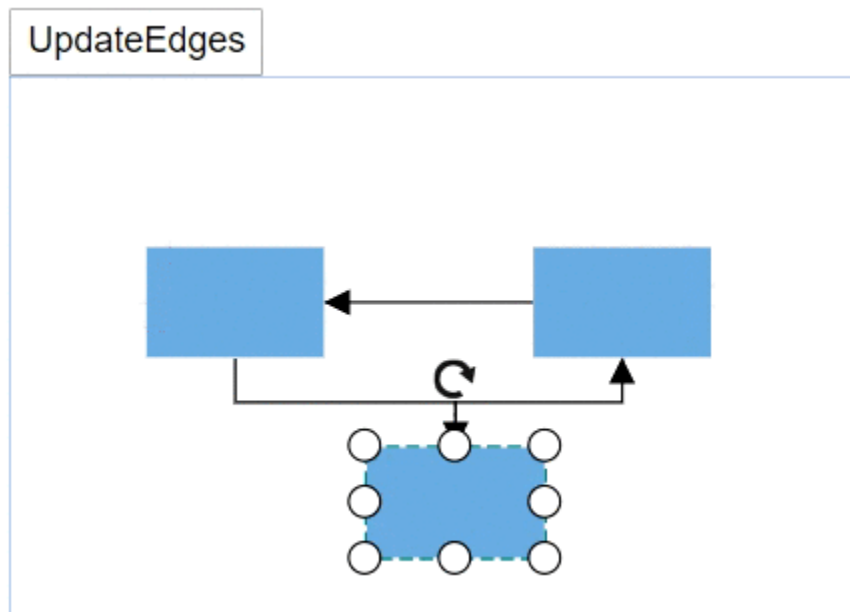
```
Id = "Node1",
// Position of the node
OffsetX = 100,
OffsetY = 100
},
new DiagramNode()
{
Id = "Node2",
// Position of the node
OffsetX = 198, OffsetY = 189
},
new DiagramNode()
{
Id = "Node3",
// Position of the node
OffsetX = 272, OffsetY = 100
}
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
new DiagramConnector()
{
// Sets the unique id, source node, and target node
Id = "Connector1", SourceID = "Node1", TargetID = "Node2",
// Sets the type of the connector
Type = Segments.Orthogonal
},
new DiagramConnector()
{
// Sets the unique id, source node, and target node
Id = "Connector2", SourceID = "Node2", TargetID = "Node3",
// Sets the type of the connector
Type = Segments.Orthogonal
},
new DiagramConnector()
{
// Sets the unique id, source node, and target node
Id = "connector3", SourceID = "Node3", TargetID = "Node1",
// Sets the type of the connector
Type = Segments.Orthogonal
}
};
protected override async Task OnAfterRenderAsync(bool firstRender)
{
if (firstRender)
{
//OnAfterRenderAsync will be triggered after the component rendered.
await Task.Delay(1500);
// Select the node
Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[1] },
null);
}
}
// The GetEdges method is used to get the connectors that connected to
nodes.
public async Task UpdateEdges()
```



```

{
    string NodeId = Diagram.SelectedItems.Nodes[0].Id;
    // Find the out connectors connected from the node
    string[] OutEdges = await Diagram.GetEdges(NodeId, true);
    // Find the in connectors connected to the node
    string[] InEdges = await Diagram.GetEdges(NodeId, false);
    string[] edges = InEdges.Concat(OutEdges).ToArray();
    for (int i = 0; i < edges.Length; i++)
    {
        // Get the connector from id
        DiagramConnector connector = Diagram.GetConnector(edges[i]);
        // Change the style of the connector
        connector.Style.StrokeColor = "#1413F8";
        connector.TargetDecorator.Style.StrokeColor = "#1413F8";
        connector.TargetDecorator.Style.Fill = "#1413F8";
    }
}

```



#### Get the parent information

You can find the parent of the object by using the [GetParentID](#) method. The following code shows how to get the parent information of the object.

#### ASPX-CS

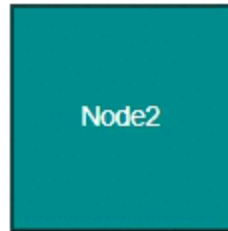
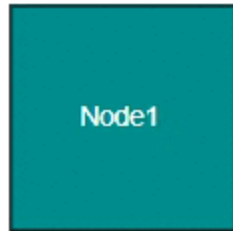
```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Get Parent Node" @onclick="@GetParentNode" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    SfDiagram Diagram;
    ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>();
    protected override void OnInitialized()

```

```
{
DiagramNode node1 = CreateNode("node1", 100, 100, "Node1");
DiagramNode node2 = CreateNode("node2", 300, 100, "Node2");
DiagramNode groupnode = new DiagramNode();
// Grouping node 1 and node 2 into a single group
groupnode.Children = new string[] { "node1", "node2" };
NodeCollection.Add(node1);
NodeCollection.Add(node2);
NodeCollection.Add(groupnode);
}
public DiagramNode CreateNode(string Id, double OffsetX, double OffsetY,
string Content)
{
DiagramNode Node = new DiagramNode()
{
Id = Id,
OffsetX = OffsetX,
OffsetY = OffsetY,
Height = 100,
Width = 100,
Style = new NodeShapeStyle() { Fill = "darkcyan" },
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation()
{
Id="annotation1",
Content = Content,
Style=new AnnotationStyle() {Color="white",
Fill="transparent",StrokeColor="None"},
}
}
};
return Node;
}
public async void GetParentNode()
{
// Get the parent id
string parentId = await Diagram.GetParentId("node2");
DiagramNode Parent = Diagram.GetNode(parentId);
Parent.Style.StrokeColor = "#6BA5D7";
}
}
```

Get Parent Node



## Connectors

### Add the connector

You can add a connector at runtime by using the server-side method [AddConnector](#) in the Diagram component. The following code explains how to add connectors at runtime.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
<input type="button" value="Add Connector" @onclick="@AddConnector">
<SfDiagram @ref="@Diagram" Height="600px">
</SfDiagram>
@code
{
    SfDiagram Diagram;
    // To hide the gridlines
    SnapConstraints Constraints = SnapConstraints.None;
    protected void AddConnector()
    {
        DiagramConnector diagramConnector = new DiagramConnector()
        {
            Id = "Connector1",
            SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
            TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
            TargetDecorator = new ConnectorTargetDecorator()
            {
                Shape = DecoratorShapes.Arrow,
                Style = new DecoratorShapeStyle()
                {
                    Fill = "#6f409f",
                    StrokeColor = "#6f409f",
                    StrokeWidth = 1
                }
            },
            Style = new ConnectorShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1
        },
    },
}
```

```
Type = Segments.Straight,
};
//Add the connector at the run time.
Diagram.AddConnector(diagramConnector);
}
}
```



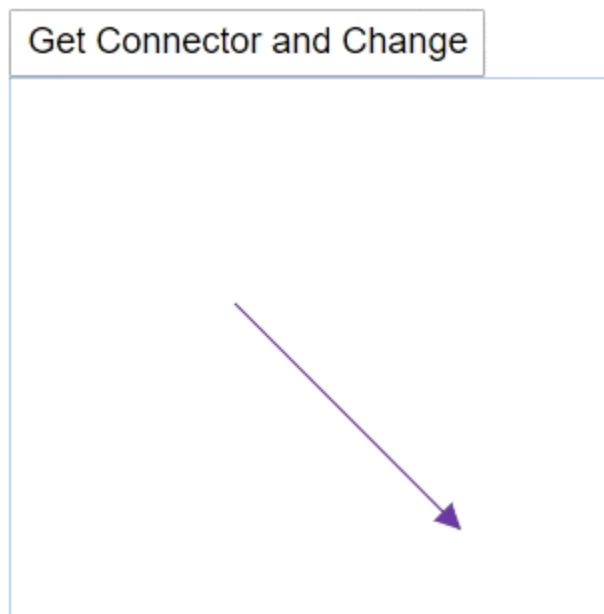
#### *Get the connector*

You can get the connector from property `Id` by using the [GetConnector](#) method. The following code shows how to find the connector in the diagram.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Get Connector and Change"
@onclick="@GetConnector" />
<SfDiagram @ref="@Diagram" Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
// A connector is created and stored in the connector collection.
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
new DiagramConnector()
{
Id = "Connector1",
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
}
```

```
TargetDecorator = new ConnectorTargetDecorator()
{
    Shape = DecoratorShapes.Arrow,
    Style = new DecoratorShapeStyle()
    {
        Fill = "#6f409f",
        StrokeColor = "#6f409f",
        StrokeWidth = 1
    }
},
Style = new ConnectorShapeStyle()
{
    StrokeColor = "#6f409f",
    StrokeWidth = 1
},
Type = Segments.Straight,
};
}
public void GetConnector()
{
    // Get the connector
    DiagramConnector Connector = Diagram.GetConnector("Connector1");
    Connector.SourcePoint.X += 10;
}
}
```



#### *Reset the segments*

You can arrange the connector segments without overlapping the nodes by using the [ResetSegments](#) method.

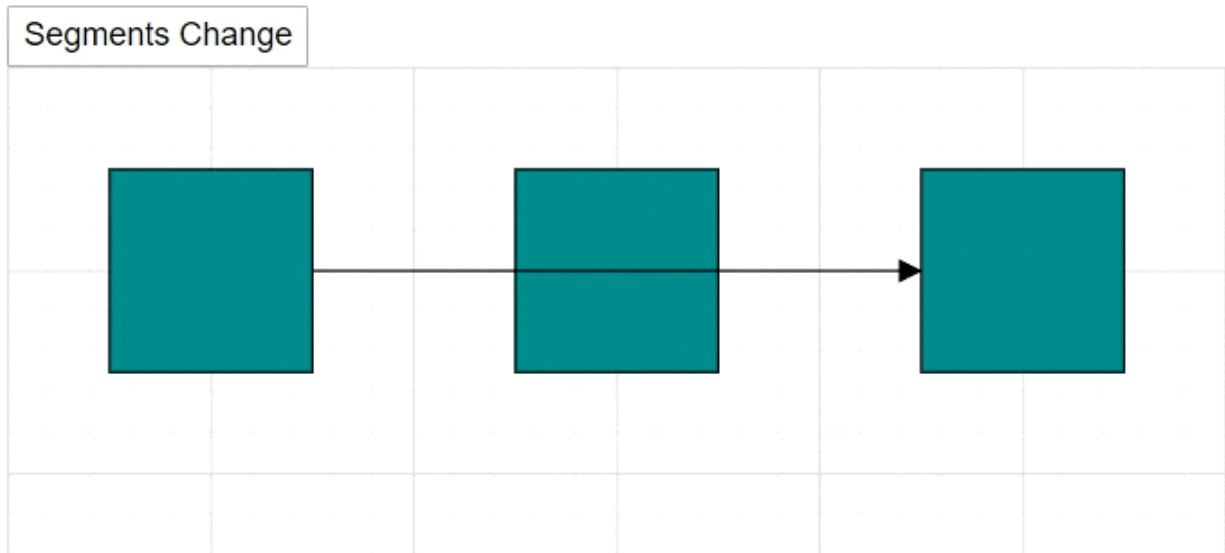
#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
```

```

<input type="button" value="Segments Change" @onclick="@ResetSegments" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code
{
    SfDiagram Diagram;
    //Defines diagram's nodes collection
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>();
    //Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection = new
    ObservableCollection<DiagramConnector>();
    public DiagramConstraints Constraints { get; set; }
    protected override void OnInitialized()
    {
        // Add nodes to collection
        NodeCollection.Add(CreateNode("Node1", 100, 100, "Node1"));
        NodeCollection.Add(CreateNode("Node2", 300, 100, "Node2"));
        NodeCollection.Add(CreateNode("Node3", 500, 100, "Node3"));
        ConnectorCollection = new ObservableCollection<DiagramConnector>()
        {
            new DiagramConnector()
            {
                Id = "Connector1", SourceID = "Node1", TargetID = "Node3",
                Type = Segments.Orthogonal
            }
        };
        public DiagramNode CreateNode(string Id, double OffsetX, double OffsetY,
        string Content)
        {
            DiagramNode Node = new DiagramNode()
            {
                Id = Id,
                OffsetX = OffsetX, OffsetY = OffsetY,
                Height = 100, Width = 100,
                Style = new NodeShapeStyle() { Fill = "darkcyan" }
            };
            return Node;
        }
        public void ResetSegments()
        {
            Diagram.BeginUpdate();
            Diagram.Constraints = DiagramConstraints.Default |
            DiagramConstraints.LineRouting;
            Diagram.EndUpdate();
            // update the segments based on routing
            Diagram.ResetSegments();
        }
    }
}

```



### Annotations

#### *Add the annotation*

Annotations can be added at runtime by using the [AddLabels](#) method. The following code explains how to add an annotation to a node.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input value="Addlabel" type="button" @onclick="@AddLabel" name="Addlabel" />
<SfDiagram Height="600px" @ref="@diagram" Nodes="@NodeCollection">
</SfDiagram>
@code
{
    // Reference to diagram
    SfDiagram diagram;
    //Defines diagram's node collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    SnapConstraints snapConstraints;
    protected override void OnInitialized()
    {
        snapConstraints = SnapConstraints.None;
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node = new DiagramNode()
        {
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
        };
        NodeCollection.Add(node);
    }
    //Method to add labels at runtime
    public void AddLabel()
    {

```

```

ObservableCollection<DiagramNodeAnnotation> annotations = new
ObservableCollection<DiagramNodeAnnotation>()
{
    new DiagramNodeAnnotation() { Content = "Annotation" },
};
// AddLabels method is used to add annotations at run time
diagram.AddLabels(diagram.Nodes[0], annotations);
}
}

```

Addlabel



#### Remove the annotation

A collection of annotations can be removed from the node by using the [RemoveLabels](#) method. The following code explains how to remove an annotation to a node.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input value="Removelabel" type="button" onclick="@RemoveLabel"
name="Removelabel" />
<SfDiagram Height="600px" @ref="@diagram" Nodes="@NodeCollection">
</SfDiagram>
@code
{
    //Reference to diagram
    SfDiagram diagram;
    //Defines diagram's node collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node = new DiagramNode()
        {
            Width = 100,
            Height = 100,

```



```

OffsetX = 100,
OffsetY = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
};
node.Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
    new DiagramNodeAnnotation() { Id="label", Content = "Annotation" },
};
NodeCollection.Add(node);
}
//Method to remove labels at runtime
public void RemoveLabel()
{
    ObservableCollection<DiagramNodeAnnotation> annotations = new
    ObservableCollection<DiagramNodeAnnotation>()
    {
        new DiagramNodeAnnotation() { Id="label", Content = "Annotation" }
    };
    // RemoveLabels method is used to remove label at run time.
    diagram.RemoveLabels(diagram.Nodes[0], annotations);
}
}

```

Addlabel



#### *Edit the annotation*

You can change the annotation content by using the [StartTextEdit](#) method. The following code shows how to edit the content of the annotation.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input value="Annotation Editing" type="button"
@onclick="@AnnotationEditing" />
<SfDiagram Height="600px" @ref="@Diagram" Nodes="@NodeCollection">
</SfDiagram>

```

```
@code
{
    // Reference to diagram
    SfDiagram Diagram;
    //Defines diagram's node collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    protected override void OnInitialized()
    {
        NodeCollection = new ObservableCollection<DiagramNode>();
        DiagramNode node = new DiagramNode()
        {
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
            Annotations = new ObservableCollection<DiagramNodeAnnotation>()
            {
                new DiagramNodeAnnotation() { Id = "Annotation1", Content = "Annotation" }
            }
        };
        NodeCollection.Add(node);
    }
    //Method to edit the annotation at runtime
    public void AnnotationEditing()
    {
        Diagram.StartTextEdit(NodeCollection[0], "Annotation1");
    }
}
```

Annotation Editing



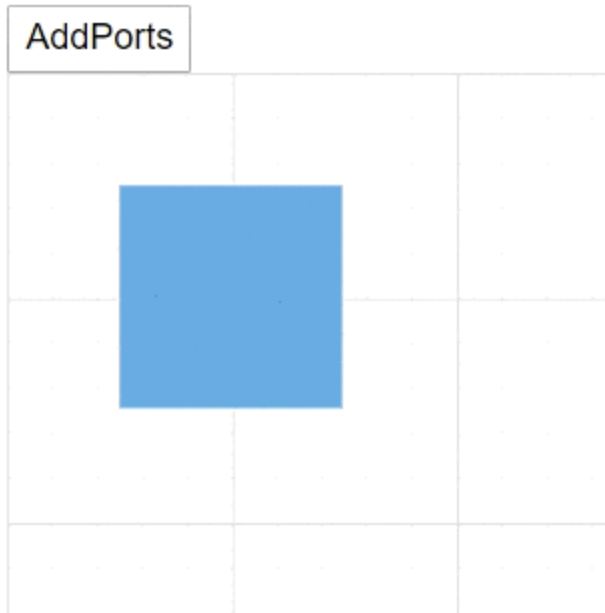
## Ports

### Add the ports

You can add ports to a node at runtime by using the [AddPorts](#) methods. The following code shows how to add new port in a node.

### CSHARP

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="AddPorts" @onclick="@AddPorts" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    SfDiagram Diagram;
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        //A node is created and stored in nodes array
        DiagramNode model = new DiagramNode()
        {
            //Position of the node
            OffsetX = 250,
            OffsetY = 250,
            //Size of the node
            Width = 100,
            Height = 100,
            Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
        };
        NodeCollection.Add(model);
    }
    public void AddPorts()
    {
        // Initialize port collection
        ObservableCollection<DiagramPort> ports = new
        ObservableCollection<DiagramPort>()
        {
            new DiagramPort() { Id = "port1", Offset = new NodePortOffset() { X = 0, Y =
            0.5 }, Visibility = PortVisibility.Visible },
            new DiagramPort() { Id = "port2", Offset = new NodePortOffset() { X = 1, Y =
            0.5 }, Visibility = PortVisibility.Visible },
            new DiagramPort() { Id = "port3", Offset = new NodePortOffset() { X = 0.5, Y
            = 0 }, Visibility = PortVisibility.Visible },
            new DiagramPort() { Id = "port4", Offset = new NodePortOffset() { X = 0.5, Y
            = 1 }, Visibility = PortVisibility.Visible }
        };
        Diagram.AddPorts(NodeCollection[0], ports);
    }
}
```



### *Remove the ports*

You can remove the existing ports in a node by using the [RemovePorts](#) method. The following code shows how to remove the ports in a diagram.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="RemovePorts" @onclick="@RemovePorts" />
<SfDiagram Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    SnapConstraints constraints;
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        constraints = SnapConstraints.None;
        // A node is created and stored in nodes array.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
            // Initialize port collection
            Ports = new ObservableCollection<DiagramPort>()
            {
                new DiagramPort()
                {
                    Id = "port1",
                    Offset = new NodePortOffset() { X = 0, Y = 0.5 },
```

```

Visibility = PortVisibility.Visible,
//Set the style for the port
Style= new PortShapeStyle(){ Fill="red", StrokeColor="black",
StrokeWidth=2},
// Sets the shape of the port as Circle
Width= 12,
Height=12,
Shape= PortShapes.Circle
}
},
};
NodeCollection.Add(node1);
}
public void RemovePorts()
{
(NodeCollection[0].Ports as ObservableCollection<DiagramPort>).RemoveAt(0);
}
}

```

RemovePorts



Print and exporting

*Print the diagram*

You can print the diagram area by using the [Print](#) method. The following code shows how to print the diagram.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Print Diagram" @onclick="@PrintDiagram" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
SfDiagram Diagram;

```

```

public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
    // A node is created and stored in the nodes collection.
    DiagramNode node1 = new DiagramNode()
    {
        // Position of the node
        OffsetX = 250,
        OffsetY = 250,
        // Size of the node
        Width = 100,
        Height = 100,
        Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
    };
    NodeCollection.Add(node1);
}
// Print the diagram
public void PrintDiagram()
{
    IPrintOptions PrintOptions = new IPrintOptions() { PageHeight = 500,
    PageWidth = 500 };
    Diagram.Print(PrintOptions);
}
}

```

#### *Export the diagram*

You can export the diagram area by using the [ExportDiagram](#) method. The following code shows how to export the diagram.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Export Diagram" @onclick="@ExportDiagram" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    SfDiagram Diagram;
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes collection.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
        };
        NodeCollection.Add(node1);
    }
}

```

```
// Export the diagram
public void ExportDiagram()
{
    IExportOptions ExportOptions = new IExportOptions() { PageWidth = 500,
    PageHeight = 500 };
    Diagram.ExportDiagram(ExportOptions);
}
}
```

### Save and load the diagram

- You can save the diagram to JSON data by using the [SaveDiagram](#) method. The following code is used how to save the diagram.
- You can load the diagram from JSON data by using the [LoadDiagram](#) method. The following code is used how to load the diagram at runtime.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Save Diagram" @onclick="@SaveDiagram" />
<input type="button" value="Load Diagram" @onclick="@LoadDiagram" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
    SfDiagram Diagram;
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>() { };
    protected override void OnInitialized()
    {
        // A node is created and stored in the nodes collection.
        DiagramNode node1 = new DiagramNode()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
        };
        NodeCollection.Add(node1);
    }
    string SaveString;
    // Save the diagram
    public async void SaveDiagram()
    {
        SaveString = await Diagram.SaveDiagram();
    }
    // Load the diagram
    public void LoadDiagram()
    {
        Diagram.LoadDiagram(SaveString);
    }
}
```



## Layers

### *Add the layers*

You can add new layers in a diagram by using the `AddLayers` method. The following code shows how to add the layers to the diagram.

### ASPX-CS

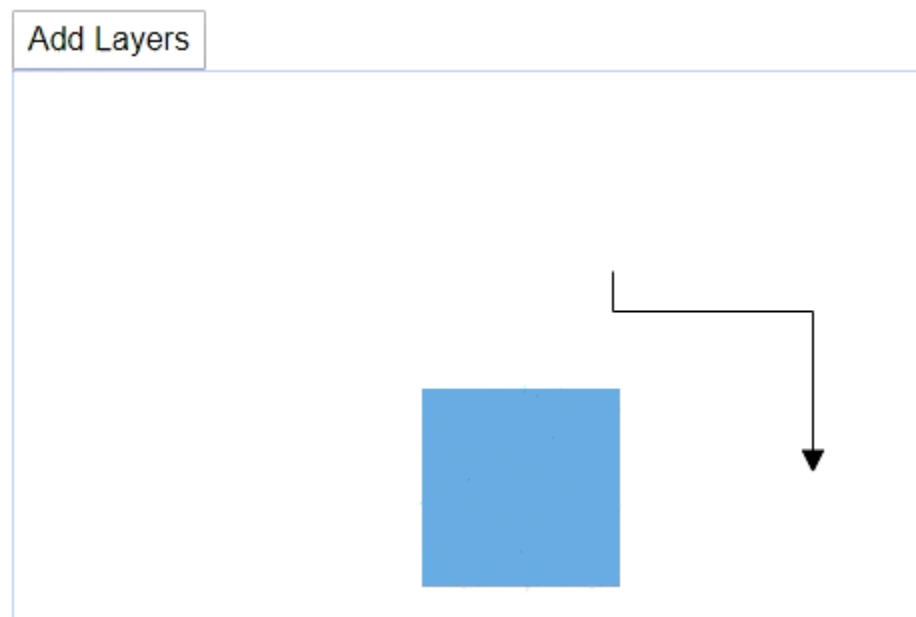
```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Add Layers" @onclick="@AddLayerCollection" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" Layers="@LayersCollection">
</SfDiagram>
@code{
    SfDiagram Diagram;
    SnapConstraints Constraints;
    public ObservableCollection<DiagramNode> NodeCollection;
    public ObservableCollection<DiagramConnector> ConnectorCollection;
    public ObservableCollection<DiagramLayer> LayersCollection;
    protected override void OnInitialized()
    {
        Constraints = SnapConstraints.None;
        // A node is created and stored in node's collection.
        NodeCollection = new ObservableCollection<DiagramNode>()
        {
            new DiagramNode()
            {
                Id = "Node1",
                // Position of the node
                OffsetX = 250, OffsetY = 250,
                // Size of the node
                Width = 100, Height = 100,
                Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
```



```

}
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
    new DiagramConnector()
    {
        Id = "Connector1",
        SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
        TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
        Type = Segments.Orthogonal
    }
};
LayersCollection = new ObservableCollection<DiagramLayer>()
{
    new DiagramLayer() { Id = "Layer1", Objects = new string[] { "Node1",
        "Connector1" } }
};
// Add layer at runtime
public void AddLayerCollection()
{
    DiagramNode Node1 = new DiagramNode()
    {
        Id = "New Node1",
        OffsetX = 100,
        OffsetY = 100,
        Width = 100,
        Height = 100,
        Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
    };
    Diagram.AddLayer(new DiagramLayer() { Id = "Layer2" }, new Object[] { Node1
    });
}
}

```



### *Remove the layer*

You can remove the layers in a diagram by using the `RemoveLayers` method. The following code shows how to remove the layers to the diagram.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Remove Layers" @onclick="@RemoveLayers" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" Layers="@LayersCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
SnapConstraints Constraints;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
public ObservableCollection<DiagramLayer> LayersCollection;
protected override void OnInitialized()
{
Constraints = SnapConstraints.None;
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 250, OffsetY = 250,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
}
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
new DiagramConnector()
{
Id = "Connector1",
SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
Type = Segments.Orthogonal
}
};
LayersCollection = new ObservableCollection<DiagramLayer>()
{
new DiagramLayer() { Id = "Layer1", Objects = new string[] { "Node1",
"Connector1" } }
};
// Remove layer at runtime
public void RemoveLayers()
{
Diagram.RemoveLayer("Layer1");
}
}
```

Remove Layers



### Clone the layer

You can clone the layers in a diagram by using the `CloneLayer` method. The following code shows how to clone the layers to the diagram.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Clone Layers" @onclick="@CloneLayers" />
<SfDiagram @ref="@Diagram" Height="600px"
Nodes="@NodeCollection" Connectors="@ConnectorCollection"
Layers="@LayersCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
SnapConstraints Constraints;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
public ObservableCollection<DiagramLayer> LayersCollection;
protected override void OnInitialized()
{
Constraints = SnapConstraints.None;
// A node is created and stored in node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 250, OffsetY = 250,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
}
};
};
```

```

ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
    new DiagramConnector()
    {
        Id = "Connector1",
        SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
        TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
        Type = Segments.Orthogonal
    }
};
LayersCollection = new ObservableCollection<DiagramLayer>()
{
    new DiagramLayer() { Id = "Layer1", Objects = new string[] { "Node1",
        "Connector1" } }
};
// Clone layer at runtime
public void CloneLayers()
{
    Diagram.CloneLayer("Layer1");
}
}

```

### Clone Layers



### Get the active layer

You can get the active layer in a diagram by using the `GetActiveLayer` method. The following code shows how to find the active layer of the diagram.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Get Active Layers" @onclick="@GetActiveLayer" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" Layers="@LayersCollection">

```

```
</SfDiagram>
@code{
SfDiagram Diagram;
SnapConstraints Constraints;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
public ObservableCollection<DiagramLayer> LayersCollection;
protected override void OnInitialized()
{
Constraints = SnapConstraints.None;
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 250, OffsetY = 250,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
Id = "Node2",
// Position of the node
OffsetX = 300, OffsetY = 200,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
}
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
new DiagramConnector()
{
Id = "Connector1",
SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
Type = Segments.Orthogonal
}
};
LayersCollection = new ObservableCollection<DiagramLayer>()
{
new DiagramLayer() { Id = "Layer1", Objects = new string[] { "Node1",
"Connector1" } },
new DiagramLayer() { Id = "Layer2", Objects = new string[] { "Node2" } }
};
// Get active layer at runtime
public void GetActiveLayer()
{
Diagram.GetActiveLayer();
}
}
```

### Set the active layer

You can set the active layer in a diagram by using the `SetActiveLayer` method. The following code shows how to set the active layer of the diagram.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Set Active Layer" @onclick="@SetActiveLayer" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" Layers="@LayersCollection">
</SfDiagram>
@code{
    SfDiagram Diagram;
    SnapConstraints Constraints;
    public ObservableCollection<DiagramNode> NodeCollection;
    public ObservableCollection<DiagramConnector> ConnectorCollection;
    public ObservableCollection<DiagramLayer> LayersCollection;
    protected override void OnInitialized()
    {
        Constraints = SnapConstraints.None;
        // A node is created and stored in node's collection.
        NodeCollection = new ObservableCollection<DiagramNode>()
        {
            new DiagramNode()
            {
                Id = "Node1",
                // Position of the node
                OffsetX = 250, OffsetY = 250,
                // Size of the node
                Width = 100, Height = 100,
                Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
            },
            new DiagramNode()
            {
                Id = "Node2",
                // Position of the node
                OffsetX = 250, OffsetY = 250,
                // Size of the node
                Width = 100, Height = 100,
                Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
            }
        };
        ConnectorCollection = new ObservableCollection<DiagramConnector>()
        {
            new DiagramConnector()
            {
                Id = "Connector1",
                SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
                TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
                Type = Segments.Orthogonal
            }
        };
        LayersCollection = new ObservableCollection<DiagramLayer>()
        {
            new DiagramLayer() { Id = "Layer1", Objects = new string[] { "Node1",
                "Connector1" } },
        }
```

```

new DiagramLayer() { Id = "Layer2", Objects = new string[] { "Node2" } }
};
}
// Set active layer at runtime
public async void SetActiveLayer()
{
    DiagramLayer layer = await Diagram.GetActiveLayer();
    await Diagram.SetActiveLayer("Layer1");
    layer = await Diagram.GetActiveLayer();
}
}

```

### *Bring the layer to forward*

You can change the layer order, to get the layer forward by using the **BringLayerForward** method. The following code shows how to bring forward the particular layer in the diagram.

### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Bring Layer To Forward"
@onclick="@BringLayerToForward" />
<SfDiagram @ref="@Diagram" Height="600px"
Nodes="@NodeCollection" Connectors="@ConnectorCollection"
Layers="@LayersCollection">
</SfDiagram>
@code{
    SfDiagram Diagram;
    SnapConstraints Constraints;
    public ObservableCollection<DiagramNode> NodeCollection;
    public ObservableCollection<DiagramConnector> ConnectorCollection;
    public ObservableCollection<DiagramLayer> LayersCollection;
    protected override void OnInitialized()
    {
        Constraints = SnapConstraints.None;
        // A node is created and stored in the node's collection.
        NodeCollection = new ObservableCollection<DiagramNode>()
        {
            new DiagramNode()
            {
                Id = "Node1",
                // Position of the node
                OffsetX = 250, OffsetY = 250,
                // Size of the node
                Width = 100, Height = 100,
                Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
            },
            new DiagramNode()
            {
                Id = "Node2",
                // Position of the node
                OffsetX = 250, OffsetY = 250,
                // Size of the node
                Width = 100, Height = 100,
                Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
            }
        }
    }
}

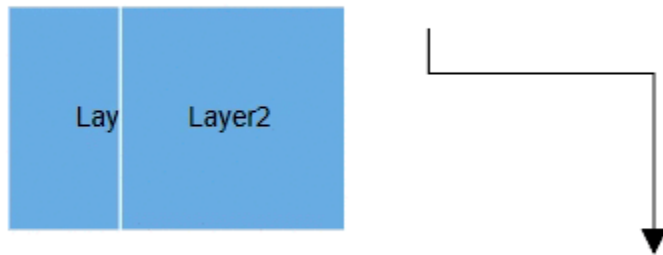
```

```

};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
    new DiagramConnector()
    {
        Id = "Connector1",
        SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
        TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
        Type = Segments.Orthogonal
    }
};
LayersCollection = new ObservableCollection<DiagramLayer>()
{
    new DiagramLayer() { Id = "Layer1", Objects = new string[] { "Node1",
        "Connector1" } },
    new DiagramLayer() { Id = "Layer2", Objects = new string[] { "Node2" } }
};
// Bring layer to forward
public void BringLayerToForward()
{
    Diagram.BringLayerForward("Layer1");
}
}

```

### Bring Layer To Forward



#### *Send the layer to backward*

You can change the layer order, to get the layer forward by using the `SendLayerBackward` method. The following code shows how to bring forward the particular layer in the diagram.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel

```



```

<input type="button" value="Send Layer To Backward"
@onclick="@SendLayerToBackward" />
<SfDiagram @ref="@Diagram" Height="600px"
Nodes="@NodeCollection" Connectors="@ConnectorCollection"
Layers="@LayersCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
SnapConstraints Constraints;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
public ObservableCollection<DiagramLayer> LayersCollection;
protected override void OnInitialized()
{
Constraints = SnapConstraints.None;
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 250, OffsetY = 250,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
Id = "Node2",
// Position of the node
OffsetX = 250, OffsetY = 250,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
}
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
new DiagramConnector()
{
Id = "Connector1",
SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
Type = Segments.Orthogonal
}
};
LayersCollection = new ObservableCollection<DiagramLayer>()
{
new DiagramLayer() { Id = "Layer1", Objects = new string[] { "Node1",
"Connector1" } },
new DiagramLayer() { Id = "Layer2", Objects = new string[] { "Node2" } }
};
// Add node at runtime
public void SendLayerToBackward()
{

```

```
Diagram.SendLayerBackward("Layer2");
}
}
```

Send Layer To Backward



#### *Move object from one to another layer*

You can move node or connector from one layer to another layer by using the [MoveObjects](#) method. The following code shows how to move an object from one layer to another.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Move Objects" @onclick="@MoveObjectsLayer" />
<SfDiagram @ref="@Diagram" Height="600px"
Nodes="@NodeCollection" Connectors="@ConnectorCollection"
Layers="@LayersCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
SnapConstraints Constraints;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
public ObservableCollection<DiagramLayer> LayersCollection;
protected override void OnInitialized()
{
Constraints = SnapConstraints.None;
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 250, OffsetY = 250,
// Size of the node
```

```

Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
    Id = "Node2",
    // Position of the node
    OffsetX = 250, OffsetY = 250,
    // Size of the node
    Width = 100, Height = 100,
    Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
    new DiagramConnector()
    {
        Id = "Connector1",
        SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
        TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
        Type = Segments.Orthogonal
    }
};
LayersCollection = new ObservableCollection<DiagramLayer>()
{
    new DiagramLayer() { Id = "Layer1", Objects = new string[] { "Node1",
    "Connector1" } },
    new DiagramLayer() { Id = "Layer2", Objects = new string[] { "Node2" } }
};
// Move object one layer to another layer
public async void MoveObjectsLayer()
{
    List<string> nodelist = new List<string>() { "Node1" };
    await Diagram.MoveObjects(nodelist, "Layer2");
}
}

```

## Layout

### *Refresh the layout*

Diagram allows you to refresh the layout at runtime. Use the following code example to refresh the layout.

### **CSHARP**

```

//update the layout at runtime.
diagram.DoLayout();
//Here, diagram is instance of SfDiagram.

```

## Sizing Commands

A sizing commands enable to equally size the selected nodes with respect to the first selected object.

**SizingOptions** are as follows:

- Width: Scales the width of the selected objects.
- Height: Scales the height of the selected objects.
- Size: Scales the selected objects both vertically and horizontally.

The following code example shows how to execute the SameSize commands.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Same Width" @onclick="@OnSameWidth" />
<input type="button" value="Same Height" @onclick="@OnSameHeight" />
<input type="button" value="Same Size" @onclick="@OnSameSize" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection;
protected override void OnInitialized()
{
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 100, OffsetY = 100,
// Size of the node
Width = 70, Height = 40,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
Id = "Node2",
// Position of the node
OffsetX = 100, OffsetY = 300,
// Size of the node
Width = 60, Height = 80,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
Id = "Node3",
// Position of the node
OffsetX = 100, OffsetY = 200,
// Size of the node
Width = 50, Height = 50,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
Id = "Node4",
// Position of the node
OffsetX = 200, OffsetY = 250,
// Size of the node
```

```
Width = 70, Height = 90,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
}
// Same width
public async void OnSameWidth()
{
    await Diagram.Select(new ObservableCollection<DiagramNode>() {
        Diagram.Nodes[0], Diagram.Nodes[1], Diagram.Nodes[2] }, true);
    Syncfusion.Blazor.Diagrams.SizingOptions sizingOptions =
        Syncfusion.Blazor.Diagrams.SizingOptions.Width;
    await Diagram.SameSize(sizingOptions);
}
// Same Height
public async void OnSameHeight()
{
    await Diagram.Select(new ObservableCollection<DiagramNode>() {
        Diagram.Nodes[0], Diagram.Nodes[1], Diagram.Nodes[2] }, true);
    //Changing the selected nodes to same height
    Syncfusion.Blazor.Diagrams.SizingOptions sizingOptions =
        Syncfusion.Blazor.Diagrams.SizingOptions.Height;
    await Diagram.SameSize(sizingOptions);
}
// Same Height
public async void OnSameSize()
{
    await Diagram.SelectAll();
    //Changing the selected nodes to same size
    Syncfusion.Blazor.Diagrams.SizingOptions sizingOptions =
        Syncfusion.Blazor.Diagrams.SizingOptions.Size;
    await Diagram.SameSize(sizingOptions);
}
}
```



### Alignment commands

Alignment commands enable you to align the selected or defined objects such as nodes and connectors with respect to the selection boundary. Refer to the Align commands that shows how to use align methods in the diagram.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Alignment" @onclick="@OnAlign" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection;
protected override void OnInitialized()
{
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 50, OffsetY = 100,
// Size of the node
Width = 70, Height = 40,
```

```
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
    Id = "Node2",
    // Position of the node
    OffsetX = 292, OffsetY = 293,
    // Size of the node
    Width = 60, Height = 80,
    Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
    Id = "Node3",
    // Position of the node
    OffsetX = 166, OffsetY = 184,
    // Size of the node
    Width = 50, Height = 50,
    Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
    Id = "Node4",
    // Position of the node
    OffsetX = 435, OffsetY = 197,
    // Size of the node
    Width = 70, Height = 90,
    Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
}
};
}
public async void OnAlign()
{
    Diagram.SelectAll();
    //Aligns the selected items to top
    Diagram.Align(AlignmentOptions.Top, null, AlignmentMode.Selector);
}
}
```

Alignment



### Distribution commands

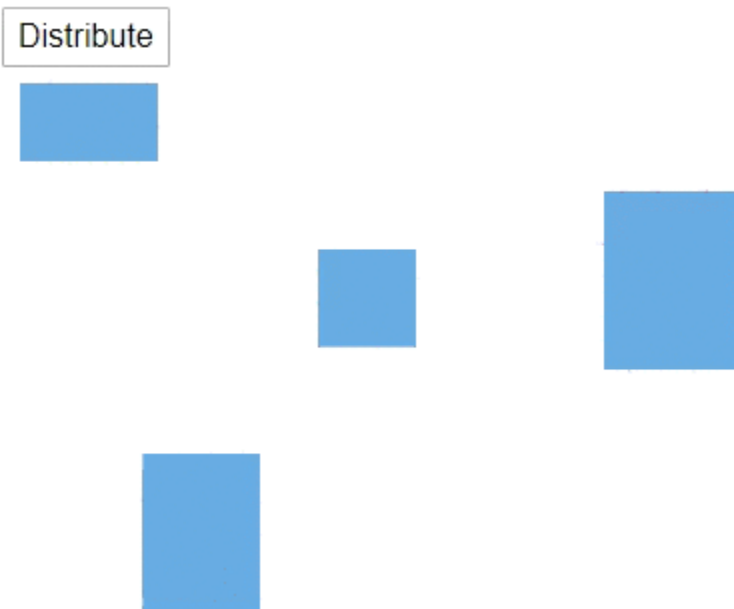
The Distribute commands enable to place the selected objects on the page at equal intervals from each other. The selected objects are equally spaced within the selection boundary.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Distribute" @onclick="@OnDistribution" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection;
protected override void OnInitialized()
{
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 100, OffsetY = 100,
// Size of the node
Width = 70, Height = 40,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
Id = "Node2",
// Position of the node
OffsetX = 156, OffsetY = 305,
// Size of the node
Width = 60, Height = 80,
```



```
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
    Id = "Node3",
    // Position of the node
    OffsetX = 239, OffsetY = 188,
    // Size of the node
    Width = 50, Height = 50,
    Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
    Id = "Node4",
    // Position of the node
    OffsetX = 392, OffsetY = 179,
    // Size of the node
    Width = 70, Height = 90,
    Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
}
};
}
public async void OnDistribution()
{
    Diagram.SelectAll();
    //Distribute the selected items to middle
    Diagram.Distribute(DistributeOptions.Middle);
}
}
```



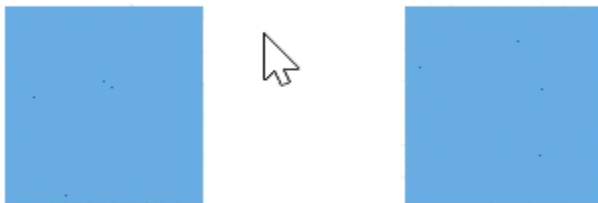
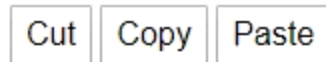
### Clipboard commands

A Clipboard commands are used to cut, copy, or paste the selected elements. Refer to the following link that shows how to use clipboard methods in the diagram.

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Cut" @onclick="@OnCut" />
<input type="button" value="Copy" @onclick="@OnCopy" />
<input type="button" value="Paste" @onclick="@OnPaste" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection;
protected override void OnInitialized()
{
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 100, OffsetY = 100,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
Id = "Node2",
// Position of the node
OffsetX = 300, OffsetY = 100,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
Id = "Node3",
// Position of the node
OffsetX = 300, OffsetY = 100,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
}
};
}
// Cut the selected node
public async void OnCut()
{
Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[0] },
true);
Diagram.Cut();
}
// Copy the selected node
public async void OnCopy()
{
}
```

```
Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[0] },
true);
Diagram.Copy();
}
// Paste the copied node
public async void OnPaste()
{
Diagram.Paste();
}
}
```



### Grouping commands

Grouping commands are used to group or ungroup the selected elements on the diagram. Refer to the following link that shows how to use grouping commands in the diagram.

- A **Group** method is used to the selected nodes and connectors in the diagram.
- A **UnGroup** method is used to the selected nodes and connectors in the diagram.

The following code shows how to execute the grouping commands.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Group" @onclick="@OnGroup" />
<input type="button" value="UnGroup" @onclick="@OnUnGroup" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
```

```
public ObservableCollection<DiagramLayer> LayersCollection;
protected override void OnInitialized()
{
    // A node is created and stored in the node's collection.
    NodeCollection = new ObservableCollection<DiagramNode>()
    {
        new DiagramNode()
        {
            Id = "Node1",
            // Position of the node
            OffsetX = 250, OffsetY = 250,
            // Size of the node
            Width = 100, Height = 100,
            Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
        }
    };
    ConnectorCollection = new ObservableCollection<DiagramConnector>()
    {
        new DiagramConnector()
        {
            Id = "Connector1",
            SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
            TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
            Type = Segments.Orthogonal
        }
    };
    // Group the selected objects
    public void OnGroup()
    {
        Diagram.SelectAll();
        Diagram.Group();
    }
    // Ungroup the selected group
    public void OnUnGroup()
    {
        Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[1] });
        Diagram.UnGroup();
    }
}
```



### Order commands

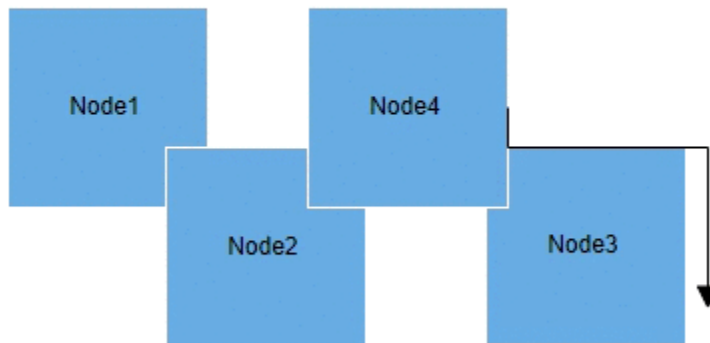
Order commands enable you to visually arrange the selected objects such as nodes and connectors on the page.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Send To Back" @onclick="@OnSendToBack" />
<input type="button" value="Bring To Front" @onclick="@OnBringToFront" />
<input type="button" value="Send BackWard" @onclick="@OnSendBackWard" />
<input type="button" value="Move Forward" @onclick="@OnMoveForward" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
protected override void OnInitialized()
{
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 100, OffsetY = 100,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
Annotations = new ObservableCollection<DiagramNodeAnnotation>()
{
new DiagramNodeAnnotation(){ Id="Annotation1", Content="Node1" }
}
},
},
```

```
new DiagramNode()
{
    Id = "Node2",
    // Position of the node
    OffsetX = 179, OffsetY = 170,
    // Size of the node
    Width = 100, Height = 100,
    Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
    Annotations = new ObservableCollection<DiagramNodeAnnotation>()
    {
        new DiagramNodeAnnotation() { Id="Annotation1", Content="Node2" }
    },
    new DiagramNode()
    {
        Id = "Node3",
        // Position of the node
        OffsetX = 339, OffsetY = 169,
        // Size of the node
        Width = 100, Height = 100,
        Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
        Annotations = new ObservableCollection<DiagramNodeAnnotation>()
        {
            new DiagramNodeAnnotation() { Id="Annotation1", Content="Node3" }
        },
        new DiagramNode()
        {
            Id = "Node4",
            // Position of the node
            OffsetX = 250, OffsetY = 100,
            // Size of the node
            Width = 100, Height = 100,
            Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
            Annotations = new ObservableCollection<DiagramNodeAnnotation>()
            {
                new DiagramNodeAnnotation() { Id="Annotation1", Content="Node4" }
            }
        };
        ConnectorCollection = new ObservableCollection<DiagramConnector>()
        {
            new DiagramConnector()
            {
                Id = "Connector1",
                SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
                TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
                Type = Segments.Orthogonal
            }
        };
    }
    // Send the object to back
    public async void OnSendToBack()
    {
        Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[1] });
        Diagram.SendToBack();
    }
}
```

```
// bring the object to front
public async void OnBringToFront()
{
    Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[1] });
    Diagram.BringToFront();
}
// Send the object to backward
public async void OnSendBackWard()
{
    Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[1] });
    Diagram.SendBackward();
}
// Move the object to forward
public async void OnMoveForward()
{
    Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[1] });
    Diagram.MoveForward();
}
}
```

[Send To Back](#)[Bring To Front](#)[Send BackWard](#)[Move Forward](#)

### Interaction

Diagram provides the support to interact the nodes, connectors, and so on.

#### Selection

- A object can be select at runtime by using the [Select](#) method.
- You can select all the objects in the diagram by using the [SelectAll](#) method.
- You can clear the selected objects by using the [ClearSelection](#) method.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
```

```

<input type="button" value="Select Node" @onclick="@OnSelectNode">
<input type="button" value="Select Connector" @onclick="@OnSelectConnector"
/>
<input type="button" value="Multiple Selection"
@onclick="@OnMultipleSelection">
<input type="button" value="Select All" @onclick="@OnSelectAll">
<input type="button" value="Clear Selection" @onclick="@OnClearSelection" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code {
// reference of the diagram
SfDiagram Diagram;
// To define node collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
// A node is created and stored in the nodes collection.
DiagramNode Node1 = new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 300, OffsetY = 100,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
// Add node
NodeCollection.Add(Node1);
DiagramConnector Connector1 = new DiagramConnector()
{
Id = "Connector1",
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
Fill = "#6f409f",
StrokeColor = "#6f409f",
StrokeWidth = 1
}
},
Style = new ConnectorShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1
},
Type = Segments.Orthogonal
};
ConnectorCollection.Add(Connector1);
}
public void OnSelectNode()
{
// Select the node

```



```

Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[0] },
null);
}
public void OnSelectConnector()
{
    // Select the connector
    Diagram.Select(new ObservableCollection<DiagramConnector>() {
    Diagram.Connectors[0] }, null);
}
public void OnMultipleSelection()
{
    // Select the node
    Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[0] },
    null);
    // Enables the multiple selection
    Diagram.Select(new ObservableCollection<DiagramConnector>() {
    Diagram.Connectors[0] }, true);
}
public void OnSelectAll()
{
    // Select all the elements in the diagram
    Diagram.SelectAll();
}
public void OnClearSelection()
{
    // clear selection in the diagram
    Diagram.ClearSelection();
}
}

```

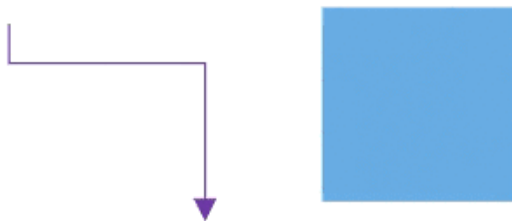
Select Node

Select Connector

Multiple Selection

Select All

Clear Selection



### Drag

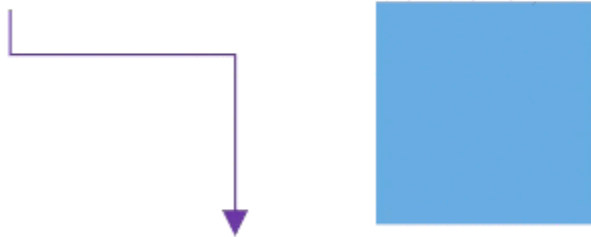
A object can be drag at runtime by using the [Drag](#) method. The following code explains how to drag the node by using the drag method.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
```

```
@using System.Collections.ObjectModel
<input type="button" value="Drag Node" @onclick="@OnDragNode">
<input type="button" value="Drag Connector" @onclick="@OnDragConnector" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code {
// reference of the diagram
SfDiagram Diagram;
// To define node collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>();
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
// A node is created and stored in the nodes collection.
DiagramNode Node1 = new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 300, OffsetY = 100,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
// Add node
NodeCollection.Add(Node1);
DiagramConnector Connector1 = new DiagramConnector()
{
Id = "Connector1",
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle()
{
Fill = "#6f409f",
StrokeColor = "#6f409f",
StrokeWidth = 1
}
},
Style = new ConnectorShapeStyle()
{
StrokeColor = "#6f409f",
StrokeWidth = 1
},
Type = Segments.Orthogonal
};
ConnectorCollection.Add(Connector1);
}
public void OnDragNode()
{
// Drag the node
Diagram.Drag(Diagram.Nodes[0], 10, 10);
```

```
}  
public void OnDragConnector()  
{  
    // Drag the connector  
    Diagram.Drag(Diagram.Connectors[0], 10, 10);  
}  
}
```



### Resize

A node can be resized at runtime by using the [Scale](#) method. The following code explains how to resize the node by using the scale method.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams  
@using System.Collections.ObjectModel  
<input type="button" value="Resize" @onclick="OnResize">  
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">  
</SfDiagram>  
@code {  
    // reference of the diagram  
    SfDiagram Diagram;  
    // To hide the gridlines  
    SnapConstraints constraints = SnapConstraints.None;  
    // To define node collection  
    public ObservableCollection<DiagramNode> NodeCollection = new  
        ObservableCollection<DiagramNode>() { };  
    protected override void OnInitialized()  
    {  
        // A node is created and stored in the nodes collection.  
    }
```

```

DiagramNode node1 = new DiagramNode()
{
    // Position of the node
    OffsetX = 250, OffsetY = 250,
    // Size of the node
    Width = 100, Height = 100,
    Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
// Add node
NodeCollection.Add(node1);
}
public void OnResize()
{
    // Resize the node
    Diagram.Scale(Diagram.Nodes[0], 0.5, 0.5, new PointModel() { X = 0, Y = 0 }
);
}
}

```

Resize



### Rotate

A node can be rotate at runtime by using the [Rotate](#) method. The following code explains how to rotate the node by using method.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Rotate" @onclick="OnRotate">
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection">
</SfDiagram>
@code {
    // reference of the diagram
    SfDiagram Diagram;
    // To hide the gridlines
    SnapConstraints constraints = SnapConstraints.None;

```

```
// To define node collection
public ObservableCollection<DiagramNode> NodeCollection = new
ObservableCollection<DiagramNode>() { };
protected override void OnInitialized()
{
    // A node is created and stored in nodes collection.
    DiagramNode node1 = new DiagramNode()
    {
        // Position of the node
        OffsetX = 250, OffsetY = 250,
        // Size of the node
        Width = 100, Height = 100,
        Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
    };
    // Add node
    NodeCollection.Add(node1);
}
public void OnRotate()
{
    // Rotate the node
    Diagram.Rotate(Diagram.Nodes[0], Diagram.Nodes[0].RotateAngle+10);
}
}
```

Rotate



#### *Drag source end*

You can drag the source point of the connector by using the [DragSourceEnd](#) method. The following code is used to change the source end.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Drag Source End" @onclick="@OnDragSourceEnd">
<SfDiagram @ref="@Diagram" Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
```

```

@code {
// reference of the diagram
SfDiagram Diagram;
// To hide the gridlines
SnapConstraints constraints = SnapConstraints.None;
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
DiagramConnector diagramConnector = new DiagramConnector()
{
Id = "Connector1",
SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
TargetDecorator = new ConnectorTargetDecorator()
{
Shape = DecoratorShapes.Arrow,
Style = new DecoratorShapeStyle() { Fill = "#6f409f", StrokeColor =
"#6f409f", StrokeWidth = 1 }
},
Style = new ConnectorShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1
},
Type = Segments.Orthogonal
};
ConnectorCollection.Add(diagramConnector);
}
public void OnDragSourceEnd()
{
// Drag the source end point
Diagram.DragSourceEnd(Diagram.Connectors[0], 10, 10);
}
}

```

### Drag target end

You can drag the target point of the connector by using the [DragTargetEnd](#) method. The following code is used to change the target end.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Drag Target End" @onclick="@OnDragTargetEnd">
<SfDiagram @ref="@Diagram" Height="600px" Connectors="@ConnectorCollection">
</SfDiagram>
@code {
// reference of the diagram
SfDiagram Diagram;
// To hide the gridlines
SnapConstraints constraints = SnapConstraints.None;
//Defines diagram's connector collection
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
DiagramConnector diagramConnector = new DiagramConnector()

```

```

{
    Id = "Connector1",
    SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
    TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
    TargetDecorator = new ConnectorTargetDecorator()
    {
        Shape = DecoratorShapes.Arrow,
        Style = new DecoratorShapeStyle()
        {
            Fill = "#6f409f",
            StrokeColor = "#6f409f",
            StrokeWidth = 1
        }
    },
    Style = new ConnectorShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1 },
    Type = Segments.Orthogonal
};
ConnectorCollection.Add(diagramConnector);
}
public void OnDragTargetEnd()
{
    // Drag the target end point
    Diagram.DragTargetEnd(Diagram.Connectors[0], 10, 10);
}
}

```

### Nudge

You can move a single or some of the distance of the selected node by using the [Nudge](#) method.

- NudgeDirection - used to consider the direction to move the selected object.
- X, Y - which distance to be moved. By default, X and Y value is set to be 1.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Select Node" @onclick="@OnSelectNode">
<input type="button" value="Select Connector" @onclick="@OnSelectConnector" />
<input type="button" value="Nudge Left" @onclick="@OnNudgeLeft">
<input type="button" value="Nudge Right" @onclick="@OnNudgeRight">
<input type="button" value="Nudge Top" @onclick="@OnNudgeTop" />
<input type="button" value="Nudge Bottom" @onclick="@OnNudgeBottom" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code {
    // reference of the diagram
    SfDiagram Diagram;
    // To define node collection
    public ObservableCollection<DiagramNode> NodeCollection = new
    ObservableCollection<DiagramNode>();
    //Defines diagram's connector collection

```

```
public ObservableCollection<DiagramConnector> ConnectorCollection = new
ObservableCollection<DiagramConnector>();
protected override void OnInitialized()
{
    // A node is created and stored in nodes collection.
    DiagramNode Node1 = new DiagramNode()
    {
        Id = "Node1"
        // Position of the node
        OffsetX = 250, OffsetY = 250,
        // Size of the node
        Width = 100, Height = 100,
        Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
    };
    // Add node
    NodeCollection.Add(Node1);
    DiagramConnector Connector1 = new DiagramConnector()
    {
        Id = "Connector1",
        SourcePoint = new ConnectorSourcePoint() { X = 100, Y = 100 },
        TargetPoint = new ConnectorTargetPoint() { X = 200, Y = 200 },
        TargetDecorator = new ConnectorTargetDecorator()
        {
            Shape = DecoratorShapes.Arrow,
            Style = new DecoratorShapeStyle()
            {
                Fill = "#6f409f",
                StrokeColor = "#6f409f",
                StrokeWidth = 1
            }
        },
        Style = new ConnectorShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1
    },
    Type = Segments.Orthogonal
    };
    ConnectorCollection.Add(Connector1);
}
public void OnSelectNode()
{
    // Select the node
    Diagram.Select(new ObservableCollection<DiagramNode>() { Diagram.Nodes[0] },
    null);
}
public void OnSelectConnector()
{
    // Select the node
    Diagram.Select(new ObservableCollection<DiagramNode>() {
    Diagram.Connectors[0] }, null);
}
public void OnNudgeLeft()
{
    // Selected objects move one step to left
    Diagram.Nudge(NudgeDirection.Left);
}
public void OnNudgeRight()
{
    // Selected objects move one step to right
```



```

Diagram.Nudge(NudgeDirection.Right);
}
public void OnNudgeTop()
{
    // Selected objects move one step to top
    Diagram.Nudge(NudgeDirection.Up);
}
public void OnNudgeBottom()
{
    // Selected objects move one step to bottom
    Diagram.Nudge(NudgeDirection.Down);
}
}

```

Select Node

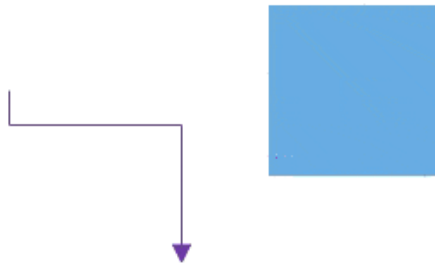
Select Connector

Nudge Left

Nudge Right

Nudge Top

Nudge Bottom


[View Port](#)
[Zoom](#)

You can zoom in or zoom out the diagram. The following code how to zoom the diagram.

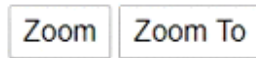
#### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Zoom" @onclick="@OnZoom" />
<input type="button" value="Zoom To" @onclick="@OnZoomRegion" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
    SfDiagram Diagram;
    public ObservableCollection<DiagramNode> NodeCollection;
    public ObservableCollection<DiagramConnector> ConnectorCollection;
    public ObservableCollection<DiagramLayer> LayersCollection;
    protected override void OnInitialized()
    {
        // A node is created and stored in the node's collection.
        NodeCollection = new ObservableCollection<DiagramNode>()
    }
}

```

```
new DiagramNode()
{
    Id = "Node1",
    // Position of the node
    OffsetX = 250, OffsetY = 250,
    // Size of the node
    Width = 100, Height = 100,
    Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
    new DiagramConnector()
    {
        Id = "Connector1",
        SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
        TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
        Type = Segments.Orthogonal
    }
};
// Zoom the diagram
public void OnZoom()
{
    Diagram.Zoom(0.5);
}
public void OnZoomRegion()
{
    ZoomOptions Options = new ZoomOptions()
    {
        FocusPoint = new PointModel()
        {
            X = 100,
            Y = 100
        },
        Type = ZoomTypes.ZoomIn,
        ZoomFactor = 2.5
    };
    Diagram.ZoomTo(Options);
}
```



### *Pan*

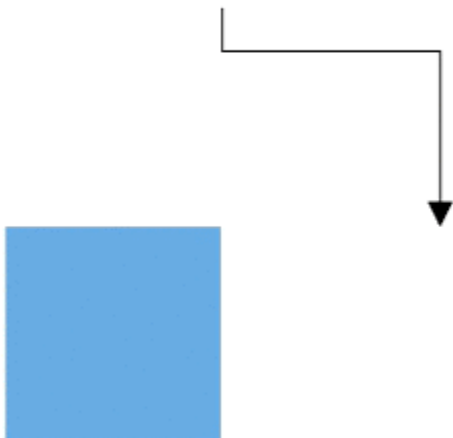
You can move the diagram view port by using the [Pan](#) method. The following code shows how to pan the diagram.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Pan" @onclick="@OnPan" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
public ObservableCollection<DiagramLayer> LayersCollection;
protected override void OnInitialized()
{
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 250, OffsetY = 250,
// Size of the node
Width = 100, Height = 100,
```

```
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }  
};  
ConnectorCollection = new ObservableCollection<DiagramConnector>()  
{  
    new DiagramConnector()  
    {  
        Id = "Connector1",  
        SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },  
        TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },  
        Type = Segments.Orthogonal  
    }  
};  
// Pan the diagram  
public void OnPan()  
{  
    Diagram.Pan(100, 20);  
}
```

Pan

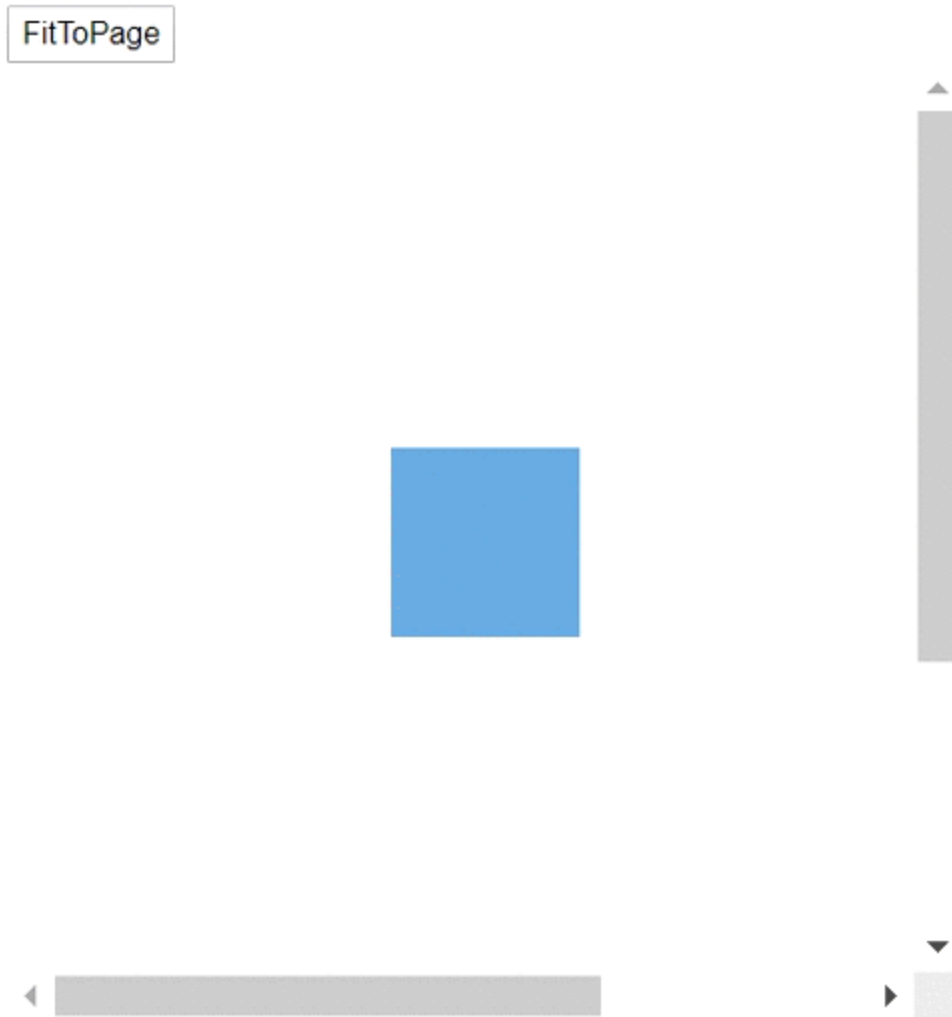


### *Fit to page*

You can fit the diagram elements within the diagram bounds. The following code is used to how to fit all the elements in the diagram area.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="FitToPage" @onclick="@OnFitToPage" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
public ObservableCollection<DiagramLayer> LayersCollection;
protected override void OnInitialized()
{
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 250, OffsetY = 250,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
}
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
new DiagramConnector()
{
Id = "Connector1",
SourcePoint = new ConnectorSourcePoint() { X = 700, Y = 500 },
TargetPoint = new ConnectorTargetPoint() { X = 500, Y = 700 },
Type = Segments.Orthogonal
}
};
// Fit all the elemnts in the diagram view port
public void OnFitToPage()
{
Diagram.FitToPage();
}
}
```



#### Update view port

You can change the size of the diagram area, after that you can update the view port by using the [UpdateViewPort](#) method. The following code shows how to update the view port.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="UpdateViewPort" @onclick="@OnUpdateViewPort" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
public ObservableCollection<DiagramLayer> LayersCollection;
protected override void OnInitialized()
{
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
```

```

{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 250, OffsetY = 250,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
}
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
new DiagramConnector()
{
Id = "Connector1",
SourcePoint = new ConnectorSourcePoint() { X = 700, Y = 500 },
TargetPoint = new ConnectorTargetPoint() { X = 500, Y = 700 },
Type = Segments.Orthogonal
}
};
}
// Update the view port
public void OnUpdateViewPort()
{
Diagram.BeginUpdate();
Diagram.Width = "700px";
Diagram.Height = "700px";
Diagram.EndUpdate();
Diagram.UpdateViewPort();
}
}

```

### *Bring to center*

You can view the particular bounds to center of the view port by using the [BringToCenter](#) method. The following code shows how to use the method.

### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="BringToCenter" @onclick="@OnBringToCenter" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
public ObservableCollection<DiagramLayer> LayersCollection;
protected override void OnInitialized()
{
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>() {
new DiagramNode()
{

```

```

Id = "Node1",
// Position of the node
OffsetX = 250, OffsetY = 250,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
    new DiagramConnector()
    {
        Id = "Connector1",
        SourcePoint = new ConnectorSourcePoint() { X = 700, Y = 500 },
        TargetPoint = new ConnectorTargetPoint() { X = 500, Y = 700 },
        Type = Segments.Orthogonal
    }
};
// Bring into center of the particular bounds
public void OnBringToCenter()
{
    System.Drawing.Rectangle Bounds = new System.Drawing.Rectangle() { X = 500,
    Y = 500, Height = 200, Width = 200 };
    Diagram.BringToCenter(Bounds);
}
}

```

### *Bring to view*

You can view the particular bounds to the view port by using the [BringIntoView](#) method. The following code show how to use the method.

### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="BringToView" @onclick="@OnBringToView" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
    SfDiagram Diagram;
    public ObservableCollection<DiagramNode> NodeCollection;
    public ObservableCollection<DiagramConnector> ConnectorCollection;
    public ObservableCollection<DiagramLayer> LayersCollection;
    protected override void OnInitialized()
    {
        // A node is created and stored in the node's collection.
        NodeCollection = new ObservableCollection<DiagramNode>()
        {
            new DiagramNode()
            {
                Id = "Node1",
                // Position of the node
                OffsetX = 250, OffsetY = 250,
                // Size of the node
            }
        }
    }
}

```



```

Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
    new DiagramConnector()
    {
        Id = "Connector1",
        SourcePoint = new ConnectorSourcePoint() { X = 700, Y = 500 },
        TargetPoint = new ConnectorTargetPoint() { X = 500, Y = 700 },
        Type = Segments.Orthogonal
    }
};
// Bring into view of the particular bounds
public void OnBringToView()
{
    System.Drawing.Rectangle Bounds = new System.Drawing.Rectangle() { X = 500,
    Y = 500, Height = 200, Width = 200 };
    Diagram.BringIntoView(Bounds);
}
}

```

### Get diagram bounds

You can get the diagram bounds by using the [GetDiagramBounds](#) method.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="GetDiagramBounds" @onclick="@OnGetDiagramBounds"
/>
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
    SfDiagram Diagram;
    public ObservableCollection<DiagramNode> NodeCollection;
    public ObservableCollection<DiagramConnector> ConnectorCollection;
    public ObservableCollection<DiagramLayer> LayersCollection;
    protected override void OnInitialized()
    {
        // A node is created and stored in the node's collection.
        NodeCollection = new ObservableCollection<DiagramNode>()
        {
            new DiagramNode()
            {
                Id = "Node1",
                // Position of the node
                OffsetX = 250, OffsetY = 250,
                // Size of the node
                Width = 100, Height = 100,
                Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
            }
        };
    }
}

```

```

ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
    new DiagramConnector()
    {
        Id = "Connector1",
        SourcePoint = new ConnectorSourcePoint() { X = 700, Y = 500 },
        TargetPoint = new ConnectorTargetPoint() { X = 500, Y = 700 },
        Type = Segments.Orthogonal
    }
};
}
// Get the diagram bounds
public async void OnGetDiagramBounds()
{
    Object Bounds = await Diagram.GetDiagramBounds();
}
}

```

### Clear

You can clear all the elements in diagram by using the [Clear](#) method.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="UpdateViewPort" @onclick="@OnUpdateViewPort" />
<input type="button" value="Refresh" @onclick="@OnRefresh" />
<input type="button" value="Clear" @onclick="@OnClear" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
    SfDiagram Diagram;
    public ObservableCollection<DiagramNode> NodeCollection;
    public ObservableCollection<DiagramConnector> ConnectorCollection;
    public ObservableCollection<DiagramLayer> LayersCollection;
    protected override void OnInitialized()
    {
        // A node is created and stored in the node's collection.
        NodeCollection = new ObservableCollection<DiagramNode>()
        {
            new DiagramNode()
            {
                Id = "Node1",
                // Position of the node
                OffsetX = 250, OffsetY = 250,
                // Size of the node
                Width = 100, Height = 100,
                Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
            }
        };
        ConnectorCollection = new ObservableCollection<DiagramConnector>()
        {
            new DiagramConnector()
            {
                Id = "Connector1",

```

```
SourcePoint = new ConnectorSourcePoint() { X = 700, Y = 500 },
TargetPoint = new ConnectorTargetPoint() { X = 500, Y = 700 },
Type = Segments.Orthogonal
}
};
}
// Bring into view of the particular bounds
public void OnUpdateViewPort()
{
    Diagram.BeginUpdate();
    Diagram.Width = "700px";
    Diagram.Height = "700px";
    Diagram.EndUpdate();
    Diagram.UpdateViewPort();
}
// Clear all objects in diagram
public void OnClear()
{
    Diagram.Clear();
}
}
```



## History

### Undo

You can reverse the last action that was performed by using the [Undo](#) method.

### **CSHARP**

```
Diagram.Undo();
```

### Redo

You can restore the last action that was reverted by using the [Redo](#) method.

**CSHARP**

```
Diagram.Redo();
```

*Group action*

- A [StartGroupAction](#) method is used to starts to group the changes to revert or restore them in a single reverse(undo) or restore(redo).
- A [EndGroupAction](#) used to end to group the changes to revert or restore them.

**CSHARP**

```
// Start group action in history  
Diagram.StartGroupAction();  
// End group action in history  
Diagram.EndGroupAction();
```

*Get history stack*

You can get the number of undo or redo actions to be stored on the history list.

**CSHARP**

```
// Get the undo list  
List<HistoryEntry> EntryStack = await Diagram.GetHistoryStack(true);  
// Get the redo list  
EntryStack = await Diagram.GetHistoryStack(false);
```

*Set stack limit*

You can restrict the undo and redo actions to a certain limit by using the [SetStackLimit](#) method.

**CSHARP**

```
Diagram.SetStackLimit(5);
```

*Clear history*

You can clear the actions that is recorded to perform undo or redo operation in the diagram by using the [ClearHistory](#) method.

**CSHARP**

```
Diagram.ClearHistory();
```

*Add the custom history entry*

You can add the history entry at runtime by using the [AddCustomHistoryEntry](#) and perform some actions.

**CSHARP**

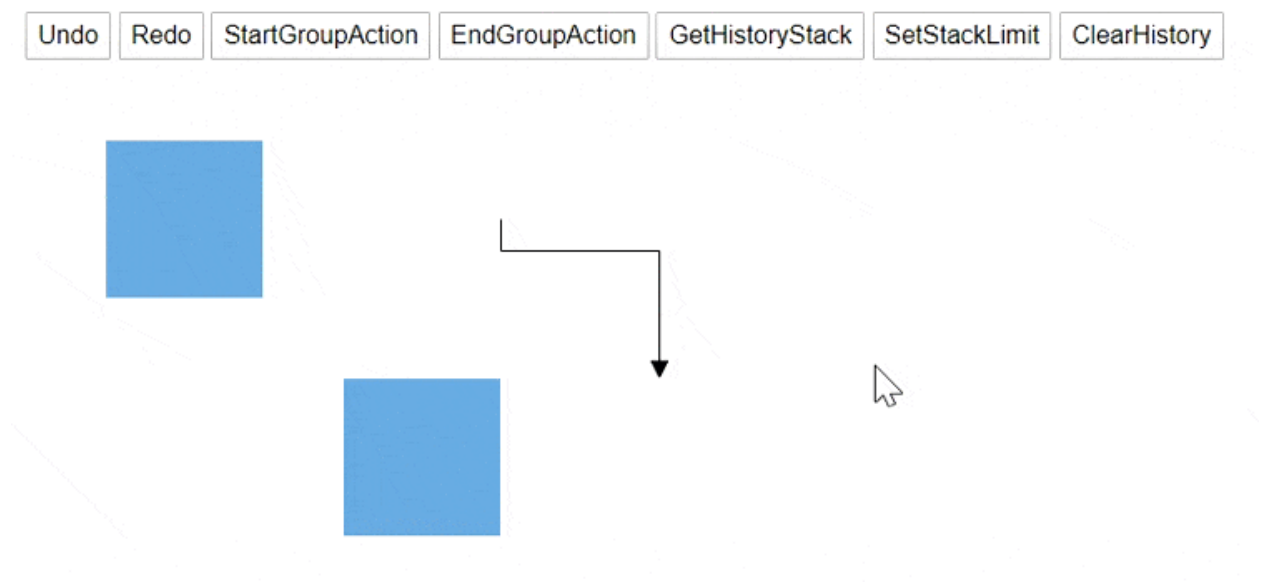
```
HistoryEntry Entry = new HistoryEntry() { BlazorHistoryEntryType =  
HistoryEntryType.Object, Category = EntryCategory.External };  
Diagram.AddCustomHistoryEntry(Entry);
```

The following code shows how to manage and get the history list.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="Undo" @onclick="@OnUndo" />
<input type="button" value="Redo" @onclick="@OnRedo" />
<input type="button" value="StartGroupAction" @onclick="@OnStartGroupAction" />
<input type="button" value="EndGroupAction" @onclick="@OnEndGroupAction" />
<input type="button" value="GetHistoryStack" @onclick="@OnGetHistoryStack" />
<input type="button" value="SetStackLimit" @onclick="@OnSetStackLimit" />
<input type="button" value="ClearHistory" @onclick="@OnClearHistory" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection">
</SfDiagram>
@code{
SfDiagram Diagram;
SnapConstraints Constraints;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
public ObservableCollection<DiagramLayer> LayersCollection;
protected override void OnInitialized()
{
Constraints = SnapConstraints.None;
// A node is created and stored in the node's collection.
NodeCollection = new ObservableCollection<DiagramNode>()
{
new DiagramNode()
{
Id = "Node1",
// Position of the node
OffsetX = 100, OffsetY = 100,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
},
new DiagramNode()
{
Id = "Node2",
// Position of the node
OffsetX = 250, OffsetY = 250,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
}
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
new DiagramConnector()
{
Id = "Connector1",
SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
Type = Segments.Orthogonal
}
```

```
}  
};  
}  
// Undo changes  
public void OnUndo()  
{  
    Diagram.Undo();  
}  
// Redo changes  
public void OnRedo()  
{  
    Diagram.Redo();  
}  
// Start group action in history  
public void OnStartGroupAction()  
{  
    Diagram.StartGroupAction();  
}  
// End group action in history  
public void OnEndGroupAction()  
{  
    Diagram.EndGroupAction();  
}  
public async void OnGetHistoryStack(){  
    // Get the undo list  
    List<HistoryEntry> EntryStack = await Diagram.GetHistoryStack(true);  
    // Get the redo list  
    EntryStack = await Diagram.GetHistoryStack(false);  
}  
public void OnSetStackLimit(){  
    Diagram.SetStackLimit(5);  
}  
public void OnClearHistory(){  
    Diagram.ClearHistory();  
}  
}
```



### Tool tip

You can show and hide the tooltip by using the [ShowTooltip](#) and [HideTooltip](#) method.

### CSHARP

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
<input type="button" value="ShowTooltip" @onclick="@OnShowTooltip" />
<input type="button" value="HideTooltip" @onclick="@OnHideTooltip" />
<SfDiagram @ref="@Diagram" Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" Tooltip="@Tooltips">
</SfDiagram>
@code{
SfDiagram Diagram;
public ObservableCollection<DiagramNode> NodeCollection;
public ObservableCollection<DiagramConnector> ConnectorCollection;
DiagramTooltip Tooltips = new DiagramTooltip()
{
    Content = "Tooltip",
    Position = Syncfusion.Blazor.Popups.Position.BottomCenter,
    RelativeMode = TooltipRelativeMode.Object,
    Animation = new Syncfusion.Blazor.Popups.AnimationModel()
    {
        Open = new Syncfusion.Blazor.Popups.TooltipAnimationSettings() { Effect =
Syncfusion.Blazor.Popups.Effect.FadeZoomIn, Delay = 1 },
        Close = new Syncfusion.Blazor.Popups.TooltipAnimationSettings() { Effect =
Syncfusion.Blazor.Popups.Effect.FadeZoomOut, Delay = 0 },
    };
protected override void OnInitialized()
{
    // A node is created and stored in the node's collection.
    NodeCollection = new ObservableCollection<DiagramNode>()
    {
        new DiagramNode()
        {
            Id = "Node1",
```

```
// Position of the node
OffsetX = 250, OffsetY = 250,
// Size of the node
Width = 100, Height = 100,
Style = new NodeShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" }
};
ConnectorCollection = new ObservableCollection<DiagramConnector>()
{
    new DiagramConnector()
    {
        Id = "Connector1",
        SourcePoint = new ConnectorSourcePoint() { X = 300, Y = 100 },
        TargetPoint = new ConnectorTargetPoint() { X = 400, Y = 200 },
        Type = Segments.Orthogonal
    }
};
// Show the tooltip
public void OnShowToolTip()
{
    Diagram.ShowToolTip(Diagram.Nodes[0]);
}
// Hide the tooltip
public void OnHideToolTip()
{
    Diagram.HideToolTip(Diagram.Nodes[0]);
}
}
```

### Property change

The diagram allows for changing the property at runtime. You can change more properties in a single update by using [BeginUpdate](#) and [EndUpdate](#) methods.

### CSHARP

```
public void UpdateNodeProperties()
{
    Diagram.BeginUpdate();
    Diagram.Nodes[0].OffsetX += 20;
    Diagram.Nodes[0].OffsetY += 30;
    Diagram.Nodes[0].Style.Fill = "red";
    Diagram.EndUpdate();
}
```

### How To

#### Styling And Appearance in Blazor Diagram Component

To modify the Diagram appearance, you need to override the default CSS of the Diagram. Please find the list of CSS classes in the Diagram.

CSS class	Purpose
----- -----	
.e-diagram-endpoint-handle	Customize the connector endpoint handle.



| .e-diagram-endpoint-handle.e-connected | Customize the connector endpoint handle when connected. |

| .e-diagram-endpoint-handle.e-disabled | Customize the connector endpoint handle when disabled. |

| .e-diagram-bezier-handle | Customize the bezier connector handle. |

| .e-diagram-bezier-line | Customize the bezier connector line. |

| .e-diagram-resize-handle | Customize the resize handle. |

| .e-diagram-pivot-line | Customize the selector pivot line. |

| .e-diagram-border | Customize the selector border. |

| .e-diagram-rotate-handle | Customize the rotate handle. |

| .e-symbolpalette .e-symbol-hover:hover | Customize the symbol palette while hovering. |

| .e-symbolpalette .e-symbol-selected | Customize the symbol palette when selected. |

| .e-diagram .e-ruler | Customize the ruler. |

| .e-diagram .e-ruler-overlap | Customize the ruler overlap. |

| .e-diagram .e-diagram-text-edit | Customize the text edit. |

| .e-diagram-text-edit::selection | Customize the text edit on selection. |

### Blazor WebAssembly Diagram in Blazor Diagram Component

This article provides a step-by-step instructions to configure Syncfusion Blazor Diagram in a simple Blazor WebAssembly application using [Visual Studio 2019](#).

**Note:** 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](#)

**Note:** .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.

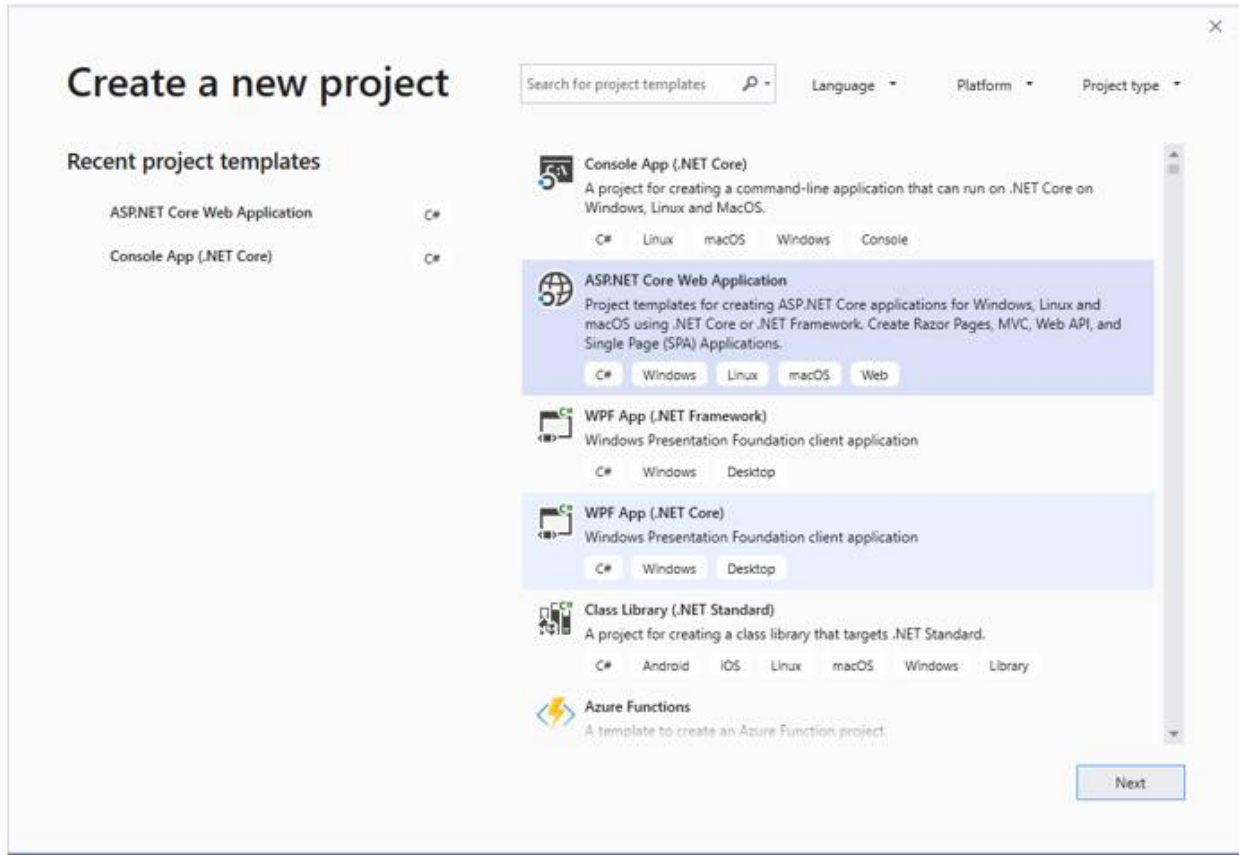
#### Create a Blazor WebAssembly project in Visual Studio 2019

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.

!blazor template](../images/blazor-template.png)

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

!asp.net core project configuration](../images/project-configuration.png)

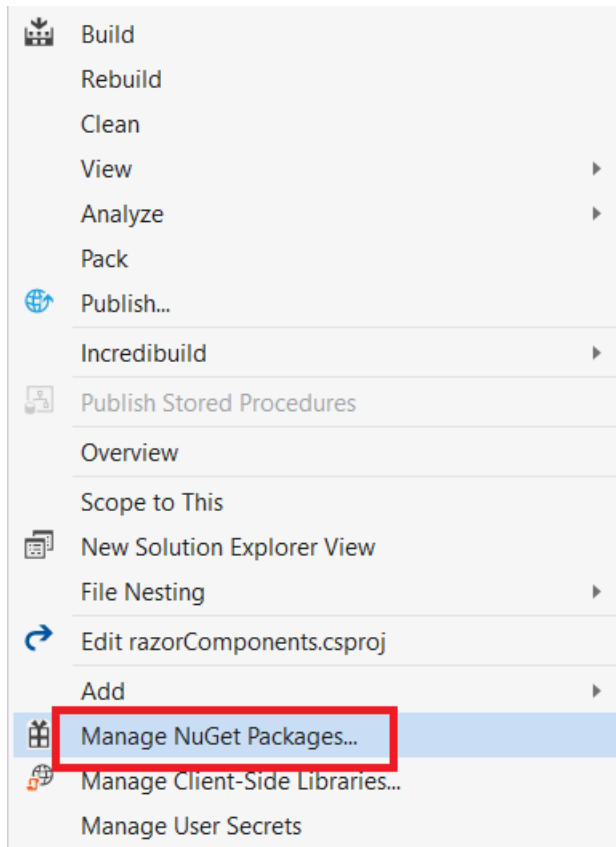
5. Choose **Blazor WebAssembly App** from the dashboard and click **Create** button to create a new Blazor WebAssembly application. Make sure **.NET Core** and **ASP.NET Core 3.1** is selected at the top.

!wasm template](../images/blazor-client-template.png)

**Note:** ASP.NET Core 3.1 is 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.

![select nuget](../images/DiagramPackages.png)

3. The Syncfusion Blazor package will be installed in the project, once the installation process is completed.
4. Install **Syncfusion.EJ2.Blazor** NuGet package to the application using the **NuGet Package Manager**.

Please ensure to check the Include prerelease option for our Beta release.

5. You can add the client-side style resources from NuGet package in the `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.razor` file and import the **Syncfusion.Blazor.Diagram** package.

### ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Diagrams
```

#### *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();
        }
    }
}
```

**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/18.4.42/syncfusion-
blazor.min.js"></script>
</environment>
```

```
</head>
```

### Adding Diagram component to the Application

Diagram component can be rendered by using the **SfDiagram** tag helper in ASP.NET Core Blazor application.

The following example shows a basic Diagram component.

#### ASPX-CS

```
<SfDiagram Width="100%" Height="600px">
</SfDiagram>
```

### Adding Nodes and Connectors

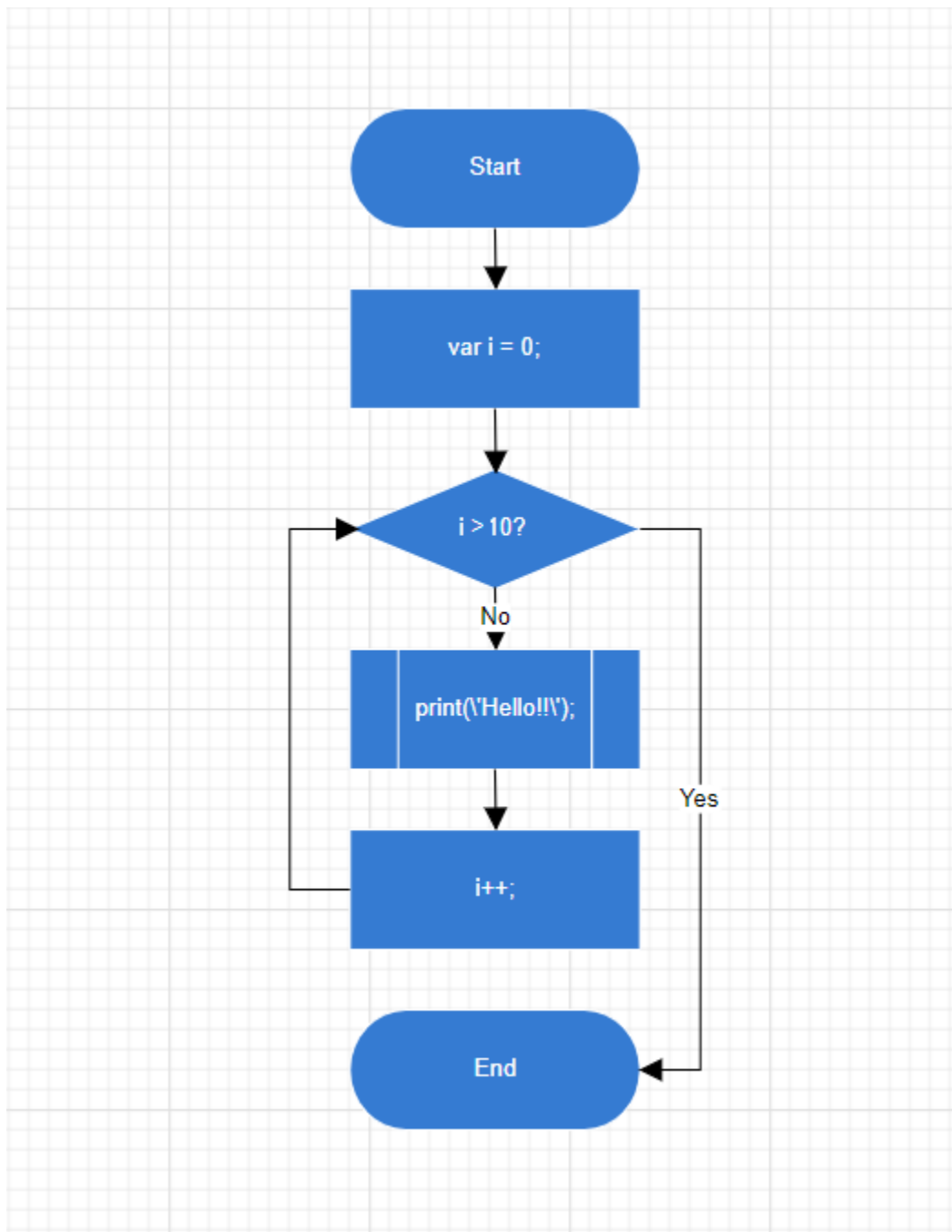
Let us create and add a **Nodes** with specific position, size, label and shape. Connect two or more Nodes by using a **Connectors**.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagrams
@using System.Collections.ObjectModel
@using DiagramShapes = Syncfusion.Blazor.Diagrams.Shapes
@using DiagramSegments = Syncfusion.Blazor.Diagrams.Segments
<SfDiagram Height="600px" Nodes="@NodeCollection"
Connectors="@ConnectorCollection" NodeDefaults="@NodeDefaults"
ConnectorDefaults="@ConnectorDefaults"></SfDiagram>
@code
{
    int connectorCount = 0;
    // Reference to diagram
    SfDiagram diagram;
    // Defines diagram's nodes collection
    public ObservableCollection<DiagramNode> NodeCollection { get; set; }
    // Defines diagram's connector collection
    public ObservableCollection<DiagramConnector> ConnectorCollection { get;
    set; }
    // Defines default values for DiagramNode object
    public DiagramNode NodeDefaults { get; set; }
    // Defines default values for DiagramConnector object
    public DiagramConnector ConnectorDefaults { get; set; }
    protected override void OnInitialized()
    {
        InitDiagramModel();
    }
    private void InitDiagramModel()
    {
        InitDiagramDefaults();
        NodeCollection = new ObservableCollection<DiagramNode>();
        ConnectorCollection = new ObservableCollection<DiagramConnector>();
        CreateNode("Start", 50, FlowShapes.Terminator, "Start");
        CreateNode("Init", 140, FlowShapes.Process, "var i = 0;");
        CreateNode("Condition", 230, FlowShapes.Decision, "i < 10?");
        CreateNode("Print", 320, FlowShapes.PreDefinedProcess,
        "print(\'Hello!!\');");
        CreateNode("Increment", 410, FlowShapes.Process, "i++;");
    }
}
```

```
CreateNode("End", 500, FlowShapes.Terminator, "End");
DiagramConnectorSegment rightSegment = new DiagramConnectorSegment()
{
    Type = DiagramSegments.Orthogonal,
    Length = 30,
    Direction = Direction.Right
};
DiagramConnectorSegment bottomSegment = new DiagramConnectorSegment()
{
    Type = DiagramSegments.Orthogonal,
    Length = 300,
    Direction = Direction.Bottom
};
DiagramConnectorSegment leftSegment = new DiagramConnectorSegment()
{
    Type = DiagramSegments.Orthogonal,
    Length = 30,
    Direction = Direction.Left
};
DiagramConnectorSegment topSegment = new DiagramConnectorSegment()
{
    Type = DiagramSegments.Orthogonal,
    Length = 200,
    Direction = Direction.Top
};
CreateConnector("Start", "Init");
CreateConnector("Init", "Condition");
CreateConnector("Condition", "Print");
CreateConnector("Condition", "End", "Yes", rightSegment, bottomSegment);
CreateConnector("Print", "Increment", "No");
CreateConnector("Increment", "Condition", null, leftSegment, topSegment);
}
private void CreateConnector(string sourceId, string targetId, string label
= default(string), DiagramConnectorSegment rightSegment = null,
DiagramConnectorSegment bottomSegment = null)
{
    DiagramConnector diagramConnector = new DiagramConnector()
    {
        Id = string.Format("connector{0}", ++connectorCount),
        SourceID = sourceId,
        TargetID = targetId
    };
    if (label != default(string))
    {
        var annotation = new DiagramConnectorAnnotation()
        {
            Content = label,
            Style = new AnnotationStyle() { Fill = "white" }
        };
        diagramConnector.Annotations = new
ObservableCollection<DiagramConnectorAnnotation>() { annotation };
    }
    if (rightSegment != null)
    {
        diagramConnector.Segments = new
ObservableCollection<DiagramConnectorSegment>() { rightSegment,
bottomSegment };
    }
}
```

```
}
ConnectorCollection.Add(diagramConnector);
}
private void InitDiagramDefaults()
{
    DiagramNodeAnnotation defaultAnnotation = new DiagramNodeAnnotation()
    {
        Style = new AnnotationStyle()
        {
            Color = "white",
            Fill = "transparent"
        }
    };
    NodeDefaults = new DiagramNode()
    {
        Width = 140,
        Height = 50,
        OffsetX = 300,
        Annotations = new ObservableCollection<DiagramNodeAnnotation>() {
            defaultAnnotation },
        Style = new NodeShapeStyle() { Fill = "#357BD2", StrokeColor = "white" }
    };
    ConnectorDefaults = new DiagramConnector()
    {
        Type = DiagramSegments.Orthogonal,
        TargetDecorator = new ConnectorTargetDecorator() { Shape =
            DecoratorShapes.Arrow, Width = 10, Height = 10 }
    };
}
private void CreateNode(string id, double y, FlowShapes shape, string label,
    bool positionLabel = false)
{
    DiagramNodeAnnotation annotation = new DiagramNodeAnnotation() { Content =
        label };
    if (positionLabel)
    {
        annotation.Margin = new NodeAnnotationMargin() { Left = 25, Right = 25 };
    };
    DiagramNode diagramNode = new DiagramNode()
    {
        Id = id,
        OffsetY = y,
        Shape = new DiagramShape() { Type = DiagramShapes.Flow, FlowShape = shape },
        Annotations = new ObservableCollection<DiagramNodeAnnotation>() { annotation
        }
    };
    NodeCollection.Add(diagramNode);
}
}
```



*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](#)



## Diagram Component

### Getting Started in Blazor Diagram Component

This section briefly explains about how to include a Diagram in your Blazor WebAssembly application. You can refer [Getting Started with Syncfusion Blazor for Blazor WebAssembly in Visual Studio 2019](#) page for the introduction and configuring the common specifications.

#### Importing Syncfusion Blazor component in the application

1. Install **Syncfusion.Blazor.Diagram** NuGet package to the application by using the **NuGet Package Manager**.
2. You can add the style resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the **~/wwwroot/index.cshtml** page.

#### ASPX-CS

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

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
```

#### 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();
    }
}
```

### Adding Diagram component to the Application

Diagram component can be rendered by using the [SfDiagramComponent](#) tag helper in ASP.NET Core Blazor application. Add the Diagram component in any web page razor in the Pages folder. For example, the Diagram component is added in the `~/Pages/Index.razor` page.

The following example shows a basic Diagram component.

#### ASPX-CS

```
<SfDiagramComponent Width="100%" Height="600px"/>
```

### Adding Nodes and Connectors

Let us create and add a [Node](#) with specific position, size, label and shape. Connect two or more nodes by using a [Connector](#).

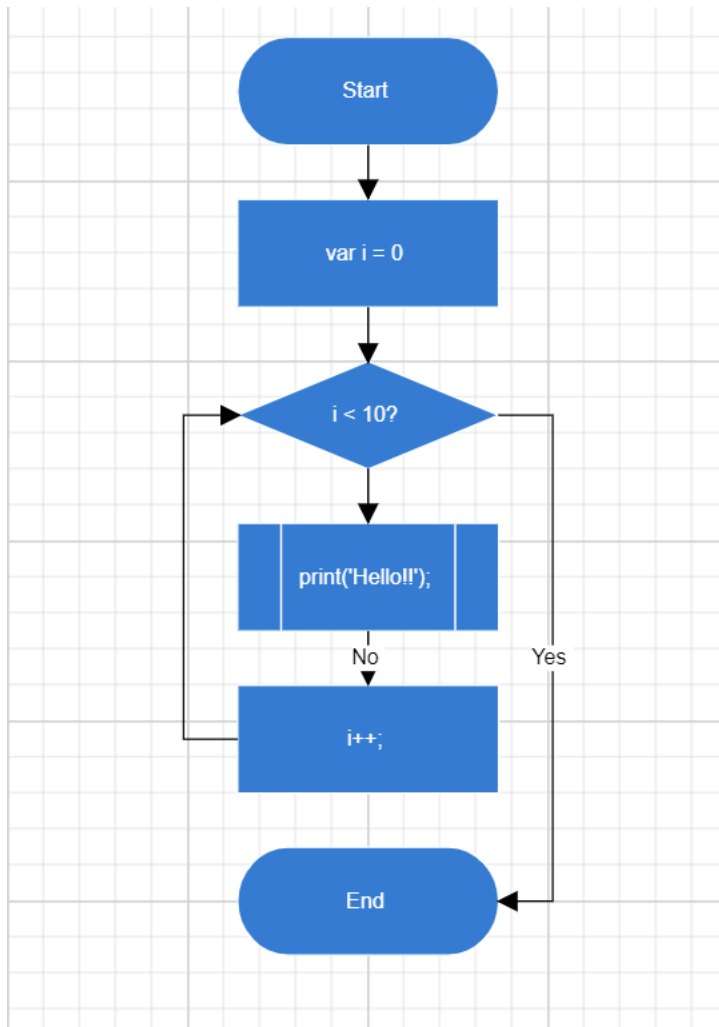
#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="@diagram" Connectors="@connectors" Height="700px"
Nodes="@nodes" />
@code
{
    SfDiagramComponent diagram;
    int connectorCount = 0;
    //Defines Diagram's nodes collection
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    //Defines Diagram's connectors collection
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        InitDiagramModel();
    }
    private void InitDiagramModel()
    {
        CreateNode("Start", 300, 50, FlowShapeType.Terminator, "Start");
        CreateNode("Init", 300, 140, FlowShapeType.Process, "var i = 0");
        CreateNode("Condition", 300, 230, FlowShapeType.Decision, "i < 10?");
        CreateNode("Print", 300, 320, FlowShapeType.PreDefinedProcess,
        "print(\'Hello!!\')");
        CreateNode("Increment", 300, 410, FlowShapeType.Process, "i++;");
        CreateNode("End", 300, 500, FlowShapeType.Terminator, "End");
    }
}
```

```
// Creates orthogonal connector.
OrthogonalSegment segment1 = new OrthogonalSegment()
{
    Type = ConnectorSegmentType.Orthogonal,
    Direction = Direction.Right,
    Length = 30,
};
OrthogonalSegment segment2 = new OrthogonalSegment()
{
    Type = ConnectorSegmentType.Orthogonal,
    Length = 300,
    Direction = Direction.Bottom,
};
OrthogonalSegment segment3 = new OrthogonalSegment()
{
    Type = ConnectorSegmentType.Orthogonal,
    Length = 30,
    Direction = Direction.Left,
};
OrthogonalSegment segment4 = new OrthogonalSegment()
{
    Type = ConnectorSegmentType.Orthogonal,
    Length = 200,
    Direction = Direction.Top,
};
CreateConnector("Start", "Init");
CreateConnector("Init", "Condition");
CreateConnector("Condition", "Print");
CreateConnector("Condition", "End", "Yes", segment1, segment2);
CreateConnector("Print", "Increment", "No");
CreateConnector("Increment", "Condition", null, segment3, segment4);
}
// Method to create connector
private void CreateConnector(string sourceId, string targetId, string label
= default(string), OrthogonalSegment segment1 = null, OrthogonalSegment
segment2 = null)
{
    Connector diagramConnector = new Connector()
    {
        // Represents the unique id of the connector.
        ID = string.Format("connector{0}", ++connectorCount),
        SourceID = sourceId,
        TargetID = targetId,
    };
    if (label != default(string))
    {
        // Represents the annotation of the connector.
        PathAnnotation annotation = new PathAnnotation()
        {
            Content = label,
            Style = new TextStyle() { Fill = "white" }
        };
        diagramConnector.Annotations = new DiagramObjectCollection<PathAnnotation>()
        { annotation };
    }
    if (segment1 != null)
    {

```

```
// Represents the segment type of the connector.
diagramConnector.Type = ConnectorSegmentType.Orthogonal;
diagramConnector.Segments = new DiagramObjectCollection<ConnectorSegment> {
    segment1, segment2 };
}
connectors.Add(diagramConnector);
}
// Method to create node
private void CreateNode(string id, double x, double y, FlowShapeType shape,
string label)
{
    Node diagramNode = new Node()
    {
        //Represents the unique id of the node.
        ID = id,
        // Defines the position of the node.
        OffsetX = x,
        OffsetY = y,
        // Defines the size of the node.
        Width = 145,
        Height = 60,
        // Defines the style of the node.
        Style = new ShapeStyle { Fill = "#357BD2", StrokeColor = "White" },
        // Defines the shape of the node.
        Shape = new FlowShape() { Type = Shapes.Flow, Shape = shape },
        // Defines the annotation collection of the node.
        Annotations = new DiagramObjectCollection<ShapeAnnotation>
        {
            new ShapeAnnotation
            {
                Content = label,
                Style = new TextStyle()
                {
                    Color="White",
                    Fill = "transparent"
                }
            }
        };
    };
    nodes.Add(diagramNode);
}
```



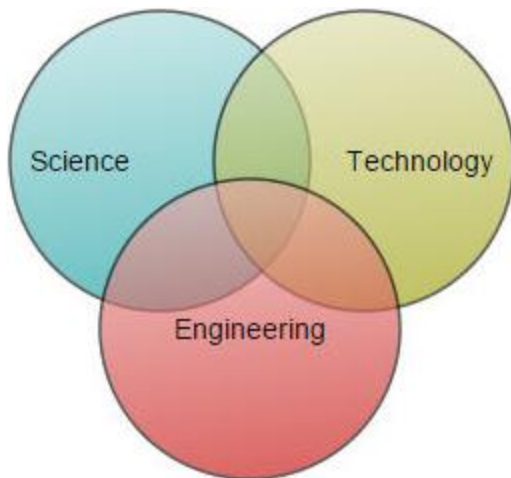
See Also

- [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](#)

## Nodes

### Node in Blazor Diagram Component

Nodes are graphical objects that are used to visually represent the geometrical information, process flow, internal business procedure, entity, or any other kind of data and it represents the functions of a complete system in regards to how it interacts with external entities.



### Create node

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

### Add node through nodes collection

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in the nodes collection.
        Node node = new Node()
        {
            ID = "node1",
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle()
            {
                Fill = "#6495ED",
                StrokeColor = "white"
            }
        };
        // Add node
        nodes.Add(node);
    }
}
```



### *Add nodes at runtime*

You can add a Node at runtime by adding it to the nodes collection of the Diagram component. The following code explains how to add a node at runtime.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<input type="button" value="Add Node" @onclick="@AddNode">
<SfDiagramComponent Width="1000px" Height="500px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node1",
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle() { Fill = "#6495ED" };
        };
        nodes.Add(node);
    }
    public void AddNode()
    {
        Node NewNode = new Node()
        {
            ID = "node2",
            // Position of the node
            OffsetX = 450,
            OffsetY = 450,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle() { Fill = "#6495ED" };
        };
        nodes.Add(NewNode);
    }
}
```

### Add node from palette

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

- Once you drag a node/connector from the palette to the diagram, the following events can be used to do the customization.
- When a symbol is dragged into a diagram from symbol palette, the [DragStart](#) event gets triggered. [DragStartEventArgs](#) notifies when the element enters into the diagram from the symbol palette.
- When a symbol is dragged over a diagram, the [Dragging](#) event gets triggered. [DraggingEventArgs](#) notifies when an element drags over another diagram element.
- When a symbol is dragged and dropped from symbol palette to diagram area, the [DragDrop](#) event gets triggered. [DropEventArgs](#) notifies when the element is dropped on the diagram.
- When a symbol is dragged outside of the diagram, the [DragLeave](#) event gets triggered. [DragLeaveEventArgs](#) notifies when the element leaves the diagram.



### Draw node using drawing object

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

For more information about drawing node, refer to the [Draw Nodes](#).





#### *Create node through data source*

Nodes can be generated automatically with the information provided through data source. The default properties for these nodes are fetched from default settings. For more information about datasource, refer to the [DataSource](#).

#### *Remove nodes at runtime*

A node can be removed from the diagram at runtime by using the **Remove** method.

The following code shows how to remove a node at runtime.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<input type="button" value="Remove Node" @onclick="@RemoveNodes">
<SfDiagramComponent Width="1000px" Height="500px" Nodes="@nodes" />
@code
{
    //Defines diagram's connector collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node1",
            // Position of the node
        }
    }
}
```

```

OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new ShapeStyle()
{
    Fill = "#6495ED",
    StrokeColor = "white"
};
// Add node
nodes.Add(node);
}
public void RemoveNodes()
{
    // Remove Node at runtime
    nodes.Remove(nodes[0]);
}
}

```

A node can be removed from the diagram by using the native **RemoveAt** method. Refer to the following example that shows how to remove the node at runtime.

#### **C#**

```

public void RemoveNodes()
{
    nodes.RemoveAt(0);
}

```

#### *Update nodes at runtime*

You can change any node's properties at runtime.

The following code example explains how to change the node properties.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="Update Node" @onclick="@UpdateNodes">
<SfDiagramComponent @ref="Diagram" Width="1000px" Height="500px"
Nodes="@nodes"/>
@code
{
    SfDiagramComponent Diagram;
    //Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node1",
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,

```

```
// Size of the node
Width = 100,
Height = 100,
Style = new ShapeStyle()
{
    Fill = "#6495ED",
    StrokeColor = "white"
};
nodes.Add(node);
}
public async void UpdateNodes()
{
    Diagram.BeginUpdate();
    Diagram.Nodes[0].Width = 50;
    Diagram.Nodes[0].Height = 50;
    await Diagram.EndUpdate();
}
}
```

[BeginUpdate](#) and [EndUpdate](#) method which allows you to stop the continuous update of control and resume it finally.

*See Also*

- [How to add annotations to the node](#)
- [How to add ports to the node](#)
- [How to enable/disable the behavior of the node](#)

#### Appearance of a Node in Blazor Diagram Component

The appearance of a node can be customized by changing its [Fill](#), [StrokeDashArray](#), [StrokeColor](#), [StrokeWidth](#), and [Shadow](#) properties. The [IsVisible](#) property of the node indicates whether the node is visible or not.

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

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes array.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
```

```
// Add node
Style = new ShapeStyle()
{
    Fill = "Green",
    StrokeDashArray = "5,5",
    StrokeColor = "red",
    StrokeWidth = 2
},
};
nodes.Add(node);
}
}
```



[ID](#) for each node should be unique and so it is further used to find the node at runtime and do any customization.

#### [NodeCreating](#)

Default values for all the Nodes can be set using the [NodeCreating](#) method. For example, if all nodes have the same type or having the same property then such properties can be moved into NodeCreating method.

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

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes"
NodeCreating="@NodeCreating" />
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes array.
        Node node1 = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Shape of the Node
            Shape = new BasicShape()
            {
                Type = Shapes.Basic,
                Shape = BasicShapeType.Rectangle
            }
        }
    }
}
```

```

};
Node node2 = new Node()
{
    // Position of the node
    OffsetX = 100,
    OffsetY = 100,
    // Shape of the Node
    Shape = new BasicShape()
    {
        Type = Shapes.Basic,
        Shape = BasicShapeType.Ellipse
    }
};
nodes.Add(node1);
nodes.Add(node2);
}
private void NodeCreating(IDiagramObject obj)
{
    Node node = obj as Node;
    node.Style = new ShapeStyle() { Fill = "#6495ED" };
    // Size of the node
    node.Width = 100;
    node.Height = 100;
}
}

```

### NodeTemplate

We can define node style using template in [NodeTemplate](#) at tag level. If we want to define separate template for each node, differentiate the nodes by the ID property. The following code explains how to define template for a node.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="diagram" Width="1200px" Height="1000px"
Nodes="@nodes">
    <DiagramTemplates>
        <NodeTemplate>
            @{
                <style>
                    th {
                        border: 5px solid #c1dad7
                    }
                    td {
                        border: 5px solid #c1dad7
                    }
                    .c1 {
                        background: transparent
                    }
                    .c2 {
                        background: transparent
                    }
                    .c3 {
                        background: transparent
                    }
                    .c4 {

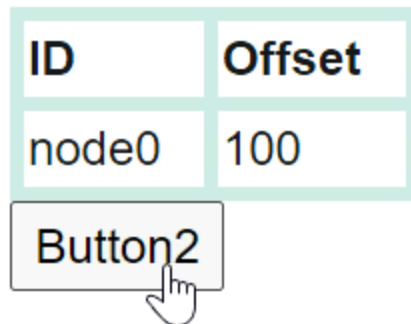
```

```

background: transparent
}
</style>
<div id="node0_content_html_element" class="foreign-object" style="height:
164px; width: 137px; left: 0px; top: 0px; position: absolute; transform:
scale(1, 1) rotate(0deg); pointer-events: all; visibility: visible; opacity:
1;">
<div style="height: 100%; width: 100%;">
<table style="width:100%;">
<tbody>
<tr><th class="c1">ID</th><th class="c2">Offset</th></tr>
<tr><td class="c1">node0</td><td class="c2">100</td></tr>
</tbody>
</table>
<input type="button" value="Button2" @onclick="@OnClick" />
</div>
</div>
}
</NodeTemplate>
</DiagramTemplates>
</SfDiagramComponent>
@code
{
SfDiagramComponent diagram;
DiagramObjectCollection<Node> nodes;
protected override void OnInitialized()
{
nodes = new DiagramObjectCollection<Node>();
Node node = new Node()
{
Width = 137,
Height = 64,
OffsetX = 300,
OffsetY = 100,
ID = "html1",
Shape = new Shape() { Type = Shapes.HTML },
};
nodes.Add(node);
}
public void OnClick()
{
nodes[0].BackgroundColor = "cornflowerblue";
StateHasChanged();
}
}

```

In the above example, node's background color is updated using the click event of the button defined in the template.



### *SetNodeTemplate*

The [SetNodeTemplate](#) method of diagram allows you to define the style for the Node. The following code demonstrates how to set different style for different node through SetNodeTemplate method.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
@using System.Collections.ObjectModel
@* Initialize Diagram *@
<SfDiagramComponent Height="600px" Nodes="@nodes"
SetNodeTemplate="@SetNodeTemplate" />
@code
{
    // Initialize node collection with node
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node1 = new Node()
        {
            ID = "node1",
            // Size of the node
            Height = 100,
            Width = 100,
            // Position of the node
            OffsetX = 100,
            OffsetY = 100,
        };
        Node node2 = new Node()
        {
```

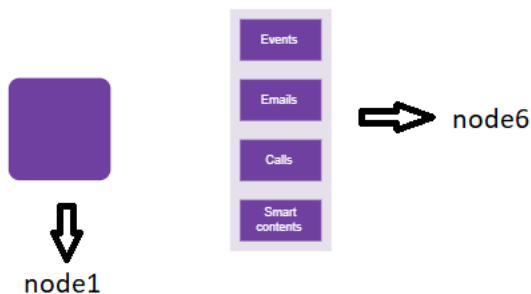
```
ID = "node6",
// Size of the node
Height = 510,
Width = 202,
// Position of the node
OffsetX = 300,
OffsetY = 100,
};
nodes.Add(node1);
nodes.Add(node2);
}
private ICommonElement SetNodeTemplate(IDiagramObject node)
{
    if ((node as Node).ID == "node6")
    {
        StackPanel table = new StackPanel();
        table.Style = new ShapeStyle() { Fill = "#e6e0eb", StrokeColor = "#e6e0eb"
        };
        table.Orientation = Orientation.Horizontal;
        StackPanel column1 = new StackPanel();
        column1.Style = new ShapeStyle() { Fill = "#6F409F", StrokeColor = "#6F409F"
        };
        column1.Margin = new Margin() { Bottom = 10, Left = 10, Right = 10, Top = 10
        };
        column1.Padding = new Thickness() { Bottom = 10, Left = 10, Right = 10, Top
        = 10 };
        column1.Children = new ObservableCollection<ICommonElement>();
        column1.Children.Add(getTextElement("Events"));
        StackPanel column2 = new StackPanel();
        column2.Margin = new Margin() { Bottom = 10, Left = 10, Right = 10, Top = 10
        };
        column2.Padding = new Thickness() { Bottom = 10, Left = 10, Right = 10, Top
        = 10 };
        column2.Children = new ObservableCollection<ICommonElement>();
        column2.Children.Add(getTextElement("Emails"));
        column2.Style = new ShapeStyle() { Fill = "#6F409F", StrokeColor = "#6F409F"
        };
        StackPanel column3 = new StackPanel();
        column3.Margin = new Margin() { Bottom = 10, Left = 10, Right = 10, Top = 10
        };
        column3.Padding = new Thickness() { Bottom = 10, Left = 10, Right = 10, Top
        = 10 };
        column3.Children = new ObservableCollection<ICommonElement>();
        column3.Children.Add(getTextElement("Calls"));
        column3.Style = new ShapeStyle() { Fill = "#6F409F", StrokeColor = "#6F409F"
        };
        StackPanel column4 = new StackPanel();
        column4.Margin = new Margin() { Bottom = 10, Left = 10, Right = 10, Top = 10
        };
        column4.Padding = new Thickness() { Bottom = 10, Left = 10, Right = 10, Top
        = 10 };
        column4.Children = new ObservableCollection<ICommonElement>();
        column4.Children.Add(getTextElement("Smart contents"));
        column4.Style = new ShapeStyle() { Fill = "#6F409F", StrokeColor = "#6F409F"
        };
        table.Orientation = Orientation.Vertical;
```



```

table.Children = new ObservableCollection<ICommonElement>() { column1,
column2, column3, column4 };
return table;
}
else
{
(node as Node).Style = new ShapeStyle()
{
Fill = "#6F409F",
StrokeColor = "#6F409F",
StrokeWidth = 2
};
(node as Node).Shape = new BasicShape()
{
Type = Shapes.Basic,
Shape = BasicShapeType.Rectangle,
CornerRadius = 10
};
}
return null;
}
private TextElement getTextElement(string text)
{
TextElement textElement = new TextElement();
textElement.Width = 60;
textElement.Height = 20;
textElement.Content = text;
textElement.Style = new TextStyle() { Color = "white" };
return textElement;
}
}

```



### Shadow

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

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{

```

```

DiagramObjectCollection<Node> nodes;
protected override void OnInitialized()
{
    nodes = new DiagramObjectCollection<Node>();
    // A node is created and stored in nodes array.
    Node node = new Node()
    {
        // Position of the node
        OffsetX = 250,
        OffsetY = 250,
        // Size of the node
        Width = 100,
        Height = 100,
        Style = new ShapeStyle()
        {
            Fill = "#6495ED",
            StrokeColor = "white"
        },
        Constraints = NodeConstraints.Default | NodeConstraints.Shadow
    };
    nodes.Add(node);
}

```



#### Customizing shadow

The [Angle](#), [Distance](#), and [Opacity](#) of the shadow can be customized with the shadow property of the node. The following code example illustrates how to customize shadow.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes array.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,

```

```

Style = new ShapeStyle()
{
    Fill = "#6495ED",
    StrokeColor = "white"
},
Constraints = NodeConstraints.Default | NodeConstraints.Shadow,
// Custom Shadow of the node
Shadow = new Shadow()
{
    Angle = 50,
    Color = "Blue",
    Opacity = 0.8,
    Distance = 20
}
};
nodes.Add(node);
}
}

```



### Gradient

The [Gradient](#) property of the node allows you to define and apply the gradient effect to the node. The gradient stops property defines the color and a position, where the previous color transition ends and a new color transition starts. The gradient stop's opacity property defines the transparency level of the region.

There are two types of gradients as follows:

- LinearGradientBrush
- RadialGradientBrush

### Linear gradient brush

[LinearGradientBrush](#) defines a smooth transition between a set of colors (so-called stops) in a line. A linear gradient's X1, Y1, X2, Y2 properties are used to define the position (relative to the node) of the rectangular region that needs to be painted.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
    }
}

```

```
// A node is created and stored in nodes array.
Node node = new Node()
{
    // Position of the node
    OffsetX = 250,
    OffsetY = 250,
    // Size of the node
    Width = 100,
    Height = 100,
    Style = new ShapeStyle()
    {
        Gradient = new LinearGradientBrush()
        {
            // Start point of linear gradient
            X1 = 0,
            Y1 = 0,
            // End point of linear gradient
            X2 = 50,
            Y2 = 50,
            //Sets an array of stop objects
            GradientStops = new DiagramObjectCollection<GradientStop>()
            {
                new GradientStop(){ Color = "white", Offset = 0},
                new GradientStop(){ Color = "#6495ED", Offset = 100}
            },
        },
    };
    nodes.Add(node);
}
```



#### Radial gradient brush

[RadialGradientBrush](#) defines a smooth transition between stops on a circle. The radial gradient brush properties are used to define the position (relative to the node) of the outermost or the innermost circle of the radial gradient.

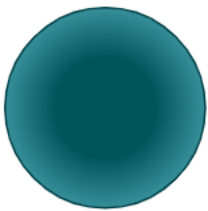
#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
    }
}
```

```

Node node = new Node()
{
    // Position of the node
    OffsetX = 250,
    OffsetY = 250,
    // Size of the node
    Width = 100,
    Height = 100,
    Shape = new BasicShape()
    {
        Type = Shapes.Basic,
        Shape = BasicShapeType.Ellipse
    },
    Style = new ShapeStyle()
    {
        Fill = "37909A#",
        StrokeColor = "#024249",
        Gradient = new RadialGradientBrush()
        {
            // Center point of outer circle
            CX = 50,
            CY = 50,
            // Center point of inner circle
            FX = 50,
            FY = 50,
            GradientStops = new DiagramObjectCollection<GradientStop>()
            {
                new GradientStop() { Color = "#00555b", Offset = 45 },
                new GradientStop() { Color= "#37909A", Offset= 90 }
            },
        },
    };
    // Add node
    nodes.Add(node);
}

```



### Custom properties

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

The following code shows how to set the AdditionalInfo value.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code

```

```

{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Dictionary<string, object> NodeInfo = new Dictionary<string, object>();
        NodeInfo.Add("nodeInfo", "Central Node");
        // A node is created and stored in nodes collection.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle()
            {
                Fill = "#6BA5D7",
                StrokeColor = "white"
            },
            AdditionalInfo = NodeInfo
        };
        // Add node
        nodes.Add(node);
    }
}

```

**Note:** We can set any type of value for the AdditionalInfo property.

*See also*

- [How to get events when they interact the node](#)

### Node Interaction in Blazor Diagram Component

Diagram provides the support to select, drag, resize, or rotate the node interactively.

#### Select

A node can be select at runtime by using the [Select](#) method and clear the selection in the diagram by using the [ClearSelection](#). The following code explains how to select and clear selection in the diagram.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
@using System.Collections.ObjectModel
<input type="button" value="Select" @onclick="OnSelect">
<input type="button" value="UnSelect" @onclick="@UnSelect" />
<SfDiagramComponent @ref="@diagram" Height="600px" Nodes="@nodes" />
@code
{
    // reference of the diagram
    SfDiagramComponent diagram;
    // To define node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()

```

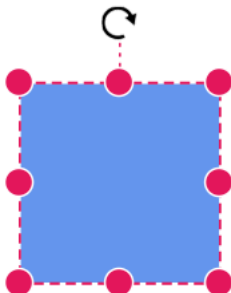
```

{
    nodes = new DiagramObjectCollection<Node>();
    // A node is created and stored in nodes collection.
    Node node = new Node()
    {
        // Position of the node
        OffsetX = 250,
        OffsetY = 250,
        // Size of the node
        Width = 100,
        Height = 100,
        Style = new ShapeStyle()
        {
            Fill = "#6495ED",
            StrokeColor = "white"
        }
    };
    // Add node
    nodes.Add(node);
}
public void OnSelect()
{
    // Select the node
    diagram.Select(new ObservableCollection<IDiagramObject> { diagram.Nodes[0]
});
}
public void UnSelect()
{
    // clear selection in the diagram
    diagram.ClearSelection();
}
}

```

And also the selection enable during the interaction.

- An element can be selected by clicking that element.
- When you select the elements in the diagram, the [SelectionChanging](#) and [SelectionChanged](#) event gets triggered and to do customization in those events.



#### Drag

A node can be drag at runtime by using the [Drag](#) method. The following code explains how to drag the node by using the drag method.

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<input type="button" value="Drag" @onclick="OnDrag">
<SfDiagramComponent @ref="@Diagram" Height="600px" Nodes="@nodes" />
@code
{
    // reference of the diagram
    SfDiagramComponent Diagram;
    // To define node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes collection.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle()
            {
                Fill = "#6495ED",
                StrokeColor = "white"
            }
        };
        // Add node
        nodes.Add(node);
    }
    public void OnDrag()
    {
        // Drag the node
        Diagram.Drag(Diagram.Nodes[0], 10, 10);
    }
}
```

And also the drag the node during the interaction.

- An object can be dragged by clicking and dragging it. When multiple elements are selected, dragging any one of the selected elements move all the selected elements.
- When you drag the elements in the diagram, the [PositionChanging](#) and [PositionChanged](#) event gets triggered and to do customization in this event.





### Resize

A node can be resized at runtime by using the [Scale](#) method. The following code explains how to resize the node by using the scale method.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<input type="button" value="Resize" @onclick="OnResize">
<SfDiagramComponent @ref="@diagram" Height="600px" Nodes="@nodes" />
@code
{
    // reference of the diagram
    SfDiagramComponent diagram;
    // To define node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes collection.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
```

```
Height = 100,
Style = new ShapeStyle()
{
    Fill = "#6495ED",
    StrokeColor = "white"
};
// Add node
nodes.Add(node);
}
public void OnResize()
{
    // Resize the node
    diagram.Scale(diagram.Nodes[0], 0.5, 0.5, new DiagramPoint() { X = 0, Y = 0
});
}
}
```

And also you can resize the node during interaction.

- Selector is surrounded by eight thumbs. When dragging these thumbs, the selected items can be resized.
- When one corner of the selector is dragged, the opposite corner will be in a static position.
- When a node is resized, the [SizeChanging](#) and [SizeChanged](#) event gets triggered.



### Rotate

A node can be rotate at runtime by using the [Rotate](#) method. The following code explains how to rotate the node by using the rotate method.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<input type="button" value="Rotate" @onclick="OnRotate">
<SfDiagramComponent @ref="@diagram" Height="600px" Nodes="@nodes" />
@code
{
    // reference of the diagram
    SfDiagramComponent diagram;
    // To define node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes collection.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
```

```
Height = 100,
Style = new ShapeStyle()
{
    Fill = "#6495ED",
    StrokeColor = "white"
};
// Add node
nodes.Add(node);
}
public void OnRotate()
{
    // Rotate the node
    diagram.Rotate(diagram.Nodes[0], diagram.Nodes[0].RotationAngle + 10);
}
}
```

And also the rotate the node during the interaction.

- A rotate handler is placed above the selector. Clicking and dragging the handler in a circular direction lead to rotate the node.
- The node is rotated with reference to the static pivot point.
- Pivot thumb (thumb at the middle of the node) appears when rotating the node to represent the static point.
- When a node is rotated, the [RotationChanging](#) and [RotationChanged](#) events getting triggered.



For more information about node interaction, refer to [Node Interaction](#).

*See also*

- [How to get events while the interact the node](#)
- [How to positioning the node](#)
- [How to customize the node](#)
- [How to interact the annotation in diagram](#)
- [How to interact the port in diagram](#)
- [How to interact the connector in diagram](#)

## Events and Constraints in Blazor Diagram Component

### *Events*

Diagram provides some events support for node that triggers when interacting the node.

### *Selection change*

- While selecting the diagram elements, the following events can be used to do the customization.
- When selecting or unselecting the diagram elements, the following events are getting triggered.

Event Name	Arguments	Description
------------	-----------	-------------

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

| [SelectionChanging](#) | [SelectionChangingEventArgs](#) | Triggers before the selection is changed in the diagram. |

| [SelectionChanged](#) | [SelectionChangedEventArgs](#) | Triggers when the node's or connector's selection is changed in the diagram. |

The following code example explains how to get the selection change event in the diagram.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px"
Nodes="@nodes"
SelectionChanging="OnSelectionChanging"
SelectionChanged="OnSelectionChanged" />
@code
{
// To define node collection
DiagramObjectCollection<Node> nodes;
protected override void OnInitialized()
{
nodes = new DiagramObjectCollection<Node>();
// A node is created and stored in nodes collection.
Node node = new Node()
{
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
// Apperence of the node
Style = new ShapeStyle()
{
Fill = "#6BA5D7",
StrokeColor = "white"
}
};
// Add node
nodes.Add(node);
}
// Event to notify the selection changing event before select/unselect the
// diagram elements
public void OnSelectionChanging(SelectionChangingEventArgs args)
{
// sets true to cancel the selection.
args.Cancel = true;
}
// Event to notify the selection changed event after select/unselect the
// diagram elements.
public void OnSelectionChange(SelectionChangedEventArgs args)
{
// Action to be performed
}
}
```

*Position change*

- While dragging the node or connector through interaction, the following events can be used to do the customization.
- When dragging the node, the following events are getting triggered.

Event Name	Arguments	Description
<a href="#">PositionChanging</a>	<a href="#">PositionChangingEventArgs</a>	Triggers while dragging the elements in the diagram.
<a href="#">PositionChanged</a>	<a href="#">PositionChangedEventArgs</a>	Triggers when the node's or connector's position is changed.

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px"
Nodes="@nodes"
PositionChanging="OnPositionChanging"
PositionChanged="OnPositionChanged" />
@code
{
    // To define node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes collection.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle()
            {
                Fill = "#6495ED",
                StrokeColor = "white"
            },
            Shape = new Shape() { Type = Shapes.Basic}
        };
        // Add node
        nodes.Add(node);
    }
    // Event to notify the position changing event while dragging the elements
    in diagram.
    public void OnPositionChanging(PositionChangingEventArgs args)
    {
        // Sets true to cancel the action.
        args.Cancel = true;
    }
}
```

```
//Event to notify the position changed event when the node's or connector's
position is changed.
public void OnPositionChanged(PositionChangedEventArgs args)
{
    // Action to be performed.
}
}
```

### Size change

- While resizing the node during the interaction, the following events can be used to do the customization.
- When resizing the node, the following events are getting triggered.

Event Name	Arguments	Description
<a href="#">SizeChanging</a>	<a href="#">SizeChangingEventArgs</a>	Triggers before the node is resized.
<a href="#">SizeChanged</a>	<a href="#">SizeChangedEventArgs</a>	Triggers when the node is resized.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px"
Nodes="@nodes"
SizeChanged="OnSizeChanged"
SizeChanging="OnSizeChanging"/>
@code
{
    // To define node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes collection.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle()
            {
                Fill = "#6BA5D7",
                StrokeColor = "white"
            }
        };
        // Add node
        nodes.Add(node);
    }
    // Event to notify the Size changing event before the diagram elements size
    is changing.
```



```

public void OnSizeChanging(SizeChangingEventArgs args)
{
    // Sets true to cancel the resize action
    args.Cancel = true;
}
// Event to notify the Size change event after the diagram elements size is
// changed.
public void OnSizeChanged(SizeChangedEventArgs args)
{
    // Action to be performed.
}
}

```

### Rotate change

- While rotating the node during the interaction, the following events can be used to do the customization.
- When rotating the node, the following events are getting triggered.

Event Name	Arguments	Description
<a href="#">RotationChanging</a>	<a href="#">RotationChangingEventArgs</a>	Triggers before the diagram elements are rotate.
<a href="#">RotationChanged</a>	<a href="#">RotationChangedEventArgs</a>	Triggers when the diagram elements are rotated.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px"
Nodes="@nodes"
RotationChanging="OnRotateChanging"
RotationChanged="OnRotateChanged" />
@code
{
    // To define node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes collection.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle()
            {
                Fill = "#6BA5D7",
                StrokeColor = "white"
            }
        };
    }
}

```

```
// Add node
nodes.Add(node);
}
// Event to notify the rotation changing event before the node is rotate.
public void OnRotateChanging(RotationChangingEventArgs args)
{
    // Sets true to cancel the rotation
    args.Cancel = true;
}
// Event to notify the rotation changed event after the node is rotated.
public void OnRotateChanged(RotationChangedEventArgs args)
{
    // Action to be performed.
}
}
```

### Constraints

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

### See also

- [How to interact the node in diagram](#)
- [How to get events when they interact the connector](#)
- [How to get events when they interact the annotation](#)

## Positioning a node in Blazor Diagram Component

### Arrange the nodes

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

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

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

The following code shows how to change the Pivot value.

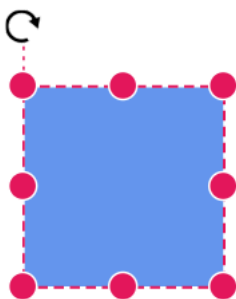
### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
```

```

@using System.Collections.ObjectModel
<SfDiagramComponent Height="600px" @ref="@diagram" Nodes="@nodes" />
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes array.
        Node node = new Node()
        {
            ID = "node",
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle()
            {
                Fill = "#6495ED",
                StrokeColor = "white"
            },
            // Pivot of the node
            Pivot = new DiagramPoint() { X = 0, Y = 0 }
        };
        nodes.Add(node);
    }
    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        if (firstRender)
        {
            //OnAfterRenderAsync will be triggered after the component rendered.
            await Task.Delay(200);
            diagram.Select(new ObservableCollection<IDiagramObject>() { diagram.Nodes[0]
        });
        }
    }
}

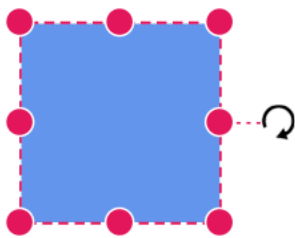
```



Rotation of a node is controlled by using the [RotationAngle](#) property. The following code shows how to change the RotationAngle value.

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" @ref="@diagram" Nodes="@nodes" />
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes array.
        Node node = new Node()
        {
            ID = "node",
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle()
            {
                Fill = "#6495ED",
                StrokeColor = "white"
            },
            // RotationAngle of the node
            RotationAngle = 90
        };
        nodes.Add(node);
    }
}
```

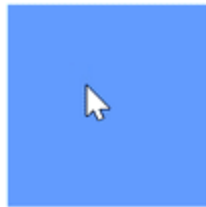
*Minimum Size and Maximum Size*

The [MinWidth](#) and [MinHeight](#) properties of node allows you to control the minimum size of the node while resizing. Similarly, the [MaxWidth](#) and [MaxHeight](#) properties of node allows you to control the maximum size of the node while resizing. The below gif explains how minimum and maximum size is controlled.

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" @ref="@diagram" Nodes="@nodes" />
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes;
```

```
protected override void OnInitialized()
{
    nodes = new DiagramObjectCollection<Node>();
    // A node is created and stored in nodes array.
    Node node = new Node()
    {
        ID = "node",
        // Position of the node
        OffsetX = 250,
        OffsetY = 250,
        // Size of the node
        Width = 100,
        Height = 100,
        //Minimum Size of the node
        MinHeight = 50,
        MinWidth = 50,
        //Maximum Size of the node
        MaxHeight = 200,
        MaxWidth = 200,
        Style = new ShapeStyle()
        {
            Fill = "#6495ED",
            StrokeColor = "white"
        },
    };
    nodes.Add(node);
}
```



*See also*

- [How to customize the node](#)
- [How to get events when they interact the node](#)

## Node Shapes in Blazor Diagram Component

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

- Image shape
- Path shape
- Basic shape
- Flow shape
- SVG shape
- HTML template

### Image node

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

The following code illustrates how an image node is created.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
```

```

@* Initialize Diagram *@
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        Node node = new Node()
        {
            ID = "node1",
            //Size of the node
            Height = 100,
            Width = 100,
            //Position of the node
            OffsetX = 100,
            OffsetY = 100,
            Shape = new ImageShape()
            {
                Type = Shapes.Image,
                //Sets the source to the image shape
                Source = "/diagram/images/syncfusion.png"
            }
        };
        nodes.Add(node);
    }
}

```



*Base64 encoded image into the image node*

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

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
@* Initialize Diagram *@
<SfDiagramComponent Height="600px" Nodes="@nodes"></SfDiagramComponent>
@code
{
    //Initialize node collection with node
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        Node node = new Node()
        {
            ID = "node1",
            //Size of the node
            Height = 100,
            Width = 100,

```

```
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Sets type of the shape as image
Shape = new ImageShape()
{
    Type = Shapes.Image,
    //Sets the Base64 image to the image shape
    Source =
    "data:image/gif;base64,R0lGODlhPQBEAeoAJosM//AwO/AwHVYZ/z595kzAP/s7P+goOXMv
8+fhw/v739/f+8PD98fH/8mJl+fn/9ZWb8/PzWlwv///6wWGbImAPgTEMImIN9gUFCEm/gDALULD
N8PAD6atYdCTX9gUNKlj8wZAKUsAOzZz+UMAOsJAP/Z2ccMDA8PD/95eX5NWvsJCOVNQPtfX/8zM
8+QePLl38MGBr8JCP+zs9myn/8GBqwpAP/GxgwJCPny78lzlYlgjAJ8vAP9fX/+MjMUCAN8zM/9wc
M8ZGcATEL+QePdZWf/29uc/P9cmJu9MTDImIN+/r7+/vz8/P8VNQGNugV8AAF9fX8swMNgtAF1DO
ICAgPNSUnNWSMQ5MBAQEJE3QPIGAM9AQMQGcG9vb6MhJsEdGM8vLx8fH98AANIWAMuQeL8fABkTE
PPQ00M5OSYdGF15jo+Pj/+pqcsTE78wMFNGQLYmID4dGPvd3UBAQJmTkP+8vH9QUK+vr8ZWSHpzc
JMmILdwcLOGCHRQUHxwcK9PT9DQ00/v70w5MLypoG8wKOUwsP/g4P/Q0IcwKEswKMl8aJ9fX2xjd
OtGRs/Pz+Dg4GImIP8gIH0sKEAwKKmTiKZ8aB/f39Wsl+Lft8dgUE9PT5x5aHBwcP+AgP+WltdgY
MyZfywz78AAAAAAD///8AAP9mZv///wAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CH5BAEAAKqALAAAAA9AEQAAj/AFEJHEiwoMGDCBMqXmiwocAbBww4nEhxoYkUpzJGrMixogkfG
UNqlNixJEIDB0SqHGmyJSojM1bKZomyop0gM3Oe2liTISKMOoPy7GnwY9CjIYcSRyM0aVKSLE6n
fq05QycVLPuhDrxBlCtYJUqNAq2bNWEBj6ZXRUyxZyDRtqwnXvkhACDV+euTeJm1Ki7A73qNwTFi
F+/gA95Gly2CJLDhweHMOUAAuOpLYDEgBxZ4GRTlC1fDnpkM+fOqD6DDj1aZpITp0dtGCDhr+fVu
Cu3zlg49ijaokTZTo27uG7Gjn2P+hI8+PDPEROUB318bWbfAJ5sUNFcuGRTYUqV/3ogfXp1rWlMc
6awJjiaAd2fm4ogXjz56aypOoIde4OE5u/F9x199dlXnnGiHZWEYbGpsAEA3QXynHwEFliKAgsWg
J8LPeiUXGwedCAKABACCN+EA1pYIIYaFlcDhytd51sGAJbo3onOpajiihl092KHGaUXGwWjUBChj
SPiWJu00/LYIm4v1tXfE6J4gCSJEZ7YgRYUNrkji9P55sF/ogxw5ZkSqIDaZBV6aSGYq/lGZplnd
kckZ98xoICbTcIJGQAZcNmdmUc210hs35nCyJ58fgmIKX5RQGOZowxaZwYA+JaoKQswGijBV4C6
SiTUmpphMspJx9unX4KaimjDv9aaXOEbteBqmuuxgEHOlX6Kqx+yXqqBANsgCtit4FWQAEkrNbpq
7HSOmtwag5w57GrmlJBASEU18ADjUYb3ADTinIttsgSB1oJFfa63bduimuqKB1keqUhoCSK374w
bujvOSu4QG6UvxBRydcPksav++Ca6G8A6Pr1x2kVMYHwsVxUALDq/knrhPSOzXG1lUTIoffqGR7
Goi2MAxbv602kEG56I7CS1RsEFKFVyoVDJoIRTg7sugNRDGqCJzJgcKE0ywc0ELm6KBCCJo8DIPF
eCWNGcyqNFE06ToAfV0HBRgxsvLThHn1oddQMrXj5DyAQgjEHS AJMWZws3HPxT/QMbabi/iBCliM
LEJKX2EEkomBAUCxRi42VDADxyTYDVogV+wSCHqmKxEKCDAYFDFj4OmwbY7bDGdBhtrnTQYOigeC
hUmclK3QTnAUfEgGFgAWt88hKA6aCRIXhxnQ1yg3BCayK44EWdkUQCBBYEQChFXfCB776aQsG0BI
lQgQgE8q026X1h8cEUep8ngRBnOy74E9QgRgEAC8SvOfQkh7FDBDms43PmGoIiKUUEGkMEC/PJHg
xw0xH74yx/3XnaYRJgMB8obxQW6kL9QYEJ0FIFgByfIL7/IQAlvQwEpnAC7DtLNJCKUoO/w45c44
GwCXiAFB/OXAATQryUxdN4LfFiwgjCNYg+kYMIEFkCKDs6PKAIJouyGWMS1FSKJOMRB/BoIXYJIU
XFUxNwoIkEKPAGCBZSQHQ1A2EWdfDEUvLYAdj5AchSIQW6gu10bE/JG2VnVnZGfo4R4d0sdQoBAHh
PjhIB94v/wRoRkQWGRHgrhGSQJxCS+0pCZbEhAAOw=="
},
Style = new ShapeStyle() { StrokeColor = "White" }
};
nodes.Add(node);
}
}
```





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

#### *Image alignment*

Stretch and align the image content anywhere but within the node boundary. The [Scale](#) property of the node allows you to stretch the image as you desire. (either to maintain proportion or to stretch). By default, the Scale property of the node is set as [Meet](#). The following code illustrates how to scale or stretch the content of the image node.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
@* Initialize Diagram *@
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        Node node = new Node()
        {
            ID = "node1",
            //Size of the node
            Height = 100,
            Width = 100,
            //Position of the node
            OffsetX = 100,
            OffsetY = 100,
            Shape = new ImageShape()
            {
                Type = Shapes.Image,
                Source = "/diagram/images/productmanager.png",
                //To stretch the image
                Scale = Scale.Meet,
                //To align the image
                ImageAlign = ImageAlignment.XMinYMax
            }
        };
        nodes.Add(node);
    }
}
```

- [ImageAlign](#) property of the shape helps to align the image based on the x and y values in the node boundary. The following table illustrates the various image alignments in the node boundary.

|Scale |ImageAlign |Result|

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



| Meet | XMinYMax |



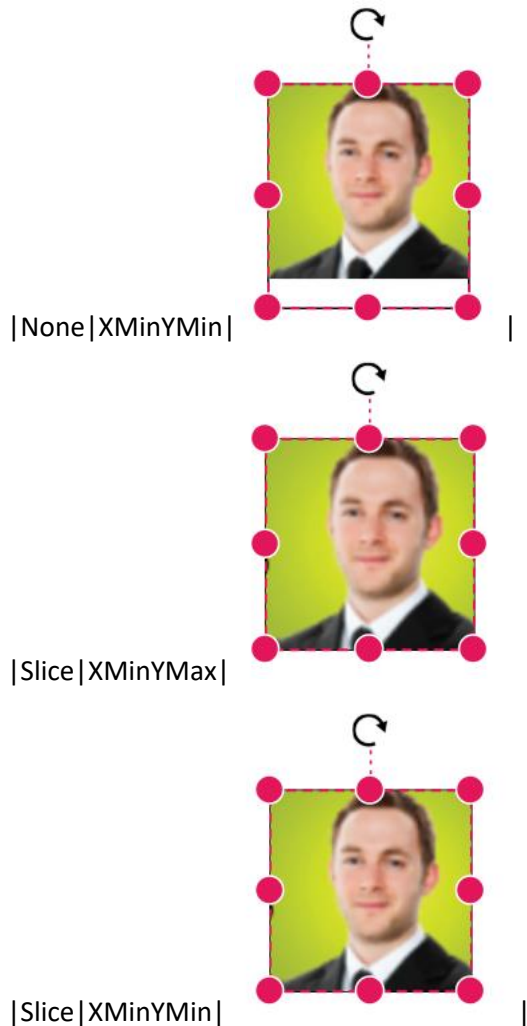
|Meet|XMinYMin|



|Meet|XMaxYMax|



|None|XMinYMax|



### HTML template shape

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
@* Add a Namespace for a syncfusion control used in Diagram HTML node *@
@using Syncfusion.Blazor.CircularGauge
@* Initialize Diagram with node template *@
<SfDiagramComponent Height="600px" Nodes="@nodes">
  <DiagramTemplates>
    <NodeTemplate>
      @ {
        <SfCircularGauge Width="300px" Height="300px">
          <CircularGaugeAxes>
            <CircularGaugeAxis StartAngle="210" EndAngle="150" Minimum="0" Maximum="120"
              Radius="80%">
              <CircularGaugeAxisLineStyle Width="10" Color="transparent">
            </CircularGaugeAxisLineStyle>
            <CircularGaugeAxisLabelStyle UseRangeColor="false">
```

```

<CircularGaugeAxisLabelFont Size="12px" FontFamily="Roboto"
FontStyle="Regular">
</CircularGaugeAxisLabelFont>
</CircularGaugeAxisLabelStyle>
<CircularGaugeAxisMajorTicks Height="10" Offset="5" UseRangeColor="false">
</CircularGaugeAxisMajorTicks>
<CircularGaugeAxisMinorTicks Height="0" Width="0" UseRangeColor="false">
</CircularGaugeAxisMinorTicks>
<CircularGaugeAnnotations>
<CircularGaugeAnnotation Radius="30%" Angle="0" ZIndex="1"
Content="Speedometer">
<ContentTemplate>
<div><span style="font-size:14px; color:#9E9E9E; font-family:Regular;
margin-left: -33px">Speedometer</span></div>
</ContentTemplate>
</CircularGaugeAnnotation>
<CircularGaugeAnnotation Radius="40%" Angle="180" ZIndex="1" Content="65
MPH">
<ContentTemplate>
<div><span style="font-size:20px; color:#424242; font-family:Regular;
margin-left: -33px">65 MPH</span></div>
</ContentTemplate>
</CircularGaugeAnnotation>
</CircularGaugeAnnotations>
<CircularGaugeRanges>
<CircularGaugeRange Start="0" End="40" Color="#30B32D" StartWidth="10"
EndWidth="10" RoundedCornerRadius="0">
</CircularGaugeRange>
<CircularGaugeRange Start="40" End="80" Color="#FFDD00" StartWidth="10"
EndWidth="10" RoundedCornerRadius="0">
</CircularGaugeRange>
<CircularGaugeRange Start="80" End="120" Color="#F03E3E" StartWidth="10"
EndWidth="10" RoundedCornerRadius="0">
</CircularGaugeRange>
</CircularGaugeRanges>
<CircularGaugePointers>
<CircularGaugePointer Value="65" Radius="60%" PointerWidth="8">
<CircularGaugePointerAnimation
Enable="true"></CircularGaugePointerAnimation>
<CircularGaugeCap Radius="7">
</CircularGaugeCap>
<CircularGaugeNeedleTail Length="18%">
</CircularGaugeNeedleTail>
</CircularGaugePointer>
</CircularGaugePointers>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
}
</NodeTemplate>
</DiagramTemplates>
</SfDiagramComponent>
@code
{
//Initialize node collection with node
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
protected override void OnInitialized()

```

```

{
Node node = new Node()
{
ID = "node1",
//Size of the node
Width = 300,
Height = 300,
//Position of the node
OffsetX = 400,
OffsetY = 300,
//sets the type of the shape as HTML
Shape = new Shape()
{
Type = Shapes.HTML,
}
};
nodes.Add(node);
}
}

```



HTML node cannot be exported to image format, like JPEG, PNG and BMP. It is by design, while exporting the diagram is drawn in a canvas. Further, this canvas is exported into image formats. Currently, drawing in a canvas equivalent from all possible HTML is not feasible. Hence, this limitation. Also, HTML node always appears as topmost layer (whose index is the higher index, even though it is defined at the last).

### Basic shapes

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

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

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
@* Initialize Diagram *@
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
//Initialize node collection with node
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
protected override void OnInitialized()
{
Node node = new Node()

```

```

{
ID = "node1",
//Size of the node
Height = 100,
Width = 100,
//Position of the node
OffsetX = 100,
OffsetY = 100,
//sets the type of the shape as basic
Shape = new BasicShape()
{
Type = Shapes.Basic,
Shape = BasicShapeType.Rectangle,
//Sets the corner radius to the node shape
CornerRadius = 10
},
Style = new ShapeStyle() { Fill = "#6495ED" }
};
nodes.Add(node);
}
}

```

By default, the [Shape](#) property of the node is set as **Basic**. Default property for Shape is Rectangle.

When the **Shape** is not set for a basic shape, it is considered as a **Rectangle**.

The **CornerRadius** property is applicable only for basic shape.

The list of basic shapes are as follows.



### Path shape

The [PathShape](#) is a commonly used basic shape that allows visually to represent the geometrical information. As node path data, any geometrical data can be provided. You can create your own Geometry and assign it to data if you want anything different from the standard figures. A geometry does not require any dimension specifications, such as width or height, because it specifies its own size. If the node's size is set, the geometry is extended to fit the node's dimensions.

To create a path node, specify the shape as **Path**. The [Data](#) property of node allows you to define the path to be drawn. The following code illustrates how a path node is created.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
@* Initialize Diagram *@
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code

```

```

{
//Initialize node collection with node
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>()
{
// Creates a path node
new Node()
{
ID = "node1",
//Size of the node
Height = 100,
Width = 100,
//Position of the node
OffsetX = 100,
OffsetY = 100,
//Sets the type of the shape as path
Shape = new PathShape()
{
Type=Shapes.Path,
Data="M 355.31 12.07 C 352.11 5.95 345.35 -1.14 331.41 0.15 C 290.33 3.93
209.61 81.48 190.42 111.69 C 189.66 107.76 187.9 101.49 184.12 98.05 C 189.5
87.75 198.01 69.64 201.57 52.48 C 202.4 52.26 203.12 51.68 203.34 50.44 C
203.72 48.34 204 46.22 204.39 44.13 C 205.01 40.62 199.99 39.43 199.42 42.91
C 199.06 45.06 198.69 47.15 198.35 49.31 C 198.16 50.6 198.69 51.62 199.54
52.14 C 196.08 68.87 187.75 86.63 182.43 96.81 C 181.52 96.29 180.53 95.87
179.41 95.7 C 179.09 95.65 178.8 95.72 178.5 95.71 C 178.19 95.72 177.91
95.65 177.59 95.7 C 176.46 95.87 175.48 96.29 174.56 96.81 C 169.24 86.63
160.92 68.87 157.46 52.14 C 158.3 51.62 158.83 50.6 158.65 49.31 C 158.3
47.15 157.93 45.06 157.58 42.91 C 157.01 39.43 151.99 40.62 152.61 44.13 C
152.99 46.22 153.28 48.34 153.66 50.44 C 153.87 51.68 154.6 52.27 155.42
52.48 C 158.98 69.63 167.49 87.75 172.87 98.05 C 169.09 101.49 167.33 107.76
166.57 111.69 C 147.39 81.48 66.67 3.93 25.59 0.15 C 11.65 -1.14 4.89 5.95
1.69 12.07 C -2.05 19.07 0.84 33.48 6.24 58.46 C 8.66 69.55 11.16 80.96
12.94 92.2 C 13.89 98.36 14.79 104.49 15.64 110.36 C 19.39 136.89 22.52
158.97 32.64 166.04 C 35.41 167.98 38.65 168.78 42.61 168.24 C 52.26 165.95
63.79 164.03 75.99 162.67 C 64.03 171.47 49.08 185.81 44.17 204.84 C 40.38
219.72 43.37 234.86 53.01 250.05 C 64.37 267.81 76.02 279.48 86.91 287.09 C
108.68 302.31 127.52 301.24 135.48 300.78 C 136.3 300.77 136.94 300.71
137.56 300.69 C 143.62 300.61 147.76 291.39 156.6 270.51 C 160.94 260.34
167.39 245.19 172.83 237.47 C 173.97 240.31 177.22 242.31 178.92 242.87 C
178.78 242.94 178.62 243.07 178.49 243.12 C 178.65 243.08 178.83 242.97 179
242.91 C 179.1 242.94 179.25 243.02 179.34 243.04 C 179.26 243.01 179.13
242.93 179.04 242.89 C 180.83 242.24 183.05 240.25 184.16 237.48 C 189.6
245.19 196.05 260.34 200.39 270.52 C 209.23 291.4 213.38 300.61 219.43 300.7
C 220.05 300.72 220.69 300.77 221.51 300.78 C 229.47 301.25 248.31 302.31
270.08 287.1 C 280.97 279.49 292.62 267.81 303.98 250.05 C 313.62 234.86
316.6 219.72 312.82 204.84 C 307.91 185.82 292.96 171.48 281 162.67 C 293.2
164.03 304.73 165.95 314.38 168.24 C 318.34 168.78 321.58 167.98 324.35
166.04 C 334.47 158.97 337.6 136.89 341.35 110.37 C 342.19 104.49 343.1
98.37 344.05 92.2 C 345.83 80.96 348.33 69.55 350.75 58.46 C 356.16 33.48
359.05 19.07 355.31 12.07 z M 183.92 237.56 C 182.35 240.3 180.35 242.13
178.97 242.86 C 177.49 242.19 174.7 240.38 173.08 237.56 C 173.08 237.56
166.19 138.22 166.91 113.58 C 166.91 113.58 169.24 97.92 178.5 97.81 C
187.76 97.92 190.09 113.58 190.09 113.58 C 190.81 138.22 183.92 237.56
183.92 237.56 z"
},
Style=new ShapeStyle()
{

```

```
Fill="Orange"
}
}
};
}
```



### Flow Shapes

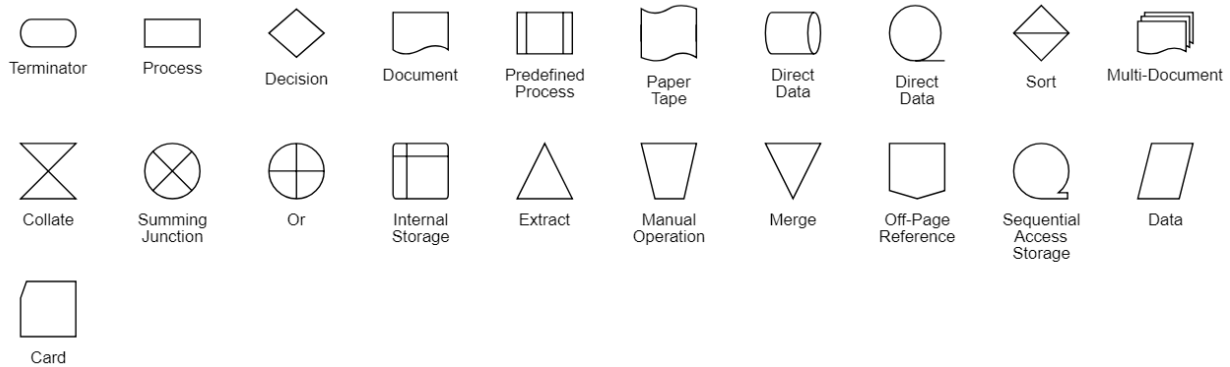
The [Flow](#) shapes are used to represent the process flow. It is used for analyzing, designing and managing the documentation process. To create a flow shape, specify the shape type as **Flow**. By default, it is considered as a **Process**. The following code example illustrates how to create a flow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

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
@* Initialize Diagram *@
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    //Initialize node collection with node
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        Node node = new Node()
        {
            ID = "node1",
            //Size of the node
            Height = 100,
            Width = 100,
            //Position of the node
            OffsetX = 100,
            OffsetY = 100,
            //Sets the type of the shape as flow
            Shape = new FlowShape()
            {
                Type = Shapes.Flow,
                Shape = FlowShapeType.DirectData
            }
        };
        nodes.Add(node);
    }
}
```

The list of flow shapes are as follows.





### SVG template shape

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
@* Initialize Diagram *@
<SfDiagramComponent Height="600px" Nodes="@nodes">
<SnapSettings Constraints="SnapConstraints.None"></SnapSettings>
<DiagramTemplates>
<NodeTemplate>
@{
if ((context as Node).Shape.Type == Shapes.SVG)
{
<svg viewBox="0 0 100 100" xmlns="http://www.w3.org/2000/svg">
<!-- Using g to inherit presentation attributes -->
<g fill="white" stroke="green" stroke-width="5">
<circle cx="40" cy="40" r="25" />
<circle cx="60" cy="60" r="25" />
</g>
</svg>
}
}
</NodeTemplate>
</DiagramTemplates>
</SfDiagramComponent>
@code
{
//Initialize node collection with node
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
protected override void OnInitialized()
{
Node node = new Node()
{
ID = "node1",
// Size of the node
Height = 100,
Width = 100,
// Position of the node
OffsetX = 100,
OffsetY = 100,
}
```

```
//Sets type of the shape as SVG
Shape = new Shape() { Type = Shapes.SVG },
};
nodes.Add(node);
}
}
```



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

## Connectors

### Connector in Blazor Diagram Component

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

#### Create connector

Connector can be created by defining the source and target point of the connector. The path to be drawn can be defined with a collection of segments.

#### Add connectors through connectors collection

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

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">
  <SnapSettings Constraints="@snapConstraints"></SnapSettings>
</SfDiagramComponent>
@code
{
  SnapConstraints snapConstraints = SnapConstraints.None;
  //Defines diagram's connector collection
  DiagramObjectCollection<Connector> connectors = new
  DiagramObjectCollection<Connector>();
  protected override void OnInitialized()
  {
    Connector Connector = new Connector()
    {
      ID = "connector1",
      // Set the source and target point of the connector
      SourcePoint = new DiagramPoint() { X = 100, Y = 100 },
      TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
      // Type of the connector segments
      Type = ConnectorSegmentType.Straight
    }
  }
}
```

```
};  
connectors.Add(Connector);  
}  
}
```



ID for each connector should be unique and so it is further used to find the connector at runtime and do any customization.

#### *Add connectors at runtime*

You can add a connector at runtime by adding connector to the connectors collection in the Diagram component. The following code explains how to add connectors at runtime.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram  
<input type="button" value="Add Connector" @onclick="@AddConnector">  
<SfDiagramComponent Width="1000px" Height="500px"  
Connectors="@connectors"></SfDiagramComponent>  
@code  
{  
    //Defines diagram's connector collection  
    DiagramObjectCollection<Connector> connectors = new  
    DiagramObjectCollection<Connector>();  
    protected override void OnInitialized()  
    {  
        Connector Connector = new Connector()  
        {  
            ID = "connector1",  
            SourcePoint = new DiagramPoint() { X = 100, Y = 100 },  
            TargetPoint = new DiagramPoint() { X = 200, Y = 200 },  
        }  
    }  
}
```

```

Type = ConnectorSegmentType.Straight
};
connectors.Add(Connector);
}
public void AddConnector()
{
Connector NewConnector = new Connector()
{
ID = "connector2",
SourcePoint = new DiagramPoint() { X = 300, Y = 300 },
TargetPoint = new DiagramPoint() { X = 400, Y = 400 },
Type = ConnectorSegmentType.Straight
};
//Add the connector at the run time.
connectors.Add(NewConnector);
}
}

```

### *Connectors from the palette*

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

### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
@using Syncfusion.Blazor.Diagram.SymbolPalette
<div style="width: 200px; float: left">
<SfSymbolPaletteComponent Height="600px" @ref="@PaletteInstance"
Palettes="@Palettes">
</SfSymbolPaletteComponent>
</div>
<SfDiagramComponent @ref="@DiagramInstance" Width="1000px" Height="500px"
Connectors="@connectors"></SfDiagramComponent>
@code
{
SfSymbolPaletteComponent PaletteInstance;
SfDiagramComponent DiagramInstance;
//Defines Symbol palette's PaletteConnector collection
DiagramObjectCollection<NodeBase> PaletteConnector = new
DiagramObjectCollection<NodeBase>();
DiagramObjectCollection<Palette> Palettes = new
DiagramObjectCollection<Palette>();
protected override async Task OnAfterRenderAsync(bool firstRender)
{
PaletteInstance.Targets = new DiagramObjectCollection<SfDiagramComponent>()
{ };
PaletteInstance.Targets.Add(DiagramInstance);
}
DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
Connector Connector = new Connector()
{
ID = "connector1",
// Set the source and target point of the connector

```

```
SourcePoint = new DiagramPoint() { X = 100, Y = 100 },
TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
// Type of the connector segments
Type = ConnectorSegmentType.Straight
};
connectors.Add(Connector);
PaletteConnector = new DiagramObjectCollection<NodeBase>();
Connector connector = new Connector
{
    ID = "Link1",
    Type = ConnectorSegmentType.Straight,
    SourcePoint = new DiagramPoint() { X = 0, Y = 0 },
    TargetPoint = new DiagramPoint() { X = 60, Y = 60 }
};
PaletteConnector.Add(connector as NodeBase);
Connector connector2 = new Connector
{
    ID = "Link2",
    Type = ConnectorSegmentType.Orthogonal,
    SourcePoint = new DiagramPoint() { X = 0, Y = 0 },
    TargetPoint = new DiagramPoint() { X = 60, Y = 60 },
    TargetDecorator = new DecoratorSettings() { Shape = DecoratorShape.OpenArrow },
    Style = new ShapeStyle() { StrokeWidth = 1 }
};
PaletteConnector.Add(connector2 as NodeBase);
Connector connector3 = new Connector
{
    ID = "Link3",
    Type = ConnectorSegmentType.Bezier,
    SourcePoint = new DiagramPoint() { X = 0, Y = 0 },
    TargetPoint = new DiagramPoint() { X = 60, Y = 60 },
    TargetDecorator = new DecoratorSettings() { Shape = DecoratorShape.None },
};
PaletteConnector.Add(connector3 as NodeBase);
Palettes = new DiagramObjectCollection<Palette>()
{
    new Palette() { Symbols = PaletteConnector, Title = "Connectors", IsExpanded = true },
};
}
```



#### *Draw connectors using drawing object*

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



#### *Remove connectors at runtime*

A connector can be removed from the diagram at runtime by using the `Remove` method.

The following code shows how to remove a connector at runtime.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<input type="button" value="Remove Connector" @onclick="@RemoveConnector">
```

```

<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">
  <SnapSettings Constraints="@snapConstraints"></SnapSettings>
</SfDiagramComponent>
@code
{
  SnapConstraints snapConstraints = SnapConstraints.None;
  //Defines diagram's connector collection
  DiagramObjectCollection<Connector> connectors = new
  DiagramObjectCollection<Connector>();
  protected override void OnInitialized()
  {
    Connector Connector = new Connector()
    {
      ID = "connector1",
      // Set the source and target point of the connector
      SourcePoint = new DiagramPoint() { X = 100, Y = 100 },
      TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
      TargetDecorator = new DecoratorSettings()
      {
        Shape = DecoratorShape.Arrow,
        // Style of the connector segment
        Style = new ShapeStyle() { Fill = "#6f409f", StrokeColor = "#6f409f",
        StrokeWidth = 1 }
      },
      Style = new ShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1 },
      // Type of the connector
      Type = ConnectorSegmentType.Straight,
    };
    connectors.Add(Connector);
  }
  public void RemoveConnector()
  {
    // Remove Node at runtime
    connectors.Remove(connectors[0]);
  }
}

```

A connector can be removed from the diagram by using the native **RemoveAt** method. Refer to the following example that shows how to remove the connector at runtime.

#### ASPX-CS

```

public void RemoveConnector()
{
  connectors.RemoveAt(0);
}

```

#### Update connectors at runtime

You can change any connector's properties at runtime.

The following code example explains how to change the connector properties.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="Update Connector" @onclick="@UpdateConnector">

```

```

<SfDiagramComponent @ref="Diagram" Width="1000px" Height="500px"
Connectors="@connectors">
<SnapSettings Constraints="@snapConstraints"></SnapSettings>
</SfDiagramComponent>
@code
{
SfDiagramComponent Diagram;
SnapConstraints snapConstraints = SnapConstraints.None;
//Defines diagram's connector collection
DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
Connector Connector = new Connector()
{
ID = "connector1",
SourcePoint = new DiagramPoint() { X = 100, Y = 100 },
TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
TargetDecorator = new DecoratorSettings() { Shape = DecoratorShape.Arrow,
Style = new ShapeStyle() { Fill = "#6f409f", StrokeColor = "#6f409f",
StrokeWidth = 1 } },
Style = new ShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1 },
// Type of the connector
Type = ConnectorSegmentType.Straight,
};
connectors.Add(Connector);
}
public void UpdateConnector()
{
Diagram.BeginUpdate();
Diagram.Connectors[0].SourcePoint.X = 50;
Diagram.Connectors[0].SourcePoint.Y = 50;
Diagram.EndUpdate();
}
}

```

---

BeginUpdate and EndUpdate method which allows you to stop the continuous update of control and resume it finally.

---

### Connections

A connectors are used to create a link between two points, nodes or ports to represent the relationships between them.

### Connections with nodes

The [SourceID](#) and [TargetID](#) properties allow to define the nodes to be connected.

The following code example illustrates how to connect two nodes.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="Diagram" Width="1000px" Height="500px"
Nodes="@nodes" Connectors="@connectors">
</SfDiagramComponent>
@code
{

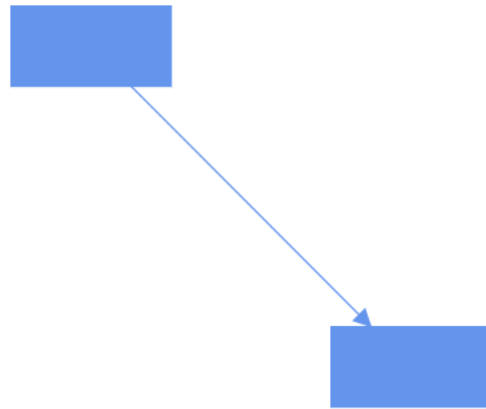
```



```

SfDiagramComponent Diagram;
DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
protected override void OnInitialized()
{
nodes = new DiagramObjectCollection<Node>()
{
new Node()
{
OffsetX = 100,
OffsetY = 100,
Height = 50,
Width = 100,
ID = "node1",
Style = new ShapeStyle(){ Fill = "#6495ED", StrokeColor = "#6495ED" },
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle }
},
new Node()
{
OffsetX = 300,
OffsetY = 300,
Height = 50,
Width = 100,
ID = "node2",
Style = new ShapeStyle(){ Fill = "#6495ED", StrokeColor = "#6495ED" },
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle }
}
};
Connector Connector = new Connector()
{
ID = "connector1",
//Source node id of the connector.
SourceID = "node1",
TargetDecorator = new DecoratorSettings()
{
Style = new ShapeStyle()
{
Fill = "#6495ED",
StrokeColor = "#6495ED",
}
},
//Target node id of the connector.
TargetID = "node2",
Style = new ShapeStyle()
{
Fill = "#6495ED",
StrokeColor = "#6495ED",
},
// Type of the connector
Type = ConnectorSegmentType.Straight,
};
connectors.Add(Connector);
}
}

```



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

#### Connections with ports

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

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

#### ASPX-CS

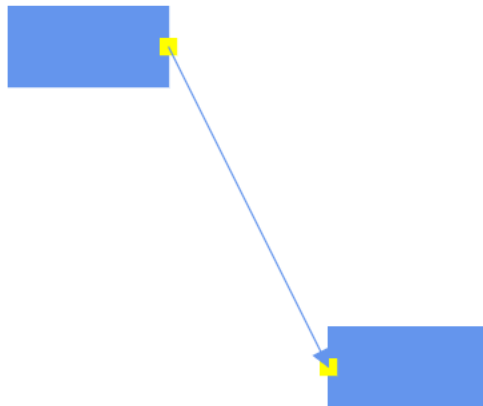
```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="Diagram" Width="1000px" Height="500px"
Nodes="@nodes" Connectors="@connectors">
</SfDiagramComponent>
@code
{
    SfDiagramComponent Diagram;
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
```

```

nodes = new DiagramObjectCollection<Node>()
{
    new Node()
    {
        OffsetX = 100,
        OffsetY = 100,
        Height = 50,
        Ports = new DiagramObjectCollection<PointPort>()
        {
            new PointPort()
            {
                ID="port1",
                Visibility = PortVisibility.Visible,
                Offset = new DiagramPoint() { X = 1, Y = 0.5},
                Height = 10, Width = 10,
                Style = new ShapeStyle(){Fill = "yellow", StrokeColor = "yellow"}
            }
        },
        Width = 100,
        ID = "node1",
        Style = new ShapeStyle(){ Fill = "#6495ED", StrokeColor = "#6495ED"},
        Shape = new BasicShape() { Type = Shapes.Basic, Shape =
        BasicShapeType.Rectangle }
    },
    new Node()
    {
        OffsetX = 300,
        OffsetY = 300,
        Height = 50,
        Width = 100,
        ID = "node2",
        Ports = new DiagramObjectCollection<PointPort>()
        {
            new PointPort()
            {
                ID="port2",
                Visibility = PortVisibility.Visible,
                Offset = new DiagramPoint() { X = 0, Y = 0.5},
                Height = 10, Width = 10,
                Style = new ShapeStyle(){Fill = "yellow", StrokeColor = "yellow"}
            }
        },
        Style = new ShapeStyle(){ Fill = "#6495ED", StrokeColor = "#6495ED" },
        Shape = new BasicShape() { Type = Shapes.Basic, Shape =
        BasicShapeType.Rectangle }
    }
};
Connector Connector = new Connector()
{
    ID = "connector1",
    //Source node id of the connector.
    SourceID = "node1",
    //source node port id.
    SourcePortID = "port1",
    //Target node id of the connector.
    TargetID = "node2",
    //Target node port id.

```

```
TargetPortID = "port2",
TargetDecorator = new DecoratorSettings()
{
    Style = new ShapeStyle()
    {
        Fill = "#6495ED",
        StrokeColor = "#6495ED",
    },
},
Style = new ShapeStyle()
{
    Fill = "#6495ED",
    StrokeColor = "#6495ED",
},
// Type of the connector
Type = ConnectorSegmentType.Straight,
};
connectors.Add(Connector);
}
```



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

*See also*

- [How to customize the connector properties](#)
- [How to interact the connector](#)
- [How to change the segments](#)
- [How to get the connector events](#)
- [How to add annotations to the connectors](#)

### Segments in Blazor Diagram Component

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

#### *Straight*

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

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">
</SfDiagramComponent>
@code
{
    //Defines diagram's connector collection
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Connector Connector = new Connector()
        {
            ID = "connector1",
            SourcePoint = new DiagramPoint()
            {
                X = 100,
                Y = 100
            },
            Style = new ShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1 },
            TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
            //Specify the segments typs as straight.
            Type = ConnectorSegmentType.Straight,
            TargetDecorator = new DecoratorSettings()
            {
                Shape = DecoratorShape.Arrow,
                Style = new ShapeStyle()
                {
                    Fill = "#6f409f",
                    StrokeColor = "#6f409f",
                    StrokeWidth = 1
                }
            }
        };
        connectors.Add(Connector);
    }
}
```



### Orthogonal

Orthogonal segments are used to create segments that are perpendicular to each other. Set the segment Type as [Orthogonal](#) to create a default orthogonal segment and need to specify Type. The following code example illustrates how to create a default orthogonal segment.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">
</SfDiagramComponent>
@code
{
    //Defines diagram's connector collection
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Connector Connector = new Connector()
        {
            ID = "connector1",
            SourcePoint = new DiagramPoint()
            {
                X = 100,
                Y = 100
            },
            Style = new ShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1 },
            TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
            //Specify the segments type as Orthogonal.
            Type = ConnectorSegmentType.Orthogonal,
            TargetDecorator = new DecoratorSettings()
            {
```

```

Shape = DecoratorShape.Arrow,
Style = new ShapeStyle()
{
    Fill = "#6f409f",
    StrokeColor = "#6f409f",
    StrokeWidth = 1
}
};
connectors.Add(Connector);
}
}

```

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

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">
</SfDiagramComponent>
@code
{
    //Defines diagram's connector collection
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Connector Connector = new Connector()
        {
            ID = "connector1",
            SourcePoint = new DiagramPoint()
            {
                X = 100,
                Y = 100
            },
            Style = new ShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1 },
            TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
            Type = ConnectorSegmentType.Orthogonal,
            //Create a new segment with length and direction
            Segments = new DiagramObjectCollection<ConnectorSegment>()
            {
                new OrthogonalSegment
                {
                    Length = 100,
                    Type = ConnectorSegmentType.Orthogonal,
                    Direction = Direction.Right
                },
                new OrthogonalSegment
                {
                    Length = 100,
                    Type = ConnectorSegmentType.Orthogonal,
                    Direction = Direction.Bottom,
                },
            },
            TargetDecorator = new DecoratorSettings()

```

```
{  
  Shape = DecoratorShape.Arrow,  
  Style = new ShapeStyle()  
  {  
    Fill = "#6f409f",  
    StrokeColor = "#6f409f",  
    StrokeWidth = 1  
  }  
};  
connectors.Add(Connector);  
}
```



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

#### Bezier

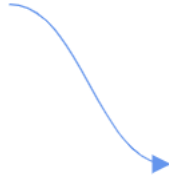
Bezier segments are used to create curve segments and the curves are configurable either with the control points or with vectors. To create a bezier segment, the Type of the segment is set as [Bezier](#) and need to specify type for the connector. The following code example illustrates how to create a default bezier segment.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram  
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">  
</SfDiagramComponent>  
@code  
{
```



```
//Defines diagram's connector collection
DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
    Connector Connector = new Connector()
    {
        ID = "connector1",
        SourcePoint = new DiagramPoint()
        {
            X = 100,
            Y = 100
        },
        Style = new ShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1 },
        TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
        Type = ConnectorSegmentType.Bezier,
        TargetDecorator = new DecoratorSettings()
        {
            Shape = DecoratorShape.Arrow,
            Style = new ShapeStyle()
            {
                Fill = "#6f409f",
                StrokeColor = "#6f409f",
                StrokeWidth = 1
            }
        }
    };
    //Add the connector into connectors's collection.
    connectors.Add(Connector);
}
```



We have properties called [Point1](#) and [Point2](#) which is used to control the points of the bezier connector . And also we have properties called [vector1](#) and [Vector2](#) which is used to defines the length and angle between the source point and target point respectively .The following code example illustrates how to use these properties in our control .

#### ASPX-CS

```
Connector Connector1 = new Connector()
{
    ID = "Connector1",
    Type = ConnectorSegmentType.Bezier,
    SourcePoint = new DiagramPoint { X = 500, Y = 200 },
    TargetPoint = new DiagramPoint { X = 600, Y = 300 },
    Segments = new DiagramObjectCollection<ConnectorSegment>
    {
        new BezierSegment()
        {
            Type = ConnectorSegmentType.Bezier,
            //Defines the point1 and point2 for the bezier connector
            Point1 = new DiagramPoint { X = 500, Y = 100 },
            Point2 = new DiagramPoint { X = 600, Y = 200 }
        }
    };
Connector Connector2 = new Connector()
{
    ID = "Connector2",
    Type = ConnectorSegmentType.Bezier,
    SourcePoint = new DiagramPoint { X = 200, Y = 100 },
    TargetPoint = new DiagramPoint { X = 300, Y = 200 },
    Segments = new DiagramObjectCollection<ConnectorSegment>
```

```
{
new BezierSegment()
{
Type = ConnectorSegmentType.Bezier,
//Defines the Vector1 and Vector2 for the bezier connector
Vector1 = new Vector() {Distance = 100 ,Angle = 90 },
Vector2 = new Vector() {Distance = 45 ,Angle = 45 }
}
}
};
```

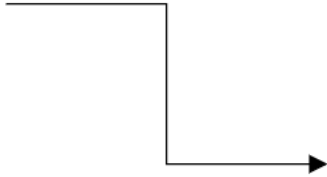
#### *How to create multiple segments*

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

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">
<SnapSettings Constraints="SnapConstraints.None"></SnapSettings>
</SfDiagramComponent>
@code
{
//Defines diagram's connector collection
DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
Connector connector1 = new Connector()
{
ID = "Connector1",
Type = ConnectorSegmentType.Orthogonal,
SourcePoint = new DiagramPoint()
{
X = 100,
Y = 100
},
TargetPoint = new DiagramPoint() { X = 300, Y = 200 },
Segments = new DiagramObjectCollection<ConnectorSegment>()
{
new OrthogonalSegment
{
Length = 100,
Type = ConnectorSegmentType.Orthogonal,
Direction = Direction.Right
},
new OrthogonalSegment
{
Length = 100,
Type = ConnectorSegmentType.Orthogonal,
Direction = Direction.Bottom
}
},
};
//Add the connector into connectors's collection.
```

```
connectors.Add(connector1);
}
}
```



- Similarly, you can create multiple segments for all the connector type.

### Segment Editing

#### Straight segment editing

- End point of each straight segment is represented by a thumb that enables to edit the segment.
- Any number of new segments can be inserted into a straight line by clicking, when Shift and Ctrl keys are pressed (Ctrl+Shift+Click).
- Straight segments can be removed by clicking the segment end point, when Ctrl and Shift keys are pressed (Ctrl+Shift+Click).

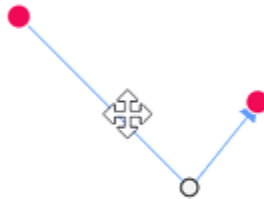
### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="Diagram" Width="1000px" Height="500px"
Connectors="@connectors">
</SfDiagramComponent>
@code
{
    SfDiagramComponent Diagram;
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Connector Connector = new Connector()
        {
            ID = "Connector1",
            Constraints = ConnectorConstraints.Default |
            ConnectorConstraints.DragSegmentThumb,
            SourcePoint = new DiagramPoint { X = 200, Y = 100 },
            TargetPoint = new DiagramPoint { X = 340, Y = 150 },
            // Enable the segment editing
            Segments = new DiagramObjectCollection<ConnectorSegment>
            {
                new StraightSegment()
                {
                    Type = ConnectorSegmentType.Straight,
                    Point = new DiagramPoint { X = 300, Y = 200 }
                }
            }
        }
    }
}
```

```

}
};
connectors.Add(Connector);
}
}

```



### Orthogonal Segment Editing

- Orthogonal thumbs allow you to adjust the length of adjacent segments by clicking and dragging it.
- When necessary, some segments are added or removed automatically, when dragging the segment. This is to maintain proper routing of orthogonality between segments.

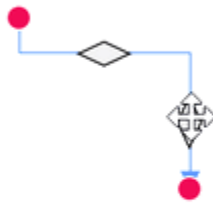
### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="Diagram" Width="1000px" Height="500px"
Connectors="@connectors">
</SfDiagramComponent>
@code
{
    SfDiagramComponent Diagram;
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Connector Connector = new Connector()
        // Enable the segment editing
        {
            ID = "Connector2",
            Constraints = ConnectorConstraints.Default |
            ConnectorConstraints.DragSegmentThumb,
            Type = ConnectorSegmentType.Orthogonal,

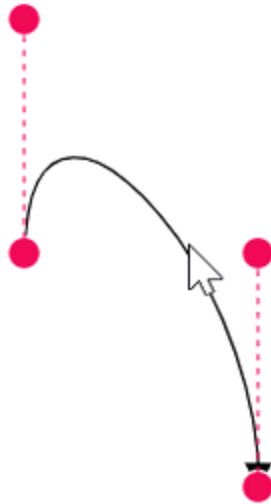
```

```
SourcePoint = new DiagramPoint { X = 400, Y = 100 },  
TargetPoint = new DiagramPoint { X = 500, Y = 200 }  
};  
connectors.Add(Connector);  
}  
}
```



### Bezier Segment Editing

- A segment control point of the Bezier connector is used to change the bezier vectors, points of the connector.



*See also*

- [How to customize the connector properties](#)
- [How to interact the connector](#)
- [How to get the connector events](#)

### Customization in Blazor Diagram Component

#### *Decorator*

Diagram allows you to customize the connector appearances . The following topics shows how to customize several properties of the connectors.

- Starting and ending points of a connector can be decorated with some customizable shapes like arrows, circles, diamond, or path. The connection end points can be decorated with the [SourceDecorator](#) and [TargetDecorator](#) properties of the connector.
- The [Shape](#) property of SourceDecorator allows to define the shape of the source decorator. Similarly, the Shape property of TargetDecorator allows to define the shape of the target decorator.
- To create custom shape for source decorator, use [PathData](#) property. Similarly, to create custom shape for target decorator, use PathData property.
- The following code example illustrates how to create decorators of various shapes.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
```

```

<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">
</SfDiagramComponent>
@code
{
//Defines diagram's connector collection
DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
Connector Connector = new Connector()
{
ID = "connector1",
SourcePoint = new DiagramPoint()
{
X = 100,
Y = 100
},
TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
Style = new ShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1 },
Type = ConnectorSegmentType.Orthogonal,
SourceDecorator = new DecoratorSettings()
{
Shape = DecoratorShape.Circle,
Style = new ShapeStyle() { StrokeColor = "#37909A", Fill = "#37909A",
StrokeWidth = 1 }
},
TargetDecorator = new DecoratorSettings()
{
Shape = DecoratorShape.Custom,
PathData = "M80.5,12.5 C80.5,19.127417 62.59139,24.5 40.5,24.5
C18.40861,24.5 0.5,19.127417 0.5,12.5 C0.5,5.872583 18.40861,0.5 40.5,0.5
C62.59139,0.5 80.5,5.872583 80.5,12.5 z",
Style = new ShapeStyle()
{
StrokeColor = "#37909A",
Fill = "#37909A",
StrokeWidth = 1,
}
}
};
connectors.Add(Connector);
}
}

```

### Decorator appearance

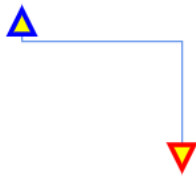
- The source decorator's [StrokeColor](#), [StrokeWidth](#) and [StrokeDashArray](#) properties are used to customize the color, width, and appearance of the decorator.
- To set the border stroke color, stroke width, and stroke dash array for the target decorator, use [StrokeColor](#), [StrokeWidth](#), and [StrokeDashArray](#).
- To set the size for source and target decorator, use [Width](#) and [Height](#) property.

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



**ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">
</SfDiagramComponent>
@code
{
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Connector Connector = new Connector()
        {
            ID = "connector1",
            Style = new ShapeStyle()
            {
                Fill = "#6495ED",
                StrokeColor = "#6495ED",
                StrokeWidth = 1
            },
            SourcePoint = new DiagramPoint()
            {
                X = 100,
                Y = 100
            },
            TargetPoint = new DiagramPoint()
            {
                X = 200,
                Y = 200
            },
            Type = ConnectorSegmentType.Orthogonal,
            SourceDecorator = new DecoratorSettings()
            {
                Shape = DecoratorShape.Arrow,
                Height = 15,
                Width = 15,
                Style = new ShapeStyle() { StrokeColor = "blue", Fill = "yellow",
                StrokeWidth = 3 }
            },
            TargetDecorator = new DecoratorSettings()
            {
                Shape = DecoratorShape.Arrow,
                Height = 15,
                Width = 15,
                Style = new ShapeStyle()
                {
                    StrokeColor = "red",
                    Fill = "yellow",
                    StrokeWidth = 3
                }
            }
        };
        connectors.Add(Connector);
    }
}
```



### Padding

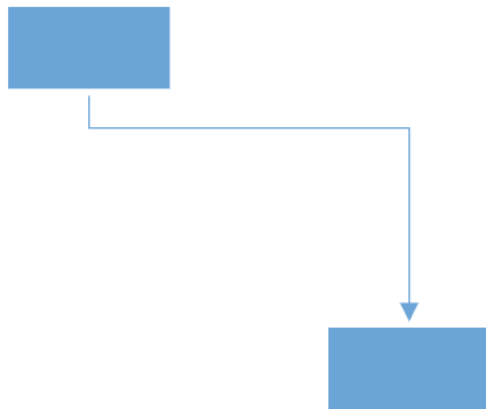
Padding is used to leave the space between the Connector's end point and the object to where it is connected.

- The [SourcePadding](#) property of connector defines space between the source point and the source node of the connector.
- The [TargetPadding](#) property of connector defines space between the end point and the target node of the connector.
- The following code example illustrates how to leave space between the connection end points and source, target nodes.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors"
Nodes="@nodes">
</SfDiagramComponent>
@code
{
//Defines diagram's nodes collection
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
//Defines diagram's connector collection
DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
nodes = new DiagramObjectCollection<Node>()
{
new Node()
```

```
{
    OffsetX = 100,
    OffsetY = 100,
    Height = 50,
    Width = 100,
    ID = "node1",
    Style = new ShapeStyle() { Fill = "#6BA5D7", StrokeColor = "#6BA5D7" },
    Shape = new BasicShape() { Type = Shapes.Basic, Shape =
    BasicShapeType.Rectangle }
},
new Node()
{
    OffsetX = 300,
    OffsetY = 300,
    Height = 50,
    Width = 100,
    ID = "node2",
    Style = new ShapeStyle(){ Fill = "#6BA5D7", StrokeColor = "#6BA5D7" },
    Shape = new BasicShape() { Type = Shapes.Basic, Shape =
    BasicShapeType.Rectangle }
}
};
Connector Connector = new Connector()
{
    ID = "connector1",
    SourceID = "node1",
    TargetID = "node2",
    //Specifies the source and target padding values.
    SourcePadding = 20,
    TargetPadding = 20,
    TargetDecorator = new DecoratorSettings()
    {
        Shape = DecoratorShape.Arrow,
        Style = new ShapeStyle() { Fill = "#6BA5D7", StrokeColor = "#6BA5D7",
        StrokeWidth = 1 }
    },
    Style = new ShapeStyle()
    {
        StrokeColor = "#6BA5D7",
        StrokeWidth = 1
    },
    Type = ConnectorSegmentType.Orthogonal
};
connectors.Add(Connector);
}
}
```



### Bridging

Line bridging creates a bridge for lines to smartly cross over the other lines, at points of intersection. By default, [BridgeDirection](#) is set to [Top](#). Depending upon the direction given bridging direction appears. Bridging can be enabled/disabled either with the [Connector.Constraints](#) or [Diagram.Constraints](#). The following code example illustrates how to enable line bridging.

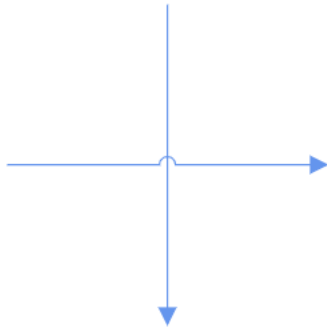
### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px"@bind-Constraints="@diagramConstraints"
Height="500px" Connectors="@connectors">
</SfDiagramComponent>
@code
{
    DiagramConstraints diagramConstraints = DiagramConstraints.Default |
    DiagramConstraints.Bridging;
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Connector Connector1 = new Connector()
        {
            ID = "connector1",
            // Bridge space value has been defined
            BridgeSpace = 20,
            Style = new ShapeStyle()
            {
                Fill = "#6495ED",
                StrokeColor = "#6495ED",
                StrokeWidth = 1
            },
        },
```

```
SourcePoint = new DiagramPoint()
{
    X = 200 , Y = 200
},
TargetPoint = new DiagramPoint()
{
    X = 400,
    Y = 200
},
Type = ConnectorSegmentType.Orthogonal,
TargetDecorator = new DecoratorSettings()
{
    Shape = DecoratorShape.Arrow ,
    Style = new ShapeStyle()
    {
        Fill = "#6495ED",
        StrokeColor = "#6495ED",
        StrokeWidth = 1
    }
};
connectors.Add(Connector1);
Connector Connector2 = new Connector()
{
    ID = "connector2",
    Style = new ShapeStyle()
    {
        Fill = "#6495ED",
        StrokeColor = "#6495ED",
        StrokeWidth = 1
    },
    SourcePoint = new DiagramPoint()
    {
        X = 300,
        Y = 100
    },
    TargetPoint = new DiagramPoint()
    {
        X = 300,
        Y = 300
    },
    Type = ConnectorSegmentType.Orthogonal,
    TargetDecorator = new DecoratorSettings()
    {
        Shape = DecoratorShape.Arrow,
        Style = new ShapeStyle()
        {
            Fill = "#6495ED",
            StrokeColor = "#6495ED",
            StrokeWidth = 1
        }
    };
connectors.Add(Connector2);
}
```

The [BridgeSpace](#) property of connectors can be used to define the width for line bridging. By default, the BridgeSpace value is 10px.

Limitation: [BezierSegment](#) do not support bridging.



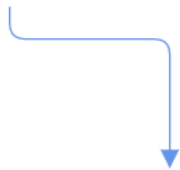
### Corner radius

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">
</SfDiagramComponent>
@code
{
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Connector Connector = new Connector()
        {
            ID = "connector1",
            Style = new ShapeStyle()
            {
                Fill = "#6495ED",
                StrokeColor = "#6495ED",
                StrokeWidth = 1
            },
            SourcePoint = new DiagramPoint()
```

```
{
X = 100,
Y = 100
},
TargetPoint = new DiagramPoint()
{
X = 200,
Y = 200
},
//specify the corner radius value.
CornerRadius = 10,
Type = ConnectorSegmentType.Orthogonal,
TargetDecorator = new DecoratorSettings()
{
Shape = DecoratorShape.Arrow,
Style = new ShapeStyle()
{
Fill = "#6495ED",
StrokeColor = "#6495ED",
StrokeWidth = 1
}
}
};
connectors.Add(Connector);
}
```



### Appearance

- The connector's [StrokeWidth](#), [StrokeColor](#), [StrokeDashArray](#), and [Opacity](#) properties are used to customize the appearance of the connector segments.
- The [IsVisible](#) property of the connector indicating whether the connector is visible in the user interface
- Default values for all the [Connectors](#) can be set by using the [ConnectorCreating](#) event callback method. For example, if all connectors have the same type or having the same property then such properties can be moved into ConnectorCreating.

### Segment appearance

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

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">
</SfDiagramComponent>
@code
{
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Connector Connector1 = new Connector()
        {
            ID = "connector1",
            SourcePoint = new DiagramPoint()
            {
                X = 100,
                Y = 100
            },
            TargetPoint = new DiagramPoint()
            {
                X = 200,
                Y = 200
            },
            Style = new ShapeStyle()
            {
                StrokeColor = "red",
                StrokeWidth = 2,
                StrokeDashArray = "2,2"
            },
            Type = ConnectorSegmentType.Orthogonal,
            TargetDecorator = new DecoratorSettings()
            {
                Shape = DecoratorShape.Arrow,
                Style = new ShapeStyle()
                {
                    Fill = "#6f409f",
                    StrokeColor = "#6f409f",
                    StrokeWidth = 1
                }
            }
        };
    }
}
```



```
connectors.Add(Connector1);  
}  
}
```

### Decorator appearance

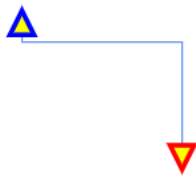
- The source decorator's [StrokeColor](#), [StrokeWidth](#) and [StrokeDashArray](#) properties are used to customize the color, width, and appearance of the decorator.
- To set the border stroke color, stroke width, and stroke dash array for the target decorator, use [StrokeColor](#), [StrokeWidth](#), and [StrokeDashArray](#).
- To set the size for source and target decorator, use [Width](#) and [Height](#) property.

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

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram  
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">  
</SfDiagramComponent>  
@code  
{  
    DiagramObjectCollection<Connector> connectors = new  
    DiagramObjectCollection<Connector>();  
    protected override void OnInitialized()  
    {  
        Connector Connector = new Connector()  
        {  
            ID = "connector1",  
            Style = new ShapeStyle()  
            {  
                Fill = "#6495ED",  
                StrokeColor = "#6495ED",  
                StrokeWidth = 1  
            },  
            SourcePoint = new DiagramPoint()  
            {  
                X = 100,  
                Y = 100  
            },  
            TargetPoint = new DiagramPoint()  
            {  
                X = 200,  
                Y = 200  
            },  
            Type = ConnectorSegmentType.Orthogonal,  
            SourceDecorator = new DecoratorSettings()  
            {  
                Shape = DecoratorShape.Arrow,  
                Height = 15,  
                Width = 15,  
                Style = new ShapeStyle()  
                {  
                    StrokeColor = "blue", Fill = "yellow", StrokeWidth = 3  
                }  
            },  
            TargetDecorator = new DecoratorSettings()  
            {
```

```
Shape = DecoratorShape.Arrow,  
Height = 15,  
Width = 15,  
Style = new ShapeStyle()  
{  
    StrokeColor = "red",  
    Fill = "yellow",  
    StrokeWidth = 3  
}  
};  
connectors.Add(Connector);  
}  
}
```



### Constraints

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

The following code illustrates how to disable selection.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram  
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">  
</SfDiagramComponent>  
@code  
{
```

```

DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
    Connector Connector = new Connector()
    {
        ID = "connector1",
        SourcePoint = new DiagramPoint()
        {
            X = 100,
            Y = 100
        },
        TargetPoint = new DiagramPoint()
        {
            X = 200,
            Y = 200
        },
        Type = ConnectorSegmentType.Orthogonal,
        TargetDecorator = new DecoratorSettings()
        {
            Shape = DecoratorShape.Arrow,
            Style = new ShapeStyle()
            {
                Fill = "black",
                StrokeColor = "black",
                StrokeWidth = 1
            }
        },
        Style = new ShapeStyle()
        {
            StrokeColor = "black",
            StrokeWidth = 1
        },
    };
    //Disable the select constraint
    Constraints = ConnectorConstraints.Default & ~ConnectorConstraints.Select,
};
connectors.Add(Connector);
}
}

```

### Custom properties

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

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@connectors">
</SfDiagramComponent>
@code
{
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()

```

```
{
Dictionary<string, object> ConnectorInfo = new Dictionary<string, object>();
ConnectorInfo.Add("connectorInfo", "Central Connector");
Connector Connector = new Connector()
{
ID = "connector1",
SourcePoint = new DiagramPoint()
{
X = 100,
Y = 100
},
TargetPoint = new DiagramPoint()
{
X = 200,
Y = 200
},
Type = ConnectorSegmentType.Orthogonal,
TargetDecorator = new DecoratorSettings()
{
Shape = DecoratorShape.Arrow,
Style = new ShapeStyle()
{
Fill = "black",
StrokeColor = "black",
StrokeWidth = 1
}
},
Style = new ShapeStyle()
{
StrokeColor = "black",
StrokeWidth = 1
},
//Define the add info value.
AdditionalInfo = ConnectorInfo
};
connectors.Add(Connector);
}
```

*See also*

- [How to interact the connector](#)
- [How to change the segments](#)
- [How to get the connector events](#)

### Interaction in Blazor Diagram Component

Connectors can be selected, dragged, and routed over the diagram page.

#### Select

A connector can be selected at runtime by using the [Select](#) method and clear the selection in the diagram using the [ClearSelection](#). The following code explains how to select and clear selection in the diagram.

#### ASPX-CS

```

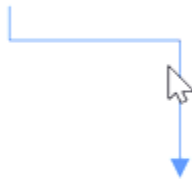
@using Syncfusion.Blazor.Diagram
@using System.Collections.ObjectModel
<input type="button" value="Select" @onclick="OnSelect">
<input type="button" value="UnSelect" @onclick="@UnSelect" />
<SfDiagramComponent @ref="Diagram" Width="1000px" Height="500px"
Connectors="@connectors">
</SfDiagramComponent>
@code
{
    SfDiagramComponent Diagram;
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Dictionary<string, object> ConnectorInfo = new Dictionary<string, object>();
        ConnectorInfo.Add("connectorInfo", "Central Connector");
        Connector Connector = new Connector()
        {
            ID = "connector1",
            SourcePoint = new DiagramPoint()
            {
                X = 100,
                Y = 100
            },
            TargetPoint = new DiagramPoint()
            {
                X = 200,
                Y = 200
            },
            Type = ConnectorSegmentType.Orthogonal,
            TargetDecorator = new DecoratorSettings()
            {
                Shape = DecoratorShape.Arrow,
                Style = new ShapeStyle()
                {
                    Fill = "#6495ED",
                    StrokeColor = "#6495ED",
                    StrokeWidth = 1
                }
            },
            Style = new ShapeStyle()
            {
                StrokeColor = "#6495ED",
                StrokeWidth = 1
            },
            AdditionalInfo = ConnectorInfo
        };
        connectors.Add(Connector);
    }
    public void OnSelect()
    {
        // Select the Connector
        Diagram.Select(new ObservableCollection<IDiagramObject> {
        Diagram.GetObject(Diagram.Connectors[0].ID) as IDiagramObject });
    }
    public void UnSelect()
    {

```

```
// clear selection in the diagram
Diagram.ClearSelection();
}
}
```

And also the selection can be enabled during the interaction.

- An element can be selected by clicking that element.
- When you select the elements in the diagram, the [SelectionChanging](#) and [SelectionChanged](#) event gets triggered and do customization in this event.



### Drag

A connector can be dragged at runtime by using the [Drag](#) method. The following code explains how to drag the connector by using the drag method.

### ASPX-CS

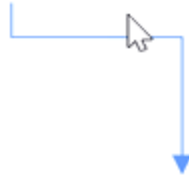
```
@using Syncfusion.Blazor.Diagram
<input type="button" value="Drag" @onclick="OnDrag">
<SfDiagramComponent @ref="Diagram" Width="1000px" Height="500px"
Connectors="@connectors">
</SfDiagramComponent>
@code
{
    SfDiagramComponent Diagram;
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Dictionary<string, object> ConnectorInfo = new Dictionary<string, object>();
        ConnectorInfo.Add("connectorInfo", "Central Connector");
        Connector Connector = new Connector()
    {
```

```
ID = "connector1",
SourcePoint = new DiagramPoint()
{
    X = 100,
    Y = 100
},
TargetPoint = new DiagramPoint()
{
    X = 200,
    Y = 200
},
Type = ConnectorSegmentType.Orthogonal,
TargetDecorator = new DecoratorSettings()
{
    Shape = DecoratorShape.Arrow,
    Style = new ShapeStyle()
    {
        Fill = "black",
        StrokeColor = "black",
        StrokeWidth = 1
    }
},
Style = new ShapeStyle()
{
    StrokeColor = "black",
    StrokeWidth = 1
},
AdditionalInfo = ConnectorInfo
};
connectors.Add(Connector);
}

public void OnDrag()
{
    // Drag the connector
    Diagram.Drag(Diagram.Connectors[0], 10, 10);
}
```

And also drag the connector during the interaction.

- An object can be dragged by clicking and dragging it. When multiple elements are selected, dragging any one of the selected elements move all the selected elements.
- When you drag the elements in the diagram, the [PositionChanging](#) and [PositionChanged](#) event gets triggered and to do customization in this event.



### End Point Dragging

The connector can be selected by clicking it. When the connector is selected, circles will be added on the starting and ending of the connector that is represented by Thumbs. Clicking and dragging those handles helps you to adjust the source and target points.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="Diagram" Width="1000px" Height="500px"
Connectors="@connectors">
</SfDiagramComponent>
@code
{
    SfDiagramComponent Diagram;
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Dictionary<string, object> ConnectorInfo = new Dictionary<string, object>();
        ConnectorInfo.Add("connectorInfo", "Central Connector");
        Connector Connector = new Connector()
        {
            ID = "connector1",
            SourcePoint = new DiagramPoint()
            {
                X = 100,
                Y = 100
            },
            TargetPoint = new DiagramPoint()
            {
                X = 200,
                Y = 200
            },
            Type = ConnectorSegmentType.Orthogonal,
            TargetDecorator = new DecoratorSettings()
```



```
{
    Shape = DecoratorShape.Arrow,
    Style = new ShapeStyle()
    {
        Fill = "black",
        StrokeColor = "black",
        StrokeWidth = 1
    },
    Style = new ShapeStyle()
    {
        StrokeColor = "black",
        StrokeWidth = 1
    },
    AdditionalInfo = ConnectorInfo
};
connectors.Add(Connector);
}
```



*See also*

- [How to customize the connector properties](#)
- [How to change the segments](#)
- [How to get the connector events](#)

Events in Blazor Diagram Component

*Selection change*

- While selecting the diagram elements, the following events can be used to do the customization.

- When selecting/unselecting the diagram elements, the following events are getting triggered and to do customization on those events

|Event Name|Arguments|Description|

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

|[SelectionChanging](#)|[SelectionChangingEventArgs](#)|Triggers before the selection is changed in the diagram.|

|[SelectionChanged](#)|[SelectionChangedEventArgs](#)|Triggers when the selection is changed in the diagram.|

The following code example explains how to get the selection change event in the diagram.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="Diagram" Width="1000px"
SelectionChanging="@OnSelectionChanging"
SelectionChanged="@OnSelectionChanged" Height="500px"
Connectors="@connectors">
</SfDiagramComponent>
@code
{
    SfDiagramComponent Diagram;
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Connector Connector = new Connector()
        {
            ID = "connector1",
            SourcePoint = new DiagramPoint()
            {
                X = 100,
                Y = 100
            },
            TargetPoint = new DiagramPoint()
            {
                X = 200,
                Y = 200
            },
            Type = ConnectorSegmentType.Orthogonal,
            TargetDecorator = new DecoratorSettings()
            {
                Shape = DecoratorShape.Arrow,
                Style = new ShapeStyle()
                {
                    Fill = "black",
                    StrokeColor = "black",
                    StrokeWidth = 1
                }
            },
            Style = new ShapeStyle()
            {
                StrokeColor = "black",
```

```

StrokeWidth = 1
},
};
connectors.Add(Connector);
}
// To notify the selection changing event before select/unselect the diagram
elements
public void OnSelectionChanging(SelectionChangingEventArgs args)
{
//sets true to cancel the selection.
args.Cancel = true;
}
// To notify the selection is changed in the diagram.
private void OnSelectionChanged(SelectionChangedEventArgs arg)
{
//Action to be performed
}
}

```

### Position change

- While dragging the diagram elements, the following events can be used to do the customization.

Event Name	Arguments	Description
<a href="#">PositionChanging</a>	<a href="#">PositionChangingEventArgs</a>	Triggers while dragging the elements in the diagram.
<a href="#">PositionChanged</a>	<a href="#">PositionChangedEventArgs</a>	Triggers when the node's/connector's position is changed.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="Diagram" Width="1000px"
PositionChanging="@OnPositionChanging" PositionChanged="@OnPositionChanged"
Height="500px" Connectors="@connectors">
</SfDiagramComponent>
@code
{
SfDiagramComponent Diagram;
DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
Connector Connector = new Connector()
{
ID = "connector1",
SourcePoint = new DiagramPoint()
{
X = 100,
Y = 100
},
TargetPoint = new DiagramPoint()
{

```

```

X = 200,
Y = 200
},
Type = ConnectorSegmentType.Orthogonal,
TargetDecorator = new DecoratorSettings()
{
    Shape = DecoratorShape.Arrow,
    Style = new ShapeStyle()
    {
        Fill = "black",
        StrokeColor = "black",
        StrokeWidth = 1
    }
},
Style = new ShapeStyle()
{
    StrokeColor = "black",
    StrokeWidth = 1
},
};
connectors.Add(Connector);
}
// To notify the position changing event before dragging the diagram
elements
public void OnPositionChanging(PositionChangingEventArgs args)
{
    //sets true to cancel the dragging.
    args.Cancel = true;
}
// To notify the position changed event after dragged the diagram elements.
private void OnPositionChanged(PositionChangedEventArgs arg)
{
    //Action to be performed
}
}

```

### Connection change

- While changing the connection of the connector, the following events can be used to do the customization.

Event Name	Arguments	Description
-----	-----	-----
<a href="#">ConnectionChanging</a>	<a href="#">ConnectionChangingEventArgs</a>	Triggers before the connector's source or target point is connect or disconnect from the source or target.
<a href="#">ConnectionChanged</a>	<a href="#">ConnectionChangedEventArgs</a>	Triggers when the connector's source or target point is connected or disconnected from the source or target.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="Diagram" Width="1000px"
    ConnectionChanging="@OnConnectionChanging"

```

```

ConnectionChanged="@OnConnectionChange" Height="500px"
Connectors="@connectors" Nodes="@nodes">
</SfDiagramComponent>
@code
{
    SfDiagramComponent Diagram;
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>()
        {
            new Node()
            {
                OffsetX = 100,
                OffsetY = 100,
                Height = 50,
                Width = 100,
                ID = "node1",
            },
        };
        Connector Connector = new Connector()
        {
            ID = "connector1",
            SourcePoint = new DiagramPoint()
            {
                X = 200,
                Y = 200
            },
            TargetID = "node1",
            Type = ConnectorSegmentType.Orthogonal,
            TargetDecorator = new DecoratorSettings()
            {
                Shape = DecoratorShape.Arrow,
                Style = new ShapeStyle()
                {
                    Fill = "black",
                    StrokeColor = "black",
                    StrokeWidth = 1
                }
            },
            Style = new ShapeStyle()
            {
                StrokeColor = "black",
                StrokeWidth = 1
            },
        };
        connectors.Add(Connector);
    }
    // To notify the connection changing event before the connection change
    private void OnConnectionChanging(ConnectionChangingEventArgs args)
    {
        //sets true to cancel the connection change.
        args.Cancel = true;
    }
    // To notify the connection changed event after connection has changed

```

```
private void OnConnectionChange(ConnectionChangedEventArgs args)
{
    //Action to be performed
}
}
```

*See also*

- [How to customize the connector properties](#)
- [How to interact the connector](#)
- [How to change the segments](#)

## NodeGroup in Blazor Diagram Component

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

### Create NodeGroup

#### Add NodeGroup when initializing diagram

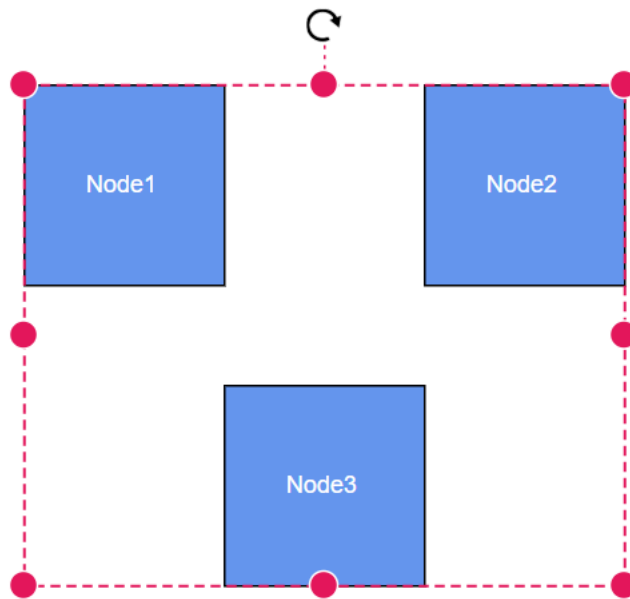
A node group can be added to the diagram model through [Nodes](#) collection. To define an object as node group, add the child objects to the [Children](#) collection of the node group. The following code illustrates how to create a node group.

- While creating node group, its child node needs to be declared before the node group declaration.
- Add a node to the existing node group child by using the [Group](#) method.
- The nodegroup's [UnGroup](#) method is used to define whether the node group can be ungrouped or not.
- A node group can be added into a child of another node group.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
@* Initialize the diagram with NodeCollection *@
<SfDiagramComponent Height="500px" @ref="diagram" Nodes="@nodes">
    <SnapSettings>
        <HorizontalGridLines LineColor="white"
            LineDashArray="2,2"></HorizontalGridLines>
        <VerticalGridLines LineColor="white"
            LineDashArray="2,2"></VerticalGridLines>
    </SnapSettings>
</SfDiagramComponent>
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        Node node1 = createNode("node1", 100, 100, "Node1");
        Node node2 = createNode("node2", 300, 100, "Node2");
        Node node3 = createNode("node3", 200, 250, "Node3");
    }
}
```

```
NodeGroup groupnode = new NodeGroup();
// Grouping node 1 and node 2 into a single nodegroup
groupnode.Children = new string[] { "node1", "node2" };
nodes.Add(node1);
nodes.Add(node2);
nodes.Add(node3);
nodes.Add(groupnode);
}
public Node createNode(string id, double offx, double offy, string content)
{
    Node node = new Node()
    {
        ID = id,
        OffsetX = offx,
        OffsetY = offy,
        Height = 100,
        Width = 100,
        Style = new ShapeStyle() { Fill = "#6495ED" }
    };
    ShapeAnnotation annotation = new ShapeAnnotation
    {
        ID = "annotation1",
        Content = content,
        Style = new TextStyle()
        {
            Color = "white",
            Fill = "transparent",
            StrokeColor = "None"
        },
    };
    node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
    {
        annotation
    };
    return node;
}
protected override async Task OnAfterRenderAsync(bool firstRender)
{
    if (firstRender)
    {
        await Task.Delay(500);
        diagram.SelectAll();
        // Adding the third node into the existing nodegroup
        diagram.Group();
    }
}
}
```



The following code illustrates how a ungroup at runtime.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
/* Initialize the diagram with nodes */
<SfDiagramComponent Height="500px" @ref="diagram" Nodes="@nodes" />
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        Node node1 = createNode("node1", 100, 100, "Node1");
        Node node2 = createNode("node2", 300, 100, "Node2");
        NodeGroup groupnode = new NodeGroup();
        // Grouping node 1 and node 2 into a single nodegroup
        groupnode.Children = new string[] { "node1", "node2" };
        nodes.Add(node1);
        nodes.Add(node2);
        nodes.Add(groupnode);
    }
    public Node createNode(string id, double offx, double offy, string content)
    {
        Node node = new Node()
        {
            ID = id,
            OffsetX = offx,
            OffsetY = offy,
            Height = 100,
            Width = 100,
        }
    }
}
```



```

Style = new ShapeStyle() { Fill = "#6495ED" }
};
ShapeAnnotation annotation = new ShapeAnnotation
{
    ID = "annotation1",
    Content = content,
    Style = new TextStyle()
    {
        Color = "white",
        Fill = "transparent",
        StrokeColor = "None"
    },
};
node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
{
    annotation
};
return node;
}
protected override async Task OnAfterRenderAsync(bool firstRender)
{
    if (firstRender)
    {
        await Task.Delay(500);
        diagram.SelectAll();
        // Ungroup the selected group into nodes
        diagram.UnGroup();
    }
}
}

```

### Add NodeGroup at runtime

A node group can be added at runtime by using Nodes collection of diagram.

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

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="AddGroup" @onclick="@AddGroup" />
/* Initialize the diagram with NodeCollection */
<SfDiagramComponent Height="500px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    NodeGroup groupnode = new NodeGroup();
    protected override void OnInitialized()
    {
        Node node1 = createNode("node1", 100, 100, "Node1");
        Node node2 = createNode("node2", 300, 100, "Node2");
        // Grouping node 1 and node 2 into a single nodegroup
        groupnode.Children = new string[] { "node1", "node2" };
        nodes.Add(node1);
        nodes.Add(node2);
    }
    public Node createNode(string id, double offx, double offy, string content)

```

```

{
Node node = new Node()
{
ID = id,
OffsetX = offx,
OffsetY = offy,
Height = 100,
Width = 100,
Style = new ShapeStyle() { Fill = "#6495ED" }
};
ShapeAnnotation annotation = new ShapeAnnotation()
{
ID = "annotation1",
Content = content,
Style = new TextStyle()
{
Color = "white",
Fill = "transparent",
StrokeColor = "None"
},
};
node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
{
annotation
};
return node;
}
private void AddGroup()
{
nodes.Add(groupnode);
}
}

```

- Also, you can add the child to the node group through [AddChild](#) method. The following code illustrates how to add child to the existing node group through AddChild method.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="AddChildToGroup" @onclick="@AddChildToGroup" />
/* Initialize the diagram with nodes */
<SfDiagramComponent @ref="@diagram" Height="500px" Nodes="@nodes"/>
@code
{
SfDiagramComponent diagram;
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
NodeGroup groupnode = new NodeGroup();
protected override void OnInitialized()
{
Node node1 = createNode("node1", 100, 100, "Node1");
Node node2 = createNode("node2", 300, 100, "Node2");
// Grouping node 1 and node 2 into a single nodegroup
groupnode.Children = new string[] { "node1", "node2" };
nodes.Add(node1);
nodes.Add(node2);
}
}

```

```
nodes.Add(groupnode);
}
public Node createNode(string id, double offx, double offy, string content)
{
    Node node = new Node()
    {
        ID = id,
        OffsetX = offx,
        OffsetY = offy,
        Height = 100,
        Width = 100,
        Style = new ShapeStyle() { Fill = "#6495ED" }
    };
    ShapeAnnotation annotation = new ShapeAnnotation()
    {
        ID = "annotation1",
        Content = content,
        Style = new TextStyle()
        {
            Color = "white",
            Fill = "transparent",
            StrokeColor = "None"
        },
    };
    node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
    {
        annotation
    };
    return node;
}
private async void AddChildToGroup()
{
    NodeGroup group = diagram.SelectionSettings.Nodes[0] as NodeGroup;
    Node node = new Node()
    {
        ID = "node" + nodes.Count.ToString(),
        OffsetX = 250,
        OffsetY = 250,
        Width = 50,
        Height = 50,
        Style = new ShapeStyle() { Fill = "#6495ED" },
        Annotations = new DiagramObjectCollection<ShapeAnnotation>()
        {
            new ShapeAnnotation()
            {
                Content = "Node" + nodes.Count.ToString(),
                Style = new TextStyle()
                {
                    Color = "white",
                    Fill = "transparent",
                    StrokeColor = "None"
                },
            },
        }
    };
    await diagram.AddChild(group as NodeGroup, node);
}
```

```
}

```

### Update position at runtime

You can change the position of the node group similar to node. For more information about node positioning, refer to [Positioning](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<input type="button" value="UpdatePosition" @onclick="@UpdatePosition" />
/* Initialize the diagram with NodeCollection */
<SfDiagramComponent Height="500px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    NodeGroup groupnode = new NodeGroup();
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node1 = createNode("node1", 100, 100, "Node1");
        Node node2 = createNode("node2", 300, 100, "Node2");
        // Grouping node 1 and node 2 into a single nodegroup
        groupnode.Children = new string[] { "node1", "node2" };
        nodes.Add(node1);
        nodes.Add(node2);
        nodes.Add(groupnode);
    }
    public Node createNode(string id, double offx, double offy, string content)
    {
        Node node = new Node()
        {
            ID = id,
            OffsetX = offx,
            OffsetY = offy,
            Height = 100,
            Width = 100,
            Style = new ShapeStyle() { Fill = "#6495ED" }
        };
        ShapeAnnotation annotation = new ShapeAnnotation
        {
            ID = "annotation1",
            Content = content,
            Style = new TextStyle()
            {
                Color = "white",
                Fill = "transparent",
                StrokeColor = "None"
            },
        };
        node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
        {
            annotation
        };
        return node;
    }
    private void UpdatePosition()

```

```
{
  nodes[2].OffsetX = 500;
  nodes[2].OffsetY = 200;
}
```

### Appearance

You can change the appearance of the node group similar to node. For more information about node appearance, refer to [Appearance](#).

### Interaction

You can edit the node group and its children at runtime. We able to interact the nodegroup as like the node interaction like resize, rotate and drag. For more information about node interaction, refer to [Interaction](#).

### Selecting a NodeGroup

When a child element of any node group is clicked, its contained node group is selected instead of the child element. With consecutive clicks on the selected element, selection is changed from top to bottom in the hierarchy of parent nodegroup to its children.

### See Also

- [How to enable/disable the behavior of the node](#)

## Annotations

### Annotation in Blazor Diagram Component

The [Annotation](#) is a block of text that can be displayed over a node or connector and it is used to textually represent an object with a string that can be edited at run time. Multiple annotations can be added to a node or connector.

### Create Annotations

An annotation can be added to a node or connector by defining the annotation object and adding that to the annotation collection of the node or connector. The [Content](#) property of annotation defines the text to be displayed. The following code explains how to create an annotation.

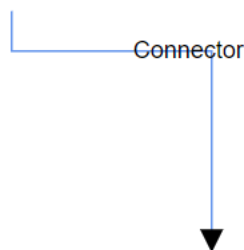
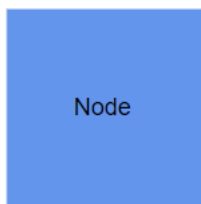
### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" Connectors="@connectors"
/>
@code
{
  // Defines diagram's connector collection
  DiagramObjectCollection<Connector> connectors;
  // Defines diagram's node collection
  DiagramObjectCollection<Node> nodes;
  protected override void OnInitialized()
  {
    nodes = new DiagramObjectCollection<Node>();
    Node node = new Node()
    {
      Width = 100,
      Height = 100,
```

```

OffsetX = 100,
OffsetY = 100,
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
Annotations = new DiagramObjectCollection<ShapeAnnotation>()
{
    // A annotation is created and stored in Annotations collection of Node.
    new ShapeAnnotation { Content = "Node" }
};
nodes.Add(node);
connectors = new DiagramObjectCollection<Connector>();
Connector connector = new Connector()
{
    SourcePoint = new DiagramPoint() { X = 300, Y = 40 },
    TargetPoint = new DiagramPoint() { X = 400, Y = 160 },
    Type = ConnectorSegmentType.Orthogonal,
    Style = new TextStyle() { StrokeColor = "#6495ED" },
    Annotations = new DiagramObjectCollection<PathAnnotation>()
    {
        // A annotation is created and stored in Annotations collection of
        Connector.
        new PathAnnotation { Content = "Connector" }
    };
connectors.Add(connector);
}
}

```



\* [ID](#) for each annotation should be unique and so it is further used to find the annotation at runtime and do any customization.

\* By default, node's annotation positioned in center point of the shape.

\* By default, connector's path annotation positioned in center point of its path.

#### *Add Annotations at runtime*

You can add Annotation at runtime to the Annotations collection of the node/connector in the diagram component by using the **Add** method.

The following code explains how to add an annotation to a node at runtime by using **Add** method.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
```

```

<input value="Addlabel" type="button" @onclick="@AddLabel" name="Addlabel"
/>
<SfDiagramComponent Height="600px" @ref="@diagram" Nodes="@nodes">
</SfDiagramComponent>
@code
{
// Reference to diagram
SfDiagramComponent diagram;
// Defines diagram's node collection
DiagramObjectCollection<Node> nodes;
protected override void OnInitialized()
{
nodes = new DiagramObjectCollection<Node>();
Node node = new Node()
{
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
};
nodes.Add(node);
}
// Method to add annotation at runtime
public void AddLabel()
{
ShapeAnnotation annotation = new ShapeAnnotation { Content = "Annotation" };
(diagram.Nodes[0].Annotations as
DiagramObjectCollection<ShapeAnnotation>).Add(annotation);
}
}

```

Also, the annotations can be added at runtime by using the [AddAsync](#) method. The `await` operator suspends evaluation of the enclosing async method until the asynchronous operation represented by its operand completes.

The following code explains how to add an annotation to a node at runtime by using `AddAsync` method.

### **C#**

```

//Method to add annotation at runtime
public async void AddLabel()
{
ShapeAnnotation annotation = new ShapeAnnotation { Content = "Annotation" };
await(diagram.Nodes[0].Annotations as
DiagramObjectCollection<ShapeAnnotation>).AddAsync(annotation);
}

```



### *Remove Annotations*

A collection of annotations can be removed from the node by using the `RemoveAt` method. The following code explains how to remove an annotation to a node.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<input value="Removelabel" type="button" @onclick="@RemoveLabel"
name="Removelabel" />
<SfDiagramComponent Height="600px" @ref="@diagram" Nodes="@nodes" />
@code
{
    // Reference to diagram
    SfDiagramComponent diagram;
    // Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            Style = new ShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
        };
        node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
        {
            new ShapeAnnotation { ID = "label", Content = "Annotation" },
        };
        nodes.Add(node);
    }
    // Method to remove annotation at runtime
    public void RemoveLabel()
    {
        (diagram.Nodes[0].Annotations as
        DiagramObjectCollection<ShapeAnnotation>).RemoveAt(0);
    }
}
```

Also, A collection of annotations can be removed from the node by using the `Remove` method.

#### **ASPX-CS**

```
// Method to remove annotation at runtime using Remove method.
```



```
public void RemoveLabel()
{
    ShapeAnnotation annotation = (diagram.Nodes[0].Annotations[0]) as
    ShapeAnnotation;
    (diagram.Nodes[0].Annotations as
    DiagramObjectCollection<ShapeAnnotation>).Remove(annotation);
}
```

\* You can delete multiple annotations from node to pass the collection of annotation objects as argument.

\* The **Add**, **Remove**, and **RemoveAt** methods are applicable for connectors too.

#### *Update Annotations at runtime*

You can get the annotation directly from the node's annotations collection property and you can change any annotation properties at runtime.

The following code sample shows how the annotation of the node changed at runtime.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<input value="Updatelabel" type="button" @onclick="@UpdateLabel"
name="Updatelabel" />
<SfDiagramComponent Height="600px" @ref="@diagram" Nodes="@nodes" />
@code
{
    // Reference to diagram
    SfDiagramComponent diagram;
    // Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            Width = 100,
            Height = 100,
            OffsetX = 100,
            Annotations = new DiagramObjectCollection<ShapeAnnotation>()
            {
                new ShapeAnnotation { Content = "Node" }
            },
            OffsetY = 100,
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
        };
        nodes.Add(node);
    }
    // Method to update the annotation at runtime
    public void UpdateLabel()
    {
        diagram.Nodes[0].Annotations[0].Content = "Updated Annotation";
    }
}
```

*See also*

- [How to add or remove annotation constraints](#)
- [How to customize the annotation](#)

### How to position node's annotation

Diagram allows you to customize the position and appearance of the annotation efficiently. Annotation can be aligned relative to the node boundaries. It has Margin, Offset, Horizontal, and Vertical alignment settings. It is quite tricky when all four alignments are used together but gives more control over alignments properties of the [ShapeAnnotation](#) class. Annotations of a node can be positioned using the following properties of [ShapeAnnotation](#).

- Offset
- HorizontalAlignment
- VerticalAlignment
- Margin

### Offset

The [Offset](#) property of an annotation is used to align the annotations based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

The following code shows the relationship between the shape annotation position and path annotation offset (fraction values).

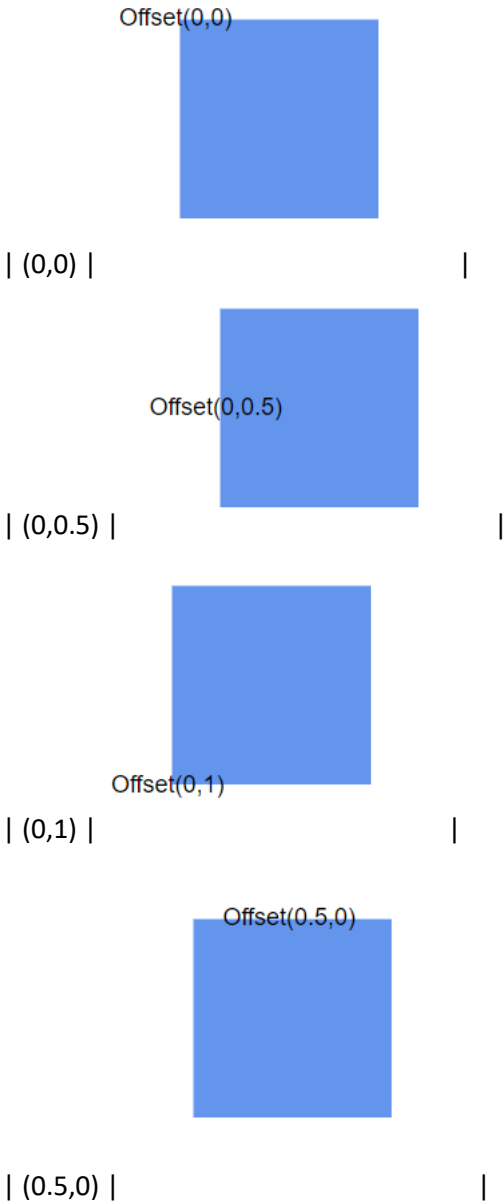
### ASPX-CS

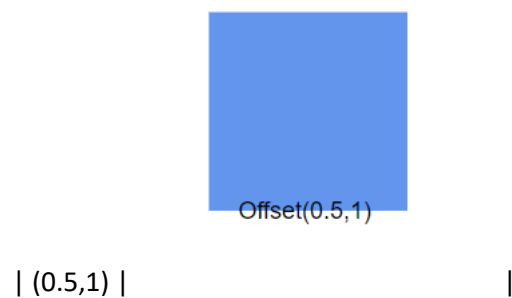
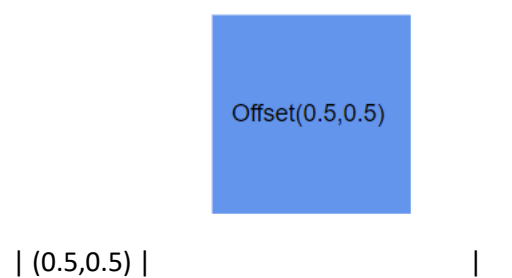
```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    // Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            Width = 100,
            Height = 100,
            OffsetX = 100,
            Annotations = new DiagramObjectCollection<ShapeAnnotation>()
            {
                new ShapeAnnotation
                {
                    Content = "Offset(0,0)",
                    // Sets the offset for an annotation's content
                    Offset = new DiagramPoint() { X = 0, Y = 0 }
                }
            },
            OffsetY = 100,
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
        };
        nodes.Add(node);
    }
}
```

```
}  
}
```

| Offset values | Output |

|---|---|





\* Type of the offset property for node’s shape annotation is [DiagramPoint](#).

\* Type of the offset property for connector’s path annotation is double.

*Horizontal and vertical alignment*


The [HorizontalAlignment](#) property of annotation is used to set how the annotation is horizontally aligned at the annotation position determined from the fraction values. The [VerticalAlignment](#) property is used to set how the annotation is vertically aligned at the annotation position.

The following table shows all the possible alignments visually with 'offset (0, 0)'.

| Horizontal Alignment | Vertical Alignment | Output with Offset(0,0) |


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

Label




| Left | Top |

Label




| Center | Top |

Label

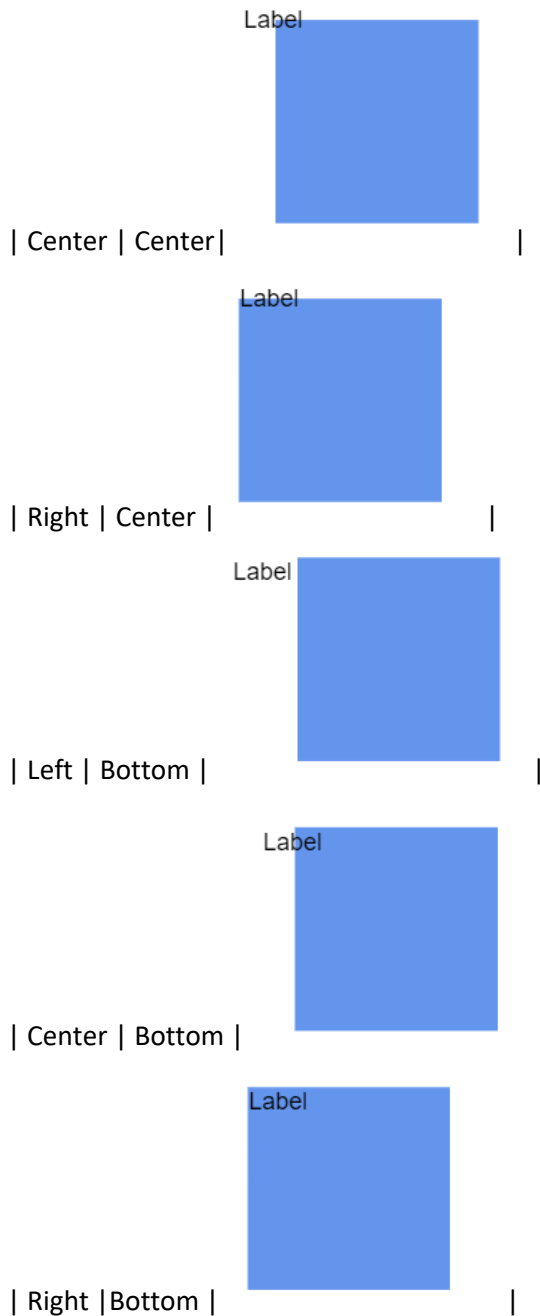


| Right | Top |

Label



| Left | Center |



The following code explains how to align annotations.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    // Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
    }
}
```

```

Node node = new Node()
{
    ID = "node1",
    Width = 100,
    Height = 100,
    OffsetX = 250,
    OffsetY = 250,
    Annotations = new DiagramObjectCollection<ShapeAnnotation>()
    {
        new ShapeAnnotation
        {
            Content = "Annotation",
            // Sets the HorizontalAlignment and VerticalAlignment for the content
            HorizontalAlignment = HorizontalAlignment.Center,
            VerticalAlignment = VerticalAlignment.Center
        }
    },
    Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
};
nodes.Add(node);
}

```

\* The value of the `HorizontalAlignment` is [Center](#) by default.

\* The value of the `VerticalAlignment` is [Center](#) by default.

\* Alignment positioned based on the offset value.

### Margin

[Margin](#) is an absolute value used to add some blank space to any one of its four sides. The annotations can be displaced with the margin property. The following code example explains how to align an annotation based on its Offset, HorizontalAlignment, VerticalAlignment, and Margin values.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    // Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node1",
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            Annotations = new DiagramObjectCollection<ShapeAnnotation>()
            {
                new ShapeAnnotation
                {

```

```

Content = "Task1",
// Sets the margin for the content
Margin = new Margin() { Top = 10},
HorizontalAlignment = HorizontalAlignment.Center,
VerticalAlignment = VerticalAlignment.Top,
Offset = new DiagramPoint() { X = 0.5 , Y = 1}
}
},
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
};
nodes.Add(node);
}
}

```

### Text align

The [TextAlign](#) property of annotation allows you to set how the text should be aligned (Left, Right, Center, or Justify) inside the text block. The following code explains how to set TextAlign for an annotation.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
// Defines diagram's node collection
DiagramObjectCollection<Node> nodes;
protected override void OnInitialized()
{
nodes = new DiagramObjectCollection<Node>();
Node node = new Node()
{
ID = "node1",
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
Annotations = new DiagramObjectCollection<ShapeAnnotation>()
{
new ShapeAnnotation
{
Content = "Text align is set as Left",
Style = new TextStyle()
{
// Sets the textAlign as left for the content
TextAlign = TextAlign.Left
}
}
},
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
};
nodes.Add(node);
}
}

```



*See also*

- [How to add annotation for Connector](#)
- [How to add or remove annotation constraints](#)
- [How to customize the annotation](#)

How to position connector's annotation

Annotations of a connector can be positioned using the following properties of Annotation class.

- Offset
- Alignment
- Displacement
- SegmentAngle
- HorizontalAlignment
- VerticalAlignment
- Margin

*Offset*

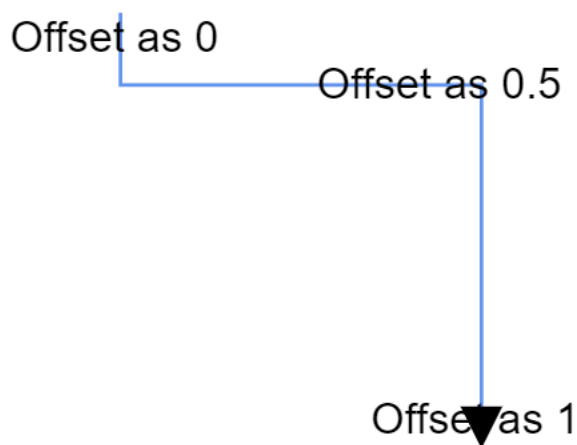
The [Offset](#) property of annotation is used to align the annotations based on fractions. 0 represents Top-Left corner, 1 represents Bottom-Right corner, and 0.5 represents half of Width/Height.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Connectors="@connectors" />
@code
{
    //Defines diagram's connector collection
    DiagramObjectCollection<Connector> connectors;
    protected override void OnInitialized()
    {
        connectors = new DiagramObjectCollection<Connector>();
        Connector connector = new Connector()
        {
            SourcePoint = new DiagramPoint() { X = 300, Y = 40 },
            TargetPoint = new DiagramPoint() { X = 400, Y = 160 },
            Type = ConnectorSegmentType.Orthogonal,
            Style = new TextStyle() { StrokeColor = "#6495ED" },
            Annotations = new DiagramObjectCollection<PathAnnotation>()
            {
                new PathAnnotation
                {
                    Content = "Offset as 0",
                    // Sets the offset of the annotation as 0
                    Offset = 0
                },
                new PathAnnotation
                {
                    Content = "Offset as 0.5",
                    // Sets the offset of the annotation as 0.5
                    Offset = 0.5
                },
                new PathAnnotation
            }
        }
    }
}
```

```
{
Content = "Offset as 1",
// Sets the offset of the annotation as 1
Offset = 1
},
}
};
connectors.Add(connector);
}
}
```

The following image shows the relationship between the annotation position and offset (fraction values).




---

By default, offset value of the connector annotation is 0.5.

---

#### *Alignment*

The connector's annotation can be aligned over its segment path using the [Alignment](#) property of annotation.

#### **ASPX-CS**

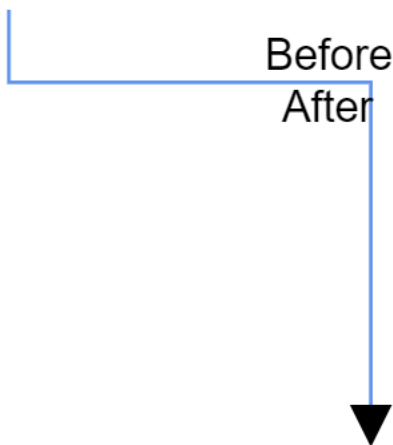
```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Connectors="@connectors" />
@code
{
//Defines diagram's connector collection
DiagramObjectCollection<Connector> connectors;
protected override void OnInitialized()
{
connectors = new DiagramObjectCollection<Connector>();
Connector connector = new Connector()
{
SourcePoint = new DiagramPoint() { X = 300, Y = 40 },
TargetPoint = new DiagramPoint() { X = 400, Y = 160 },
```

```

Type = ConnectorSegmentType.Orthogonal,
Style = new TextStyle() { StrokeColor = "#6495ED" },
Annotations = new DiagramObjectCollection<PathAnnotation>()
{
    new PathAnnotation
    {
        Content = "Before",
        // Sets the alignment of the annotation as Before
        Alignment = AnnotationAlignment.Before
    },
    new PathAnnotation
    {
        Content = "After",
        // Sets the alignment of the annotation as After
        Alignment = AnnotationAlignment.After
    },
};
connectors.Add(connector);
}
}

```

The following screenshot shows how the annotation of the connector aligned over its path.



By default, Alignment value of the connector annotation is **Center**.

#### *Displacement*

The [Displacement](#) property is used to dislocate the annotation by the value given. By default, annotation will be in center of the connector path. When you assign value to the Displacement property, annotation will be displaced from its position by displacement value.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Connectors="@connectors">

```

```

</SfDiagramComponent>
@code
{
//Defines diagram's connector collection
public DiagramObjectCollection<Connector> connectors { get; set; }
protected override void OnInitialized()
{
connectors = new DiagramObjectCollection<Connector>();
Connector connector = new Connector()
{
SourcePoint = new DiagramPoint() { X = 300, Y = 40 },
TargetPoint = new DiagramPoint() { X = 400, Y = 160 },
Type = ConnectorSegmentType.Orthogonal,
Style = new ShapeStyle()
{
StrokeColor = "#6BA5D7"
},
Annotations = new DiagramObjectCollection<PathAnnotation>()
{
new PathAnnotation()
{
Content = "After",
// Set the displacement to the annotation
Displacement = new DiagramPoint() { X = 50, Y = 50 },
Alignment = AnnotationAlignment.After
},
}
};
connectors.Add(connector);
}
}

```

### Segment angle

The [SegmentAngle](#) property is used to rotate the annotation based on the connectors segment direction. By default, annotation will be rotated in the connector path. When you assign value to the [SegmentPath](#) property, annotation will be rotated from its position based on the connector segment direction.

The following code example shows how the connector annotation rotated in its path direction.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Connectors="@connectors" />
@code
{
// Defines diagram's connector collection
DiagramObjectCollection<Connector> connectors;
protected override void OnInitialized()
{
connectors = new DiagramObjectCollection<Connector>();
Connector connector = new Connector()
{
SourcePoint = new DiagramPoint() { X = 300, Y = 40 },
TargetPoint = new DiagramPoint() { X = 400, Y = 160 },
Type = ConnectorSegmentType.Orthogonal,

```

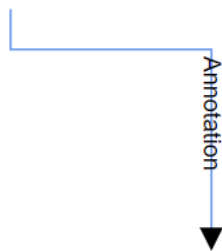
```

Style = new TextStyle() { StrokeColor = "#6495ED" },
Annotations = new DiagramObjectCollection<PathAnnotation>()
{
    new PathAnnotation
    {
        Content = "Annotation",
        // Set the segment angle for the connector's annotation
        SegmentAngle = true,
        Offset = 0.7
    },
};
connectors.Add(connector);
}
}

```

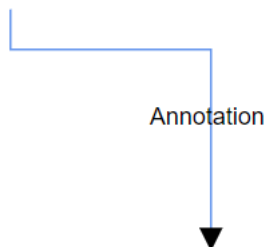
| SegmentAngle | Output |

|---|---|



| True |

|



| False |

|

By default, the SegmentAngle will be disabled.

The `HorizontalAlignment`, `VerticalAlignment` and `Margin` properties were explained in the [NodeAnnotation](#).

*See also*

- [How to add annotation for Node](#)

- [How to add or remove annotation constraints](#)
- [How to customize the annotation](#)

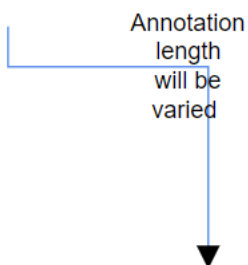
### Appearance in Blazor Diagram Component

#### *Size for an annotation*

Diagram allows you set size for annotations by using the Height and Width properties. The default value of the [Width](#) and [Height](#) properties are 0, and it takes the node or connector size as default. The following code example shows how the annotation size is customized.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Connectors="@connectors" />
@code
{
    // Defines diagram's connector collection
    DiagramObjectCollection<Connector> connectors;
    protected override void OnInitialized()
    {
        connectors = new DiagramObjectCollection<Connector>();
        Connector connector = new Connector()
        {
            SourcePoint = new DiagramPoint() { X = 300, Y = 40 },
            TargetPoint = new DiagramPoint() { X = 400, Y = 160 },
            Type = ConnectorSegmentType.Orthogonal,
            Style = new TextStyle() { StrokeColor = "#6495ED" },
            Annotations = new DiagramObjectCollection<PathAnnotation>()
            {
                new PathAnnotation
                {
                    Content = "Annotation length will be varied",
                    // Sets the size of the annotation
                    Width = 50,
                    Height = 50
                },
            },
        };
        connectors.Add(connector);
    }
}
```



### Hyperlink

Diagram provides support to add a [Hyperlink](#) to the node's or connector's annotation. It can also be customized.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    // Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node",
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            // Sets the annotation for the Node
            Annotations = new DiagramObjectCollection<ShapeAnnotation>()
            {
                // Add text as hyperlink.
                new ShapeAnnotation
                {
                    Hyperlink = new HyperlinkSettings
                    {
                        Url = "https://www.syncfusion.com"
                    }
                },
            },
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
        };
        nodes.Add(node);
    }
}
```



### Hyperlink with content

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
```

```

<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    // Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node",
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            // Sets the annotation for the Node
            Annotations = new DiagramObjectCollection<ShapeAnnotation>()
            {
                // Add text as hyperlink.
                new ShapeAnnotation
                {
                    Hyperlink = new HyperlinkSettings
                    {
                        Content = "Syncfusion",
                        Url = "https://www.syncfusion.com"
                    }
                },
                Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
            };
            nodes.Add(node);
        }
    }
}

```



### Text wrapping

The [TextWrapping](#) property of the annotation defines how the text should be wrapped. When text overflows node boundaries, you can control it by using the [TextWrapping](#). So, it is wrapped into multiple lines. The following code explains how to wrap a text in a node.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{

```



```
// Defines diagram's node collection
DiagramObjectCollection<Node> nodes;
protected override void OnInitialized()
{
    nodes = new DiagramObjectCollection<Node>();
    Node node = new Node()
    {
        ID = "node",
        Width = 100,
        Height = 100,
        OffsetX = 100,
        OffsetY = 100,
        // Sets the annotation for the node
        Annotations = new DiagramObjectCollection<ShapeAnnotation>()
        {
            new ShapeAnnotation
            {
                Content = "Annotation Text Wrapping",
                Style = new TextStyle()
                {
                    // Sets the text wrapping of the annotation as Wrap
                    TextWrapping = TextWrap.Wrap
                }
            },
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
        };
        nodes.Add(node);
    }
}
```

| TextWrapping | Description | Image |

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

Label Text Wrapping

| No Wrap | Text will not be wrapped. |

| Wrap | Text-wrapping occurs, when the text overflows beyond the available node width. |



|

| WrapWithOverflow (Default) | Text-wrapping occurs, when the text overflows beyond the available node width. However, the text may overflow beyond the node width in the case of a very long word. |



|

#### Text overflow

The [TextOverflow](#) property specifies how the overflowed content that is not displayed should be signaled to the user. The TextOverflow can have the following values.

- **Wrap:** Wraps the text to next line, when it exceeds its bounds.
- **Ellipsis:** It truncates the overflowed text and render an ellipsis ("...") to represent the clipped text.
- **Clip:** The text is clipped and the overflow text will not be shown.

The following code sample shows how the different types of overflow property working for the different types of text wrapping.

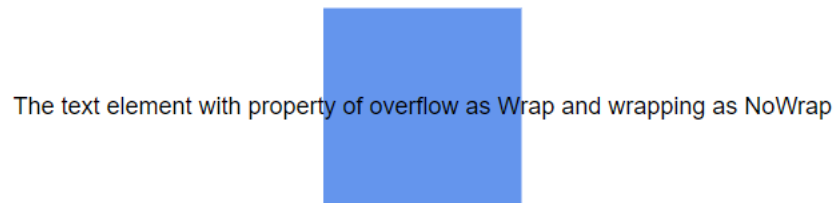
#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    // Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node",
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
```

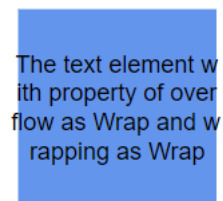
```
// Sets the style for the text to be displayed
Annotations = new DiagramObjectCollection<ShapeAnnotation>()
{
    new ShapeAnnotation
    {
        Content = "The text element with property of overflow as Wrap and wrapping
as NoWrap",
        Style = new TextStyle()
        {
            // Sets the text overflow of the annotation as Wrap
            TextOverflow = TextOverflow.Wrap,
            TextWrapping = TextWrap.NoWrap
        },
    },
    new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
};
nodes.Add(node);
}
```

| TextOverflow | Wrapping | Image |

\_\_\_\_\_



Wrap	No Wrap
	



| Wrap| Wrap |

The text element  
with property of  
overflow as Wrap  
and wrapping as  
WrapWithOverflow

| Wrap | WrapWithOverflow |

The text eleme...

| Ellipsis | No Wrap |

The text element w  
ith property of over  
flow as Ellipsis and  
wrapping as Wrap

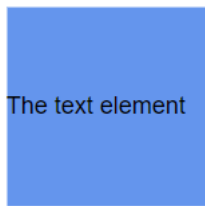
| Ellipsis | Wrap |

The text element  
with property of  
overflow as  
Ellipsis and  
wrapping as wrap  
withOverFlow

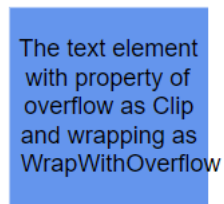
| Ellipsis | WrapWithOverflow |

The text element w  
ith property of over  
flow as Clip and w  
rapping as wrap

| Clip | No Wrap |



| Clip | Wrap |



| Clip | WrapWithOverflow |

---

**Note :** All the customization over the overflow is also applicable to connector's annotation.

---

#### *Customize the appearance of an annotation*

You can change the font style of the annotations with the font specific properties (FontSize, FontFamily, Color). The following code explains how to customize the appearance of the annotation.

- The annotation's [Bold](#), [Italic](#), and [TextDecoration](#) properties are used to style the annotation's text.
- The annotation's [Fill](#), [StrokeColor](#), and [StrokeWidth](#) properties are used to define the background color and border color of the annotation and the [Opacity](#) property is used to define the transparency of the annotations.
- The [Visibility](#) property of the annotation enables or disables the visibility of annotation.

The Fill, Border, and Opacity appearances of the text can also be customized with appearance specific properties of annotation. The following code explains how to customize the appearance of the annotation.

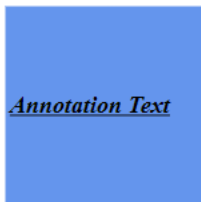
#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    // Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node",
            Width = 100,
```

```

Height = 100,
OffsetX = 100,
OffsetY = 100,
// Sets the annotation for the node
Annotations = new DiagramObjectCollection<ShapeAnnotation>()
{
    new ShapeAnnotation
    {
        Content = "Annotation Text",
        Style = new TextStyle()
        {
            // Sets the style for the annotation
            Color="black",
            Bold = true,
            Italic = true,
            TextDecoration = TextDecoration.Underline,
            FontSize = 12,
            FontFamily = "TimesNewRoman"
        }
    },
    Style = new ShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
};
nodes.Add(node);
}
}

```



#### Update the annotation style at runtime

You can change the font style of the annotations with the font specific properties ([FontSize](#), [FontFamily](#), and [Color](#)). The following code explains how to update the font style of the annotation.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="Update Style" @onclick="@UpdateStyle" />
<SfDiagramComponent @ref="@Diagram" Height="600px" Nodes="@nodes" />
@code
{
    // Reference of the diagram
    SfDiagramComponent Diagram;
    // Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {

```

```
nodes = new DiagramObjectCollection<Node>();
Node node = new Node()
{
    ID = "node1",
    Width = 100,
    Height = 100,
    OffsetX = 100,
    OffsetY = 100,
    // Sets the annotation for the node
    Annotations = new DiagramObjectCollection<ShapeAnnotation>()
    {
        new ShapeAnnotation
        {
            Content = "Annotation Text",
            Style = new TextStyle()
            {
                Color = "black",
                Bold = true,
                Italic = true,
                TextDecoration = TextDecoration.Underline,
                FontSize = 12,
                FontFamily = "TimesNewRoman"
            }
        }
    },
    Style = new ShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
};
nodes.Add(node);
}
public void UpdateStyle()
{
    // Change the style of the annotation
    Diagram.BeginUpdate();
    Diagram.Nodes[0].Annotations[0].Style.Bold = false;
    Diagram.Nodes[0].Annotations[0].Style.TextDecoration = TextDecoration.None;
    Diagram.Nodes[0].Annotations[0].Style.Color = "Red";
    Diagram.EndUpdate();
}
```

### *Change the editing mode*

Diagram provides support to edit an annotation at runtime, either programmatically or interactively. By default, the annotation is in view mode. But it can be brought to edit mode in two ways.

- You can edit the annotation programmatically by using the [StartTextEdit](#) method.
- Also, you can edit the annotation interactively.
- By double-clicking the annotation.
- By selecting the item and pressing the F2 key.

Double-clicking any annotation will enable the editing and the node enables first annotation editing. When the focus of editor is lost, the annotation for the node is updated.

### *Set Annotation to read only*

Diagram allows to create read-only annotations. You have to set the read-only property of annotation to enable or disable the [ReadOnly](#) constraints. The following code explains how to enable read-only mode.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    // Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node",
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            Annotations = new DiagramObjectCollection<ShapeAnnotation>()
            {
                new ShapeAnnotation
                {
                    Content = "Annotation Text",
                    // Sets the constraints as Read only
                    Constraints = AnnotationConstraints.ReadOnly
                }
            },
            Style = new ShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
        };
        nodes.Add(node);
    }
}
```

### *Create Multiple Annotations*

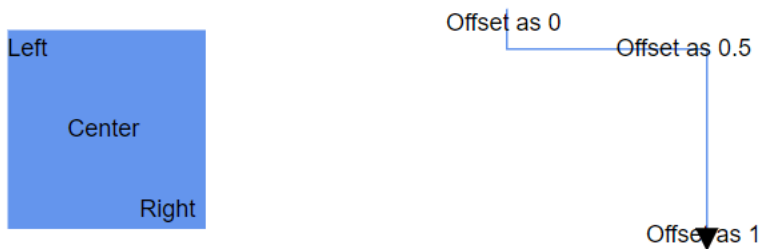
You can add any number of annotations to a node or connector. The following code example shows how to add multiple annotations to a node.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" Connectors="@connectors" />
@code
{
    // Defines diagram's node collection
    DiagramObjectCollection<Node> nodes;
    // Defines diagram's connector collection
    DiagramObjectCollection<Connector> connectors;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
```



```
{
ID = "node",
Width = 100,
Height = 100,
OffsetX = 100,
OffsetY = 100,
Style = new ShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
// Sets the multiple annotation for the node
Annotations = new DiagramObjectCollection<ShapeAnnotation>()
{
new ShapeAnnotation
{
Content = "Left",
Offset = new DiagramPoint() { X = .12, Y = .1}
},
new ShapeAnnotation
{
Content = "Center",
Offset = new DiagramPoint() { X = .5, Y = .5}
},
new ShapeAnnotation
{
Content = "Right",
Offset = new DiagramPoint() { X = .82, Y = .9}
}
},
};
nodes.Add(node);
connectors = new DiagramObjectCollection<Connector>();
Connector connector = new Connector()
{
SourcePoint = new DiagramPoint() { X = 300, Y = 40 },
TargetPoint = new DiagramPoint() { X = 400, Y = 160 },
Type = ConnectorSegmentType.Orthogonal,
Style = new TextStyle() { StrokeColor = "#6495ED" },
Annotations = new DiagramObjectCollection<PathAnnotation>()
{
new PathAnnotation
{
Content = "Offset as 0",
Offset = 0
},
new PathAnnotation
{
Content = "Offset as 0.5",
Offset = 0.5
},
new PathAnnotation
{
Content = "Offset as 1",
Offset = 1
}
},
};
connectors.Add(connector);
}
```




---

\* Type of the annotation's property of the node or connector was ObservableCollection.

\* Default value of the annotation will be null.

\* All the same customization can be applicable for the annotations.

\* Text Editing can be stated only the first annotation of the annotation collection when you double click the node or connector.

---

#### Constraints

[AnnotationConstraints](#) are used to enable or disable certain behaviors of the annotation. Constraints are provided as flagged enumerations, so that multiple behaviors can be enabled or disabled with bitwise operators.

AnnotationConstraints may have multiple behaviors as follows:

| Constraints | Usages |

|---|---|

| ReadOnly | Enables or disables whether the annotation to be read only or not. |

| None | Disables all behaviors of Annotation. |

| InheritReadOnly | Enables or disables to inherit the ReadOnly option from the parent object. |

---

The default value is [InheritReadOnly](#) for constraints property of the annotation.

---

Refer to [Constraints](#) to learn about how to enable or disable the annotation constraints.

#### See also

- [How to add or remove annotation constraints](#)
- [How to add annotation for Node](#)
- [How to add annotation for Connector](#)

#### Events in Blazor Diagram Component

##### Text Change

- While editing the node's or connector's annotation, [TextChanged](#) event can be used to do the customization.
- When the node's or connector's annotation is changed in the diagram, this event is getting triggered.

The [TextChangeEventArgs](#) notifies when the annotation of an element undergoes editing.

The following code example shows how to register and get the notification from the TextChanged event.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" TextChanged="OnTextChanged"
Nodes="@nodes" />
@code
{
    // Defines diagram's nodes collection
    DiagramObjectCollection<Node> nodes;
    // Triggered this event when complete the editing for Annotation and update
    the old text and new text values.
    private void OnTextChanged(TextChangeEventArgs args)
    {
        Console.WriteLine("Oldvalue", args.OldValue);
        Console.WriteLine("NewValue", args.NewValue);
    }
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            Width = 100,
            Height = 100,
            OffsetX = 100,
            OffsetY = 100,
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
        };
        node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
        {
            new ShapeAnnotation { Content = "Annotation" }
        };
        nodes.Add(node);
    }
}
```

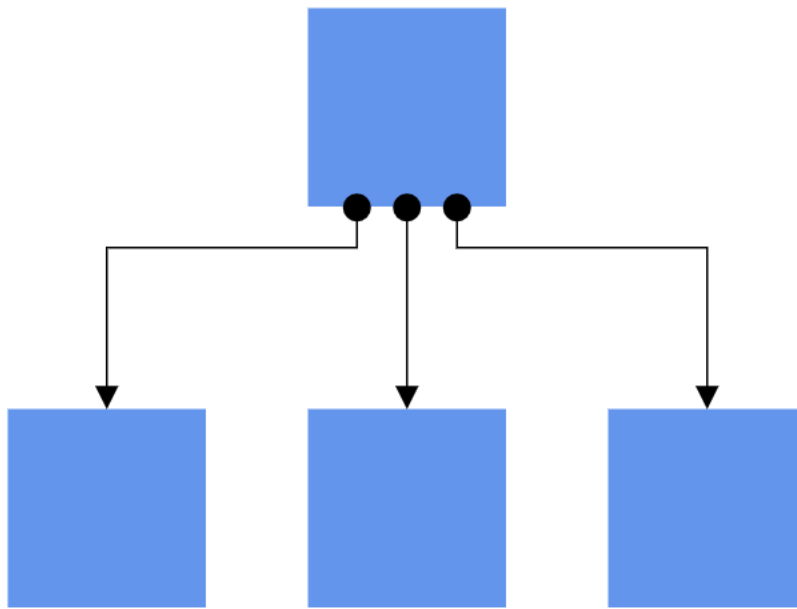
*See also*

- [How to add or remove annotation constraints](#)
- [How to customize the annotation](#)
- [How to add annotation for Node](#)
- [How to add annotation for Connector](#)

### Ports

#### Ports in Blazor Diagram Component

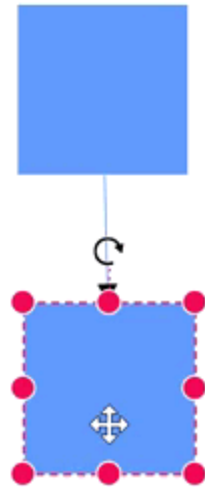
Port is a special connection point in a Node that you can glue the connectors. When you glue a connector to a node or port, they stay connected, even if one of the node is moved.



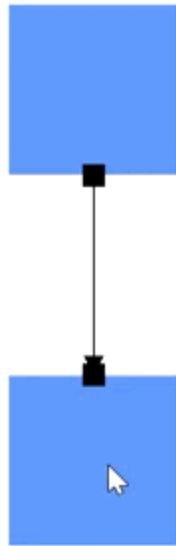
### Connections

There are two main types of connections, node to node and port to port. The difference between these two connections is whether or not a connector remains glued to a specific connection point when you move the attached node or connector.

A node to node connection is one where the connector will move around the node as you move the node. Diagram will always ensure the connector is the shortest, most direct line possible. You can create a node to node connection by selecting the entire node (rather than the port) and connect it to another shape (rather than to a port).



Ports act as the connection points of the node and allows creating connections with only those specific points as shown in the following image.



### Create Ports

To add a connection port, define the port object and add it to node's ports collection. The [Offset](#) property of the port accepts an object of fractions and used to determine the position of ports. The following code explains how to add ports when initializing the node.

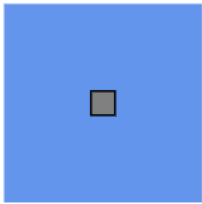
### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes collection.
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
```

```

Height = 100,
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
// Initialize port collection
Ports = new DiagramObjectCollection<PointPort>()
{
    new PointPort()
    {
        Style = new ShapeStyle(){ Fill = "gray" },
        // Sets the position for the port
        Offset = new DiagramPoint() { X = 0.5, Y = 0.5 },
        Visibility = PortVisibility.Visible
    }
};
nodes.Add(node);
}
}

```



#### Add Ports at runtime

You can add Ports at runtime to the nodes collection in the Diagram component by using the **Add** method.

The following code explains how to add ports to node at runtime by using **Add** method. The port's **ID** property is used to define the unique ID for the port and it is further used to find the port at runtime.

If **ID** is not set, then default **ID** is automatically set.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="AddPorts" @onclick="@AddPorts" />
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        //A node is created and stored in nodes array
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            //Position of the node
            OffsetX = 250,
            OffsetY = 250,
            //Size of the node
            Width = 100,

```

```

Height = 100,
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
};
nodes.Add(node);
}
public void AddPorts()
{
    PointPort port = new PointPort()
    {
        ID = "port",
        Style = new ShapeStyle(){ Fill = "gray" },
        Offset = new DiagramPoint() { X = 0, Y = 0.5 },
        Visibility = PortVisibility.Visible
    };
    // Initialize port collection
    nodes[0].Ports.Add(port);
}
}

```

Also, the Port can be added at runtime by using the `AddAsync` method. The `await` operator suspends evaluation of the enclosing async method until the asynchronous operation represented by its operand completes.

The following code explains how to add ports to node at runtime by using `AddAsync` method.

#### CSHARP

```

//Method to add Port at runtime
public async void AddPorts()
{
    PointPort port = new PointPort()
    {
        Style = new ShapeStyle() { Fill = "gray" },
        Offset = new DiagramPoint() { X = 0, Y = 0.5 },
        Visibility = PortVisibility.Visible
    };
    await ((nodes[0].Ports) as
    DiagramObjectCollection<PointPort>).AddAsync(port);
}

```



#### Add Multiple Ports at runtime

Add Multiple ports at runtime by using the method `Add` in the port collection. The following code explains how to add two or more ports to node at runtime.



The port's [ID](#) property is used to define the unique ID for the port and it is further used to find the port at runtime. If **ID** is not set, then default **ID** is automatically set.

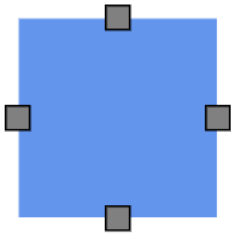
### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<input type="button" value="AddPorts" @onclick="@AddPorts" />
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        //A node is created and stored in nodes array
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            //Position of the node
            OffsetX = 250,
            OffsetY = 250,
            //Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle()
            {
                Fill = "#6495ED",
                StrokeColor = "white"
            },
        };
        nodes.Add(node);
    }
    public void AddPorts()
    {
        PointPort port1 = new PointPort()
        {
            ID = "port1",
            Offset = new DiagramPoint() { X = 0, Y = 0.5 },
            Visibility = PortVisibility.Visible,
            Style = new ShapeStyle() { Fill = "gray" }
        };
        PointPort port2 = new PointPort()
        {
            ID = "port2",
            Offset = new DiagramPoint() { X = 1, Y = 0.5 },
            Visibility = PortVisibility.Visible,
            Style = new ShapeStyle() { Fill = "gray" }
        };
        PointPort port3 = new PointPort()
        {
            ID = "port3",
            Offset = new DiagramPoint() { X = 0.5, Y = 0 },
            Visibility = PortVisibility.Visible,
            Style = new ShapeStyle() { Fill = "gray" }
        };
        PointPort port4 = new PointPort()
        {
            ID = "port4", Offset = new DiagramPoint() { X = 0.5, Y = 1 },
```

```

Visibility = PortVisibility.Visible,
Style = new ShapeStyle() { Fill = "gray" }
};
// Add multiple ports in the port collection
nodes[0].Ports.Add(port1);
nodes[0].Ports.Add(port2);
nodes[0].Ports.Add(port3);
nodes[0].Ports.Add(port4);
}
}

```



#### *Remove ports at runtime*

A collection of ports can be removed from the node by using the native `RemoveAt` method. Refer to the following example that shows how to remove ports at runtime.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="RemovePorts" @onclick="@RemovePorts" />
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        // A node is created and stored in nodes array.
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
            // Initialize port collection
            Ports = new DiagramObjectCollection<PointPort>()
            {
                new PointPort()
                {
                    ID = "port1",
                    Offset = new DiagramPoint() { X = 0, Y = 0.5 },

```

```

Visibility = PortVisibility.Visible,
//Set the style for the port
Style= new ShapeStyle()
{
    Fill = "red",
    StrokeColor = "black",
    StrokeWidth = 2
},
Width = 12,
Height = 12,
// Sets the shape of the port as Circle
Shape = PortShapes.Circle
}
},
};
nodes.Add(node);
}
public void RemovePorts()
{
    (nodes[0].Ports as DiagramObjectCollection<PointPort>).RemoveAt(0);
}
}

```

#### Update Ports at runtime

You can change any port properties at runtime.

The following code example explains how to change the port properties at runtime.

#### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="Update Port" @onclick="@UpdatePort" />
<SfDiagramComponent @ref="diagram" Height="600px" Nodes="@nodes" />
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        // Initialize port collection
        DiagramObjectCollection<PointPort> ports = new
        DiagramObjectCollection<PointPort>();
        ports.Add(new PointPort()
        {
            ID = "port",
            Offset = new DiagramPoint()
            {
                X = 0,
                Y = 0.5
            },
            Visibility = PortVisibility.Visible
        });
        // A node is created and stored in nodes array
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {

```

```
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new ShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },
Ports = ports
};
nodes.Add(node);
}
public async void UpdatePort()
{
//Update ports at run time
diagram.BeginUpdate();
nodes[0].Ports[0].Offset.X = 1;
nodes[0].Ports[0].Offset.Y = 1;
await diagram.EndUpdate();
}
}
```

See also

- [How to create a node](#)
- [How to customize the ports](#)
- [How to interact the ports](#)
- [How to set the position of the port](#)

### How to position node's port

Diagram allows you to customize the position and appearance of the port efficiently. Port can be aligned relative to the node boundaries. It has Margin, Offset, Horizontal, and Vertical alignment settings. It is quite tricky when all four alignments are used together but gives more control over alignments properties of the [PointPort](#) class. Ports of a node can be positioned using the following properties of [PointPort](#).

- Offset
- HorizontalAlignment
- VerticalAlignment
- Margin

### Offset

The [Offset](#) property is used to align the Ports based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes"/>
@code
{
DiagramObjectCollection<Node> nodes;
protected override void OnInitialized()
```

```

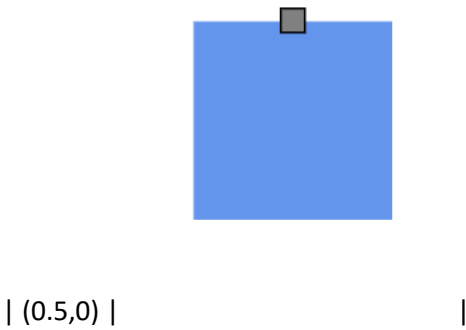
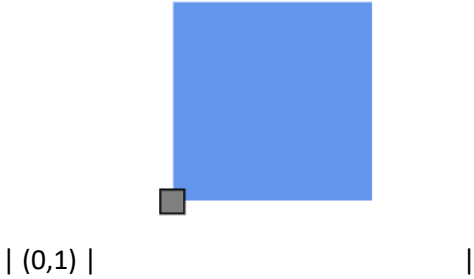
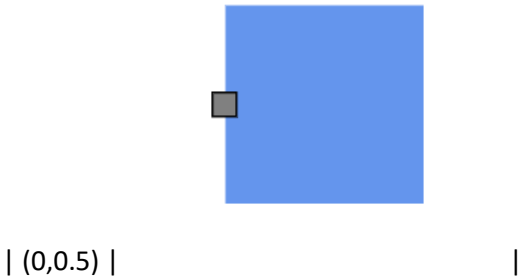
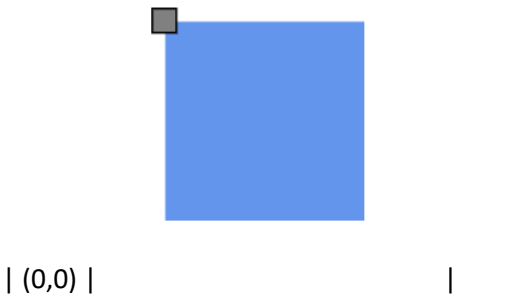
{
nodes = new DiagramObjectCollection<Node>();
// A node is created and stored in nodes collection.
Node node = new Node()
{
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
// Initialize port collection
Ports = new DiagramObjectCollection<PointPort>()
{
new PointPort()
{
ID = "port1",
// Sets the offset for the port
Offset = new DiagramPoint() { X = 0, Y = 0.5 },
Visibility = PortVisibility.Visible,
//Set the style for the port
Style= new ShapeStyle() { Fill = "gray", StrokeColor = "black" },
Width = 12,
Height = 12,
// Sets the shape of the port as Square
Shape = PortShapes.Square
}
}
};
nodes.Add(node);
}
}

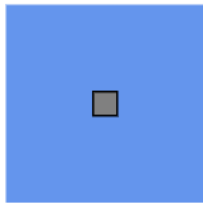
```



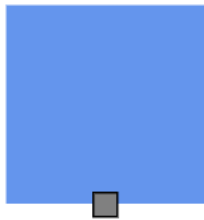
The following table shows the relationship between the shape port position and path port offset (fraction values).

Offset values	Output
--- ---	



 $| (0.5, 0.5) |$ 

|

 $| (0.5, 1) |$ 

|

 $| (1, 0) |$ 

|

 $| (1, 0.5) |$ 

|



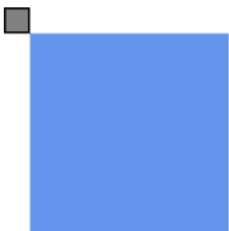
| (1,1) |

Horizontal and Vertical alignment

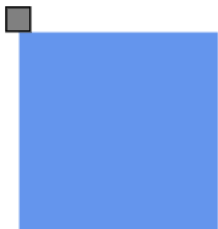
The [HorizontalAlignment](#) property of the port is used to set how the port is horizontally aligned at the port position determined from the fraction values. The [VerticalAlignment](#) property is used to set how the port is vertically aligned at the port position.

The following table shows all the possible alignments visually with offset (0, 0).

Horizontal Alignment	Vertical Alignment	Output with Offset(0,0)
-----	-----	-----



| Left | Top |



| Center | Top |

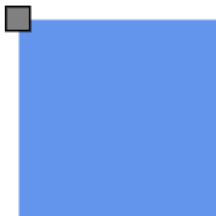




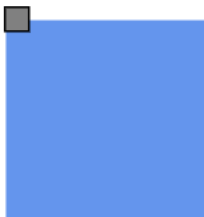
| Right | Top |



| Left | Center |



| Center | Center |



| Right | Center |



| Left | Bottom |



| Center | Bottom |



| Right | Bottom |

The following code explains how to align ports.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes"/>
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes array.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
        };
        // Initialize port collection
```

```

Ports = new DiagramObjectCollection<PointPort>()
{
    new PointPort()
    {
        ID = "port1",
        Offset = new DiagramPoint() { X = 0, Y = 0 },
        Visibility = PortVisibility.Visible,
        //Set the style for the port
        Style = new ShapeStyle() { Fill="gray", StrokeColor="black"},
        Width = 12,
        Height = 12,
        // Sets the shape of the port as Square
        Shape = PortShapes.Square,
        HorizontalAlignment = HorizontalAlignment.Center,
        VerticalAlignment = VerticalAlignment.Center
    }
};
nodes.Add(node);
}

```



The value of the `HorizontalAlignment` is `Center` by default. The value of the `VerticalAlignment` is `Center` by default. Alignment positioned based on the offset value.

### Margin

[Margin](#) is an absolute value used to add some blank space to any one of its four sides. The ports can be displaced with the margin property. The following code example explains how to align a port based on its Offset, HorizontalAlignment, VerticalAlignment, and Margin values.

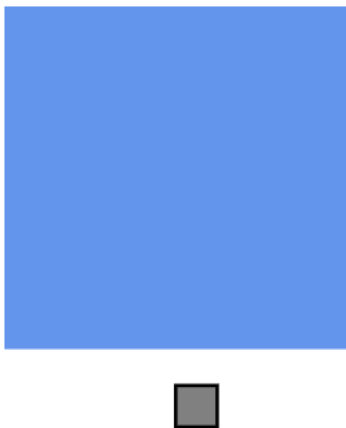
### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes"/>
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes array.
        Node node = new Node()
        {

```

```
// Position of the node
OffsetX = 250,
OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
// Initialize port collection
Ports = new DiagramObjectCollection<PointPort>()
{
    new PointPort()
    {
        ID = "port1",
        Offset = new DiagramPoint() { X = 0.5, Y = 1 },
        Visibility = PortVisibility.Visible,
        //Set the style for the port
        Style= new ShapeStyle() { Fill = "gray", StrokeColor = "black" },
        Width = 12,
        Height = 12,
        // Sets the shape of the port as Square
        Shape = PortShapes.Square,
        HorizontalAlignment = HorizontalAlignment.Left,
        VerticalAlignment = VerticalAlignment.Top,
        Margin = new Margin() { Top = 10 }
    }
};
nodes.Add(node);
}
```



*See also*

- [How to create a node](#)
- [How to customize the ports](#)

- [How to interact the ports](#)

## Port appearance and positioning

### Appearance

- The shape of a port can be changed by using the [Shape](#) property. To explore the different types of port shapes, refer to Port Shapes. If you need to render a custom shape, then you can set shape to path and define path using the path data property.
- The appearance of the ports can be customized by using the [StrokeColor](#), [StrokeWidth](#), and [Fill](#) properties.
- Customize the port size by using the [Width](#) and [Height](#) properties of port.
- The ports [Visibility](#) property allows you to define when the port should be visible.

The following code explains how to change the appearance of the port.

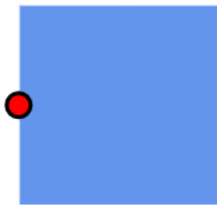
### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes array.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
            // Initialize port collection
            Ports = new DiagramObjectCollection<PointPort>()
            {
                new PointPort()
                {
                    ID = "port1",
                    Offset = new DiagramPoint() { X = 0, Y = 0.5 },
                    Visibility = PortVisibility.Visible,
                    //Set the style for the port
                    Style = new ShapeStyle()
                    {
                        Fill = "red",
                        StrokeColor = "black",
                        StrokeWidth = 2
                    },
                    Width = 12,
                    Height = 12,
                    // Sets the shape of the port as Circle
                    Shape = PortShapes.Circle
                }
            }
        }
    }
}
```

```

}
},
};
nodes.Add(node);
}
}

```



### Visibility

The [Visibility](#) of the ports depends upon the properties of Connect, Hidden, Hover, and Visible. By default, [PortVisibility](#) is set to Hidden.

| Property | Definition |

|---|---|

| Hover | Port is visible when mousehover the DiagramElement. |

| Hidden | Port is not visible for the DiagramElement. |

| Connect | Specifies to visible the port when mousehover the DiagramElement and enable the PortConstraints as InConnect and OutConnect. |

| Visible | Port is always visible for the DiagramElement. |

### Types of port shapes

We have provided some basic built-in [PortShapes](#) for the port. Please find the shapes as follows.

- Circle
- Custom
- Square
- X

### Custom shape

We have provided custom shape support for port. You can able to add the custom path data instead of build-in shapes. Please find the code example that explains how to change the custom shape for port.

### ASPX-CS

```

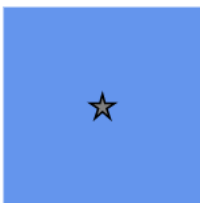
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;

```

```

protected override void OnInitialized()
{
    nodes = new DiagramObjectCollection<Node>();
    // A node is created and stored in nodes array.
    Node node = new Node()
    {
        // Position of the node
        OffsetX = 250,
        OffsetY = 250,
        // Size of the node
        Width = 100,
        Height = 100,
        Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
        // Initialize port collection
        Ports = new DiagramObjectCollection<PointPort>()
        {
            new PointPort()
            {
                ID = "port1",
                Offset = new DiagramPoint() { X = 0.5, Y = 0.5 },
                Visibility = PortVisibility.Visible,
                //Set the style for the port
                Style = new ShapeStyle() { Fill = "gray", StrokeColor = "black" },
                Width = 12,
                Height = 12,
                // Sets the shape of the port as Custom
                Shape = PortShapes.Custom,
                // Sets the PathData for port
                PathData =
                "M540.3643,137.9336L546.7973,159.7016L570.3633,159.7296L550.7723,171.9366L558.9053,194.9966L540.3643,179.4996L521.8223,194.9966L529.9553,171.9366L510.3633,159.7296L533.9313,159.7016L540.3643,137.9336z"
            }
        },
    };
    nodes.Add(node);
}

```



### Constraints

The constraints property allows you to enable or disable certain behaviors of ports. For more information about port constraints, refer to [Port Constraints](#). You can verify the [Constraints](#) to learn how to enable or disable the port constraints.

The PortConstraints may have multiple behaviors listed as follows:

- | Constraints | Usages |
- |---|---|
- | None | Disables all behaviors of Port. |
- | Draw | Enables or disables to draw a connector. |
- | InConnect | Enables or disables connecting to the incoming Connector. |
- | OutConnect | Enables or disables connecting the outgoing Connector. |

#### Custom properties

The [AdditionalInfo](#) property of the port allows you to maintain additional information to the port.

#### See also

- [How to create a node](#)
- [How to customize the ports](#)
- [How to set the position of the port](#)
- [How to interact the ports](#)

#### Interaction in Blazor Diagram Component

The port can be used to create connector by enable the **Draw** in the [PortConstraints](#).

#### Draw

Diagram provides the support to draw the connector in the port.

The following code explains how to draw the connector by using the port constraints.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes array.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
            // Initialize port collection
            Ports = new DiagramObjectCollection<PointPort>()
            {
                new PointPort()
                {
                    ID = "port1",
```



```
Offset = new DiagramPoint() { X = 0.5, Y = 0.5 },
Visibility = PortVisibility.Visible,
//Set the style for the port
Style = new ShapeStyle()
{
    Fill = "gray",
    StrokeColor = "black"
},
Width = 12,
Height = 12,
// Sets the shape of the port as Square
Shape = PortShapes.Square,
// Enable draw operation for Port
Constraints = PortConstraints.Default | PortConstraints.Draw
}
},
};
nodes.Add(node);
}
```



*See also*

- [How to create a node](#)
- [How to customize the ports](#)
- [How to set the position of the port](#)

## Constraints in Blazor Diagram Component

Constraints are used to enable or disable certain behaviors of the diagram, nodes, and connectors. Constraints are provided as flagged enumerations, so that multiple behaviors can be enabled or disabled using the Bitwise operators (&, |, ~, <<, etc.).

To know more about Bitwise operators, refer to the [Bitwise Operations](#).

### Diagram constraints

[DiagramConstraints](#) allow you to enable or disable the following behaviors.

- PageEditable
- Bridging
- Zoom
- UndoRedo
- UserInteraction

Constraints	Description
-----	-----
None	Disable all diagram functionalities
Bridging	Enables or Disable Bridging support for connector in diagram
Undo/redo	Enables or Disable the Undo/Redo support for the diagram
UserInteraction	Enables or Disable user interaction support for the diagram
ApiUpdate	Enables or Disable update API support for the diagram
PageEditable	Enables or Disable Page Editable support for the diagram
Zoom	Enables or Disable Zoom support for the diagram
PanX	Enables or Disable Paning X coordinate support for the diagram
PanY	Enables or Disable Paning Y coordinate support for the diagram
Pan	Enables or Disable panning both X and Y coordinates support for the diagram
ZoomTextEdit	Enables or Disables zooming the text box while editing the text
Default	Enables or Disable all constraints in diagram

The following example shows how to disable PageEditable constraint from default diagram constraints.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
/* Initialize the diagram with constraints */
<SfDiagramComponent Height="600px" Nodes="@nodes"
Constraints="@DiagramConstraints" />
@code
{
//sets the Diagram constraints...
DiagramConstraints DiagramConstraints = DiagramConstraints.Default &
~DiagramConstraints.PageEditable;
//Initialize the Nodes Collection.
DiagramObjectCollection<Node> nodes;
```

```
protected override void OnInitialized()
{
    nodes = new DiagramObjectCollection<Node>();
    Node node = new Node()
    {
        ID = "node1",
        Height = 100,
        Width = 100,
        OffsetX = 100,
        OffsetY = 100,
    };
    nodes.Add(node);
}
```

The following example shows how to add Bridging constraint to the default constraints of diagram.

#### **C#**

```
DiagramConstraints DiagramConstraints = DiagramConstraints.Default |
DiagramConstraints.Bridging;
```

The diagram constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#) in the diagram.

#### **C#**

```
//Removing multiple constraints from default
DiagramConstraints DiagramConstraints = DiagramConstraints.Default &
~(DiagramConstraints.PageEditable|DiagramConstraints.Zoom);
```

For more information about diagram constraints, refer to the [Diagram constraints](#).

By default, the following constraints are enabled in the diagram,

- \* ApiUpdate
- \* PanX
- \* PanY
- \* Pan
- \* ZoomTextEdit
- \* Default
- \* None

#### **Node constraints**

The [Constraints](#) property of the Node, allows you to enable or disable the following behaviors.

- Select
- Drag
- Resize
- Rotate

- Delete
- InConnect
- OutConnect

| Constraints | Description |

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

|None|Disable all node Constraints|

|Select|Enables or Disables node to be selected|

|Drag|Enables or Disables node to be Dragged|

|Rotate|Enables or Disables node to be rotating|

|Shadow|Enables or disables node to display shadow|

|PointerEvents|Enables or disables node to provide pointer option|

|Delete|Enables or Disables node to be deleting|

|InConnect|Enables or disables node to provide in connect option|

|OutConnect|Enables or disables node to provide out connect option|

|AllowDrop|Enables node to provide allow to drop option|

|ResizeNorthEast|Enable or disable to Resizing NorthEast side of the node|

|ResizeEast|Enable or disable to Resizing East side of the node|

|ResizeSouthEast|Enable or disable to Resizing SouthEast side of the node|

|ResizeSouth|Enable or disable to Resizing South side of the node|

|ResizeSouthWest|Enable or disable to Resizing SouthWest side of the node|

|ResizeWest|Enable or disable to Resizing West side of the node|

|ResizeNorthWest|Enable or disable to Resizing NorthWest side of the node|

|ResizeNorth|Enable or disable to Resizing North side of the node|

|AspectRatio|Enables the Aspect ratio of the node|

|ReadOnly|Enables the ReadOnly support for annotation in node|

|HideThumbs|Enable to hide all resize thumbs for the node|

|Resize|Enables or Disables the expansion or compression of a node|

|Inherit|Enables the node to inherit the interaction option from the parent object|

|Default|Enables all default constraints for the node|

The following example shows how to disable rotate constraint from the default node constraints.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
@* Initialize the diagram with NodeCollection *@
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
```

```

{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node1",
            Height = 100,
            Width = 100,
            OffsetX = 100,
            OffsetY = 100,
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "White" },
            //sets the NodeConstraints constraints...
            Constraints = NodeConstraints.Default & ~NodeConstraints.Rotate
        };
        nodes.Add(node);
    }
}

```



The following example shows how to add Shadow constraint to the default constraints of node.

#### **C#**

```

NodeConstraints NodeConstraints = NodeConstraints.Default |
NodeConstraints.Shadow;

```

The node constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#).

#### **C#**

```

//Removing multiple constraints from default
NodeConstraints NodeConstraints = NodeConstraints.Default & ~
(NodeConstraints.Select | NodeConstraints.Drag);

```

For more information about node constraints, refer to the [NodeConstraints](#).

**Note :** By default, the following constraints are enabled for the node,

- \* Shadow
- \* PointerEvents
- \* AllowDrop
- \* ResizeNorthEast

---

- \* ResizeEast
- \* ResizeSouthEast
- \* ResizeSouth
- \* ResizeSouthWest
- \* ResizeWest
- \* ResizeNorthWest
- \* ResizeNorth
- \* AspectRatio
- \* ReadOnly
- \* HideThumbs
- \* Inherit
- \* Default

---

### Connector constraints

The [Constraints](#) property of the Connector, allow you to enable or disable the following behaviors of connectors.

- Select
- Drag
- DragSourceEnd
- DragTargetEnd
- Delete
- InheritBridging
- PointerEvents

Constraints	Description
None	Disable all connector Constraints
Select	Enables or Disables node to be selected
Delete	Enables or Disables node to be deleting
Drag	Enables or Disables node to be Dragged
DragSourceEnd	Enables connectors source end to be selected
DragTargetEnd	Enables connectors target end to be selected
DragSegmentThumb	Enables control point and end point of every segment in a connector for editing
Interaction	Enables or disables Interaction for the connector
AllowDrop	Enables allow drop support to the connector
Bridging	Enables bridging to the connector
InheritBridging	Enables to inherit bridging option from the parent object

|PointerEvents| Enables to set the pointer events|

|ConnectToNearByNode| Enables to connect to the nearest node|

|ConnectToNearByPort| Enables to connect to the nearest port|

|ConnectToNearByElement| Enables to connect to the nearest elements|

|ReadOnly| Enables or disables readonly for the connector|

|Default| Enables all constraints for the connector|

The following code shows how to disable select constraint from the default constraints of connector.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
@* Initialize the diagram with connectors *@
<SfDiagramComponent Height="600px" Connectors="@connectors" />
@code
{
    DiagramObjectCollection<Connector> connectors;
    protected override void OnInitialized()
    {
        connectors = new DiagramObjectCollection<Connector>();
        Connector connector = new Connector()
        {
            ID = "connector1",
            Type = ConnectorSegmentType.Straight,
            SourcePoint = new DiagramPoint() { X = 100, Y = 100 },
            TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
            //sets the ConnectorConstraints...
            Constraints = ConnectorConstraints.Default & ~ConnectorConstraints.Select
        };
        connectors.Add(connector);
    }
}
```

The following example shows how to add Bridging constraint to the default constraints of connector.

#### CSHARP

```
ConnectorConstraints ConnectorConstraints = ConnectorConstraints.Default |
ConnectorConstraints.Bridging;
```

The connector constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#).

#### CSHARP

```
//Removing multiple constraints from default
ConnectorConstraints ConnectorConstraints = ConnectorConstraints.Default & ~
(ConnectorConstraints.Select | ConnectorConstraints.Drag);
```

For more information about connector constraints, refer to the [ConnectorConstraints](#).

By default, the following constraints are enabled for the connector,

- \* DragSegmentThumb
- \* Interaction
- \* AllowDrop
- \* Bridging
- \* InheritBridging
- \* ConnectToNearByNode
- \* ConnectToNearByPort
- \* ConnectToNearByElement
- \* ReadOnly
- \* Default

### Port constraints

The [Constraints](#) property of the Port, allow you can enable or disable the following behaviors of port.

- InConnect
- OutConnect

| Constraints | Description |

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

| None | Disable all port Constraints |

| Draw | Enables to create the connection when mouse hover on the port |

| InConnect | Enables or disables to only connect the target end of connector |

| OutConnect | Enables or disables to only connect the source end of connector |

| Default | Enables all constraints for the port |

The following code shows how to disable creating connections with a port.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
/* Initialize the diagram with NodeCollection */
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        // Initialize the NodeCollection.
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node1",
            Height = 100,
            Width = 100,
            OffsetX = 100,
            OffsetY = 100,
```



```
};
node.Ports = new DiagramObjectCollection<PointPort>()
{
    new PointPort()
    {
        ID="port1",
        Offset=new DiagramPoint(){X=0,Y=0.5},
        Shape=PortShapes.Circle,
        Visibility=PortVisibility.Visible,
        // Sets the PortConstraints...
        Constraints=PortConstraints.None
    }
};
nodes.Add(node);
}
```

The following another code example shows to modify the port constraints to accept target connection alone.

#### **CSHARP**

```
//Enable to create target connection alone.
port.Constraints = PortConstraints.InConnect;
```

The port constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#).

#### **CSHARP**

```
//Enable to create target connection alone.
port.Constraints = PortConstraints.Default | PortConstraints.Draw;
```

For more information about port constraints, refer to the [PortConstraints](#).

By default, the following constraints are enabled for the port,

\* Draw

\* Default

#### **Annotation constraints**

The [Constraints](#) property of the Annotations, allow you can enable or disable read-only mode for the annotations by using the annotation constraints.

Constraints	Description
-----	-----
ReadOnly	Enables or Disables the ReadOnly Constraints
InheritReadOnly	Enables or Disables to inherit the ReadOnly option from the parent object
None	Disables all constraints for the annotation

The following code shows how to enable read-only mode for the annotations.

**ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
@* Initialize the diagram with NodeCollection *@
<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        //Initialize the NodeCollection.
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node1",
            Height = 100,
            Width = 100,
            OffsetX = 100,
            OffsetY = 100,
        };
        node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
        {
            new ShapeAnnotation()
            {
                ID="annotation1",
                Content="Annotation Text Wrapping",
                // Sets the Constraints for Annotation...
                Constraints=AnnotationConstraints.ReadOnly,
                Style= new TextStyle()
                {
                    Color="#000000",
                    Fill="Transparent",
                    FontFamily="TimesNewRoman",
                    FontSize=12,
                    Bold=true,
                    Italic=true
                },
            },
        };
        nodes.Add(node);
    }
}

```

For more details about annotation constraints, refer to the [AnnotationConstraints](#).

**Selector constraints**

Selector visually represents the selected elements with certain editable thumbs. The visibility of the thumbs can be controlled with selector constraints. The part of selector is categorized as follows:

- ResizeAll
- UserHandle
- Rotate

| Constraints | Description |

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

|None|Hides all the selector elements|

|ConnectorSourceThumb|Shows or hides the source thumb of the connector|

|ConnectorTargetThumb|Shows or hides the target thumb of the connector|

|ResizeSouthEast|Shows or hides the bottom right resize handle of the selector|

|ResizeSouthWest|Shows or hides the bottom left resize handle of the selector|

|ResizeNorthEast|Shows or hides the top right resize handle of the selector|

|ResizeNorthWest|Shows or hides the top left resize handle of the selector|

|ResizeEast|Shows or hides the middle right resize handle of the selector|

|ResizeWest|Shows or hides the middle left resize handle of the selector|

|ResizeSouth|Shows or hides the bottom center resize handle of the selector|

|ResizeNorth|Shows or hides the top center resize handle of the selector|

|Rotate|Shows or hides the rotate handle of the selector|

|UserHandle|Shows or hides the user handles of the selector|

|ResizeAll|Shows or hides all resize handles of the selector|

|All|Shows all handles of the selector|

The following code shows how to hide rotator.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes"
SelectionSettings="@selectionSettings"/>
@code
{
    DiagramObjectCollection<Node> nodes;
    public DiagramSelectionSettings selectionSettings = new
    DiagramSelectionSettings()
    {
        Constraints = SelectorConstraints.All & ~SelectorConstraints.Rotate
    };
    protected override void OnInitialized()
    {
        //Initialize the NodeCollection.
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "node1",
            Height = 100,
            Width = 100,
            OffsetX = 100,
            OffsetY = 100,
        };
        nodes.Add(node);
    }
}
```




---

Element should be in selected state, then only Rotator, UserHandle and Resizer thumbs will be visible.

---

The following another code example shows how to disable the userhandle functionality for the selected item.

#### **CSHARP**

```
//Enable userhandle constraint for the selected item.
selectedItems.Constraints = SelectorConstraints.All &~
SelectorConstraints.UserHandle;
```

For more information about selector constraints, refer to the [SelectorConstraints](#).

---

By default, the following constraints are enabled for the selected items,

- \* ConnectorSourceThumb
  - \* ConnectorTargetThumb
  - \* ResizeSouthEast
  - \* ResizeSouthWest
  - \* ResizeEast
  - \* ResizeWest
  - \* ResizeSouth
  - \* ResizeNorth
  - \* All
- 

#### **Snap constraints**

The [Constraints](#) property of the SnapConstraints control the visibility of gridlines and enable or disable snapping. Snap constraints allow to set the following behaviors.

- Show only horizontal or vertical gridlines.
- Show both horizontal and vertical gridlines.
- Snap to either horizontal or vertical gridlines.
- Snap to both horizontal and vertical gridlines.

The following list of snapping constraints are used to Enables or Disables certain features of snapping.

Constraints	Description
-----	-----

|None| Disable to snapping the nodes/connectors in diagram|

|ShowHorizontalLines| Displays only the horizontal gridlines in diagram|

|ShowVerticalLines| Displays only the Vertical gridlines in diagram|

|ShowLines| Display both Horizontal and Vertical gridlines|

|SnapToHorizontalLines| Enables the object to snap only with horizontal gridlines|

|SnapToVerticalLines| Enables the object to snap only with Vertical gridlines|

|SnapToLines| Enables the object to snap with both horizontal and Vertical gridlines|

|SnapToObject| Enables the object to snap with the other objects in the diagram|

|All| Shows gridlines and enables snapping|

The following code shows how to show only horizontal gridlines.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes">
  /* Initialize the snapsettings with constraints */
  <SnapSettings Constraints="SnapConstraints.ShowHorizontalLines" />
</SfDiagramComponent>

@code
{
  DiagramObjectCollection<Node> nodes;
  protected override void OnInitialized()
  {
    //Initialize the NodeCollection.
    nodes = new DiagramObjectCollection<Node>();
    Node node = new Node()
    {
      ID = "node1",
      Height = 100,
      Width = 100,
      OffsetX = 100,
      OffsetY = 100,
    };
    nodes.Add(node);
  }
}
```

The snap constraints are provided as flagged enumerations, so that multiple behaviors can be added or removed from the default constraints using the [Bitwise Operations](#).

### C#

```
snapconstraints = SnapConstraints.ShowHorizontalLines |
SnapConstraints.ShowVerticalLines | SnapConstraints.ShowLines;
```

For more information about snap constraints, refer to the [SnapConstraints](#).

By default, the following constraints are enabled for the snap functionality in the diagram,

\* ShowLines

- \* ShowVerticalLines
- \* ShowHorizontalLines
- \* SnapToHorizontalLines
- \* SnapToObject
- \* All

### Boundary constraints

Boundary constraints defines a boundary for the diagram inside that the interaction should be done. Boundary constraints allow to set the following behaviors.

- Infinity
- Diagram
- Page

The following list of constraints are used to Enables or Disables certain features of boundary interactions of the diagram.

Constraints	Description
-----	-----
Infinity	Allow the interactions to take place at the infinite height and width
Diagram	Allow the interactions to take place around the diagram height and width
Page	Allow the interactions to take place around the page height and width

The following code shows how to limit the interaction done inside a diagram within a page.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes">
  @* Initialize the pagesettings with boundary constraints *@
  <PageSettings Width="600" Height="500"
    BoundaryConstraints="BoundaryConstraints.Page"/>
</SfDiagramComponent>
@code
{
  DiagramObjectCollection<Node> nodes;
  protected override void OnInitialized()
  {
    //Initialize the NodeCollection.
    nodes = new DiagramObjectCollection<Node>();
    Node node = new Node()
    {
      ID = "node1",
      Height = 100,
      Width = 100,
      OffsetX = 100,
      OffsetY = 100,
    };
    nodes.Add(node);
  }
}
```

```
}
```

For more information about selector constraints, refer to the [BoundaryConstraints](#).

By default, the following boundary constraints are enabled for the snap functionality in the diagram,

\* Diagram

### Inherit behaviors

Some of the behaviors can be defined through both the specific object (node or connector) and diagram. When the behaviors are contradictorily defined through both, the actual behavior is set through inherit options.

The following code example shows how to inherit the line bridging behavior from the diagram.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
/* Initialize the diagram with constraints */
<SfDiagramComponent Height="600px"
Connectors="@connectors"
Constraints="@diagramConstraints">
</SfDiagramComponent>
@code
{
//Sets the diagram constraints
DiagramConstraints diagramConstraints = DiagramConstraints.Default |
DiagramConstraints.Bridging;
DiagramObjectCollection<Connector> connectors;
protected override void OnInitialized()
{
connectors = new DiagramObjectCollection<Connector>();
Connector connector = new Connector()
{
ID = "connector1",
SourcePoint = new DiagramPoint() { X = 100, Y = 100 },
TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
//sets the ConnectorConstraints...
Constraints = ConnectorConstraints.Default |
ConnectorConstraints.InheritBridging
};
Connector connector1 = new Connector()
{
ID = "connector2",
SourcePoint = new DiagramPoint() { X = 200, Y = 100 },
TargetPoint = new DiagramPoint() { X = 100, Y = 200 },
};
connectors.Add(connector);
connectors.Add(connector1);
}
}
```

### Bitwise operations

Bitwise operations are used to manipulate the flagged enumerations `enum`. In this section, Bitwise operations are shown by using the node constraints. The same is applicable when working with node constraints, connector constraints, or port constraints.

### Add operation

You can add or enable multiple values at a time by using the Bitwise `|` (OR) operator.

The following code shows to add bridging constraints into the default diagram constraints to enable bridging functionality into the diagram.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Constraints="@diagramConstraint"/>
@code
{
    //To adding line routing constraint with default constraints.
    DiagramConstraints diagramConstraint = DiagramConstraints.Default |
    DiagramConstraints.Bridging;
}
```

### Remove Operation

You can remove or disable values by using the Bitwise `&~` (XOR) operator.

The following code shows to remove zoom and pan constraints from the default constraints to disable zoom and panning functionality in the diagram.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Constraints="@diagramConstraint"/>
@code
{
    //To removing zoom and panning constraints from the default constraints
    //It has disabled zoom and panning functionality for the diagram.
    DiagramConstraints diagramConstraint = DiagramConstraints.Default &~
    (DiagramConstraints.Zoom | DiagramConstraints.Pan);
}
```

### Check operation

You can check any value by using the Bitwise `&` (AND) operator.

#### C#

```
if ((node.constraints & (NodeConstraints.Rotate)) ==
    (NodeConstraints.Rotate));
```

In the previous example, check whether the rotate constraints are enabled in a node. When node constraints have rotated constraints, the expression returns a rotate constraint.



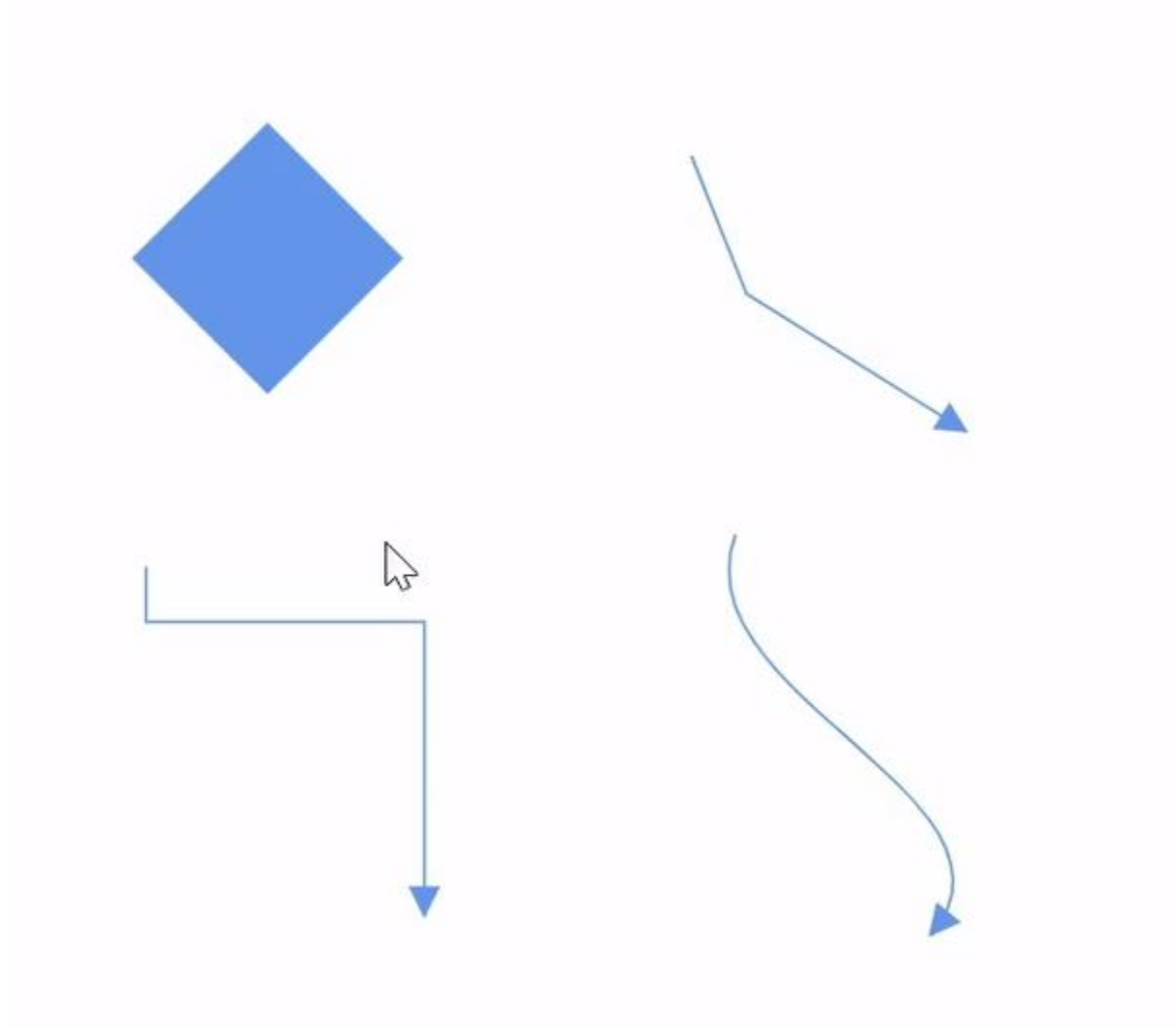
## Interaction in Blazor Diagram Component

### Selection

[DiagramSelectionSettings](#) provides a visual representation of selected elements. It behaves like a container and allows to update the size, position, and rotation angle of the selected elements through interaction and by using program. Single or multiple elements can be selected at a time.

### Single selection

An element can be selected by clicking that element. During single click, all previously selected items are cleared. The following image shows how the selected elements are visually represented.



- While selecting the diagram elements, the following events can be used to do your customization.
- When selecting/unselecting the diagram elements, the following events are gets triggered and to do customization on those events.

Events	EventArgs	Description
-----	-----	-----

| [SelectionChanging](#) | [SelectionChangingEventArgs](#) | Notify when click to selecting the elements in the diagram |

| [SelectionChanged](#) | [SelectionChangedEventArgs](#) | Notify after click to selected the elements in the diagram |

### CSHARP

```
<SfDiagramComponent Height="600px" Nodes="@NodeCollection"
SelectionChanging="OnSelectionChanging"
SelectionChanged="OnSelectionChanged">
</SfDiagramComponent>
@code
{
    public DiagramObjectCollection<Node> NodeCollection = new
    DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        Node node = new Node()
        {
            OffsetX = 100,
            OffsetY = 200,
            Height = 100,
            Width = 100,
            ID = "node",
        };
        NodeCollection.Add(node);
    }
    //To notify selection changing event, before select the nodes/connector in
    diagram.
    private void OnSelectionChanging(SelectionChangingEventArgs args)
    {
        //sets true to cancel the element's selection
        args.Cancel = true;
    }
    //To notify selection changed event, after selected the nodes/connector in
    diagram.
    private void OnSelectionChanged(SelectionChangedEventArgs args)
    {
        //Action to be performed.
    }
}
```

### Selecting a group

When a child element of any group is clicked, its contained group is selected instead of the child element. With consecutive clicks on the selected element, selection is changed from top to bottom in the hierarchy of parent group to its children.

### Multiple selection

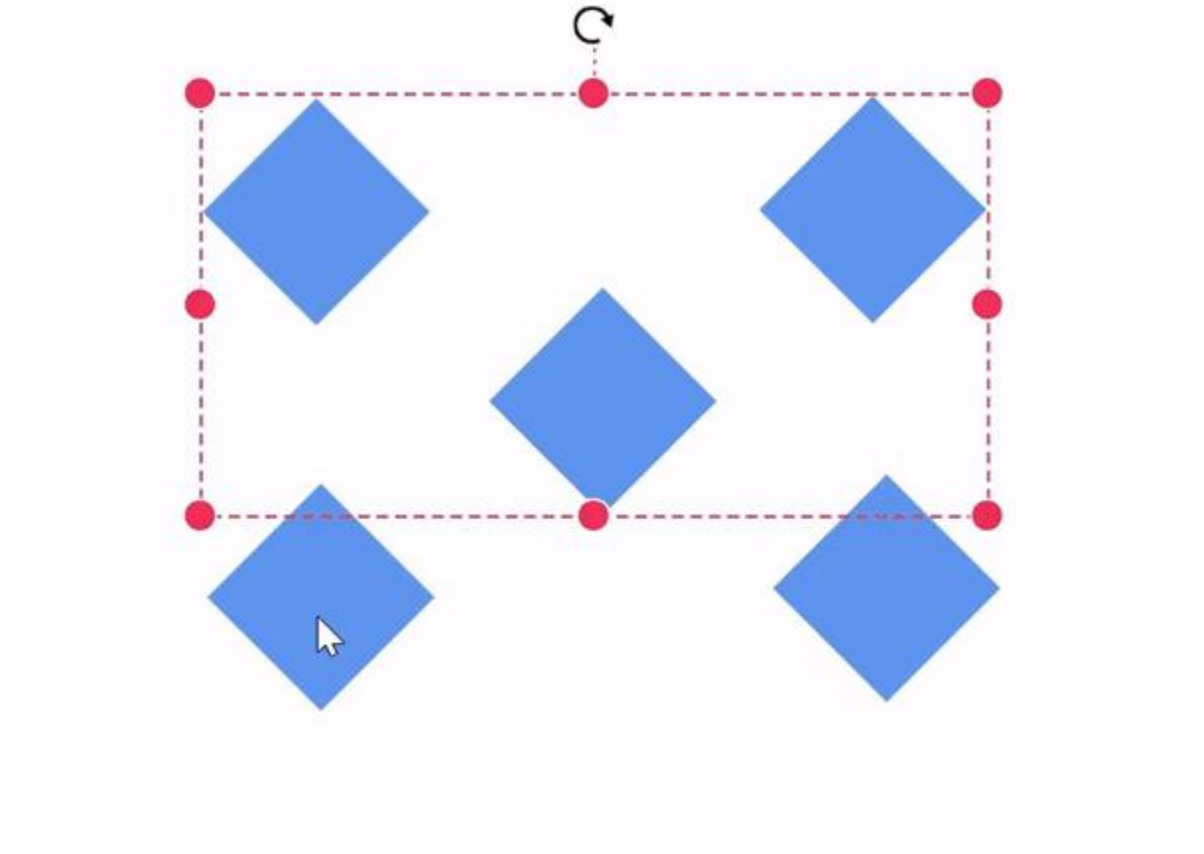
Multiple elements can be selected with the following ways:

- Ctrl+Click

During single click, any existing item in the selection list be cleared, and only the item clicked recently is there in the selection list. To avoid clearing the old selected item, Ctrl key must be on hold when clicking.

- Selection rectangle/rubber band selection

Clicking and dragging the diagram area allows to create a rectangular region. The elements that are covered under the rectangular region are selected at the end.



#### Select/Unselect elements using program

The [Select](#) and [ClearSelection](#) methods help to select or clear the selection of the elements at runtime.

Get the current selected items from the [Nodes](#) and [Connectors](#) collection of the [SelectionSettings](#) property of the diagram model.

#### Select entire elements in diagram programmatically

The [SelectAll](#) method used to select all the elements such as nodes/connectors in the diagram. Refer to the following link which shows how to use SelectAll method on the diagram.

#### Drag

- An object can be dragged by clicking and dragging it. When multiple elements are selected, dragging any one of the selected elements move every selected element.

- When you drag the elements in the diagram, the following events are triggered and to do customization on those events.

Events	EventArgs	Description
<a href="#">PositionChanging</a>	<a href="#">PositionChangingEventArgs</a>	Notify while dragging the elements in the diagram
<a href="#">PositionChanged</a>	<a href="#">PositionChangedEventArgs</a>	Notify when the elements's position has changed in the diagram

### CSHARP

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@NodeCollection"
PositionChanging="OnPositionChanging"
PositionChanged="OnPositionChanged">
</SfDiagramComponent>
@code
{
    public DiagramObjectCollection<Node> NodeCollection = new
    DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        Node node = new Node()
        {
            OffsetX = 100,
            OffsetY = 200,
            Height = 100,
            Width = 100,
            ID = "node",
        };
        NodeCollection.Add(node);
    }
    //Event to notify while dragging the elements in the diagram.
    private void OnPositionChanging(PositionChangingEventArgs args)
    {
        //sets true to cancel the element's dragging
        args.Cancel = true;
    }
    //Event to notify once element's position has changed in the diagram.
    private void OnPositionChanged(PositionChangedEventArgs args)
    {
        //Action to be performed.
    }
}
```

For more information about dragging , refer [Node Drag](#)

### Resize

- Selector is surrounded by eight thumbs. When dragging these thumbs, selected items can be resized.
- When one corner of the selector is dragged, opposite corner is in a static position.
- When a node is resized, the following events are gets triggered.

| Events | EventArgs | Description |

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

| [SizeChanging](#) | [SizeChangingEventArgs](#) | Notify while resizing the elements in the diagram |

| [SizeChanged](#) | [SizeChangedEventArgs](#) | Notify when the element's size has changed in the diagram |

### CSHARP

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@NodeCollection"
SizeChanging="OnSizeChanging"
SizeChanged="OnSizeChanged">
</SfDiagramComponent>
@code
{
    public DiagramObjectCollection<Node> NodeCollection = new
    DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        Node node = new Node()
        {
            OffsetX = 100,
            OffsetY = 200,
            Height = 100,
            Width = 100,
            ID = "node",
        };
        NodeCollection.Add(node);
    }
    //Event to notify while resizing the elements in the diagram.
    private void OnSizeChanging(SizeChangingEventArgs args)
    {
        //sets true to cancel the element's resizing
        args.Cancel = true;
    }
    //Event to notify once element's size has changed in the diagram.
    private void OnSizeChanged(SizeChangedEventArgs args)
    {
        //Action to be performed.
    }
}
```

For more information about resizing , refer [Node Resize](#)

Note: While dragging and resizing, the objects are snapped towards the nearest objects to make better alignments. For better alignments, refer to [Snapping](#).

### Rotate

- A rotate handler is placed above the selector. Clicking and dragging the handler in a circular direction lead to rotate the node.
- The node is rotated with reference to the static pivot point.
- Pivot thumb (thumb at the middle of the node) appears while rotating the node to represent the static point.

- When a node is rotated, the following events are gets triggered.

Events	EventArgs	Description
<a href="#">RotationChanging</a>	<a href="#">RotationChangingEventArgs</a>	Notify while rotating the elements in the diagram
<a href="#">RotationChanged</a>	<a href="#">RotationChangedEventArgs</a>	Notify when the element's rotate angle has changed in the diagram

### CSHARP

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@NodeCollection"
RotationChanging="OnRotationChanging"
RotationChanged="OnRotationChanged">
</SfDiagramComponent>
@code
{
    public DiagramObjectCollection<Node> NodeCollection = new
    DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        Node node = new Node()
        {
            OffsetX = 100,
            OffsetY = 200,
            Height = 100,
            Width = 100,
            ID = "node",
        };
        NodeCollection.Add(node);
    }
    //Event to notify while rotating the elements in the diagram.
    private void OnRotationChanging(RotationChangingEventArgs args)
    {
        //sets true to cancel the element's rotation
        args.Cancel = true;
    }
    //Event to notify once element's rotate angle has changed in the diagram.
    private void OnRotationChanged(RotationChangedEventArgs args)
    {
        //Action to be performed.
    }
}
```

For more information about resizing , refer [Node Rotate](#)

### Connection editing

- Each segment of a selected connector is editable with some specific handles/thumbs.

### End point handles

Source and target points of the selected connectors are represented with two handles. Clicking and dragging those handles help you to adjust the source and target points.

For more information , refer [End Point Dragging](#)

- If you drag the connector end points, then the following events can be used to do your customization.
- When you connect connector with ports/node or disconnect from it, the following events are gets triggered.

Events	EventArgs	Description
<a href="#">ConnectionChanging</a>	<a href="#">ConnectionChangingEventArgs</a>	Notify while creating the connection between the nodes in the diagram
<a href="#">ConnectionChanged</a>	<a href="#">ConnectionChangedEventArgs</a>	Notify once the connection has created between the nodes in the diagram

### CSHARP

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@NodeCollection"
Connectors="@Connectors"
ConnectionChanging="OnConnectionChanging"
ConnectionChanged="OnConnectionChanged">
</SfDiagramComponent>
@code
{
    public DiagramObjectCollection<Node> NodeCollection = new
    DiagramObjectCollection<Node>();
    public DiagramObjectCollection<Connector> Connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Node node1 = new Node()
        {
            OffsetX = 100,
            OffsetY = 200,
            Height = 100,
            Width = 100,
            ID = "node1",
        };
        NodeCollection.Add(node1);
        Node node2 = new Node()
        {
            OffsetX = 300,
            OffsetY = 200,
            Height = 100,
            Width = 100,
            ID = "node2",
        };
        NodeCollection.Add(node2);
        Connector Connector = new Connector()
```

```

{
    ID = "connector1",
    //Source node id of the connector.
    SourceID = "node1",
    TargetDecorator = new DecoratorSettings()
    {
        Style = new ShapeStyle()
        {
            Fill = "#6495ED",
            StrokeColor = "#6495ED",
        }
    },
    //Target node id of the connector.
    TargetID = "node2",
    Style = new ShapeStyle()
    {
        Fill = "#6495ED",
        StrokeColor = "#6495ED",
    },
    // Type of the connector
    Type = ConnectorSegmentType.Straight,
};
Connectors.Add(Connector);
}
//Event to notify while creating the connection between the nodes in the diagram.
private void OnConnectionChanging(ConnectionChangingEventArgs args)
{
    //sets true to cancel the element's resizing
    args.Cancel = true;
}
//Event to notify once created the connection between the nodes in the diagram.
private void OnConnectionChanged(ConnectionChangedEventArgs args)
{
    //Action to be performed.
}
}

```

### Straight segment editing

- End point of each straight segment is represented by a thumb that enables to edit the segment.
- Any number of new segments can be inserted into a straight line by clicking, when Shift and Ctrl keys are pressed (Ctrl+Shift+Click).
- Straight segments can be removed by clicking the segment end point, when Ctrl and Shift keys are pressed (Ctrl+Shift+Click).

For more information about straight segment editing , refer [Straight Segment Editing](#)

### Orthogonal segment editing

- Orthogonal thumbs allow you to adjust the length of adjacent segments by clicking and dragging it.



- When necessary, some segments are added or removed automatically, when dragging the segment. This is to maintain proper routing of orthogonality between segments.
- When you editing the segment collection of connector, the following event gets triggered.

| Events | EventArgs | Description |

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

| [SegmentCollectionChange](#) | [SegmentCollectionChangeEventArgs](#) | Notify when the connector's segment collection has modified in the diagram. |

### CSHARP

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px" Height="500px" Connectors="@Connectors"
SegmentCollectionChange="OnSegmentChange">
</SfDiagramComponent>
@code{
    //Defines diagram's connector collection
    DiagramObjectCollection<Connector> Connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Connector connector = new Connector()
        {
            ID = "connector1",
            SourcePoint = new DiagramPoint()
            {
                X = 100,
                Y = 100
            },
            // Enable DragSegmentThumb constraints to segment editing
            Constraints = ConnectorConstraints.Default |
            ConnectorConstraints.DragSegmentThumb,
            Style = new ShapeStyle() { StrokeColor = "#6f409f", StrokeWidth = 1 },
            TargetPoint = new DiagramPoint() { X = 300, Y = 300 },
            //Specify the segments type as Orthogonal.
            Type = ConnectorSegmentType.Orthogonal,
            //Create a new segment with length and direction
            Segments = new DiagramObjectCollection<ConnectorSegment>()
            {
                new OrthogonalSegment
                {
                    Length = 100,
                    Type = ConnectorSegmentType.Orthogonal,
                    Direction = Direction.Right,
                },
                new OrthogonalSegment
                {
                    Length = 100,
                    Type = ConnectorSegmentType.Orthogonal,
                    Direction = Direction.Bottom,
                },
            },
            TargetDecorator = new DecoratorSettings()
            {
                Shape = DecoratorShape.Arrow,
```

```
Style = new ShapeStyle()
{
    Fill = "#6f409f",
    StrokeColor = "#6f409f",
    StrokeWidth = 1
}
};
Connectors.Add(connector);
}
//Event to notify while modifying the segment collection for the connector.
private void OnSegmentChange(SegmentCollectionChangeEventArgs args)
{
    //Action to be perform...
}
}
```

For more information about orthogonal segment editing , refer [Orthogonal Segment Editing](#)

### User handles

- User handles are used to add some frequently used commands around the selector. To create user handles, define and add them to the [UserHandles](#) collection of the [SelectionSettings](#) property.
- The [Name](#) property of user handle is used to define the name of the user handle and its further used to find the user handle at runtime and do any customization.

### Alignment

User handles can be aligned relative to the node boundaries. It has [Margin](#), [Offset](#), [Side](#), [HorizontalAlignment](#), and [VerticalAlignment](#) settings. It is quite tricky when all four alignments are used together but gives more control over alignment.

### Offset

The [Offset](#) property of [UserHandle](#) is used to align the user handle based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

### Side

The [Side](#) property of [UserHandle](#) is used to align the user handle by using the [Top](#), [Bottom](#), [Left](#), and [Right](#) options.

### Horizontal and vertical alignments

The [HorizontalAlignment](#) property of [UserHandle](#) is used to set how the user handle is horizontally aligned at the position based on the [Offset](#). The [VerticalAlignment](#) property is used to set how user handle is vertically aligned at the position.

### Margin

Margin is an absolute value used to add some blank space in any one of its four sides. The [UserHandle](#) can be displaced with the [Margin](#) property.

### Notification for the mouse button clicked

The diagram component notifies the mouse button clicked. For example, whenever the right mouse button is clicked, the clicked button is notified as right. The mouse click is notified with,

| Notification | Description |

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

| Left | When the left mouse button is clicked, left is notified |

| Middle | When the mouse wheel is clicked, middle is notified |

| Right | When the right mouse button is clicked, right is notified |

### CSHARP

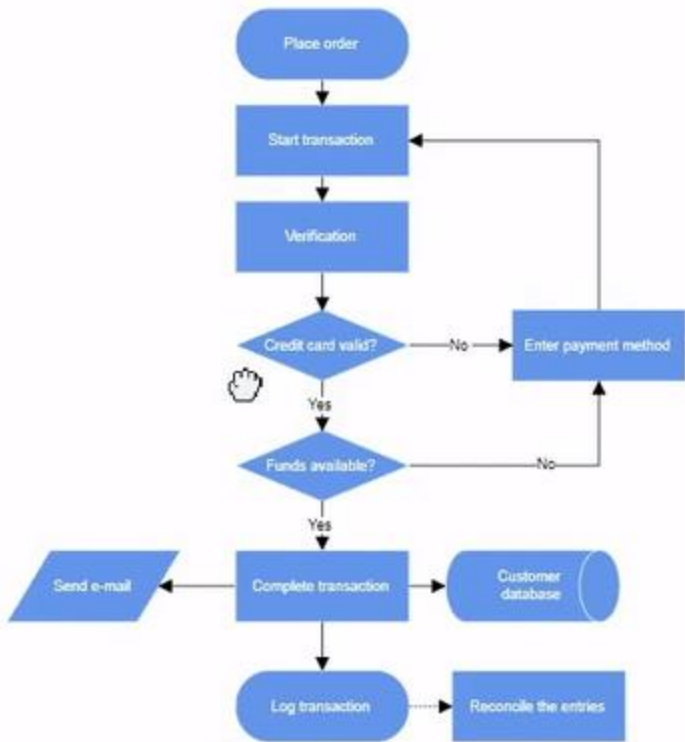
```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" Click='@OnClick' />
@code
{
    public DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in nodes array.
        Node node = new Node()
        {
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            // Add node
            Style = new ShapeStyle() { Fill = "#6BA5D7", StrokeDashArray = "5,5",
            StrokeColor = "red", StrokeWidth = 2 },
        };
        nodes.Add(node);
    }
    private void OnClick(ClickEventArgs args)
    {
        Console.WriteLine("Button", args.Button);
    }
}
```

### Appearance

The appearance of the user handle can be customized by using the [Size](#), [BorderColor](#), [BackgroundColor](#), [Visible](#), [PathData](#), and [PathColor](#) properties of the [UserHandle](#).

### Zoom pan

- When a large diagram is loaded, only certain portion of the diagram is visible. The remaining portions are clipped. Clipped portions can be explored by scrolling the scrollbars or panning the diagram.
- Diagram can be zoomed in or out by using Ctrl + mouse wheel.



Keyboard

Diagram provides support to interact with the elements with key gestures. By default, some in-built commands are bound with a relevant set of key combinations.

The following table illustrates those commands with the associated key values.

Shortcut Key	Command	Description
-----	-----	-----
Ctrl + A	<a href="#">SelectAll</a>	Select all nodes/connectors in the diagram.
Ctrl + C	<a href="#">Copy</a>	Copy the diagram selected elements.
Ctrl + V	<a href="#">Paste</a>	Pastes the copied elements.
Ctrl + X	<a href="#">Cut</a>	Cuts the selected elements.
Ctrl + Z	<a href="#">Undo</a>	Reverses the last editing action performed on the diagram.
Ctrl + Y	<a href="#">Redo</a>	Restores the last editing action when no other actions have occurred since the last undo on the diagram.
Delete	Delete	Deletes the selected elements.
Ctrl/Shift + Click on object		Multiple selection (Selector binds all selected nodes/connectors).
Up Arrow	<a href="#">Nudge(Direction.Up)</a>	<b>NudgeUp</b> : Moves the selected elements towards up by one pixel.

| Down Arrow | [Nudge\(Direction.Down\)](#) | **NudgeDown**: Moves the selected elements towards down by one pixel. |

| Left Arrow | [Nudge\(Direction.Left\)](#) | **NudgeLeft**: Moves the selected elements towards left by one pixel. |

| Right Arrow | [Nudge\(Direction.Right\)](#) | **NudgeRight**: Moves the selected elements towards right by one pixel. |

| Ctrl + MouseWheel | [Zoom](#) | Zoom (Zoom in/Zoom out the diagram). |

| F2 | [StartTextEdit](#) | Starts to edit the label of selected element. |

| Esc | | Sets the label mode as view and stops editing. |

See Also

- [How to control the diagram history](#)

## Tools in Blazor Diagram Component

### Drawing tools

Drawing tool allows you to draw any kind of [Node](#) or [Connector](#) during runtime by clicking and dragging on the diagram page.

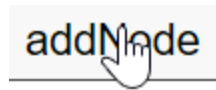
### Shapes

To draw a [shape](#), you have to activate the drawing tool by using the [InteractionController](#) property and you need to set the shape by using the [DrawingObject](#) property. The following code example illustrates how to draw a rectangle at runtime.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<input Type="button" value="addNode" @onclick="AddNode" />
<SfDiagramComponent @ref="diagram" Nodes="@nodes" Height="600px" />
@code
{
    //Reference to diagram
    SfDiagramComponent diagram;
    //Defines diagram's nodes collection
    public DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node = new Node()
        {
            ID = "group",
            OffsetX = 200,
            OffsetY = 200,
            Width = 100,
            Height = 100,
            Annotations = new DiagramObjectCollection<ShapeAnnotation>()
            {
                new ShapeAnnotation()
                {
                    Content = "Node",
                    Style = new TextStyle()
```

```
{
    Color = "white",
}
},
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" }
};
nodes.Add(node);
}
private void AddNode()
{
    //To draw an object once, activate draw once
    diagram.InteractionController = InteractionController.DrawOnce;
    //Initialize the drawing object to draw the shape
    diagram.DrawingObject = new Node()
    {
        Shape = new BasicShape() { Type = Shapes.Basic, Shape =
        BasicShapeType.Rectangle },
        Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" }
    };
}
}
```



### Connectors

To draw a [Connector](#), you have to activate the drawing tool by using the [InteractionController](#) property and you need to set the connector by using the [DrawingObject](#) property. The following code example illustrates how to draw a [StraightSegment](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<input Type="button" value="AddConnector" @onclick="AddConnector" />
<SfDiagramComponent @ref="diagram" Nodes="@nodes" Height="600px" />
@code
{
    //Reference to diagram
    SfDiagramComponent diagram;
    //Defines diagram's nodes collection
    public DiagramObjectCollection<Node> nodes;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
    }
}
```

```
Node node = new Node()
{
    ID = "group",
    OffsetX = 200,
    OffsetY = 200,
    Width = 100,
    Height = 100,
    Annotations = new DiagramObjectCollection<ShapeAnnotation>()
    {
        new ShapeAnnotation()
        {
            Content = "Node",
            Style = new TextStyle()
            {
                Color = "white",
            }
        },
    },
    Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" }
};
nodes.Add(node);
}

private void AddConnector()
{
    //To draw an object once, activate draw once
    diagram.InteractionController = InteractionController.DrawOnce;
    //Initialize the drawing object to draw the connectors
    diagram.DrawingObject = new Connector()
    {
        ID = "connector1",
        Type = ConnectorSegmentType.Straight,
    };
}
}
```





### Polygon shape

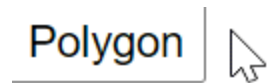
Diagram allows to create the [Polygon](#) shape by clicking and moving the mouse at runtime on the diagram page.

The following code illustrates how to draw a polygon shape.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<input Type="button" value="Polygon" @onclick="Polygon" />
<SfDiagramComponent @ref="diagram" Nodes="@nodes" Height="600px">
  <SnapSettings Constraints="SnapConstraints.None"></SnapSettings>
</SfDiagramComponent>
@code
{
  //Reference to diagram
  SfDiagramComponent diagram;
  //Defines diagram's nodes collection
  public DiagramObjectCollection<Node> nodes;
  protected override void OnInitialized()
```

```
{
nodes = new DiagramObjectCollection<Node>();
Node node = new Node()
{
ID = "group",
OffsetX = 200,
OffsetY = 200,
Width = 100,
Height = 100,
Annotations = new DiagramObjectCollection<ShapeAnnotation>()
{
new ShapeAnnotation()
{
Content = "Node",
Style = new TextStyle()
{
Color = "white",
}
},
},
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" }
};
nodes.Add(node);
}
private void Polygon()
{
//To draw an object once, activate draw once
diagram.InteractionController = InteractionController.DrawOnce;
//Initialize the drawing object to draw the polygon shape
diagram.DrawingObject = new Node()
{
ID = "polygon",
Shape = new BasicShape()
{
Type = Shapes.Basic,
Shape = BasicShapeType.Polygon,
},
};
}
}
```



### Tool selection

There are some functionalities that can be achieved by clicking and dragging on the diagram surface. They are as follows,

- Draw selection rectangle: MultipleSelect interaction Controller
- Pan the diagram: Zoom pan
- Draw nodes/connectors: DrawOnce/DrawOnce

As all the three behaviors are completely different, you can achieve only one behavior at a time based on the interaction controller that you choose.

When more than one of those interaction controllers are applied, a interaction controller is activated based on the precedence given in the following table.

Precedence	InteractionControllers	Description
-----	----	-----

|1st| [ContinuesDraw](#) | Allows you to draw the nodes or connectors continuously. Once it is activated, you cannot perform any other interaction in the diagram. |

|2nd| [DrawOnce](#) | Allows you to draw a single node or connector. Once you complete the DrawOnce action, SingleSelect, and MultipleSelect interaction controllers are automatically enabled. |

|3rd| [ZoomPan](#) | Allows you to pan the diagram. When you enable both the SingleSelect and ZoomPan interaction controllers, you can perform the basic interaction as the cursor hovers node/connector. Panning is enabled when cursor hovers the diagram. |

|4th| [MultipleSelect](#) | Allows you to select multiple nodes and connectors. When you enable both the MultipleSelect and ZoomPan interaction controllers, cursor hovers the diagram. When panning is enabled, you cannot select multiple nodes. |

|5th| [SingleSelect](#) | Allows you to select individual nodes or connectors. |

|6th| None | Disables all interaction controllers. |

|7th| [Default](#) | Allows users to select an individual as well as multiple nodes and connectors. |

Set the desired [InteractionController](#) to the diagram.

The following code illustrates how to enable single interaction controller,

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Connectors="@connectors" Height="600px"
InteractionController="@tool" />
@code
{
    //Enable the single tool
    public InteractionController tool = InteractionController.DrawOnce;
    //Defines diagram's connectors collection
    public DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
}
```

The following code illustrates how to enable multiple interaction controllers,

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Connectors="@connectors" @ref="diagram" Height="600px"
InteractionController="@tool" />
@code
{
    //Reference to diagram
    SfDiagramComponent diagram;
    //Enable the multiple tools
    public InteractionController tool = InteractionController.DrawOnce |
    InteractionController.ZoomPan;
    //Defines diagram's connectors collection
    public DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
}
```

## Gridlines in Blazor Diagram Component

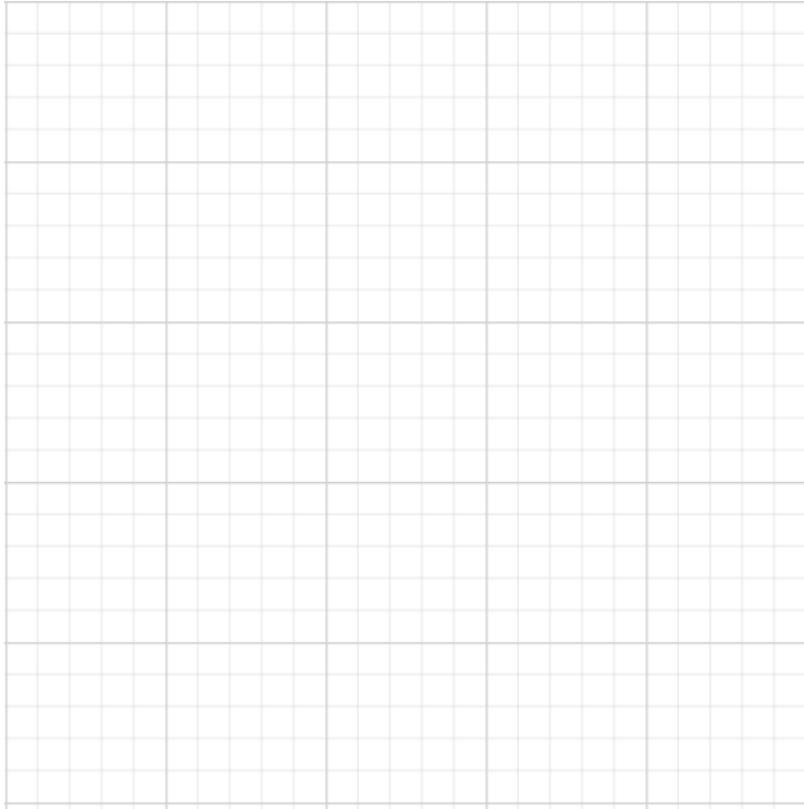
Gridlines are the pattern of lines drawn behind the diagram elements. It provides a visual guidance while dragging or arranging the objects on the diagram surface. The [SnapSettings](#) property is used to customize the gridlines and control the snapping behavior in the diagram.

### Customize the Gridlines visibility

The [Constraints](#) property of SnapSettings class allows you to control the visibility of the gridlines.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes">
  /* Initialize the diagram snapping the custom interval */
  <SnapSettings Constraints="SnapConstraints.All">
    <HorizontalGridLines SnapIntervals="@SnapInterval">
    </HorizontalGridLines>
    <VerticalGridLines SnapIntervals="@SnapInterval">
    </VerticalGridLines>
  </SnapSettings>
</SfDiagramComponent>
@code
{
  //Sets the snapinterval...
  public double[] SnapInterval { get; set; } = new double[]
  {
    10
  };
  DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
  protected override void OnInitialized()
  {
    nodes = new DiagramObjectCollection<Node>();
    Node diagramNode = new Node();
    diagramNode.OffsetX = 100;
    diagramNode.OffsetY = 100;
    diagramNode.Width = 100;
    diagramNode.Height = 100;
    diagramNode.Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor =
    "#6495ED" };
    diagramNode.ID = "node1";
    nodes.Add(diagramNode);
  }
}
```



To show only horizontal/vertical gridlines or to hide gridlines, refer to [SnapConstraints](#).

### Appearance

The appearance of the gridlines can be customized by using a set of predefined properties.

- The [HorizontalGridLines](#) and the [VerticalGridLines](#) properties allow to customize the appearance of the horizontal and vertical gridlines respectively.
- The horizontal gridlines [LineColor](#) and [LineDashArray](#) properties are used to customize the line color and line style of the horizontal gridlines.
- The vertical gridlines [LineColor](#) and [LineDashArray](#) properties are used to customize the line color and line style of the vertical gridlines.

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

### ASPX-CS

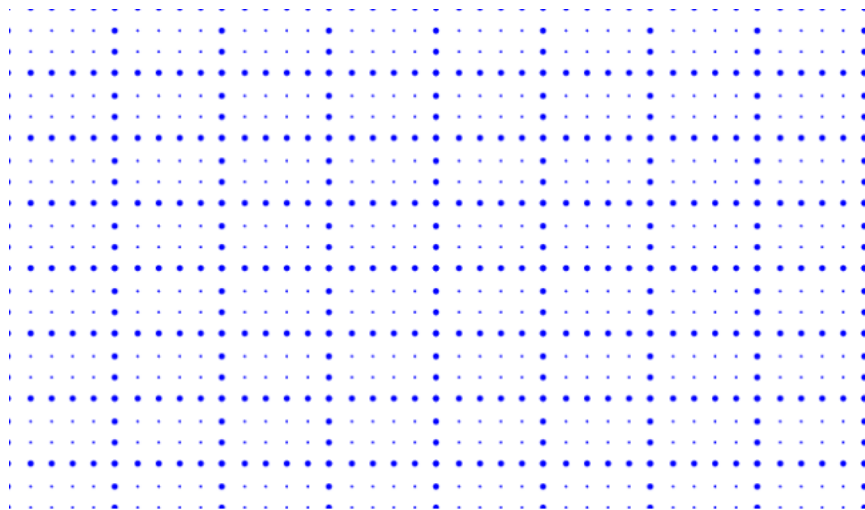
```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px">
  @* Shows both horizontal and vertical gridlines *@
  <SnapSettings Constraints="SnapConstraints.ShowLines">
    @* Customizes the line color and line style to the gridlines *@
    <HorizontalGridLines LineColor="blue" LineDashArray="2,2" />
    <VerticalGridLines LineColor="blue" LineDashArray="2,2" />
  </SnapSettings>
</SfDiagramComponent>
```

### How to create dot grid patterns

The appearance of the grid lines can be changed into dots by using the [GridType](#) as Dots. The following code illustrates how to render grid patterns as Dots.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Width="1000px" Height="500px">
  @* Customizes the appearance of the gridlines *@
  <SnapSettings GridType="GridType.Dots">
    <HorizontalGridLines LineColor="Blue" @bind-LineIntervals="@HInterval"
      @bind-DotIntervals="@HDotInterval"></HorizontalGridLines>
    <VerticalGridLines LineColor="Blue" @bind-LineIntervals="@VInterval"
      @bind-DotIntervals="@VDotInterval"></VerticalGridLines>
  </SnapSettings>
</SfDiagramComponent>
@code
{
  public double[] HDotInterval { get; set; } = new double[] { 3, 20, 1, 20, 1, 20 };
  public double[] VDotInterval { get; set; } = new double[] { 3, 20, 1, 20, 1, 20, 1, 20, 1, 20 };
  public double[] HInterval { get; set; } = new double[] { 1.25, 18.75, 0.25, 19.75, 0.25, 19.75, 0.25, 19.75, 0.25, 19.75 };
  public double[] VInterval { get; set; } = new double[] { 1.25, 18.75, 0.25, 19.75, 0.25, 19.75, 0.25, 19.75, 0.25, 19.75 };
}
```



### Line intervals

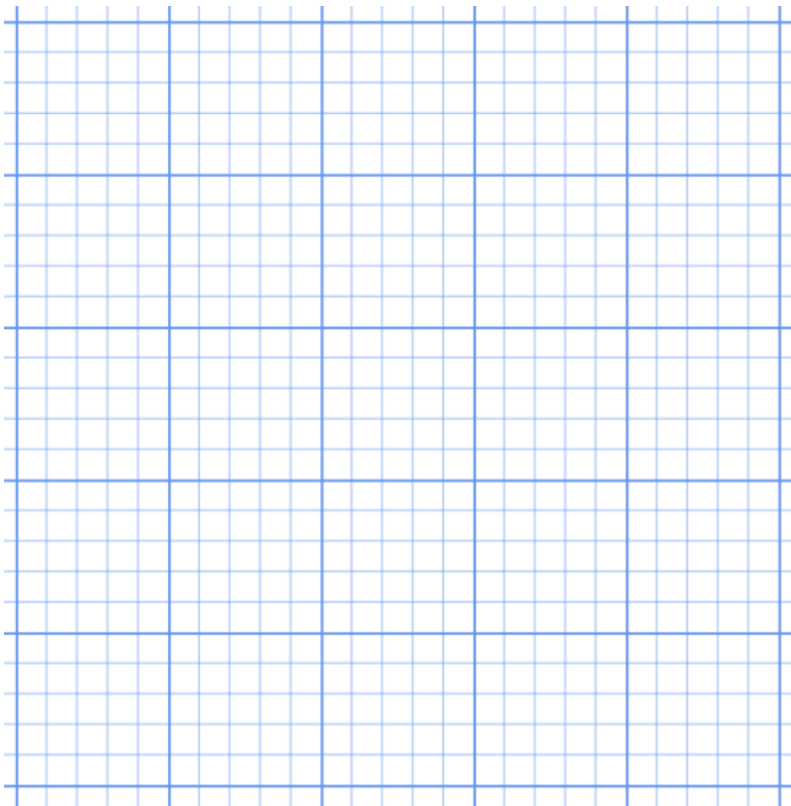
Thickness and the space between gridlines can be customized by using [LinesIntervals](#) property of the HorizontalGridLines and VerticalGridLines. In the line intervals collections, values at the odd places are referred as the thickness of lines and values at the even places are referred as the space between gridlines.

The following code example illustrates how to customize the thickness of lines and the line intervals.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
```

```
<SfDiagramComponent Height="600px">
  @* Customize the appearance of the grid lines *@
  <SnapSettings Constraints="SnapConstraints.ShowLines">
    <HorizontalGridLines LineColor="blue" LineDashArray="2,2"
      LineIntervals="@LineInterval">
    </HorizontalGridLines>
    <VerticalGridLines LineColor="blue" LineDashArray="2,2"
      LineIntervals="@LineInterval">
    </VerticalGridLines>
  </SnapSettings>
</SfDiagramComponent>
@code
{
  //Sets the line intervals for the gridlines
  public double[] LineInterval { get; set; } = new double[]
  {
    1.25, 14, 0.25, 15, 0.25, 15, 0.25, 15, 0.25, 15
  };
}
```



## Snapping

### *Snap to lines*

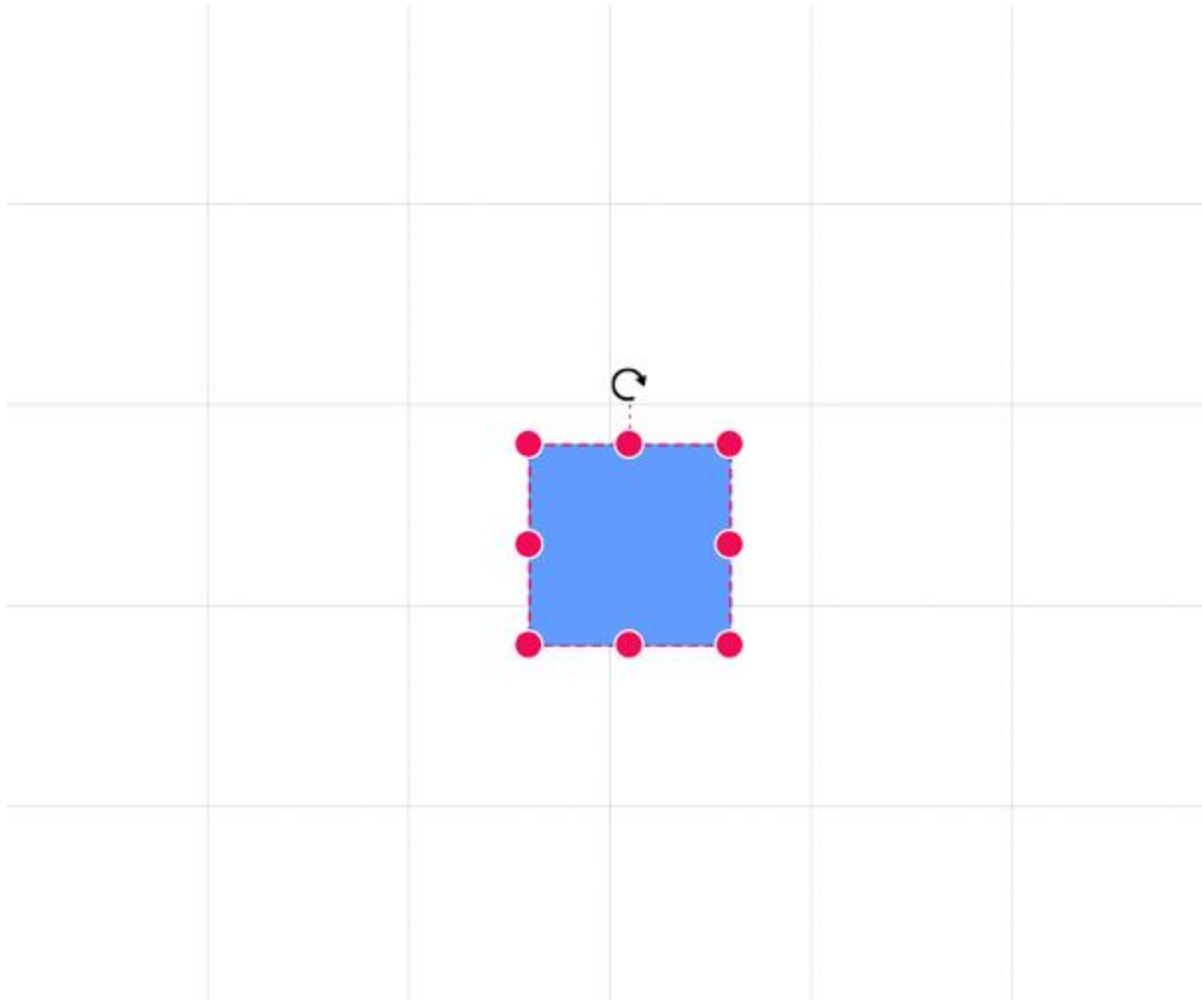
This feature allows the diagram objects to snap to the nearest intersection of gridlines while being dragged or resized. This feature enables easier alignment during layout or design.

Snapping to gridlines can be enabled/disabled with the [SnapConstraints](#). The following code example illustrates how to enable/disable the snapping to gridlines.



**ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes">
  <SnapSettings Constraints="@snapConstraints"></SnapSettings>
</SfDiagramComponent>
@code
{
  //Sets the snap constraints
  public SnapConstraints snapConstraints = SnapConstraints.ShowLines |
  SnapConstraints.SnapToLines;
  DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
  protected override void OnInitialized()
  {
    nodes = new DiagramObjectCollection<Node>();
    Node diagramNode = new Node();
    diagramNode.OffsetX = 100;
    diagramNode.OffsetY = 100;
    diagramNode.Width = 100;
    diagramNode.Height = 100;
    diagramNode.Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor =
    "#6495ED" };
    diagramNode.ID = "node1";
    nodes.Add(diagramNode);
  }
}
```



### Customization of snap intervals

By default, the objects are snapped towards the nearest gridline. The gridline or position towards where the diagram object snaps can be customized by using [SnapIntervals](#) property of the `HorizontalGridLines` and `VerticalGridLines`.

### ASPX-CS

```
@page "/CustomSnapLineInterval Sample"
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes">
  /* Initialize the diagram snapping the custom interval */
  <SnapSettings Constraints="SnapConstraints.All">
    <HorizontalGridLines SnapIntervals="@SnapInterval">
    </HorizontalGridLines>
    <VerticalGridLines SnapIntervals="@SnapInterval">
    </VerticalGridLines>
  </SnapSettings>
</SfDiagramComponent>
@code
{
  //Sets the snapinterval...
  public double[] SnapInterval { get; set; } = new double[]
  {
```

```

10
};
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
protected override void OnInitialized()
{
    nodes = new DiagramObjectCollection<Node>();
    Node diagramNode = new Node();
    diagramNode.OffsetX = 100;
    diagramNode.OffsetY = 100;
    diagramNode.Width = 100;
    diagramNode.Height = 100;
    diagramNode.Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor =
"#6495ED" };
    diagramNode.ID = "node1";
    nodes.Add(diagramNode);
}
}

```

### *Snap to objects*

The snap to object provides visual cues to assist with aligning and spacing diagram elements. A node can be snapped with its neighboring objects based on certain alignments. Such alignments are visually represented as smart guides.

- The [SnapDistance](#) property allows you to define minimum distance between the selected object and the nearest object.
- The [SnapAngle](#) property allows you to define the snap angle by which the object needs to be rotated.
- The Constraints property of SnapSettings class allows you to enable or disable the certain features of the snapping, refer to [Constraints](#).

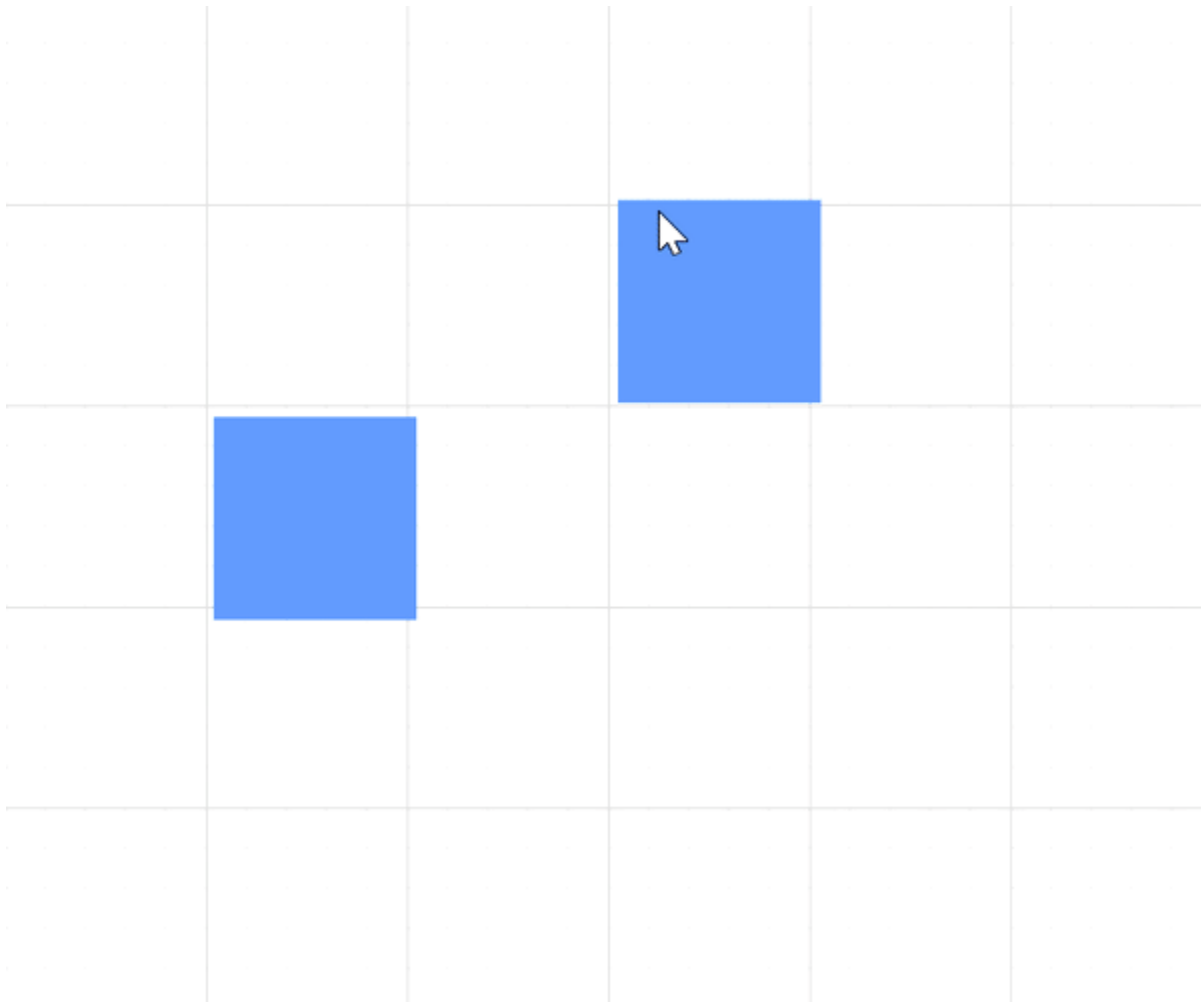
### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes">
<SnapSettings Constraints="@snapConstraints" SnapAngle="10"
SnapDistance="10">
</SnapSettings>
</SfDiagramComponent>
@code
{
    //Sets the Snap to objects constraints...
    public SnapConstraints snapConstraints = SnapConstraints.ShowLines |
    SnapConstraints.SnapToObject;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node diagramNode = new Node();
        diagramNode.OffsetX = 100;
        diagramNode.OffsetY = 100;
        diagramNode.Width = 100;
        diagramNode.Height = 100;
        diagramNode.Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor =
"#6495ED" };
    }
}

```

```
diagramNode.ID = "node1";  
nodes.Add(diagramNode);  
diagramNode = new Node();  
diagramNode.OffsetX = 300;  
diagramNode.OffsetY = 100;  
diagramNode.Width = 100;  
diagramNode.Height = 100;  
diagramNode.Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor =  
"#6495ED" };  
diagramNode.ID = "node2";  
nodes.Add(diagramNode);  
}  
}
```



### Data Binding in Blazor Diagram Component

- [Diagram](#) can be populated with the [Nodes](#) and [Connectors](#) based on the information provided from an external data source.
- Diagram exposes its specific data-related properties allowing you to specify the data source fields from where the node information has to be retrieved from.

- The [DataSource](#) property is used to define the data source either as a collection of objects or as an instance of `DataSource` that needs to be populated in the diagram.
- The [ID](#) property is used to define the unique field of each JSON data.
- The [ParentID](#) property is used to define the parent field which builds the relationship between ID and parent field.
- The [Root](#) property is used to define the root node for the diagram populated from the data source.
- To explore those properties, see [DataSourceSettings](#).
- Diagram supports two types of data binding. They are:
  1. Local data
  2. Remote data

### Local data

Diagram can be populated based on the user defined JSON data (Local Data) by mapping the relevant data source fields.

To map the user defined JSON data with diagram, configure the fields of `DataSourceSettings`. The following code example illustrates how to bind local data with the diagram.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="@Diagram"
Height="499px"
InteractionController="InteractionController.ZoomPan"
ConnectorCreating="@ConnectorDefaults"
NodeCreating="@NodeDefaults">
  <DataSourceSettings ID="Name" ParentID="Category" DataSource="DataSource"/>
  <Layout @bind-Type="type"
@bind-HorizontalSpacing="@HorizontalSpacing"
@bind-Orientation="@orientation"
@bind-VerticalSpacing="@VerticalSpacing"
@bind-HorizontalAlignment="@horizontalAlignment"
@bind-VerticalAlignment="@verticalAlignment"
GetLayoutInfo="GetLayoutInfo">
    <LayoutMargin Top="50" Bottom="50" Right="50" Left="50"/>
  </Layout>
</SfDiagramComponent>
@code
{
    SfDiagramComponent Diagram;
    // Specify the layout type.
    LayoutType type = LayoutType.HierarchicalTree;
    // Specify the orientation of the layout.
    LayoutOrientation orientation = LayoutOrientation.TopToBottom;
    HorizontalAlignment horizontalAlignment = HorizontalAlignment.Auto;
    VerticalAlignment verticalAlignment = VerticalAlignment.Auto;
    int HorizontalSpacing = 30;
    int VerticalSpacing = 30;
    // Defines the connector's default values.
    private void ConnectorDefaults(IDiagramObject connector)
    {
        (connector as Connector).Type = ConnectorSegmentType.Orthogonal;
        (connector as Connector).TargetDecorator.Shape = DecoratorShape.None;
    }
}
```

```

(connector as Connector).Style = new ShapeStyle() { StrokeColor = "#6d6d6d"
};
(connector as Connector).Constraints = 0;
(connector as Connector).CornerRadius = 5;
}
// Create the layout info
private TreeInfo GetLayoutInfo(IDiagramObject obj, TreeInfo options)
{
    // Enable the sub-tree.
    options.EnableSubTree = true;
    // Specify the subtree orientation.
    options.Orientation = Orientation.Horizontal;
    return options;
}
// Defines the node's default values.
private void NodeDefaults(IDiagramObject obj)
{
    Node node = obj as Node;
    if (node.Data is System.Text.Json.JsonElement)
    {
        node.Data =
            System.Text.Json.JsonSerializer.Deserialize<HierarchicalDetails>(node.Data.ToOString());
    }
    HierarchicalDetails hierarchicalData = node.Data as HierarchicalDetails;
    node.Style = new ShapeStyle() { Fill = "#659be5", StrokeColor = "none",
        StrokeWidth = 2, };
    node.BackgroundColor = "#659be5";
    node.Width = 150;
    node.Height = 50;
    node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
    {
        new ShapeAnnotation()
        {
            Content = hierarchicalData.Name,
            Style = new TextStyle() { Color = "white" }
        }
    };
}
// Create the hierarchical details with needed properties.
public class HierarchicalDetails
{
    public string Name { get; set; }
    public string FillColor { get; set; }
    public string Category { get; set; }
}
// Create the data source with node name and fill color values.
public List<HierarchicalDetails> DataSource = new
List<HierarchicalDetails>()
{
    new HierarchicalDetails() { Name = "Diagram",
        Category = "", FillColor = "#659be5" },
    new HierarchicalDetails() { Name = "Layout",
        Category = "Diagram", FillColor = "#659be5" },
    new HierarchicalDetails() { Name = "Tree layout",
        Category = "Layout", FillColor = "#659be5" },

```

```

new HierarchicalDetails(){ Name ="Organizational chart",
Category="Layout",FillColor="#659be5"},
new HierarchicalDetails(){ Name ="Hierarchical tree", Category="Tree
layout",FillColor="#659be5"},
new HierarchicalDetails(){ Name ="Radial tree", Category="Tree
layout",FillColor="#659be5"},
new HierarchicalDetails(){ Name ="Mind map", Category="Hierarchical
tree",FillColor="#659be5"},
new HierarchicalDetails(){ Name ="Family tree", Category="Hierarchical
tree",FillColor="#659be5"},
new HierarchicalDetails(){ Name ="Management", Category="Organizational
chart",FillColor="#659be5"},
new HierarchicalDetails(){ Name ="Human resources",
Category="Management",FillColor="#659be5"},
new HierarchicalDetails(){ Name ="University",
Category="Management",FillColor="#659be5"},
new HierarchicalDetails(){ Name ="Business",
Category="#Management",FillColor="#659be5"}
};
}

```

### JSON Data

Local JSON data can be bound to the Diagram 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 Diagram component,

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
@using Syncfusion.Blazor.Data
<SfDiagramComponent @ref="Diagram" Width="1000px" Height="500px"
NodeCreating="NodeDefaults" SetNodeTemplate="SetTemplate">
<DataSourceSettings ID="Name" ParentID="Category">
<SfDataManager Json="DataSource"
Adaptor="Syncfusion.Blazor.Adaptors.JsonAdaptor"></SfDataManager>
</DataSourceSettings>
<Layout HorizontalSpacing="40" VerticalSpacing="40"
Type="LayoutType.HierarchicalTree"></Layout>
</SfDiagramComponent>
@code
{
SfDiagramComponent Diagram;
float x = 100;
float y = 100;
Query Query = new Query().Select(new List<string>() { "EmployeeID",
"ReportsTo", "FirstName" }).Take(9);
// Create the hierarchical details with needed properties.
public class HierarchicalDetails
{
public string Name { get; set; }
public string FillColor { get; set; }
public string Category { get; set; }
}
// Create the data source with node name and fill color values.

```

```

public HierarchicalDetails[] DataSource = new HierarchicalDetails[]
{
    new HierarchicalDetails() { Name = "Diagram",
    Category="", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Layout",
    Category="Diagram", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Organizational chart",
    Category="Diagram", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Tree layout",
    Category="Layout", FillColor="#659be5"},
    new HierarchicalDetails() { Name = "Hierarchical tree", Category="Tree
    layout", FillColor="#659be5"},
};
// Defines the node's default values.
private void NodeDefaults(IDiagramObject obj)
{
    Node node = obj as Node;
    node.OffsetX = x;
    node.OffsetY = y;
    x += 100;
    HierarchicalDetails hierarchicalData = node.Data as HierarchicalDetails;
    node.Style.Fill = hierarchicalData.FillColor;
    node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
    {
        new ShapeAnnotation()
        {
            Content = hierarchicalData.Name
        }
    };
}
private ICommonElement SetTemplate(IDiagramObject node)
{
    return null;
}
}

```

### Remote Data

To bind remote data to [Diagram 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 diagram component. By default, SfDataManager uses ODataAdaptor for remote data-binding.

### 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.Diagram
@using Syncfusion.Blazor.Data
<div style="width:100%">
<div style="width:70%">

```



```

<SfDiagramComponent Height="400px"
InteractionController="@InteractionController.ZoomPan"
NodeCreating="OnNodeCreating" ConnectorCreating="OnConnectorCreating"
SetNodeTemplate="SetTemplate">
  <DataSourceSettings Id="EmployeeID" ParentId="ReportsTo">
    <SfDataManager
      Url="https://services.odata.org/V4/Northwind/Northwind.svc/Employees"
      Adaptor="Syncfusion.Blazor.Adaptors.ODataV4Adaptor"></SfDataManager>
    </DataSourceSettings>
    <SnapSettings Constraints="SnapConstraints.None"></SnapSettings>
    <Layout HorizontalSpacing="40" VerticalSpacing="40"
      Type="LayoutType.HierarchicalTree"></Layout>
    </SfDiagramComponent>
  </div>
</div>
@code
{
  SfDiagramComponent Diagram;
  private float x = 100;
  private float y = 100;
  public class Employee
  {
    public int? EmployeeID { get; set; }
    public string FirstName { get; set; }
    public int? ReportsTo { get; set; }
  }
  private Query Query = new Query().Select(new List<string>() { "EmployeeID",
    "ReportsTo", "FirstName" }).Take(9);
  private void OnNodeCreating(IDiagramObject obj)
  {
    Node node = obj as Node;
    node.OffsetX = x;
    node.OffsetY = y;
    node.Width = 80;
    node.Height = 40;
    node.Shape = new BasicShape() { Type =
    Syncfusion.Blazor.Diagram.Shapes.Basic, Shape = BasicShapeType.Rectangle,
    CornerRadius = 8 };
    node.Style = new ShapeStyle() { StrokeWidth = 0, Fill = "" };
    x += 100;
    Dictionary<string, object> data = node.Data as Dictionary<string, object>;
    node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
    {
      new ShapeAnnotation()
      {
        Content = data["FirstName"].ToString(),
        Style = new TextStyle() { Color = "white" }
      }
    };
    if (data["FirstName"].ToString() == "Andrew")
    {
      node.Style.Fill = "#3A4857";
    }
    else if (data["FirstName"].ToString() == "Nancy")
    {
      node.Style.Fill = "#2B8C68";
    }
  }
}

```

```
else if (data["FirstName"].ToString() == "Janet")
{
    node.Style.Fill = "#488CC1";
}
else if (data["FirstName"].ToString() == "Janet")
{
    node.Style.Fill = "#488CC1";
}
else if (data["FirstName"].ToString() == "Margaret")
{
    node.Style.Fill = "#4C888F";
}
else if (data["FirstName"].ToString() == "Steven")
{
    node.Style.Fill = "#8E4DB4";
}
else if (data["FirstName"].ToString() == "Laura")
{
    node.Style.Fill = "#CD6A32";
}
else
{
    node.Style.Fill = "#8E4DB4";
}
}
private void OnConnectorCreating(IDiagramObject obj)
{
    Connector connector = obj as Connector;
    connector.Style.StrokeColor = "#048785";
    connector.Type = ConnectorSegmentType.Orthogonal;
    connector.TargetDecorator.Shape = DecoratorShape.None;
    connector.SourceDecorator.Shape = DecoratorShape.None;
    connector.Style = new ShapeStyle() { StrokeColor = "#3A4857", Fill = "#3A4857", StrokeWidth = 1, StrokeDashArray = "3,3" };
}
private ICommonElement SetTemplate(IDiagramObject node)
{
    return null;
}
}
```

### See Also

- [How to arrange the diagram nodes and connectors using varies layout](#)

### Layout

#### Automatic Layout in Blazor Diagram Component

Diagram provides support to auto-arrange the nodes in the diagram area that is referred as [Layout](#). It includes the following layout modes:

#### *Layout modes*

- Hierarchical layout

- Organization chart
- Mind Map layout

See also

- [How to create a node](#)
- [How to create a connector](#)
- [How to generate the organization chart](#)
- [How to generate the hierarchical layout](#)

### Hierarchical layout in Blazor Diagram Component

The hierarchical tree layout arranges nodes in a tree-like structure, where the nodes in the hierarchical layout may have multiple parents. There is no need to specify the layout root. To arrange the nodes in a hierarchical structure, specify the layout [Type](#) as [HierarchicalTree](#). The following example shows how to arrange the nodes in a hierarchical structure.

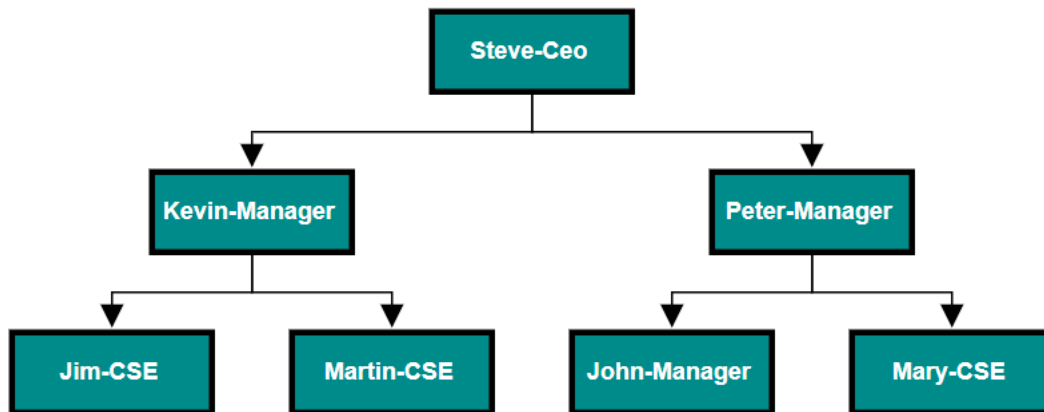
#### CSHARP

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes" Connectors="@connectors"
NodeCreating="@OnNodeCreating" ConnectorCreating="@OnConnectorCreating">
<Layout Type="LayoutType.HierarchicalTree" @bind-
HorizontalSpacing="@HorizontalSpacing" @bind-
VerticalSpacing="@VerticalSpacing">
</Layout>
<SnapSettings>
<HorizontalGridLines LineColor="white" LineDashArray="2,2">
</HorizontalGridLines>
<VerticalGridLines LineColor="white" LineDashArray="2,2">
</VerticalGridLines>
</SnapSettings>
</SfDiagramComponent>
@code
{
    int left = 40;
    int top = 50;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    int HorizontalSpacing = 40;
    int VerticalSpacing = 40;
    private void OnNodeCreating(IDiagramObject obj)
    {
        Node node = obj as Node;
        node.Height = 40;
        node.Width = 100;
        //Initializing the default node's shape style
        node.Style = new ShapeStyle() { Fill = "darkcyan", StrokeWidth = 3,
        StrokeColor = "Black" };
        node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
        {
            new ShapeAnnotation
            {
                Content = node.Annotations[0].Content,
```

```

Style = new TextStyle() { Color = "white", Bold = true },
};
}
private void OnConnectorCreating(IDiagramObject connector)
{
    (connector as Connector).Type = ConnectorSegmentType.Orthogonal;
}
protected override void OnInitialized()
{
    //Initializing node and connectors
    nodes = new DiagramObjectCollection<Node>()
    {
        new Node() { ID="node1", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new
        ShapeAnnotation{Content="Steve-Ceo"} } },
        new Node() { ID="node2", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new
        ShapeAnnotation{Content="Kevin-Manager"} } },
        new Node() { ID="node3", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new
        ShapeAnnotation{Content="Peter-Manager"} } },
        new Node() { ID="node4", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new
        ShapeAnnotation{Content="Jim-CSE"} } },
        new Node() { ID="node5", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new
        ShapeAnnotation{Content="Martin-CSE"} } },
        new Node() { ID="node6", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new
        ShapeAnnotation{Content="John-Manager"} } },
        new Node() { ID="node7", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new
        ShapeAnnotation{Content="Mary-CSE"} } },
    };
    connectors = new DiagramObjectCollection<Connector>()
    {
        new Connector() { ID="connector1", SourceID="node1", TargetID="node2" },
        new Connector() { ID="connector2", SourceID="node1", TargetID="node3" },
        new Connector() { ID="connector3", SourceID="node2", TargetID="node4" },
        new Connector() { ID="connector4", SourceID="node2", TargetID="node5" },
        new Connector() { ID="connector5", SourceID="node3", TargetID="node6" },
        new Connector() { ID="connector6", SourceID="node3", TargetID="node7" },
    };
}
}

```



### Customizing the properties

#### Orientation

You can use [Orientation](#) property of the Layout to change the orientation at runtime. The following code is used to how to change the layout.

#### CSHARP

```

<SfDiagramComponent Height="600px" Width="500px" >
<Layout Type="LayoutType.HierarchicalTree" @bind-
Orientation="@orientation"></Layout>
</SfDiagramComponent>
// Initializing the orientation value
LayoutOrientation orientation = LayoutOrientation.TopToBottom;
public void UpdateOrientation()
{
// Update LayoutOrientation in runtime
orientation = LayoutOrientation.BottomToTop;
}
  
```

#### Spacing

You can change the horizontal and vertical spacing for the diagram layout by using [HorizontalSpacing](#) and [VerticalSpacing](#) properties of the layout.

#### CSHARP

```

<SfDiagramComponent @ref="diagram" Width="900px" Height="800px">
<Layout Type="LayoutType.HierarchicalTree" @bind-
HorizontalSpacing="@HorizontalSpacing" @bind-
VerticalSpacing="@VerticalSpacing"/>
</SfDiagramComponent>
// Initializing the Horizontal and Vertical value
int HorizontalSpacing = 40;
int VerticalSpacing = 50;
// Update the spacing
  
```

```
public void UpdateSpacing()
{
    Diagram.BeginUpdate();
    HorizontalSpacing += 10;
    VerticalSpacing += 10;
    Diagram.EndUpdate();
}
```

### Margin

You can also change the margin values for the diagram layout by using [Margin](#) property.

### CSHARP

```
<SfDiagramComponent @ref="diagram" Width="900px" Height="800px" >
<Layout Type="LayoutType.HierarchicalTree">
<LayoutMargin Top="@top" Left="@left"></LayoutMargin>
</Layout>
</SfDiagramComponent>
// Initializing the Margin Top and Left value
int left = 40;
int top = 50;
// Update the margin values
public void UpdateMargin()
{
    Diagram.BeginUpdate();
    left += 10;
    top += 10;
    Diagram.EndUpdate();
}
```

### See also

- [How to create a node](#)
- [How to create a connector](#)

### Organizational Chart in Blazor Diagram Component

An organizational chart is a diagram that displays the structure of an organization and relationships. To create an organizational chart, the [Type](#) of layout should be set as an [OrganizationalChart](#). The following code example illustrates how to create an organizational chart.

### CSHARP

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@NodeCollection"
Connectors="@connectors" NodeCreating="@OnNodeCreating"
ConnectorCreating="@OnConnectorCreating">
<Layout Type="LayoutType.OrganizationalChart" @bind-
HorizontalSpacing="@HorizontalSpacing" @bind-
VerticalSpacing="@VerticalSpacing">
</Layout>
</SfDiagramComponent>
@code
{
    //Defines diagram's connector collection
```

```

DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
//Defines diagram's node collection
DiagramObjectCollection<Node> NodeCollection = new
DiagramObjectCollection<Node>();
int HorizontalSpacing = 40;
int VerticalSpacing = 40;
private void OnNodeCreating(IDiagramObject obj)
{
    Node node = obj as Node;
    node.Height = 40;
    node.Width = 100;
    //Initializing the default node's shape style
    node.Style = new ShapeStyle() { Fill = "darkcyan", StrokeWidth = 3,
    StrokeColor = "Black" };
    node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
    {
        new ShapeAnnotation { Style = new TextStyle() { Color = "white", Bold = true
        }, Content = node.Annotations[0].Content }
    };
}
private void OnConnectorCreating(IDiagramObject connector)
{
    (connector as Connector).Type = ConnectorSegmentType.Orthogonal;
}
protected override void OnInitialized()
{
    NodeCollection = new DiagramObjectCollection<Node>()
    {
        new Node() { ID = "node1", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
        "Project Management" } } },
        new Node() { ID = "node2", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
        "R&D Team" } } },
        new Node() { ID = "node3", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
        "Philosophy" } } },
        new Node() { ID = "node4", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
        "Organization" } } },
        new Node() { ID = "node5", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
        "Technology" } } },
        new Node() { ID = "node6", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
        "Funding" } } },
        new Node() { ID = "node7", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
        "Resource-Allocation" } } },
        new Node() { ID = "node8", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
        "Targeting" } } },
        new Node() { ID = "node9", Annotations = new
        DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
        "Evaluation" } } },
    }
}

```

```

new Node() { ID = "node10",Annotations = new
DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
"HR-Team" } } },
new Node() { ID = "node11",Annotations = new
DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
"Recruitment" } } },
new Node() { ID = "node12",Annotations = new
DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
"Employee-Relation" } } },
new Node() { ID = "node13",Annotations = new
DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
"Production & Sales Team" } } },
new Node() { ID = "node14",Annotations = new
DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
"Design" } } },
new Node() { ID = "node15",Annotations = new
DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
"Operation" } } },
new Node() { ID = "node16",Annotations = new
DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
"Support" } } },
new Node() { ID = "node17",Annotations = new
DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
"Quality Assurance" } } },
new Node() { ID = "node18",Annotations = new
DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
"Customer Interaction" } } },
new Node() { ID = "node19",Annotations = new
DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
"Support and Maintenance" } } },
new Node() { ID = "node20",Annotations = new
DiagramObjectCollection<ShapeAnnotation>() { new ShapeAnnotation { Content =
"Task Coordination" } } }
};
connectors = new DiagramObjectCollection<Connector>()
{
new Connector() { ID = "connector1", SourceID = "node1", TargetID = "node2"
},
new Connector() { ID = "connector2", SourceID = "node1", TargetID = "node10"
},
new Connector() { ID = "connector3", SourceID = "node1", TargetID = "node13"
},
new Connector() { ID = "connector4", SourceID = "node2", TargetID = "node3"
},
new Connector() { ID = "connector5", SourceID = "node2", TargetID = "node4"
},
new Connector() { ID = "connector6", SourceID = "node2", TargetID = "node5"
},
new Connector() { ID = "connector7", SourceID = "node2", TargetID = "node6"
},
new Connector() { ID = "connector8", SourceID = "node2", TargetID = "node7"
},
new Connector() { ID = "connector9", SourceID = "node2", TargetID = "node8"
},
new Connector() { ID = "connector10", SourceID = "node2", TargetID = "node9"
},
},

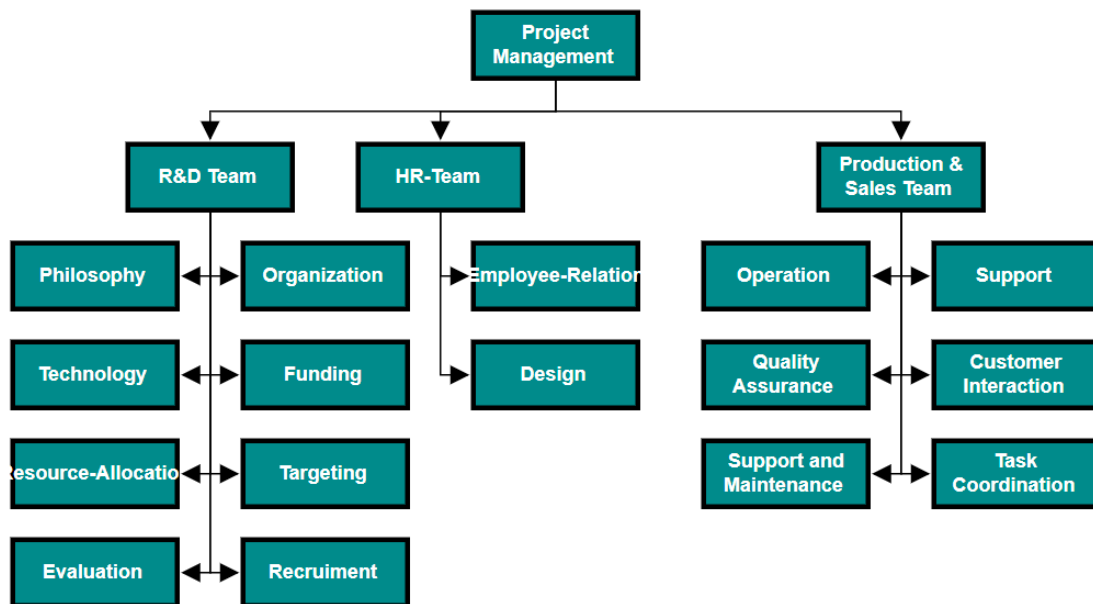
```



```

new Connector() { ID = "connector11", SourceID = "node2", TargetID = "node11" },
new Connector() { ID = "connector12", SourceID = "node10", TargetID = "node12" },
new Connector() { ID = "connector13", SourceID = "node10", TargetID = "node14" },
new Connector() { ID = "connector14", SourceID = "node13", TargetID = "node15" },
new Connector() { ID = "connector15", SourceID = "node13", TargetID = "node16" },
new Connector() { ID = "connector16", SourceID = "node13", TargetID = "node17" },
new Connector() { ID = "connector17", SourceID = "node13", TargetID = "node18" },
new Connector() { ID = "connector18", SourceID = "node13", TargetID = "node19" },
new Connector() { ID = "connector19", SourceID = "node13", TargetID = "node20" }
};
}
}

```



Organizational chart layout starts parsing from root and iterate through all its child elements. The [GetLayoutInfo](#) event callback method provides necessary information of a node's children and the way to arrange (direction, orientation, offsets, etc.) them. The arrangements can be customized by overriding this function as explained.

**GetLayoutInfo** set chart orientations, chart types, and offset to be left between parent and child nodes. The GetLayoutInfo event callback method is called to configure every subtree of the organizational chart. It takes the following arguments.

1. **IDiagramObject**: Parent node to that options are to be customized.
2. **TreeInfo**: Object to set the customizable properties.
3. **TreeInfo**: Returns an object value to be the customized.

### Customize layout

Orientation, spacings, and position of the layout can be customized with a set of properties.

To explore layout properties, refer to [Layout Properties](#).

### Layout bounds

Diagram provides support to align the layout within any custom rectangular area. For more information about bounds, refer to [Bounds](#).

### Layout alignment

The layout can be aligned anywhere over the layout bounds/viewport using the [HorizontalAlignment](#) and [VerticalAlignment](#) properties of the layout.

The following code illustrates how to align the layout at the top-left of the layout bounds.

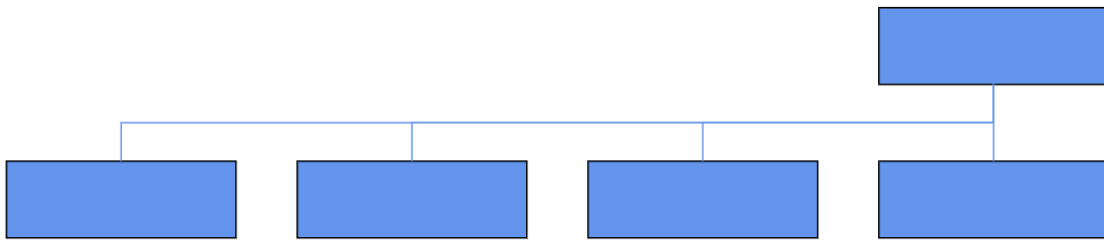
### CSHARP

```
<SfDiagramComponent @ref="diagram" Width="900px" Height="800px">
<Layout Type="LayoutType.OrganizationalChart" @bind-
HorizontalSpacing="@HorizontalSpacing" @bind-
VerticalSpacing="@VerticalSpacing" @bind-
VerticalAlignment="@verticalAlignment"></Layout>
</SfDiagramComponent>
int HorizontalSpacing = 40;
int VerticalSpacing = 40;
VerticalAlignment verticalAlignment = VerticalAlignment.Bottom;
```

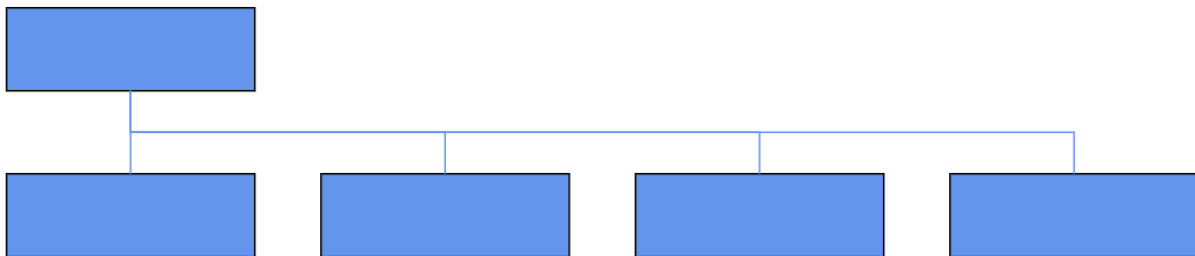
The following table illustrates the different chart orientations and chart types.

Orientation	Type	Description	Example
-----	-----	-----	-----

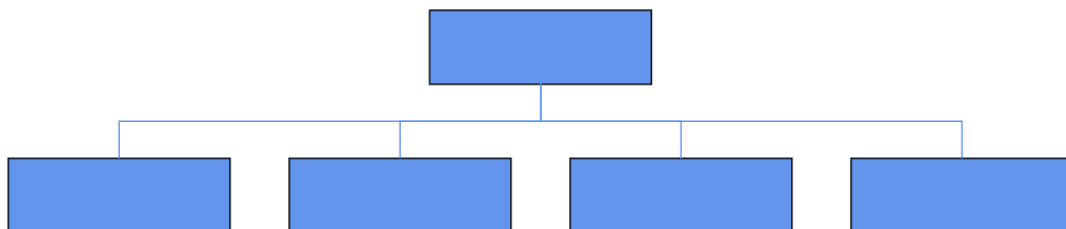
|Horizontal|Left|Arranges the child nodes horizontally at the left side of the parent.|



|  
||Right|Arranges the child nodes horizontally at the right side of the parent.|

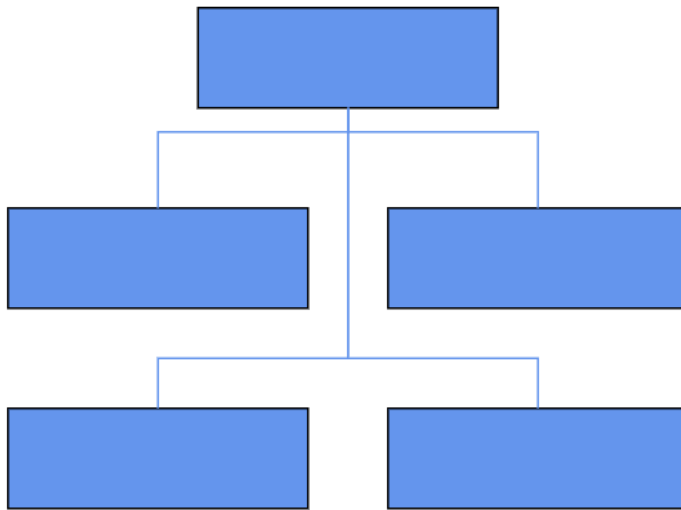


|  
||Center|Arranges the children like standard tree layout orientation.|

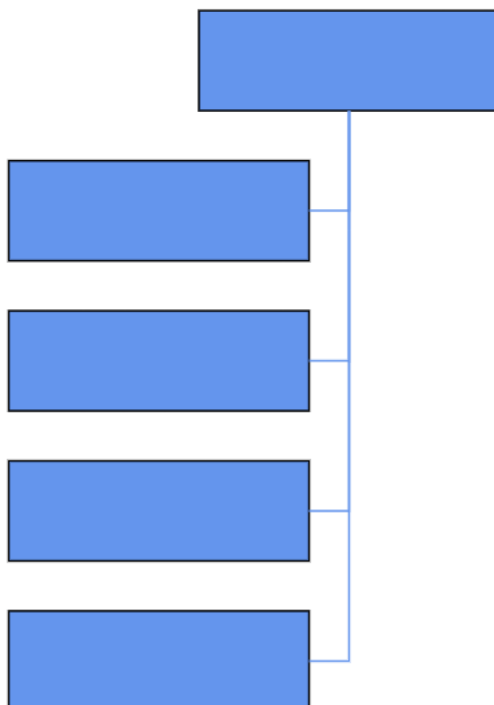


|

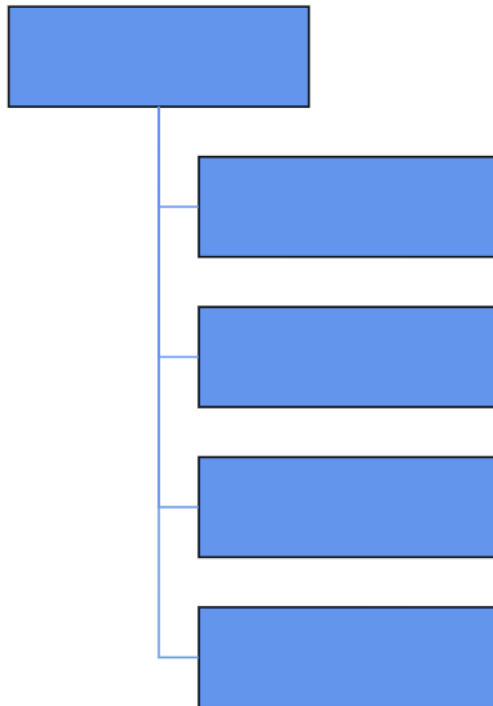
| |Balanced|Arranges the leaf level child nodes in multiple rows.|



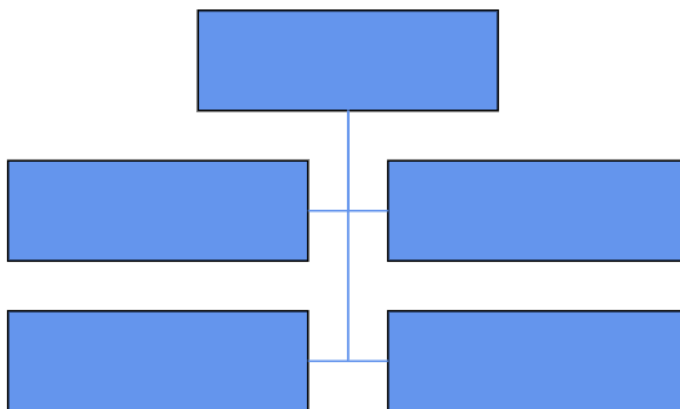
|Vertical|Left|Arranges the children vertically at the left side of the parent.|



||Right|Arranges the children vertically at the right side of the parent.



||Alternate|Arranges the children vertically at both left and right sides of the parent.



The following code example illustrates how to set the vertical right arrangement to the leaf level trees.

#### **CSHARP**

```
@using Syncfusion.Blazor.Inputs  
@using Syncfusion.Blazor.Diagram
```

```

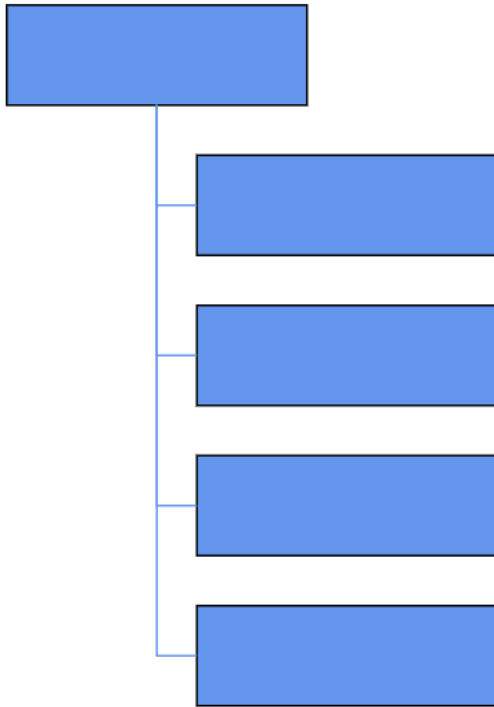
<SfDiagramComponent Height="600px" NodeCreating="@OnNodeCreating"
ConnectorCreating="@OnConnectorCreating">
  <DataSourceSettings ID="Id" ParentID="Team"
DataSource="@DataSource"></DataSourceSettings>
  <SnapSettings>
    <HorizontalGridLines LineColor="white" LineDashArray="2,2">
    </HorizontalGridLines>
    <VerticalGridLines LineColor="white" LineDashArray="2,2">
    </VerticalGridLines>
  </SnapSettings>
  <Layout Type="LayoutType.OrganizationalChart" @bind-
HorizontalSpacing="@HorizontalSpacing" @bind-
VerticalSpacing="@VerticalSpacing" GetLayoutInfo="GetLayoutInfo">
  </Layout>
</SfDiagramComponent>
@code
{
  //Initializing layout
  int HorizontalSpacing = 40;
  int VerticalSpacing = 50;
  //To configure every subtree of the organizational chart
  private TreeInfo GetLayoutInfo(IDiagramObject obj, TreeInfo options)
  {
    options.AlignmentType = SubTreeAlignmentType.Right;
    options.Orientation = Orientation.Vertical;
    return options;
  }
  //Creates node with some default values
  private void OnNodeCreating(IDiagramObject obj)
  {
    Node node = obj as Node;
    node.Height = 50;
    node.Width = 150;
    node.Style = new ShapeStyle() { Fill = "#6495ED", StrokeWidth = 1,
StrokeColor = "Black" };
  }
  //Creates connectors with some default values
  private void OnConnectorCreating(IDiagramObject connector)
  {
    Connector connectors = connector as Connector;
    connectors.Type = ConnectorSegmentType.Orthogonal;
    connectors.Style = new TextStyle() { StrokeColor = "#6495ED", StrokeWidth =
1 };
    connectors.TargetDecorator = new DecoratorSettings
    {
      Shape = DecoratorShape.None,
      Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED", }
    };
  }
  public class OrgChartDataModel
  {
    public string Id { get; set; }
    public string Team { get; set; }
    public string Role { get; set; }
  }
  public object DataSource = new List<object>()
  {

```

```

new OrgChartDataModel() { Id= "1", Role= "General Manager" },
new OrgChartDataModel() { Id= "2", Role= "Human Resource Manager", Team= "1"
},
new OrgChartDataModel() { Id= "3", Role= "Design Manager", Team= "1" },
new OrgChartDataModel() { Id= "4", Role= "Operation Manager", Team= "1" },
new OrgChartDataModel() { Id= "5", Role= "Marketing Manager", Team= "1" }
};
}

```



#### Layout spacing

Layout provides support to add space horizontally and vertically between the nodes. The [HorizontalSpacing](#) and [VerticalSpacing](#) properties of the layout allows you to set the space between the nodes in horizontally and vertically.

#### Layout margin

Layout provides support to add some blank space between the layout bounds/viewport and the layout. The [Margin](#) property of the layout allows you to set the blank space.

#### CSHARP

```

//Initialize the organizational chart layout with Margin
<SfDiagramComponent @ref="diagram" Width="900px" Height="800px" >
<Layout Type="LayoutType.HierarchicalTree">
<LayoutMargin Top="@top" Left="@left"></LayoutMargin>
</Layout>

```

```

</SfDiagramComponent>
@code
{
    //Initializing the Mergin Top and Left value
    int left = 40;
    int top = 50;
}

```

### Layout orientation

Diagram provides support to customize the **Orientation** of layout. You can set the desired orientation using **LayoutOrientation**.

The following code illustrates how to arrange the nodes in a BottomToTop orientation.

### CSHARP

```

//Initialize the layout with layout orientation as BottomToTop in page
<SfDiagramComponent Height="600px" Width="500px" >
<Layout Type="LayoutType.HierarchicalTree" @bind-
Orientation="@orientation"></Layout>
</SfDiagramComponent>
@code
{
    //Initializing the orientation value
    LayoutOrientation orientation = LayoutOrientation.TopToBottom;
}

```

### Fixed node

Layout provides support to arrange the nodes with reference to the position of a fixed node and set it to the **FixedNode** of the layout property. This is helpful when you try to expand/collapse a node. It might be expected that the position of the double-clicked node should not be changed.

### CSHARP

```

//Initialize the organizational chart layout with FixedNode
<SfDiagramComponent Height="600px" Width="500px" >
<Layout Type="LayoutType.OrganizationalChart" FixedNode="Node1" @bind-
HorizontalSpacing="@HorizontalSpacing" @bind-
VerticalSpacing="@VerticalSpacing" @bind-
Orientation="@orientation"></Layout>
</SfDiagramComponent>
@code
{
    //Initializing the orientation value
    LayoutOrientation orientation = LayoutOrientation.TopToBottom;
    //Initializing the Horizontal and Vertical value
    int HorizontalSpacing = 40;
    int VerticalSpacing = 50;
}

```

### Refresh layout

Diagram allows to refresh the layout at runtime by using **DoLayout** method. Use the below code example to refresh the layout.



**CSHARP**

```
//update the layout at runtime.
diagram.DoLayout();
//Here, diagram is instance of SfDiagramComponent.
```

See also

- [How to create a node](#)
- [How to create a connector](#)

### Mind map Layout in Blazor Diagram Component

A mind map is a diagram that displays the nodes as a spider diagram organizes information around a central concept. To create mind map, the [Type](#) of layout should be set as [MindMap](#). The following code example illustrates how to create a mind map layout.

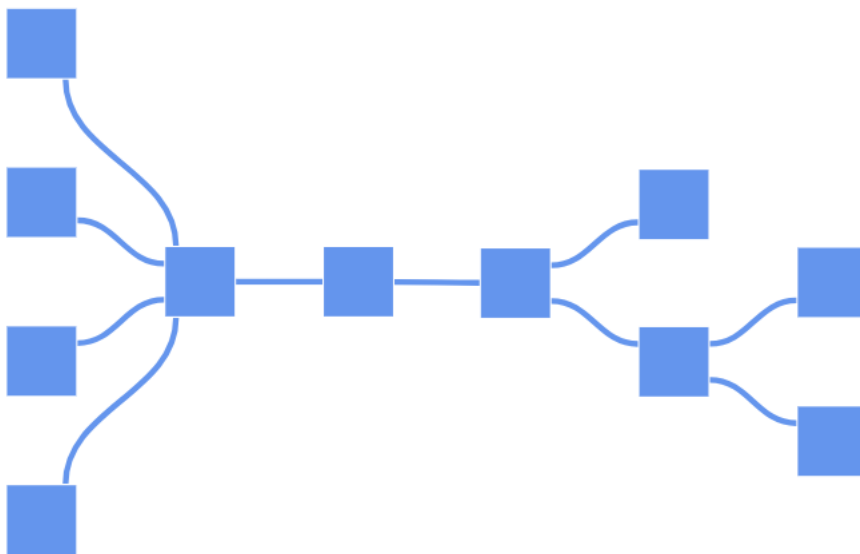
**CSHARP**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" NodeCreating="@OnNodeCreating"
ConnectorCreating="@OnConnectorCreating">
<DataSourceSettings ID="Id" ParentID="ParentId"
DataSource="@DataSource"></DataSourceSettings>
<Layout Type="LayoutType.MindMap">
<LayoutMargin Top="20" Left="20"></LayoutMargin>
</Layout>
</SfDiagramComponent>
@code
{
    //Creates nodes with some default values
    private void OnNodeCreating(IDiagramObject obj)
    {
        Node node = obj as Node;
        node.Height = 25;
        node.Width = 25;
        node.BackgroundColor = "#6BA5D7";
        node.Style = new ShapeStyle() { Fill = "#6495ED", StrokeWidth = 1,
        StrokeColor = "white" };
        node.Shape = new BasicShape() { Type = Shapes.Basic };
    }
    //Creates connectors with some default values
    private void OnConnectorCreating(IDiagramObject connector)
    {
        Connector connectors = connector as Connector;
        connectors.Type = ConnectorSegmentType.Bezier;
        connectors.Style = new ShapeStyle() { StrokeColor = "#6495ED", StrokeWidth =
        2 };
        connectors.TargetDecorator = new DecoratorSettings
        {
            Shape = DecoratorShape.None,
        };
    }
    public class MindMapDetails
    {
        public string Id { get; set; }
    }
```

```

public string Label { get; set; }
public string ParentId { get; set; }
public string Branch { get; set; }
public string Fill { get; set; }
}
public object DataSource = new List<object>()
{
    new MindMapDetails() { Id= "1",Label="Creativity", ParentId="", Branch =
    "Root"},
    new MindMapDetails() { Id= "2", Label="Brainstorming", ParentId ="1",
    Branch = "Right" },
    new MindMapDetails() { Id= "3", Label="Complementing", ParentId ="1",
    Branch = "Left" },
    new MindMapDetails() { Id= "4", Label="Sessions", ParentId ="2", Branch =
    "subRight" },
    new MindMapDetails() { Id= "5", Label="Complementing", ParentId ="2",
    Branch = "subRight" },
    new MindMapDetails() { Id= "6", Label= "Local", ParentId ="3", Branch =
    "subRight" },
    new MindMapDetails() { Id= "7", Label= "Remote", ParentId ="3", Branch =
    "subRight" },
    new MindMapDetails() { Id= "8", Label= "Individual", ParentId ="3", Branch =
    "subRight" },
    new MindMapDetails() { Id= "9", Label= "Teams", ParentId ="3", Branch =
    "subRight" },
    new MindMapDetails() { Id= "10", Label= "Ideas", ParentId ="5", Branch =
    "subRight" },
    new MindMapDetails() { Id= "11", Label= "Engagement", ParentId ="5", Branch
    = "subRight" },
};
}

```



You can also decide the branch for mind map using [GetBranch](#) method. The following code demonstrates how to set all branches on the right side for mind map layout using [GetBranch](#) method.

**CSHARP**

```

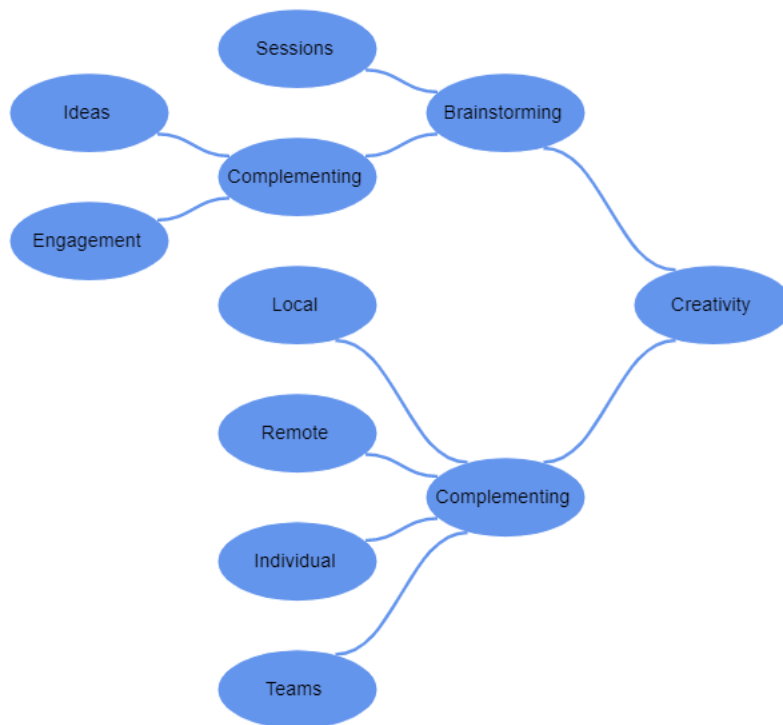
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" NodeCreating="@OnNodeCreating"
ConnectorCreating="@OnConnectorCreating">
  <DataSourceSettings ID="Id" ParentID="ParentId"
DataSource="@DataSource"></DataSourceSettings>
  <Layout Type="LayoutType.MindMap" GetBranch="@GetBranch">
  <LayoutMargin Top="20" Left="20"></LayoutMargin>
</Layout>
</SfDiagramComponent>
@code
{
  //Set all branches on the right side for mind map layout
  private BranchType GetBranch(IDiagramObject obj)
  {
    if ((obj as Node).ID == "1")
    {
      return BranchType.Root;
    }
    return BranchType.Right;
  }
  //Creates connectors with some default values
  private void OnNodeCreating(IDiagramObject obj)
  {
    Node node = obj as Node;
    node.Height = 50;
    node.Width = 100;
    node.Style = new ShapeStyle() { Fill = "#6495ED", StrokeWidth = 1,
    StrokeColor = "white" };
    node.Shape = new BasicShape() { Type = Shapes.Basic, Shape =
    BasicShapeType.Ellipse };
    MindMapDetails mindmapData = node.Data as MindMapDetails;
    node.Annotations = new DiagramObjectCollection<ShapeAnnotation>()
    {
      new ShapeAnnotation()
      {
        Content = mindmapData.Label
      }
    };
  }
  //Creates node with some default values
  private void OnConnectorCreating(IDiagramObject connector)
  {
    Connector connectors = connector as Connector;
    connectors.Type = ConnectorSegmentType.Bezier;
    connectors.Style = new TextStyle() { StrokeColor = "#6495ED", StrokeWidth =
    2 };
    connectors.TargetDecorator = new DecoratorSettings
    {
      Shape = DecoratorShape.None,
    };
  }
  public class MindMapDetails
  {
    public string Id { get; set; }
    public string Label { get; set; }
  }
}

```

```

public string ParentId { get; set; }
public string Branch { get; set; }
public string Fill { get; set; }
}
public object DataSource = new List<object>()
{
    new MindMapDetails() { Id= "1",Label="Creativity", ParentId =""},
    new MindMapDetails() { Id= "2", Label="Brainstorming", ParentId ="1"},
    new MindMapDetails() { Id= "3", Label="Complementing", ParentId ="1"},
    new MindMapDetails() { Id= "4", Label="Sessions", ParentId ="2"},
    new MindMapDetails() { Id= "5", Label="Complementing", ParentId ="2"},
    new MindMapDetails() { Id= "6", Label= "Local", ParentId ="3"},
    new MindMapDetails() { Id= "7", Label= "Remote", ParentId ="3"},
    new MindMapDetails() { Id= "8", Label= "Individual", ParentId ="3"},
    new MindMapDetails() { Id= "9", Label= "Teams", ParentId ="3"},
    new MindMapDetails() { Id= "10", Label= "Ideas", ParentId ="5"},
    new MindMapDetails() { Id= "11", Label= "Engagement", ParentId ="5"},
};
}

```



See also

- [How to create a node](#)
- [How to create a connector](#)

## Commands in Blazor Diagram Component

There are several commands available in the diagram as follows.

- Alignment commands
- Distribute commands
- Sizing commands
- Clipboard commands
- Grouping commands
- Zoom commands
- Undo/Redo commands

### Alignment commands

Alignment commands enable you to align the selected or defined objects such as nodes and connectors with respect to the selection boundary. Following are the [AlignmentOptions](#) in [SetAlign](#) commands which shows how to use align methods in the diagram.

#### Align Left

The following code example illustrates how to align all the selected objects at the left side of the selection boundary.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<input type="button" value="Left" @onclick="@OnAlignLeft" />
<SfDiagramComponent @ref="diagram" Width="1000px" Height="500px"
Nodes="@nodes" Connectors="@Connectors"/>
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    DiagramObjectCollection<Connector> Connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Node node1 = new Node()
        {
            ID = "node1",
            Width = 50,
            Height = 30,
            OffsetX = 500,
            OffsetY = 100,
            Shape = new BasicShape() { Type = Shapes.Basic, Shape =
            BasicShapeType.Rectangle }
        };
        nodes.Add(node1);
        Node node2 = new Node()
        {
            ID = "node2",
            Width = 60,
            Height = 40,
            OffsetX = 500,
            OffsetY = 300,
            Shape = new BasicShape() { Type = Shapes.Basic, Shape =
            BasicShapeType.Rectangle }
        };
        nodes.Add(node2);
        Node node3 = new Node()
        {
```

```

ID = "node3",
Width = 70,
Height = 50,
OffsetX = 500,
OffsetY = 500,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle }
};
nodes.Add(node3);
}
private void OnAlignLeft()
{
//Aligns the selected objects at the left side of the selection boundary
diagram.SetAlign(AlignmentOptions.Left);
}
}

```

### *Align Right*

The following code example illustrates how to align all the selected objects at the right side of the selection boundary.

### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="Right" @onclick="@OnAlignRight" />
<SfDiagramComponent @ref="diagram" Width="1000px" Height="500px"
Nodes="@nodes" Connectors="@Connectors"/>
@code
{
SfDiagramComponent diagram;
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
DiagramObjectCollection<Connector> Connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
Node node1 = new Node()
{
ID = "node1",
Width = 50,
Height = 30,
OffsetX = 500,
OffsetY = 100,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle }
};
nodes.Add(node1);
Node node2 = new Node()
{
ID = "node2",
Width = 60,
Height = 40,
OffsetX = 500,
OffsetY = 300,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle }
};
}

```

```

nodes.Add(node2);
Node node3 = new Node()
{
    ID = "node3",
    Width = 70,
    Height = 50,
    OffsetX = 500,
    OffsetY = 500,
    Shape = new BasicShape() { Type = Shapes.Basic, Shape =
    BasicShapeType.Rectangle }
};
nodes.Add(node3);
}
private void OnAlignRight()
{
    //Aligns the selected objects at the right side of the selection boundary
    diagram.SetAlign(AlignmentOptions.Right);
}
}

```

### *Align Top*

The following code example illustrates how to align all the selected objects at the top of the selection boundary.

### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="Top" @onclick="@OnAlignTop" />
<SfDiagramComponent @ref="diagram" Width="1000px" Height="500px"
Nodes="@nodes" Connectors="@Connectors"/>
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    DiagramObjectCollection<Connector> Connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Node node1 = new Node()
        {
            ID = "node1",
            Width = 50,
            Height = 30,
            OffsetX = 500,
            OffsetY = 100,
            Shape = new BasicShape() { Type = Shapes.Basic, Shape =
            BasicShapeType.Rectangle }
        };
        nodes.Add(node1);
        Node node2 = new Node()
        {
            ID = "node2",
            Width = 60,
            Height = 40,
            OffsetX = 500,
            OffsetY = 300,

```

```

Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle }
};
nodes.Add(node2);
Node node3 = new Node()
{
ID = "node3",
Width = 70,
Height = 50,
OffsetX = 500,
OffsetY = 500,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle }
};
nodes.Add(node3);
}
private void OnAlignTop()
{
//Aligns the selected objects at the top of the selection boundary
diagram.SetAlign(AlignmentOptions.Top);
}
}

```

### *Align Bottom*

The following code example illustrates how to align all the selected objects at the bottom of the selection boundary.

### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="Bottom" @onclick="@OnAlignBottom" />
<SfDiagramComponent @ref="diagram" Width="1000px" Height="500px"
Nodes="@nodes" Connectors="@Connectors"/>
@code
{
SfDiagramComponent diagram;
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
DiagramObjectCollection<Connector> Connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
Node node1 = new Node()
{
ID = "node1",
Width = 50,
Height = 30,
OffsetX = 500,
OffsetY = 100,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle }
};
nodes.Add(node1);
Node node2 = new Node()
{
ID = "node2",
Width = 60,

```



```

Height = 40,
OffsetX = 500,
OffsetY = 300,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle }
};
nodes.Add(node2);
Node node3 = new Node()
{
ID = "node3",
Width = 70,
Height = 50,
OffsetX = 500,
OffsetY = 500,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle }
};
nodes.Add(node3);
}
private void OnAlignBottom()
{
//Aligns the selected objects at the bottom of the selection boundary
diagram.SetAlign(AlignmentOptions.Bottom);
}
}

```

### *Align Middle*

The following code example illustrates how to align all the selected objects at the middle of the selection boundary.

### **ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="Middle" @onclick="@OnAlignMiddle" />
<SfDiagramComponent @ref="diagram" Width="1000px" Height="500px"
Nodes="@nodes" Connectors="@Connectors"/>
@code
{
SfDiagramComponent diagram;
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
DiagramObjectCollection<Connector> Connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
Node node1 = new Node()
{
ID = "node1",
Width = 50,
Height = 30,
OffsetX = 500,
OffsetY = 100,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle }
};
nodes.Add(node1);
Node node2 = new Node()

```

```

{
    ID = "node2",
    Width = 60,
    Height = 40,
    OffsetX = 500,
    OffsetY = 300,
    Shape = new BasicShape() { Type = Shapes.Basic, Shape =
    BasicShapeType.Rectangle }
};
nodes.Add(node2);
Node node3 = new Node()
{
    ID = "node3",
    Width = 70,
    Height = 50,
    OffsetX = 500,
    OffsetY = 500,
    Shape = new BasicShape() { Type = Shapes.Basic, Shape =
    BasicShapeType.Rectangle }
};
nodes.Add(node3);
}
private void OnAlignMiddle()
{
    //Aligns the selected objects at the middle of the selection boundary
    diagram.SetAlign(AlignmentOptions.Middle);
}
}

```

### Align Center

The following code example illustrates how to align all the selected objects at the center of the selection boundary.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="Center" @onclick="@OnAlignCenter" />
<SfDiagramComponent @ref="diagram" Width="1000px" Height="500px"
Nodes="@nodes" Connectors="@Connectors"/>
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    DiagramObjectCollection<Connector> Connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Node node1 = new Node()
        {
            ID = "node1",
            Width = 50,
            Height = 30,
            OffsetX = 500,
            OffsetY = 100,
            Shape = new BasicShape() { Type = Shapes.Basic, Shape =
            BasicShapeType.Rectangle }
        }
    }
}

```

```
};
nodes.Add(node1);
Node node2 = new Node()
{
    ID = "node2",
    Width = 60,
    Height = 40,
    OffsetX = 500,
    OffsetY = 300,
    Shape = new BasicShape() { Type = Shapes.Basic, Shape =
    BasicShapeType.Rectangle }
};
nodes.Add(node2);
Node node3 = new Node()
{
    ID = "node3",
    Width = 70,
    Height = 50,
    OffsetX = 500,
    OffsetY = 500,
    Shape = new BasicShape() { Type = Shapes.Basic, Shape =
    BasicShapeType.Rectangle }
};
nodes.Add(node3);
}
private void OnAlignCenter()
{
    //Aligns the selected objects at the center of the selection boundary
    diagram.SetAlign(AlignmentOptions.Center);
}
}
```

### Distribute

The [SetDistribute](#) commands enable to place the selected objects on the page at equal intervals from each other. The selected objects are equally spaced within the selection boundary.

The factor to distribute the shapes [DistributeOptions](#) are listed as follows:

- RightToLeft: Distributes the objects based on the distance between the right and left sides of the adjacent objects.
- Left: Distributes the objects based on the distance between the left sides of the adjacent objects.
- Right: Distributes the objects based on the distance between the right sides of the adjacent objects.
- Center: Distributes the objects based on the distance between the center of the adjacent objects.
- BottomToTop: Distributes the objects based on the distance between the bottom and top sides of the adjacent objects.
- Top: Distributes the objects based on the distance between the top sides of the adjacent objects.
- Bottom: Distributes the objects based on the distance between the bottom sides of the adjacent objects.

- Middle: Distributes the objects based on the distance between the vertical center of the adjacent objects.

The following code example illustrates how to execute the space commands.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<input type="button" value="Left" @onclick="@OnDistributeLeft" />
<input type="button" value="Right" @onclick="@OnDistributeRight" />
<input type="button" value="Top" @onclick="@OnDistributeTop" />
<input type="button" value="Bottom" @onclick="@OnDistributeBottom" />
<input type="button" value="Center" @onclick="@OnDistributeCenter" />
<input type="button" value="Middle" @onclick="@OnDistributeMiddle" />
<input type="button" value="BottomToTop" @onclick="@OnDistributeBottomToTop" />
<input type="button" value="RightToLeft" @onclick="@OnDistributeRightToLeft" />
<SfDiagramComponent @ref="diagram" Width="1000px" Height="500px"
Nodes="@nodes" Connectors="@Connectors"/>
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    DiagramObjectCollection<Connector> Connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Node node1 = new Node()
        {
            ID = "node1",
            Width = 50,
            Height = 30,
            OffsetX = 500,
            OffsetY = 100,
            Shape = new BasicShape() { Type = Shapes.Basic, Shape =
            BasicShapeType.Rectangle }
        };
        nodes.Add(node1);
        Node node2 = new Node()
        {
            ID = "node2",
            Width = 60,
            Height = 40,
            OffsetX = 400,
            OffsetY = 200,
            Shape = new BasicShape() { Type = Shapes.Basic, Shape =
            BasicShapeType.Rectangle }
        };
        nodes.Add(node2);
        Node node3 = new Node()
        {
            ID = "node3",
            Width = 70,
            Height = 50,
            OffsetX = 500,
            OffsetY = 300,
```

```
Shape = new BasicShape() { Type = Shapes.Basic, Shape =  
BasicShapeType.Rectangle }  
};  
nodes.Add(node3);  
}  
private void OnDistributeLeft()  
{  
//distributes the objects based on the distance between the left sides of  
the adjacent objects.  
diagram.SetDistribute(DistributeOptions.Left);  
}  
private void OnDistributeRight()  
{  
//distributes the objects based on the distance between the right sides of  
the adjacent objects.  
diagram.SetDistribute(DistributeOptions.Right);  
}  
private void OnDistributeTop()  
{  
//distributes the objects based on the distance between the top sides of the  
adjacent objects.  
diagram.SetDistribute(DistributeOptions.Top);  
}  
private void OnDistributeBottom()  
{  
//distributes the objects based on the distance between the bottom sides of  
the adjacent objects.  
diagram.SetDistribute(DistributeOptions.Bottom);  
}  
private void OnDistributeMiddle()  
{  
//distributes the objects based on the distance between vertical centers of  
the adjacent objects.  
diagram.SetDistribute(DistributeOptions.Middle);  
}  
private void OnDistributeCenter()  
{  
//distributes the objects based on the distance between the centers of the  
adjacent objects.  
diagram.SetDistribute(DistributeOptions.Center);  
}  
private void OnDistributeBottomToTop()  
{  
//distributes the objects based on the distance between bottom and top sides  
of adjacent objects.  
diagram.SetDistribute(DistributeOptions.BottomToTop);  
}  
private void OnDistributeRightToLeft()  
{  
//distributes the objects based on the distance between right and left sides  
of adjacent objects.  
diagram.SetDistribute(DistributeOptions.RightToLeft);  
}  
}
```

### Sizing Commands

Sizing [SetSameSize](#) commands are used to resize all selected object based on width, height, and size of the reference object (FirstSelectedItem).

[SizingMode](#) are as follows:

- [Width](#) : Scales the width of the selected objects.
- [Height](#) : Scales the height of the selected objects.
- [Size](#) : Scales the selected objects both vertically and horizontally.

The following code example illustrates how to execute the size commands.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<input type="button" value="SameSize" @onclick="@OnSameSize" />
<input type="button" value="SameWidth" @onclick="@OnSameWidth" />
<input type="button" value="SameHeight" @onclick="@OnSameHeight" />
<SfDiagramComponent @ref="@diagram" Height="600px" Nodes="@nodes"
Connectors="@connectors" />
@code
{
    //Reference to diagram
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Node node1 = new Node()
        {
            ID = "node1",
            Width = 50,
            Height = 30,
            OffsetX = 500,
            OffsetY = 100,
            Shape = new BasicShape() { Type = Shapes.Basic, Shape =
            BasicShapeType.Rectangle },
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" }
        };
        nodes.Add(node1);
        Node node2 = new Node()
        {
            ID = "node2",
            Width = 60,
            Height = 40,
            OffsetX = 400,
            OffsetY = 200,
            Shape = new BasicShape() { Type = Shapes.Basic, Shape =
            BasicShapeType.Rectangle },
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" }
        };
        nodes.Add(node2);
        Node node3 = new Node()
        {
            ID = "node3",
```

```

Width = 70,
Height = 50,
OffsetX = 500,
OffsetY = 300,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle },
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" }
};
nodes.Add(node3);
}
private void OnSameSize()
{
//Scales the selected objects both vertically and horizontally.
diagram.SetSameSize(SizingMode.Size);
}
private void OnSameWidth()
{
//Scales the width of the selected objects.
diagram.SetSameSize(SizingMode.Width);
}
private void OnSameHeight()
{
//Scales the height of the selected objects.
diagram.SetSameSize(SizingMode.Height);
}

```

## Clipboard

Clipboard commands are used to cut, copy, or paste the selected elements.

- Cuts the selected elements from the diagram to the diagram's clipboard using [Cut](#) command.
- Copies the selected elements from the diagram to the diagram's clipboard using [Copy](#) command.
- Pastes the diagram's clipboard data (nodes/connectors) into the diagram using [Paste](#) command.

The following code illustrates how to execute the clipboard commands.

## ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="Cut" @onclick="@OnCut" />
<input type="button" value="Copy" @onclick="@OnCopy" />
<input type="button" value="Paste" @onclick="@OnPaste" />
<SfDiagramComponent @ref="diagram" Height="600px" Nodes="@nodes"
Connectors="@Connectors"/>
@code
{
//Reference to diagram
SfDiagramComponent diagram;
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
DiagramObjectCollection<Connector> Connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
Node node1 = new Node()
{

```

```

ID = "node1",
Width = 50,
Height = 30,
OffsetX = 500,
OffsetY = 100,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle },
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" }
};
nodes.Add(node1);
Node node2 = new Node()
{
ID = "node2",
Width = 60,
Height = 40,
OffsetX = 400,
OffsetY = 200,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle },
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" }
};
nodes.Add(node2);
}
private void OnCut()
{
//Removes the selected objecte
diagram.Cut();
}
private void OnCopy()
{
//copies the selected object
diagram.Copy();
}
private void OnPaste()
{
//pastes the copied object
diagram.Paste();
}
}

```

### Grouping

**Grouping commands** are used to group/ungroup the selected elements on the diagram. To group the elements, select the elements using select all command and group the selected elements using group command.

[Group](#) command is used to group the selected nodes and connectors in the diagram.

[UnGroup](#) command is used to ungroup the selected nodes and connectors in the diagram.

The following code illustrates how to execute the grouping commands.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<input type="button" value="Group" @onclick="@OnGroup" />
<input type="button" value="UnGroup" @onclick="@OnUnGroup" />
<input type="button" value="SelectAll" @onclick="@OnSelectAll" />

```



```

<SfDiagramComponent @ref="@diagram" Height="600px" Nodes="@nodes"
Connectors="@Connectors" />
@code
{
//Reference to diagram
SfDiagramComponent diagram;
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
DiagramObjectCollection<Connector> Connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
Node node1 = new Node()
{
ID = "node1",
Width = 50,
Height = 30,
OffsetX = 500,
OffsetY = 100,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle },
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" }
};
nodes.Add(node1);
Node node2 = new Node()
{
ID = "node2",
Width = 60,
Height = 40,
OffsetX = 400,
OffsetY = 200,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle },
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" }
};
nodes.Add(node2);
}
private void OnGroup()
{
//group the selected items in the diagram
diagram.Group();
}
private void OnUnGroup()
{
//ungroup the selected items in the diagram
diagram.UnGroup();
}
private void OnSelectAll()
{
//select all the items in the diagram
diagram.SelectAll();
}
}

```

### Zoom

The [Zoom](#) command is used to zoom-in and zoom-out the diagram view.

The following code illustrates how to zoom-in/zoom out the diagram.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<input type="button" value="Zoom" @onclick="@OnZoom" />
<SfDiagramComponent @ref="@diagram" Height="600px" Nodes="@nodes"
Connectors="@Connectors"/>
@code
{
    //Reference to diagram
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    DiagramObjectCollection<Connector> Connectors = new
    DiagramObjectCollection<Connector>();
    protected override void OnInitialized()
    {
        Node node1 = new Node()
        {
            ID = "node1",
            Width = 50,
            Height = 30,
            OffsetX = 500,
            OffsetY = 100,
            Shape = new BasicShape() { Type = Shapes.Basic, Shape =
            BasicShapeType.Rectangle },
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" }
        };
        nodes.Add(node1);
    }
    private void OnZoom()
    {
        // Sets the ZoomFactor
        // Defines the FocusPoint to zoom the Diagram with respect to any point
        // When you do not set focus point, zooming is performed with reference to
        the center of current Diagram view.
        diagram.Zoom(1.2, new DiagramPoint() { X = 100, Y = 100 });
    }
}
```

### Nudge command

The [Nudge](#) commands repositions the selected object by the specified delta in the given direction.

[Direction](#) nudge command moves the selected elements towards the specified direction by 1 pixel, by default.

The accepted values of the argument direction are as follows:

- Up: Moves the selected elements towards up by the specified delta value.
- Down: Moves the selected elements towards down by the specified delta value.
- Left: Moves the selected elements towards left by the specified delta value.
- Right: Moves the selected elements towards right by the specified delta value.

The following code illustrates how to execute nudge command.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="diagram" Height="600px" />
@code
{
    SfDiagramComponent diagram;
    private void NudgeLeft()
    {
        //Repositions the selected objects by 50 towards left direction.
        diagram.Nudge(Direction.Left, 50);
    }
}
```

#### Nudge by using arrow keys

The corresponding arrow keys are used to move the selected elements towards up, down, left, or right direction by 1 pixel.



Nudge commands are particularly useful for accurate placement of elements.

#### Undo and Redo command

The [Undo](#) and [Redo](#) commands help you to revert/restore the changes.

### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="@diagram" Height="600px">
</SfDiagramComponent>
@code
{
    //Reference to diagram
    SfDiagramComponent diagram;
    private void Undo()
    {
        //Revert the changes
        diagram.Undo();
    }
    private void Redo()
    {
        //Restore the changes
        diagram.Redo();
    }
}
```

#### Command manager

Diagram provides support to map or bind command execution with desired combination of key gestures. Diagram provides some built-in commands.

The [CommandManager](#) provides support to define custom commands. The custom commands are executed when the specified key gesture is recognized.

### Command Execution

The [Execute](#) event call back method will invoke when execute the custom command in the diagram.

The [CanExecute](#) event determines whether this command can execute in its current state.

### Custom command

To define a custom command, specify the following properties:

- [Gesture](#): A combination of [Keys](#) and [Modifiers](#).
- [Name](#): Defines the name of the command.

To explore the properties of custom commands, refer to the [Commands](#).

The following code example shows how to define a custom command.

### ASPX-CS

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="@diagram" Height="600px" Nodes="@nodes">
  @* Initializing the custom commands *@
  <CommandManager>
    <CommandManager Commands="@command" Execute="@CommandExecute"
    CanExecute="@Canexe">
  </CommandManager>
</CommandManager>
</SfDiagramComponent>
@code
{
  // Reference to diagram
  SfDiagramComponent diagram;
  DiagramObjectCollection<KeyboardCommand> command = new
  DiagramObjectCollection<KeyboardCommand>()
  {
    new KeyboardCommand()
    {
      Name = "CustomGroup",
      Gesture = new KeyGesture() { Key = Keys.G, Modifiers = ModifierKeys.Control
    },
  },
  new KeyboardCommand()
  {
    Name = "CustomUnGroup",
    Gesture = new KeyGesture() { Key = Keys.U, Modifiers = ModifierKeys.Control
  },
  },
};
  // Defines diagram's nodes collection
  DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
  protected override void OnInitialized()
  {
    Node node1 = new Node()
    {
      ID = "node1",
      OffsetX = 100,
      OffsetY = 100,
      Width = 100,
      Height = 100,
    }
  }
}

```

```
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" },
Annotations = new DiagramObjectCollection<ShapeAnnotation>()
{
    new ShapeAnnotation() { Content = "Node" }
};
nodes.Add(node1);
Node node2 = new Node()
{
    ID = "node2",
    OffsetX = 300,
    OffsetY = 100,
    Width = 100,
    Height = 100,
    Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" },
    Annotations = new DiagramObjectCollection<ShapeAnnotation>()
    {
        new ShapeAnnotation() { Content = "Node1" }
    }
};
nodes.Add(node2);
}
public void Canexe(CommandKeyArgs args)
{
    args.CanExecute = true;
}
/// <summary>
/// Custom command execution
/// </summary>
public void CommandExecute(CommandKeyArgs args)
{
    if (args.Gesture.Modifiers == ModifierKeys.Control && args.Gesture.Key ==
Keys.G)
    {
        //Custom command to group the selected nodes
        diagram.Group();
    }
    if (args.Gesture.Modifiers == ModifierKeys.Control && args.Gesture.Key ==
Keys.U)
    {
        DiagramSelectionSettings selector = diagram.SelectionSettings;
        //Custom command to ungroup the selected items
        if (selector.Nodes.Count > 0 && selector.Nodes[0] is NodeGroup)
        {
            if ((selector.Nodes[0] as NodeGroup).Children.Length > 0)
            {
                diagram.UnGroup();
            }
        }
    }
}
}
```

*Modify the existing command*

When any one of the default commands is not desired, they can be disabled. To change the functionality of a specific command, the command can be completely modified.

The following code example shows how to disable a command and how to modify the built-in commands.

**ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent @ref="@diagram" Height="600px" Nodes="@nodes">
  @* Initializing the custom commands *@
  <CommandManager>
    <CommandManager Commands="@commands" Execute="@CommandExecute"
    CanExecute="@Canexe">
  </CommandManager>
</CommandManager>
</SfDiagramComponent>
@code
{
  // Reference to diagram
  SfDiagramComponent diagram;
  // Defines diagram's nodes collection
  DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
  DiagramObjectCollection<KeyboardCommand> commands = new
  DiagramObjectCollection<KeyboardCommand>()
  {
    new KeyboardCommand()
    {
      Name = "SelectAll",
      Gesture = new KeyGesture() { Key = Keys.A, Modifiers = ModifierKeys.Control
    },
    },
    new KeyboardCommand()
    {
      Name = "Copy",
      Gesture = new KeyGesture() { Key = Keys.C, Modifiers = ModifierKeys.Control
    },
    },
  };
  protected override void OnInitialized()
  {
    Node node1 = new Node()
    {
      ID = "node1",
      OffsetX = 100,
      OffsetY = 100,
      Width = 100,
      Height = 100,
      Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" },
      Annotations = new DiagramObjectCollection<ShapeAnnotation>()
      {
        new ShapeAnnotation() { Content = "Node" }
      }
    };
    nodes.Add(node1);
    Node node2 = new Node()
```

```

{
    ID = "node2",
    OffsetX = 300,
    OffsetY = 100,
    Width = 100,
    Height = 100,
    Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" },
    Annotations = new DiagramObjectCollection<ShapeAnnotation>()
    {
        new ShapeAnnotation() { Content = "Node1" }
    }
};
nodes.Add(node2);
}
public void Canexe(CommandKeyArgs args)
{
    args.CanExecute = true;
}
/// <summary>
/// Custom command execution event
/// </summary>
public void CommandExecute(CommandKeyArgs args)
{
    if (args.Gesture.Modifiers == ModifierKeys.Control && args.Gesture.Key ==
    Keys.A)
    {
        //to disable a built-in command and none of action execute
    }
    if (args.Gesture.Modifiers == ModifierKeys.Control && args.Gesture.Key ==
    Keys.C)
    {
        //Modify the existing copy command to cut command
        diagram.Cut();
    }
}
}
}

```

## Undo Redo support in Blazor Diagram Component

Diagram tracks the history of actions that are performed after initializing the diagram and provides support to reverse and restore those changes.

### Undo and redo

[Diagram](#) provides built-in support to track the changes that are made through interaction and through public APIs. The changes can be reverted or restored either through shortcut keys or through commands.

### Undo/redo through shortcut keys

Undo/redo commands can be executed through shortcut keys. Shortcut key for undo is Ctrl+Z and shortcut key for redo is Ctrl+Y.

### Undo/redo through public APIs

The [Undo](#) and [Redo](#) methods helps you to revert/restore the changes. The following code example illustrates how to undo/redo the changes through code.

### **ASPX-CS**

```
SfDiagramComponent diagram;
// Reverts the last action performed
diagram.Undo();
// Restores the last undone action
diagram.Redo();
```

When a change in the diagram is reverted or restored (undo/redo), the [HistoryChanged](#) event gets triggered.

#### *Group multiple changes*

History list allows to revert or restore multiple changes through a single undo/redo command. For example, revert/restore the fill color change of multiple elements at a time.

[StartGroupAction](#) is used to notify the diagram to start grouping the changes. [EndGroupAction](#) is used to notify to stop grouping the changes. The following code illustrates how to undo/redo to change of multiple elements at a time.

#### **ASPX-CS**

```
SfDiagramComponent diagram;
//Starts grouping the changes
diagram.StartGroupAction();
//Ends grouping the changes
diagram.EndGroupAction();
```

#### History change event

- While interacting the elements in the diagram, this event can be used to do the customization.
- When interacting the node or connector, the entries getting added to the history list to trigger this event.

The [HistoryChangedEventArgs](#) notifies while the changes occurs during undo/redo process.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Diagram
@* Initialize Diagram *@
<SfDiagramComponent @ref="@diagram" Height="600px"
HistoryChanged="@Onhistorychange">
</SfDiagramComponent>
@code
{
SfDiagramComponent diagram;
public void Onhistorychange(HistoryChangedEventArgs args)
{
//causes of history change
HistoryChangedAction ActionTrigger = args.ActionTrigger;
}
}
```

#### Track custom entry

[Diagram](#) provides options to track the changes that are made to custom properties. The following example illustrates how to track such custom property changes.



**ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
<input value="CustomEntry" type="button" @onclick="@OnCustomEntry"
name="CustomEntry" />
@* Initialize Diagram *@
<SfDiagramComponent @ref="@diagram" Height="600px" Nodes="@nodes">
</SfDiagramComponent>
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Dictionary<string, object> NodeInfo = new Dictionary<string, object>();
        NodeInfo.Add("nodeInfo", "Central Node");
        // A node is created and stored in the nodes collection.
        Node node = new Node()
        {
            ID = "node1",
            // Position of the node
            OffsetX = 250,
            OffsetY = 250,
            // Size of the node
            Width = 100,
            Height = 100,
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
            AdditionalInfo = NodeInfo
        };
        // Add node
        nodes.Add(node);
    }
    private void OnCustomEntry()
    {
        HistoryEntry entry = new HistoryEntry();
        entry.UndoObject = diagram.Nodes[0];
        diagram.AddHistoryEntry(entry);
    }
}

```

*HistoryAdding*

[HistoryAdding](#) in the [DiagramHistoryManager](#), which takes a history entry as argument and returns whether the specific entry can be added or not.

**ASPX-CS**

```

@using Syncfusion.Blazor.Diagram
@* Initialize Diagram *@
<SfDiagramComponent @ref="@diagram" Height="600px" Nodes="@nodes">
<DiagramHistoryManager HistoryAdding="@OnHistoryAdding"/>
</SfDiagramComponent>
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();

```

```
protected override void OnInitialized()
{
    nodes = new DiagramObjectCollection<Node>();
    // A node is created and stored in the nodes collection.
    Node node = new Node()
    {
        ID = "node1",
        // Position of the node
        OffsetX = 250,
        OffsetY = 250,
        // Size of the node
        Width = 100,
        Height = 100,
        Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" }
    };
    // Add node
    nodes.Add(node);
}
private void OnHistoryAdding(HistoryAddingEventArgs entry)
{
    // Sets true to cancel undo/redo action.
    entry.Cancel = true;
}
}
```

#### Custom undo redo

The purpose of custom undo redo process is to store actions which are not done through default undo redo history list.

[Undo](#) method in [DiagramHistoryManager](#) is getting triggered when the custom entry is in undo stage and [Redo](#) method in [DiagramHistoryManager](#) is getting triggered when the custom entry is in redo stage.

#### ASPX-CS

```
@using Syncfusion.Blazor.Diagram
<input value="CustomEntry" type="button" @onclick="@OnCustomEntry"
name="CustomEntry" />
@* Initialize Diagram *@
<SfDiagramComponent @ref="@diagram" Height="600px" Nodes="@nodes">
<DiagramHistoryManager Undo="@onCustomUndo" Redo="@onCustomRedo"/>
</SfDiagramComponent>
@code
{
    SfDiagramComponent diagram;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    string EventValue = string.Empty;
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        // A node is created and stored in the nodes collection.
        Node node = new Node()
        {
            ID = "node1",
            // Position of the node
            OffsetX = 250,
```

```

OffsetY = 250,
// Size of the node
Width = 100,
Height = 100,
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" }
};
// Add node
nodes.Add(node);
}
private void OnCustomEntry()
{
    HistoryEntry entry = new HistoryEntry();
    entry.UndoObject = diagram.Nodes[0];
    diagram.AddHistoryEntry(entry);
}
private void onCustomUndo(HistoryEntryBase entry)
{
    (entry.RedoObject) = entry.UndoObject.Clone() as Node;
    (entry.UndoObject as Node).AdditionalInfo[(entry.UndoObject as Node).ID] =
    "Start";
    EventValue += "UndoObject:" + (entry.UndoObject as
    Node).AdditionalInfo[(entry.UndoObject as Node).ID];
}
private void onCustomRedo(HistoryEntryBase entry)
{
    EventValue += "RedoObject:" + (entry.RedoObject as
    Node).AdditionalInfo[(entry.RedoObject as Node).ID];
    Node current = entry.UndoObject.Clone() as Node;
    (entry.UndoObject as Node).AdditionalInfo[(entry.UndoObject as Node).ID] =
    "Description";
    entry.RedoObject = current;
}
}

```

## User Handles for node, connector in Blazor Diagram Component

The user handles are customizable handles that can be used to perform custom actions and default clipboard actions.

### Initialization an user handle

The user handle can enables for the selected nodes/connectors by setting a [SelectorConstraints](#) as [UserHandle](#) and then use the [UserHandle](#) class to define the userhandle object and add that to [UserHandles](#) collection of the DiagramSelectionSettings. The following code example used to enable and create an user handles for the diagram nodes/connectors.

### CSHARP

```

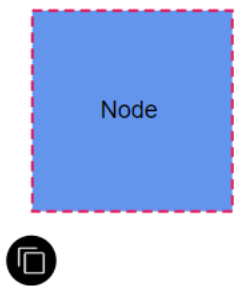
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px"
Nodes="@nodes"
SelectionSettings="@SelectedModel">
    <SnapSettings>
        <HorizontalGridLines LineColor="White" LineDashArray="2,2" />
        <VerticalGridLines LineColor="White" LineDashArray="2,2" />
    </SnapSettings>
</SfDiagramComponent>

```

```

@code
{
    // Defines diagram's nodes collection
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    // Defines diagram's SelectionSettings
    DiagramSelectionSettings SelectedModel = new DiagramSelectionSettings();
    DiagramObjectCollection<UserHandle> UserHandles = new
    DiagramObjectCollection<UserHandle>();
    protected override void OnInitialized()
    {
        //Creating the userhandle for cloning the objects
        UserHandle cloneHandle = new UserHandle()
        {
            //Name of the user handle
            Name = "clone",
            //Set pathdata for userhandle
            PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
            M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
            2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
            30V34.4h30V72.5z",
            //Set visibility for the user handle
            Visible = true,
            //Set the position for the user handle
            Offset = 0,
            //Set side based on the given offset
            Side = Direction.Bottom,
            //set margin for the user handle
            Margin = new Margin() { Top = 0, Bottom = 0, Left = 0, Right = 0 };
        };
        //Add user handle to the collection...
        UserHandles = new DiagramObjectCollection<UserHandle>()
        {
            cloneHandle
        };
        SelectedModel = new DiagramSelectionSettings()
        {
            //Enable userhandle for selected model...
            Constraints = SelectorConstraints.UserHandle,
            UserHandles = this.UserHandles
        };
        nodes = new DiagramObjectCollection<Node>();
        Node diagramNode = new Node()
        {
            ID = "node1",
            OffsetX = 100,
            OffsetY = 100,
            Width = 100,
            Height = 100,
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "none" },
            Annotations = new DiagramObjectCollection<ShapeAnnotation>() { new
            ShapeAnnotation { Content = "Node" } }
        };
        nodes.Add(diagramNode);
    }
}

```



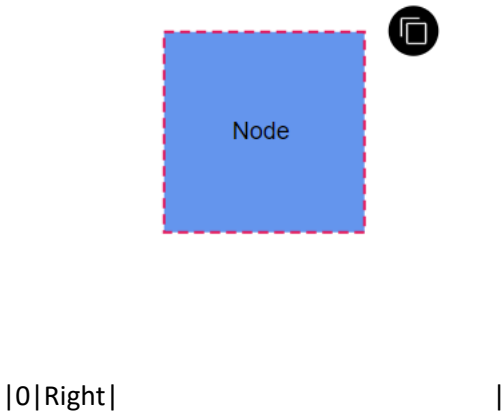
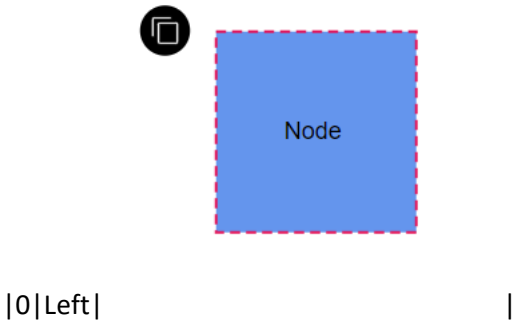
Customization

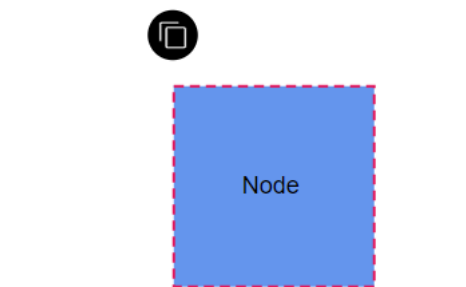
Position

The [Offset](#) property of user handle is used to align the user handles based on fractions. 0 represents Top-Left corner, 1 represents Bottom-Right corner, and 0.5 represents half of Width or Height. The [Side](#) property is used to set how the user handle is aligned based on the given [Offset](#).

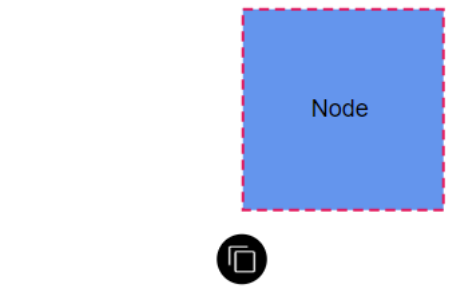
The following table shows all the possible alignments visually shows the user handle positions.

Offset	Side	Output
-----	-----	-----

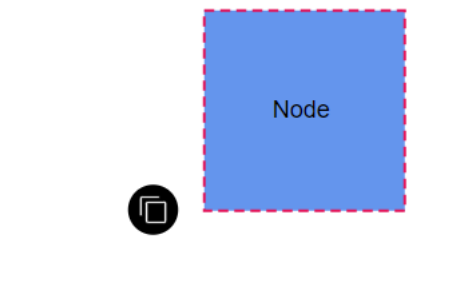




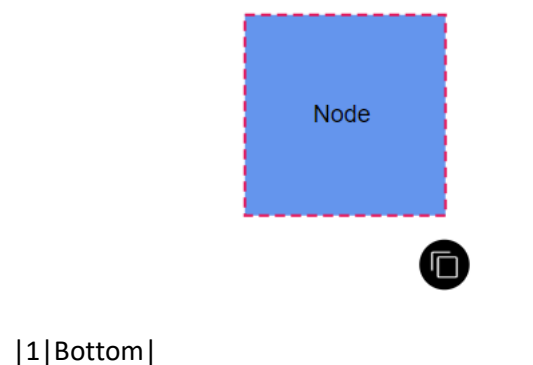
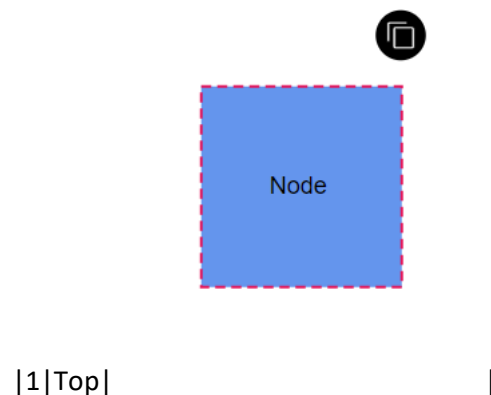
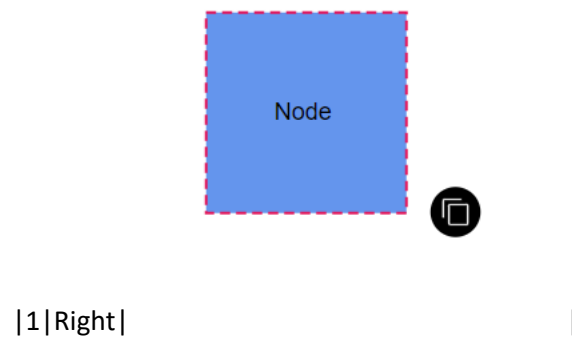
|0|Top|



|0|Bottom|



|1|Left|



### Size

Diagram allows you set size for user handles by using the [Size](#) property. The default value of the `Size` property is 25.

### Style

You can change the style of the user handles with the specific properties of `PathColor`, `BorderColor`, `BackgroundColor` and `BorderWidth`. The following code explains how to customize the appearance of the user handles.

- The user handle's [PathColor](#) property used to change the color of the given [PathData](#) of the user handle.

- The user handle [BorderColor](#), [BackgroundColor](#) properties are used to define the background color and border color of the user handle and the [BorderWidth](#) property is used to define the border width of the user handles.
- The [Visible](#) property indicating whether the user handle is visible in the user interface.

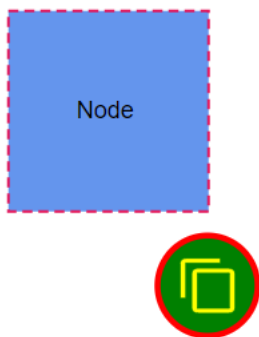
The following code explains how to customize the appearance of the user handle.

### CSHARP

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes"
SelectionSettings="@SelectedModel">
<SnapSettings>
<HorizontalGridLines LineColor="White" LineDashArray="2,2"/>
<VerticalGridLines LineColor="White" LineDashArray="2,2"/>
</SnapSettings>
</SfDiagramComponent>
@code
{
// Defines diagram's nodes collection
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
// Defines diagram's SelectionSettings
DiagramSelectionSettings SelectedModel = new DiagramSelectionSettings();
DiagramObjectCollection<UserHandle> UserHandles = new
DiagramObjectCollection<UserHandle>();
protected override void OnInitialized()
{
//Creating the userhandle for cloning the objects
UserHandle cloneHandle = new UserHandle()
{
//Name of the user handle
Name = "clone",
//Set pathdata for userhandle
PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z",
//Set visibility for the user handle
Visible = true,
//Set the position for the user handle
Offset = 1,
//Set side based on the given offset
Side = Direction.Bottom,
//set margin for the user handle
Margin = new Margin() { Top = 0, Bottom = 0, Left = 0, Right = 0 },
//Set size of the user handle
Size = 50,
//Set pathcolor for given pathdata
PathColor = "yellow",
//Set Border color of the user handle
BorderColor = "red",
//Set Background Color of the user handle
BackgroundColor = "green",
//Set Border Width Color of the user handle
BorderWidth = 3,
};
};
```



```
//Add user handle to the collection...
UserHandles = new DiagramObjectCollection<UserHandle>()
{
    cloneHandle
};
SelectedModel = new DiagramSelectionSettings()
{
    //Enable userhandle for selected model...
    Constraints = SelectorConstraints.UserHandle,
    UserHandles = this.UserHandles
};
nodes = new DiagramObjectCollection<Node>();
Node diagramNode = new Node()
{
    ID = "node1",
    OffsetX = 100,
    OffsetY = 100,
    Width = 100,
    Height = 100,
    Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "none" },
    Annotations = new DiagramObjectCollection<ShapeAnnotation>() { new
    ShapeAnnotation { Content = "Node" } }
};
nodes.Add(diagramNode);
}
}
```



### Fixed user handles

The [FixedUserHandle](#) is used to add some frequently used commands around the node and connector even without selecting it.

### Initialization an fixed user handles

To create the fixed user handles, define and add them to the collection of nodes and connectors property. The following code example used to create an fixed user handles for the nodes and connectors.

### **CSHARP**

```
@using Syncfusion.Blazor.Diagram
```

```

<SfDiagramComponent Height="600px" Nodes="@nodes" />
@code
{
    // Defines diagram's nodes collection
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    protected override void OnInitialized()
    {
        nodes = new DiagramObjectCollection<Node>();
        Node node1 = new Node()
        {
            OffsetX = 250,
            OffsetY = 250,
            Width = 100,
            Height = 100,
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
            // A fixed user handle is created and stored in fixed user handle collection
            // of Node.
            FixedUserHandles = new DiagramObjectCollection<NodeFixedUserHandle>()
            {
                new NodeFixedUserHandle()
                {
                    ID = "user1",
                    Height = 20,
                    Width = 20,
                    Visibility = true,
                    Padding = new Margin() { Bottom = 1, Left = 1, Right = 1, Top = 1 },
                    Margin = new Margin() { Right = 20 }, Offset = new DiagramPoint() { X = 0 ,
                    Y = 0 },
                    PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
                    M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5h30c3,0,5.5-
                    2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
                    30V34.4h30V72.5z"
                },
            };
            nodes.Add(node1);
        }
    }
}

```

### Customization the fixed user handle

- The [ID](#) property of fixed user handle is used to define the unique identification of the fixed user handle and it is further used to add custom events to the fixed user handle.
- The fixed user handle can be positioned relative to the node and connector boundaries. It has [Offset](#), [Padding](#) and [CornerRadius](#) settings. It is used to position and customize the fixed user handle.
- The [Padding](#) is used to leave the space that is inside the fixed user handle between the icon and border.
- The [CornerRadius](#) allows to create fixed user handles with rounded corners. The radius of the rounded corner is set with the [CornerRadius](#) property.

---

The [PathData](#) needs to be provided to render fixed user handle.

---

### Size

Diagram allows you set size for the fixed user handles by using the [Width](#) and [Height](#) property. The default value of the [Width](#) and [Height](#) property is 10.

### Style

- You can change the style of the fixed user handles with the specific properties of [borderColor](#), [borderWidth](#), and [backgroundColor](#) by using the [Stroke](#), [StrokeThickness](#), and [Fill](#) properties, and the icon [BorderColor](#), and [BorderWidth](#) by using the [IconStroke](#) and [IconStrokeThickness](#) properties.
- The fixed user handle's [IconStroke](#) and [IconStrokeThickness](#) property used to change the stroke color and stroke width of the given [PathData](#).
- The fixed user handle [Stroke](#) and [Fill](#) properties are used to define the background color and border color of the user handle and the [StrokeThickness](#) property is used to define the border width of the fixed user handle.
- The [Visibility](#) property indicating whether the fixed user handle is visible in the user interface.

The following code explains how to customize the appearance of the fixed user handles.

### CSHARP

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Connectors="@connectors">
<SnapSettings>
<HorizontalGridLines LineColor="White" LineDashArray="2,2"/>
<VerticalGridLines LineColor="White" LineDashArray="2,2"/>
</SnapSettings>
</SfDiagramComponent>
@code
{
DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
connectors = new DiagramObjectCollection<Connector>();
Connector connector = new Connector()
{
SourcePoint = new DiagramPoint() { X = 100, Y = 100 },
TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
Type = ConnectorSegmentType.Orthogonal,
Style = new ShapeStyle() { StrokeColor = "#6495ED" },
// A fixed user handle is created and stored in fixed user handle collection
of Connector.
FixedUserHandles = new DiagramObjectCollection<ConnectorFixedUserHandle>()
{
new ConnectorFixedUserHandle()
{
ID = "user1",
Height = 25,
Width = 25,
Offset = 0.5,
Alignment = FixedUserHandleAlignment.After,
Displacement = new DiagramPoint() { Y = 10 },
```

```

Visibility = true, Padding = new Margin() { Bottom = 1, Left = 1, Right = 1,
Top = 1 },
PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z"
}
},
};
connectors.Add(connector);
}
}

```

---

The fixed user handle id need to be unique.

---

### Customizing the node fixed user handle

The node fixed user handle can be aligned relative to the node boundaries. It has [Margin](#) and [Offset](#) settings. It is quite useful to position the node fixed user handle and used together and gives you more control over the node fixed user handle positioning.

#### *Margin for the node fixed user handle*

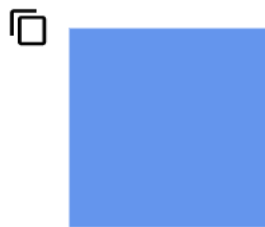
[Margin](#) is an absolute value used to add some blank space in any one of its four sides. The fixed user handle can be displaced with the `Margin` property.

#### *Offset for the node fixed user handle*

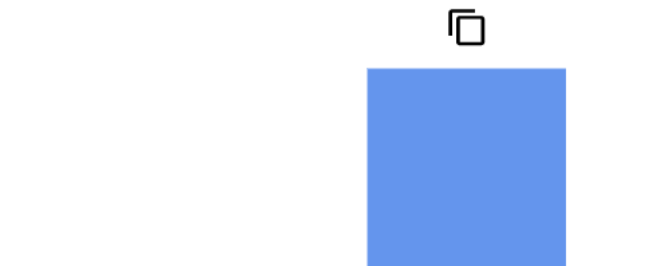
The [Offset](#) property of fixed user handle is used to align the user handle based on the `X` and `Y` points. (0,0) represents the top-left corner and (1,1) represents the bottom-right corner.

The following table shows all the possible alignments visually shows the fixed user handle positions.

Offset	Margin	Output
-----	-----	-----



(0,0)   Right = 20	
--------------------	--



| (0.5,0) | Bottom = 20 |



| (1,0) | Left = 20 |



| (0,0.5) | Right = 20 |



| (1,0.5) | Left = 20 |



| (0,1) | Right = 20 |

|



| (0.5,1) | Top = 20 |

|



| (1,1) | Left = 20 |

|

The following code explains how to customize the node fixed user handle.

#### **CSHARP**

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Nodes="@nodes">
  <SnapSettings>
    <HorizontalGridLines LineColor="White" LineDashArray="2,2"/>
    <VerticalGridLines LineColor="White" LineDashArray="2,2"/>
  </SnapSettings>
</SfDiagramComponent>
@code
{
```

```
// Defines diagram's nodes collection
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
protected override void OnInitialized()
{
    //Creating the userhandle for cloning the objects
    nodes = new DiagramObjectCollection<Node>();
    Node diagramNode = new Node()
    {
        OffsetX = 250,
        OffsetY = 250,
        Width = 100,
        Height = 100,
        Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "white" },
        // A fixed user handle is created and stored in fixed user handle collection
        // of Node.
        FixedUserHandles = new DiagramObjectCollection<NodeFixedUserHandle>()
        {
            new NodeFixedUserHandle()
            {
                ID = "user1",
                Height = 20,
                Width = 20,
                Visibility = true,
                Padding = new Margin() { Bottom = 1, Left = 1, Right = 1, Top = 1 },
                Margin = new Margin() { Left = 20 },
                Offset = new DiagramPoint() { Y = 0 },
                PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
                M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
                2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
                30V34.4h30V72.5z"
            },
        },
    };
    nodes.Add(diagramNode);
}
```

### Customizing the connector fixed user handle

- The connector fixed user handle can be aligned relative to the connector boundaries. It has alignment, displacement and offset settings. It is useful to position the connector fixed user handle and used together and gives you more control over the connector fixed user handle positioning.
- The [Offset](#) and [Alignment](#) properties of fixed user handle allows you to align the connector fixed user handles to the segments.

#### Offset for the connector fixed user handle

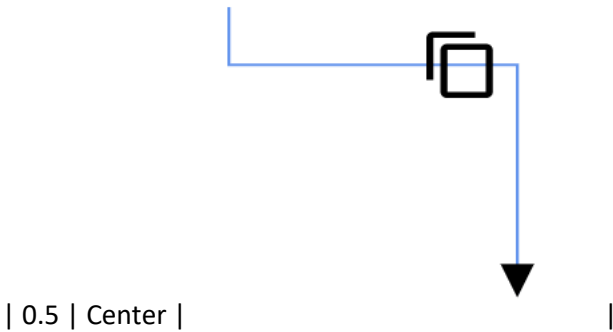
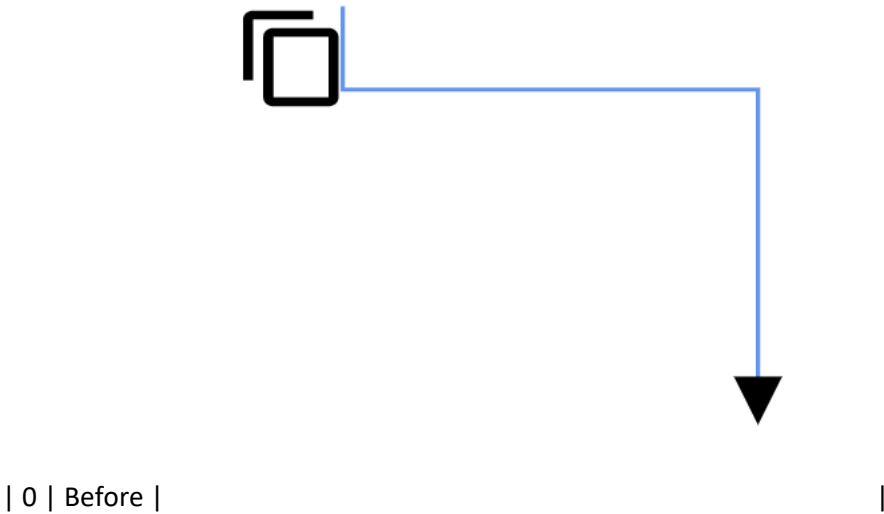
The [Offset](#) property of connector fixed user handle is used to align the user handle based on fractions. 0 represents the connector source point, 1 represents the connector target point, and 0.5 represents the center point of the connector segment.

Alignment

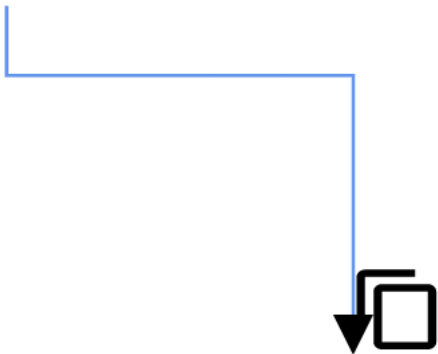
The connector's fixed user handle can be aligned over its segment path using the [Alignment](#) property of fixed user handle.

The following table shows all the possible alignments visually shows the fixed user handle positions.

Offset	Alignment	Output
-----	-----	-----







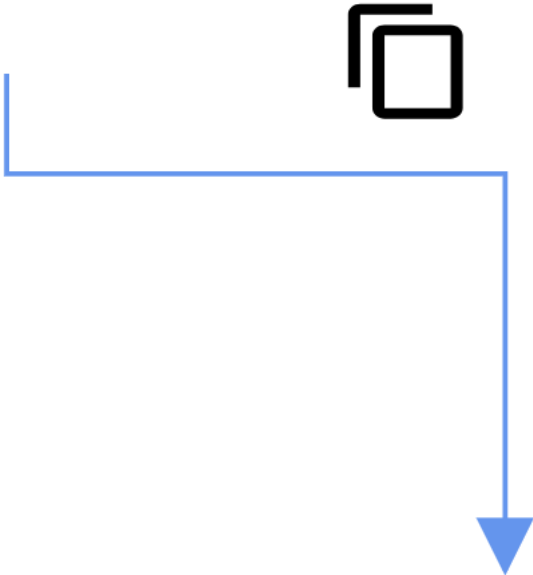
| 1 | After |

Displacement

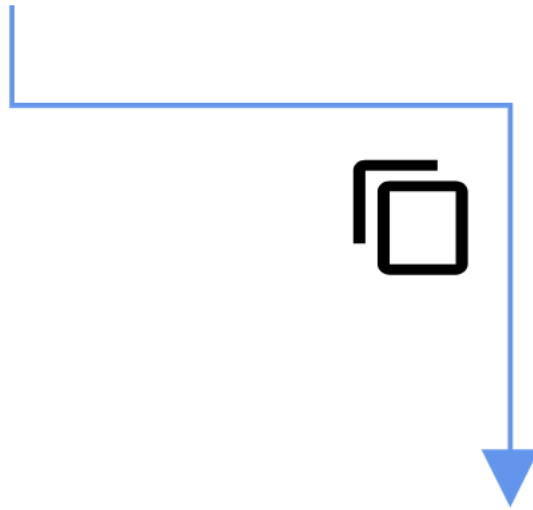
- The [Displacement](#) property allows you to specify the space to be left from the connector segment based on the x and y value provided.

The following table shows all the possible alignments visually shows the fixed user handle positions.

Displacement	Alignment	Output
-----	-----	-----



| y = 10 | Before |



| y = 10 | After |

Displacement will not be done if the alignment is set to be center.

The following code explains how to customize the connector fixed user handle.

#### CSHARP

```
@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px" Connectors="@connectors">
<SnapSettings>
<HorizontalGridLines LineColor="White" LineDashArray="2,2"/>
<VerticalGridLines LineColor="White" LineDashArray="2,2"/>
</SnapSettings>
</SfDiagramComponent>
@code
{
DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
protected override void OnInitialized()
{
connectors = new DiagramObjectCollection<Connector>();
Connector connector = new Connector()
{
SourcePoint = new DiagramPoint() { X = 100, Y = 100 },
TargetPoint = new DiagramPoint() { X = 200, Y = 200 },
Type = ConnectorSegmentType.Orthogonal,
Style = new ShapeStyle() { StrokeColor = "#6495ED" },
// A fixed user handle is created and stored in fixed user handle collection
of Connector.
FixedUserHandles = new DiagramObjectCollection<ConnectorFixedUserHandle>()
{
new ConnectorFixedUserHandle()
{
ID = "user1",
Height = 25,
```

```

Width = 25,
Offset = 0.5,
Alignment = FixedUserHandleAlignment.After,
Displacement = new DiagramPoint { Y = 10 },
Visibility = true, Padding = new Margin() { Bottom = 1, Left = 1, Right = 1,
Top = 1 },
PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z"
}
}
};
connectors.Add(connector);
}
}

```

### FixedUserHandle Event

The Diagram control provides following event for the fixed user handle.

Event Name	Event Type	Description
------------	------------	-------------

-----	-----	-----
-------	-------	-------

<a href="#">FixedUserHandleClick</a>	<a href="#">FixedUserHandleClickEventArgs</a>	Triggered when the mouse pointer is over the user handle and mouse button is up.
--------------------------------------	---	--

### CSHARP

```

@using Syncfusion.Blazor.Diagram
<SfDiagramComponent Height="600px"
FixedUserHandleClick="Changed" Nodes="@nodes" @ref="diagram">
</SfDiagramComponent>
@code
{
SfDiagramComponent diagram;
public void Changed(FixedUserHandleClickEventArgs args)
{
if ((args.Element as Node).ID == "node1" && args.FixedUserHandle.ID ==
"user1")
{
diagram.Copy();
diagram.Paste();
}
}
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
protected override void OnInitialized()
{
nodes = new DiagramObjectCollection<Node>();
Node node1 = new Node()
{
OffsetX = 250,
OffsetY = 250,
ID = "node1",
Width = 100,
Height = 100,
Style = new ShapeStyle() { Fill = "#6BA5D7", StrokeColor = "white" },

```

```
// A fixed user handle is created and stored in fixed user handle collection
// of Node.
FixedUserHandles = new DiagramObjectCollection<NodeFixedUserHandle>()
{
    new NodeFixedUserHandle()
    {
        ID = "user1",
        Height = 20,
        Width = 20,
        Visibility = true,
        Padding = new Margin() { Bottom = 1, Left = 1, Right = 1, Top = 1 },
        Margin = new Margin() { Right = 20 },
        Offset = new DiagramPoint() { X = 0, Y = 0 },
        PathData = "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z"
    },
}
};
nodes.Add(node1);
}
```

## Symbol Palette in Blazor Diagram Component

The [SymbolPalette](#) displays a collection of [Palettes](#). The palette shows a set of nodes and connectors. It allows to drag and drop the nodes and connectors into the diagram.

### Create symbol palette

The [Width](#) and [Height](#) properties of the symbol palette allows to define the size of the symbol palette.

#### CSHARP

```
@using Syncfusion.Blazor.Diagram.SymbolPalette
@using Syncfusion.Blazor.Diagram
@* Initializes the symbol palette *@
<SfSymbolPaletteComponent Height="600px"
SymbolHeight="80" SymbolWidth="80">
</SfSymbolPaletteComponent>
```

### Add node to palette

[SymbolWidth](#) and [SymbolHeight](#) properties of the [SfSymbolPaletteComponent](#) should be defined to render the symbol(node, connector or nodegroup) in the palette. The following code example illustrates how to add node to a palette.

- To render a node in a palette, first create [SymbolPalette](#) and initialize palettes collection.

#### CSHARP

```
@using Syncfusion.Blazor.Diagram
@using Syncfusion.Blazor.Diagram.SymbolPalette
<div class="control-section">
<div style="width:20%">
```

```

<div id="palette-space" class="sb-mobile-palette" style="border: 2px solid #b200ff">
<SfSymbolPaletteComponent @ref="@symbolpalette" Height="300px" Width="200px"
Palettes="@Palettes" SymbolHeight="60" SymbolWidth="60"
SymbolMargin="@SymbolMargin">
</SfSymbolPaletteComponent>
</div>
</div>
</div>
@code
{
SymbolMargin SymbolMargin = new SymbolMargin
{
Left = 15,
Right = 15,
Top = 15,
Bottom = 15
};
SfSymbolPaletteComponent symbolpalette;
//Define palattes collection
DiagramObjectCollection<Palette> Palettes = new
DiagramObjectCollection<Palette>();
}

```

- Create node and add that node to the DiagramObjectCollection.

### CSHARP

```

// Defines palette's flow-shape collection
DiagramObjectCollection<NodeBase> PaletteNodes = new
DiagramObjectCollection<NodeBase>();
protected override void OnInitialized()
{
InitPaletteModel();
}
private void InitPaletteModel()
{
CreatePaletteNode(FlowShapeType.Terminator, "Terminator");
}
private void CreatePaletteNode(FlowShapeType flowShape, string id)
{
Node node = new Node()
{
ID = id,
Shape = new FlowShape() { Type = Shapes.Flow, Shape = flowShape },
Style = new ShapeStyle() { Fill="#6495ED", StrokeColor = "#757575" },
};
PaletteNodes.Add(node);
}

```

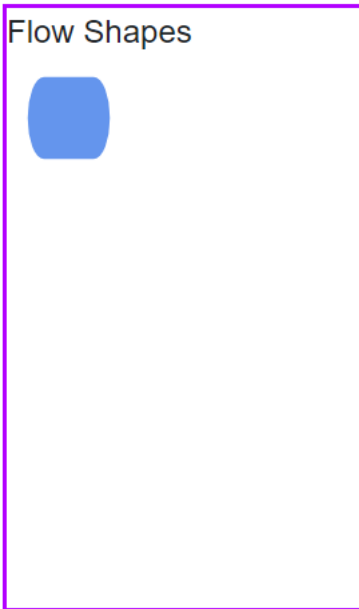
- Complete code to add node to the palette.

### CSHARP

```

@using Syncfusion.Blazor.Diagram
@using Syncfusion.Blazor.Diagram.SymbolPalette
<div class="control-section">
<div style="width:20%">
<div id="palette-space" class="sb-mobile-palette" style="border: 2px solid
#b200ff">
<SfSymbolPaletteComponent @ref="@symbolpalette" Height="300px" Width="200px"
Palettes="@Palettes" SymbolHeight="60" SymbolWidth="60"
SymbolMargin="@SymbolMargin">
</SfSymbolPaletteComponent>
</div>
</div>
</div>
@code
{
SymbolMargin SymbolMargin = new SymbolMargin
{
Left = 15,
Right = 15,
Top = 15,
Bottom = 15
};
SfSymbolPaletteComponent symbolpalette;
//Define palattes collection
DiagramObjectCollection<Palette> Palettes = new
DiagramObjectCollection<Palette>();
// Defines palette's flow-shape collection
DiagramObjectCollection<NodeBase> PaletteNodes = new
DiagramObjectCollection<NodeBase>();
protected override void OnInitialized()
{
InitPaletteModel();
}
private void InitPaletteModel()
{
CreatePaletteNode(FlowShapeType.Terminator, "Terminator");
Palettes = new DiagramObjectCollection<Palette>()
{
new Palette(){Symbols =PaletteNodes, Title="Flow Shapes", ID="Flow Shapes"
},
};
}
private void CreatePaletteNode(FlowShapeType flowShape, string id)
{
Node node = new Node()
{
ID = id,
Shape = new FlowShape() { Type = Shapes.Flow, Shape = flowShape },
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" },
};
PaletteNodes.Add(node);
}
}

```



*Add connector to palette*

The following code example illustrates how to add connector to a palette.

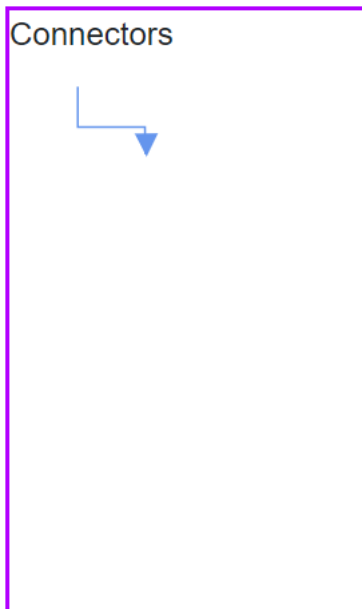
#### **C#**

```
@using Syncfusion.Blazor.Diagram
@using Syncfusion.Blazor.Diagram.SymbolPalette
<div class="control-section">
<div style="width:20%">
<div id="palette-space" class="sb-mobile-palette" style="border: 2px solid
#b200ff">
<SfSymbolPaletteComponent @ref="@symbolpalette" Height="300px" Width="200px"
Palettes="@Palettes" SymbolHeight="60" SymbolWidth="100">
</SfSymbolPaletteComponent>
</div>
</div>
</div>
@code
{
SfSymbolPaletteComponent symbolpalette;
//Define palattes collection
DiagramObjectCollection<Palette> Palettes = new
DiagramObjectCollection<Palette>();
// Defines palette's flow-shape collection
DiagramObjectCollection<NodeBase> PaletteNodes = new
DiagramObjectCollection<NodeBase>();
// Defines palette's connector collection
DiagramObjectCollection<NodeBase> PaletteConnectors = new
DiagramObjectCollection<NodeBase>();
protected override void OnInitialized()
{
InitPaletteModel();
}
private void InitPaletteModel()
{
```

```

CreatePaletteConnector("Link1", ConnectorSegmentType.Orthogonal,
DecoratorShape.Arrow);
Palettes = new DiagramObjectCollection<Palette>()
{
    new Palette(){Symbols = PaletteConnectors, Title = "Connectors" , IsExpanded
= true},
};
}
private void CreatePaletteConnector(string id, ConnectorSegmentType type,
DecoratorShape decoratorShape)
{
    Connector connector = new Connector()
    {
        ID = id,
        Type = type,
        SourcePoint = new DiagramPoint() { X = 0, Y = 0 },
        TargetPoint = new DiagramPoint() { X = 100, Y = 100 },
        Style = new ShapeStyle() { StrokeWidth = 1, StrokeColor = "#6495ED" },
        TargetDecorator = new DecoratorSettings()
        {
            Shape = decoratorShape,
            Style = new ShapeStyle() { StrokeColor = "#6495ED", Fill = "#6495ED" }
        }
    };
    PaletteConnectors.Add(connector);
}
}

```



*Add nodegroup to palette*

The following code example illustrates how to add nodegroup to a palette.

#### **CSHARP**

```
@using Syncfusion.Blazor.Diagram
```



```

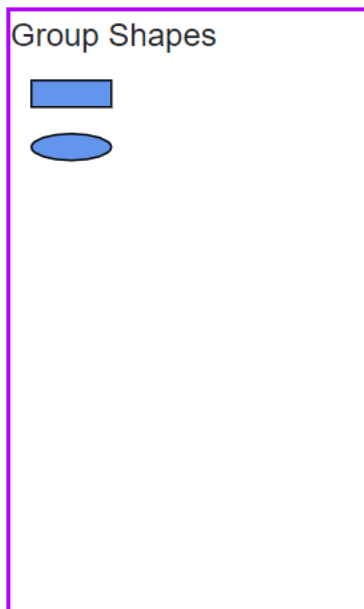
@using Syncfusion.Blazor.Diagram.SymbolPalette
<div class="control-section">
<div style="width:20%">
<div id="palette-space" class="sb-mobile-palette" style="border: 2px solid
#b200ff">
<SfSymbolPaletteComponent @ref="@symbolpalette" Height="300px" Width="200px"
Palettes="@Palettes" SymbolHeight="60" SymbolWidth="60"
SymbolMargin="@SymbolMargin">
</SfSymbolPaletteComponent>
</div>
</div>
</div>
@code{
SymbolMargin SymbolMargin = new SymbolMargin
{
Left = 15,
Right = 15,
Top = 15,
Bottom = 15
};
SfSymbolPaletteComponent symbolpalette;
//Define palattes collection
DiagramObjectCollection<Palette> Palettes = new
DiagramObjectCollection<Palette>();
// Defines palette's group collection
DiagramObjectCollection<NodeBase> PaletteGroup = new
DiagramObjectCollection<NodeBase>();
protected override void OnInitialized()
{
InitPaletteModel();
}
private void InitPaletteModel()
{
CreatePaletteGroup();
Palettes = new DiagramObjectCollection<Palette>()
{
new Palette() { Symbols = PaletteGroup, Title = "Group Shapes", IsExpanded =
true }
};
}
private void CreatePaletteGroup()
{
Node node1 = new Node()
{
ID = "node1",
Width = 50,
Height = 50,
OffsetX = 100,
OffsetY = 100,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle },
Style = new ShapeStyle() { Fill = "#6495ed" },
};
Node node2 = new Node()
{
ID = "node2",
Width = 50,

```

```

Height = 50,
OffsetX = 100,
OffsetY = 200,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Ellipse },
Style = new ShapeStyle() { Fill = "#6495ed" },
};
PaletteGroup.Add(node1);
PaletteGroup.Add(node2);
NodeGroup group = new NodeGroup()
{
ID = "group1",
Children = new string[] { "node1", "node2" }
};
PaletteGroup.Add(group);
}
}

```



#### Add palette to SymbolPalette

A palette allows to display a group of related symbols and it textually annotates the group with its header.

A [Palette](#) can be added as a collection of symbol groups.

The collection of predefined symbols can be added in palettes using the [Symbols](#) property.

To initialize a palette, define a JSON object with the property [ID](#) that is unique ID is set to the palettes.

The following code example illustrates how to define a palette.

#### CSHARP

```

@using Syncfusion.Blazor.Diagram.SymbolPalette
@using Syncfusion.Blazor.Diagram
/* Initializes the symbol palette */

```

```

<SfSymbolPaletteComponent @ref="SymbolPalette" Height="600px"
SymbolHeight="80" SymbolWidth="80" Palettes="@palettes">
</SfSymbolPaletteComponent>
@code
{
SfSymbolPaletteComponent SymbolPalette;
DiagramObjectCollection<Palette> palettes = new
DiagramObjectCollection<Palette>();
}

```

The following code example illustrates how to add nodes, connectors, nodegroups to the palette and add palette to the palettes collection of the symbol palette.

### CSHARP

```

Palettes = new DiagramObjectCollection<Palette>()
{
new Palette() { Symbols = PaletteNodes, Title = "Flow Shapes", ID = "Flow
Shapes" },
new Palette() { Symbols = PaletteConnectors, Title = "Connectors", IsExpanded
= true},
new Palette() { Symbols = PaletteGroup, Title = "Group Shapes", IsExpanded =
true}
};

```

- Complete code to render palette with node, connector and nodegroup.

### CSHARP

```

@using Syncfusion.Blazor.Diagram
@using Syncfusion.Blazor.Diagram.SymbolPalette
<div class="control-section">
<div style="width:20%">
<div id="palette-space" class="sb-mobile-palette" style="border: 2px solid
#b200ff">
<SfSymbolPaletteComponent @ref="@symbolpalette" Height="300px" Width="200px"
Palettes="@Palettes" SymbolHeight="60" SymbolWidth="60"
SymbolMargin="@SymbolMargin">
</SfSymbolPaletteComponent>
</div>
</div>
</div>
@code
{
SymbolMargin SymbolMargin = new SymbolMargin
{
Left = 15,
Right = 15,
Top = 15,
Bottom = 15
};
SfSymbolPaletteComponent symbolpalette;
//Define palattes collection
DiagramObjectCollection<Palette> Palettes = new
DiagramObjectCollection<Palette>();

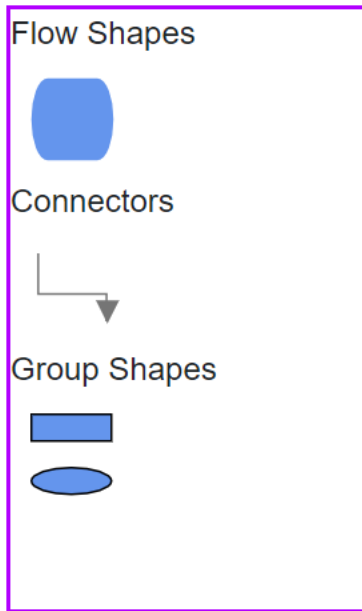
```

```

// Defines palette's flow-shape collection
DiagramObjectCollection<NodeBase> PaletteNodes = new
DiagramObjectCollection<NodeBase>();
// Defines palette's group collection
DiagramObjectCollection<NodeBase> PaletteGroup = new
DiagramObjectCollection<NodeBase>();
// Defines palette's connector collection
DiagramObjectCollection<NodeBase> PaletteConnectors = new
DiagramObjectCollection<NodeBase>();
protected override void OnInitialized()
{
    InitPaletteModel();
}
private void InitPaletteModel()
{
    CreatePaletteNode(FlowShapeType.Terminator, "Terminator");
    CreatePaletteConnector("Link1", ConnectorSegmentType.Orthogonal,
    DecoratorShape.Arrow);
    CreatePaletteGroup();
    Palettes = new DiagramObjectCollection<Palette>()
    {
        new Palette() { Symbols = PaletteNodes, Title = "Flow Shapes", ID = "Flow
        Shapes" },
        new Palette() { Symbols = PaletteConnectors, Title = "Connectors", IsExpanded =
        true },
        new Palette() { Symbols = PaletteGroup, Title = "Group Shapes", IsExpanded =
        true }
    };
}
private void CreatePaletteNode(FlowShapeType flowShape, string id)
{
    Node node = new Node()
    {
        ID = id,
        Shape = new FlowShape() { Type = Shapes.Flow, Shape = flowShape },
        Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" },
    };
    PaletteNodes.Add(node);
}
private void CreatePaletteConnector(string id, ConnectorSegmentType type,
DecoratorShape decoratorShape)
{
    Connector connector = new Connector()
    {
        ID = id,
        Type = type,
        SourcePoint = new DiagramPoint() { X = 0, Y = 0 },
        TargetPoint = new DiagramPoint() { X = 60, Y = 60 },
        Style = new ShapeStyle() { StrokeWidth = 1, StrokeColor = "#757575" },
        TargetDecorator = new DecoratorSettings()
        {
            Shape = decoratorShape,
            Style = new ShapeStyle() { StrokeColor = "#757575", Fill = "#757575" }
        }
    };
    PaletteConnectors.Add(connector);
}

```

```
private void CreatePaletteGroup()
{
    Node node1 = new Node()
    {
        ID = "node1",
        Width = 50,
        Height = 50,
        OffsetX = 100,
        OffsetY = 100,
        Shape = new BasicShape() { Type = Shapes.Basic, Shape =
        BasicShapeType.Rectangle },
        Style = new ShapeStyle() { Fill = "#6495ed" },
    };
    Node node2 = new Node()
    {
        ID = "node2",
        Width = 50,
        Height = 50,
        OffsetX = 100,
        OffsetY = 200,
        Shape = new BasicShape() { Type = Shapes.Basic, Shape =
        BasicShapeType.Ellipse },
        Style = new ShapeStyle() { Fill = "#6495ed" },
    };
    PaletteGroup.Add(node1);
    PaletteGroup.Add(node2);
    NodeGroup group = new NodeGroup()
    {
        ID = "group1",
        Children = new string[] { "node1", "node2" }
    };
    PaletteGroup.Add(group);
}
```



How to drag and drop symbols from palette to diagram

To initialize drag and drop, you must add the diagram to the [Targets](#) collection of the symbol palette. The below code illustrates how to add diagram to the Targets collection.

#### CSHARP

```
@code
{
    SfDiagramComponent diagram;
    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        symbolpalette.Targets = new DiagramObjectCollection<SfDiagramComponent>() {
        };
        symbolpalette.Targets.Add(diagram);
    }
}
```

- Complete code to drag and drop symbols from palette to diagram.

#### CSHARP

```
@using Syncfusion.Blazor.Diagram
@using Syncfusion.Blazor.SymbolPalette
<div class="control-section">
<div style="width: 100%">
<div class="sb-mobile-palette-bar">
<div id="palette-icon" style="float: right;" role="button" class="e-ddb-
icons1 e-toggle-palette"></div>
</div>
<div id="palette-space" class="sb-mobile-palette">
<SfSymbolPaletteComponent @ref="@symbolpalette" Height="700px"
Palettes="@Palettes" SymbolHeight="60" SymbolWidth="60"
SymbolMargin="@SymbolMargin">
```

```

</SfSymbolPaletteComponent>
</div>
<div id="diagram-space" class="sb-mobile-diagram">
<div class="content-wrapper" style="border: 1px solid #D7D7D7">
<SfDiagramComponent @ref="@diagram" Height="700px" Connectors="@connectors"
Nodes="@nodes">
</SfDiagramComponent>
</div>
</div>
</div>
</div>
@code
{
SymbolMargin SymbolMargin = new SymbolMargin
{
Left = 15,
Right = 15,
Top = 15,
Bottom = 15
};
SfDiagramComponent diagram;
SfSymbolPaletteComponent symbolpalette;
//Define nodes collection
DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
//Define connectors collection
DiagramObjectCollection<Connector> connectors = new
DiagramObjectCollection<Connector>();
//Define palattes collection
DiagramObjectCollection<Palette> Palettes = new
DiagramObjectCollection<Palette>();
// Defines palette's flow-shape collection
DiagramObjectCollection<NodeBase> PaletteNodes = new
DiagramObjectCollection<NodeBase>();
// Defines palette's group collection
DiagramObjectCollection<NodeBase> PaletteGroup = new
DiagramObjectCollection<NodeBase>();
// Defines palette's connector collection
DiagramObjectCollection<NodeBase> PaletteConnectors = new
DiagramObjectCollection<NodeBase>();
protected override async Task OnAfterRenderAsync(bool firstRender)
{
symbolpalette.Targets = new DiagramObjectCollection<SfDiagramComponent>() {
};
symbolpalette.Targets.Add(diagram);
}
protected override void OnInitialized()
{
InitPaletteModel();
}
private void InitPaletteModel()
{
CreatePaletteNode(FlowShapeType.Terminator, "Terminator");
CreatePaletteConnector("Link1", ConnectorSegmentType.Orthogonal,
DecoratorShape.Arrow);
CreatePaletteGroup();
Palettes = new DiagramObjectCollection<Palette>()
{

```

```

new Palette() { Symbols = PaletteNodes, Title = "Flow Shapes", ID = "Flow
Shapes" },
new Palette() { Symbols = PaletteConnectors, Title = "Connectors", IsExpanded =
true},
new Palette() { Symbols = PaletteGroup, Title = "Group Shapes", IsExpanded =
true}
};
}
private void CreatePaletteNode(FlowShapeType flowShape, string id)
{
Node node = new Node()
{
ID = id,
Shape = new FlowShape() { Type = Shapes.Flow, Shape = flowShape },
Style = new ShapeStyle() { Fill= "#6495ED", StrokeColor = "#6495ED" },
};
PaletteNodes.Add(node);
}
private void CreatePaletteConnector(string id, ConnectorSegmentType type,
DecoratorShape decoratorShape)
{
Connector connector = new Connector()
{
ID = id,
Type = type,
SourcePoint = new DiagramPoint() { X = 0, Y = 0 },
TargetPoint = new DiagramPoint() { X = 60, Y = 60 },
Style = new ShapeStyle() { StrokeWidth = 1, StrokeColor = "#757575" },
TargetDecorator = new DecoratorSettings()
{
Shape = decoratorShape,
Style = new ShapeStyle() { StrokeColor = "#757575", Fill = "#757575" }
}
};
PaletteConnectors.Add(connector);
}
private void CreatePaletteGroup()
{
Node node1 = new Node()
{
ID = "node1",
Width = 50,
Height = 50,
OffsetX = 100,
OffsetY = 100,
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle },
Style = new ShapeStyle() { Fill = "#6495ed" },
};
Node node2 = new Node()
{
ID = "node2",
Width = 50,
Height = 50,
OffsetX = 100,
OffsetY = 200,

```



```
Shape = new BasicShape() { Type = Shapes.Basic, Shape =  
BasicShapeType.Ellipse },  
Style = new ShapeStyle() { Fill = "#6495ed" },  
};  
PaletteGroup.Add(node1);  
PaletteGroup.Add(node2);  
NodeGroup group = new NodeGroup()  
{  
ID = "group1",  
Children = new string[] { "node1", "node2" }  
};  
PaletteGroup.Add(group);  
}  
}
```



#### Customize the palette header

Palettes can be annotated with its header texts.

The [Title](#) displayed as the header text of palette.

The [IsExpanded](#) property of palette allows to expand/collapse its palette items.

The following code illustrates how to change the Title and IsExpanded properties at runtime.

#### CSHARP

```
symbolpalette.Palettes[0].Title = "NewTitle";  
symbolpalette.Palettes[0].IsExpanded = false;
```

### Add/Remove symbols to palette at runtime

- Symbols can be added to palette at runtime by using public method, [AddPalettetItem](#). The following code sample illustrates how to add symbol using AddPalettetItem method.

#### CSHARP

```
Node decision = new Node()
{
    ID = "Decision",
    Shape = new FlowShape() { Type = Shapes.Flow, Shape = FlowShapeType.Decision
}
};
symbolpalette.AddPalettetItem("Flow Shapes", decision, false);
```

Also, you can add symbol to the palette at runtime by using **Add** method. The following code sample illustrates how to add symbol using Add method.

#### CSHARP

```
Node decision = new Node()
{
    ID = "Decision",
    Shape = new FlowShape() { Type = Shapes.Flow, Shape = FlowShapeType.Decision
}
};
symbolpalette.Palettes[0].Symbols.Add(Tnode2);
```

- Symbols can be removed from palette at runtime by using public method, [RemovePalettetItem](#). The following code sample illustrates how to remove symbol using RemovePalettetItem method.

#### CSHARP

```
symbolpalette.RemovePalettetItem("Flow Shapes", "Decision");
```

### Add/Remove palettes at runtime

- Palettes can be added to the symbol palette at runtime by using public method, [AddPalettes](#). The following code sample illustrates how to add palette using AddPalettes method.

#### CSHARP

```
DiagramObjectCollection<NodeBase> newNodees = new
DiagramObjectCollection<NodeBase>();
Node newNode = new Node()
{
    ID = "newNode",
    Shape = new FlowShape() { Type = Shapes.Flow, Shape = FlowShapeType.Process
}
};
newNodes.Add(newNode as NodeBase);
```

```
DiagramObjectCollection<Palette> newPalettes = new
DiagramObjectCollection<Palette>()
{
    new Palette() { Symbols = newNodes, Title = "FlowShapes", ID = "FlowShapes" },
};
symbolpalette.AddPalettes(newPalettes);
```

Also, you can add palette to the symbol palette at runtime by using `Add` method. The following code sample illustrates how to add palette using `Add` method.

### CSHARP

```
DiagramObjectCollection<NodeBase> Newnodes = new
DiagramObjectCollection<NodeBase>();
Newnodes = new DiagramObjectCollection<NodeBase>();
Node newNode = new Node()
{
    ID = "newNode",
    Shape = new FlowShape() { Type = Shapes.Flow, Shape = FlowShapeType.Process
};
Newnodes.Add(newNode as NodeBase);
Palette newpalette = new Palette()
{
    Symbols = Newnodes,
    Title = "Flow Shapes",
    ID = "Flow Shapes"
};
symbolpalette.Palettes.Add(newpalette);
```

- Palettes can be removed from the symbol palette at runtime by using public method, [RemovePalettes](#). The following code sample illustrates how to remove palette using `RemovePalettes` method.

### CSHARP

```
symbolpalette.RemovePalettes("Basic Shapes");
```

### Customize the size of symbols

The size of the individual symbol can be customized. The [SymbolWidth](#) and [SymbolHeight](#) properties of symbol palette enables you to define the size of the symbols.

- Also, you can update the size of the symbols at runtime.

The following code example illustrates how to change the size of a symbol and how to update the size at runtime.

### CSHARP

```
@using Syncfusion.Blazor.Diagram
@using Syncfusion.Blazor.Diagram.SymbolPalette
<div class="control-section">
```

```

<div class="properties">
<button @onclick="UpdateSize">
UpdateSize
</button>
</div>
<div style="width:20%">
<div id="palette-space" class="sb-mobile-palette" style="border: 2px solid
#b200ff">
<SfSymbolPaletteComponent @ref="@symbolpalette" Height="300px" Width="200px"
Palettes="@Palettes" SymbolHeight="@symbolwidth" SymbolWidth="@symbolheight"
SymbolMargin="@SymbolMargin">
</SfSymbolPaletteComponent>
</div>
</div>
</div>
@code
{
DiagramSize SymbolPreview;
SymbolMargin SymbolMargin = new SymbolMargin
{
Left = 15,
Right = 15,
Top = 15,
Bottom = 15
};
double symbolwidth = 60;
double symbolheight = 60;
SfSymbolPaletteComponent symbolpalette;
//Define palattes collection
DiagramObjectCollection<Palette> Palettes = new
DiagramObjectCollection<Palette>();
// Defines palette's flow-shape collection
DiagramObjectCollection<NodeBase> PaletteNodes = new
DiagramObjectCollection<NodeBase>();
protected override void OnInitialized()
{
InitPaletteModel();
}
private void InitPaletteModel()
{
Node node1 = new Node()
{
ID = "Rectangle",
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Rectangle },
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" },
};
Node node2 = new Node()
{
ID = "Ellipse",
Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Ellipse },
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" },
};
Node node3 = new Node()
{
ID = "Diamond",

```

```

Shape = new BasicShape() { Type = Shapes.Basic, Shape =
BasicShapeType.Diamond },
Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" },
};
PaletteNodes.Add(node1);
PaletteNodes.Add(node2);
PaletteNodes.Add(node3);
Palettes = new DiagramObjectCollection<Palette>()
{
    new Palette() { Symbols = PaletteNodes, Title = "Basic Shapes", ID = "Basic
Shapes" },
};
}
private void UpdateSize()
{
    symbolwidth = 80;
    symbolheight = 80;
}
}

```

The [SymbolMargin](#) property is used to create the space around the elements, outside of any defined borders.

#### SymbolDragPreviewSize

The symbol preview size of the palette items can be customized using [SymbolDragPreviewSize](#) property.

The [Width](#) and [Height](#) properties of SymbolDragPreviewSize enables you to define the preview size to all the symbol palette items.

The following code example illustrates how to change the preview size of a palette item.

#### CSHARP

```

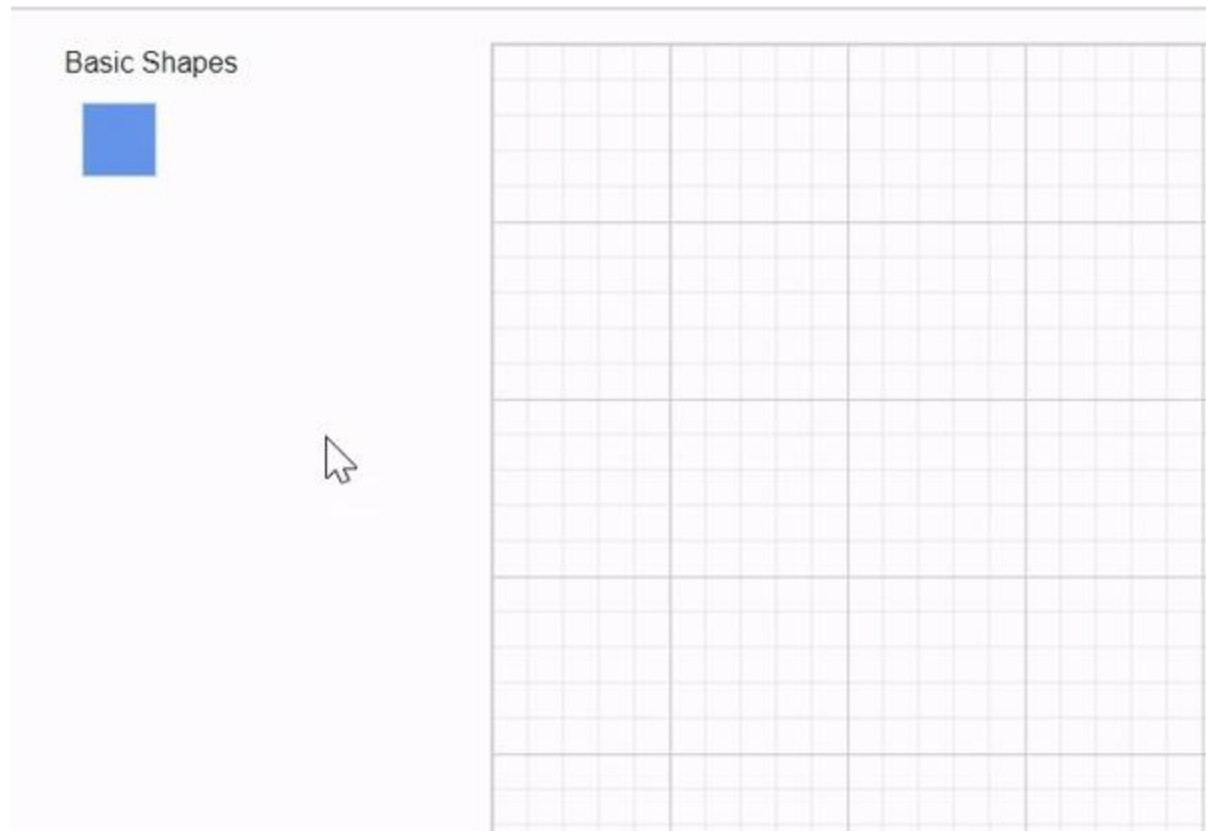
@using Syncfusion.Blazor.Diagram
@using Syncfusion.Blazor.Diagram.SymbolPalette
<div class="control-section">
<div style="width: 100%">
<div class="sb-mobile-palette-bar">
<div id="palette-icon" style="float: right;" role="button" class="e-ddb-
icons1 e-toggle-palette"></div>
</div>
<div id="palette-space" class="sb-mobile-palette">
<SfSymbolPaletteComponent @ref="@symbolpalette" Height="300px" Width="200px"
SymbolDragPreviewSize="@symbolDragPreviewSize"
Palettes="@Palettes" SymbolHeight="60" SymbolWidth="60"
SymbolMargin="@SymbolMargin">
</SfSymbolPaletteComponent>
</div>
<div id="diagram-space" class="sb-mobile-diagram">
<div class="content-wrapper" style="border: 1px solid #D7D7D7">
<SfDiagramComponent @ref="@diagram" Height="700px" Connectors="@connectors"
Nodes="@nodes">
</SfDiagramComponent>
</div>
</div>
</div>
</div>

```

```

@code
{
    Size symbolDragPreviewSize;
    SymbolMargin SymbolMargin = new SymbolMargin
    {
        Left = 15,
        Right = 15,
        Top = 15,
        Bottom = 15
    };
    SfDiagramComponent diagram;
    SfSymbolPaletteComponent symbolpalette;
    DiagramObjectCollection<Node> nodes = new DiagramObjectCollection<Node>();
    DiagramObjectCollection<Connector> connectors = new
    DiagramObjectCollection<Connector>();
    //Define palattes collection
    DiagramObjectCollection<Palette> Palettes = new
    DiagramObjectCollection<Palette>();
    // Defines palette's flow-shape collection
    DiagramObjectCollection<NodeBase> PaletteNodes = new
    DiagramObjectCollection<NodeBase>();
    protected override void OnInitialized()
    {
        InitPaletteModel();
    }
    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        symbolpalette.Targets = new DiagramObjectCollection<SfDiagramComponent>() {
        };
        symbolpalette.Targets.Add(diagram);
    }
    private void InitPaletteModel()
    {
        symbolDragPreviewSize = new DiagramSize();
        symbolDragPreviewSize.Width = 100;
        symbolDragPreviewSize.Height = 100;
        CreatePaletteNode(BasicShapeType.Rectangle, "Rectangle");
        Palettes = new DiagramObjectCollection<Palette>()
        {
            new Palette() { Symbols = PaletteNodes, Title = "Basic Shapes", ID = "Basic
            Shapes" },
        };
    }
    private void CreatePaletteNode(BasicShapeType basicShape, string id)
    {
        Node node = new Node()
        {
            ID = id,
            Shape = new BasicShape() { Type = Shapes.Basic, Shape = basicShape },
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" },
        };
        PaletteNodes.Add(node);
    }
}

```



### Default settings

While adding more number of symbols such as nodes and connectors to the palette, define the default settings for those objects through the [NodeCreating](#) and the [ConnectorCreating](#) properties of diagram allows to define the default settings for nodes and connectors.

### Adding symbol description for symbols in the palette

The diagram provides support to add symbol description below each symbol of a palette. This descriptive representation of each symbol will enhance the details of the symbol visually. The height and width of the symbol description can also be set individually.

- The method [GetSymbolInfo](#), can be used to add the symbol description at runtime.

The following code is an example to set a symbol description for symbols in the palette.

### CSHARP

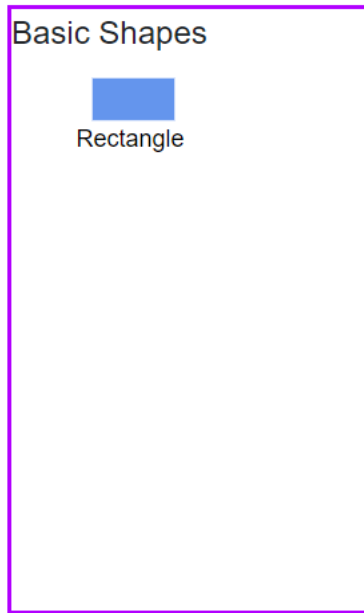
```
@using Syncfusion.Blazor.Diagram
@using Syncfusion.Blazor.Diagram.SymbolPalette
<div class="control-section">
<div style="width: 100%">
<div id="palette-space" class="sb-mobile-palette" style="border: 2px solid #b200ff">
<SfSymbolPaletteComponent @ref="@symbolpalette" Height="300px" Width="200px"
GetSymbolInfo="GetSymbolInfo"
Palettes="@Palettes" SymbolHeight="60" SymbolWidth="120"
SymbolMargin="@SymbolMargin">
</SfSymbolPaletteComponent>
```

```

</div>
</div>
</div>
@code
{
    SymbolMargin SymbolMargin = new SymbolMargin
    {
        Left = 15,
        Right = 15,
        Top = 15,
        Bottom = 15
    };
    SfSymbolPaletteComponent symbolpalette;
    //Define palattes collection
    DiagramObjectCollection<Palette> Palettes = new
    DiagramObjectCollection<Palette>();
    // Defines palette's flow-shape collection
    DiagramObjectCollection<NodeBase> PaletteNodes = new
    DiagramObjectCollection<NodeBase>();
    protected override void OnInitialized()
    {
        InitPaletteModel();
    }
    private void InitPaletteModel()
    {
        CreatePaletteNode(BasicShapeType.Rectangle, "Rectangle");
        Palettes = new DiagramObjectCollection<Palette>()
        {
            new Palette() { Symbols = PaletteNodes, Title = "Basic Shapes", ID = "Basic
            Shapes" },
        };
    }
    private void CreatePaletteNode(BasicShapeType basicShape, string id)
    {
        Node node = new Node()
        {
            ID = id,
            Shape = new BasicShape() { Type = Shapes.Basic, Shape = basicShape },
            Style = new ShapeStyle() { Fill = "#6495ED", StrokeColor = "#6495ED" },
        };
        PaletteNodes.Add(node);
    }
    private SymbolInfo GetSymbolInfo(IDiagramObject symbol)
    {
        SymbolInfo SymbolInfo = new SymbolInfo();
        string text = null;
        text = (symbol as Node).ID;
        SymbolInfo.Description = new SymbolDescription() { Text = text };
        return SymbolInfo;
    }
}

```





#### Palette interaction

Palette interaction notifies the element enter, leave, and dragging of the symbols into the diagram.

#### Escape Key function

The diagram provides support to cancel the node drop from symbol palette when the ESC key is pressed.

#### See Also

- [How to add the symbol to the diagram](#)

## Dialog

### Getting Started with Blazor Dialog Component

This section briefly explains how to include a Dialog component in your Blazor Server-side application. You can refer to our Getting Started with [Syncfusion Blazor Server-Side Popups in Visual Studio page](#) for the introduction and configuring the common specifications.

To get start quickly with Blazor Dialog component, you can check on this video:

{% youtube

"youtube:https://www.youtube.com/watch?v=uSyGKuB8ghg"%}

#### Importing Syncfusion Blazor component in the application

- Install **Syncfusion.Blazor.Popups** 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.

#### 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.Popups** package.

### ASPX-CS

```
@using Syncfusion.Blazor.Popups
```

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();
        }
    }
}
```

### Add Dialog component

To initialize the Dialog component, add the below code to your **Index.razor** view page which is present under **~/Pages** folder.

The following code explains how to initialize a simple Dialog in Razor page.

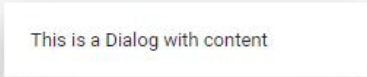
#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog Width="250px">
  <DialogTemplates>
    <Content> This is a Dialog with content </Content>
  </DialogTemplates>
</SfDialog>
```

### Run the application

After successful compilation of your application, simply run the application.

The output will be as follows.



This is a Dialog with content

---

\* In the dialog control, max-height is calculated based on the dialog target element height. If the **Target** property is not configured, the **document.body** is considered as a target. Therefore, to show a dialog in proper height, you need to add min-height to the target element.

\* If the dialog is rendered based on the body, then the dialog will get the height based on its body element height. If the height of the dialog is larger than the body height, then the dialog's height will not be set. For this scenario, we can set the CSS style for the html and body to get the dialog height.

---

#### CSS

```
html, body {
  height: 100%;
}
```

### Modal dialog

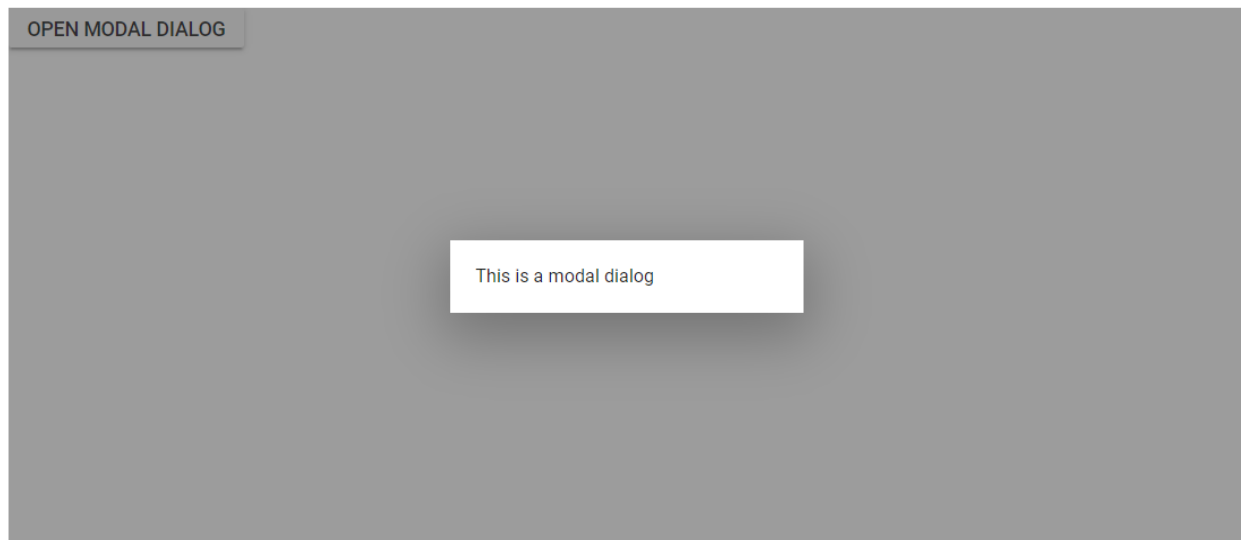
A **modal** shows an overlay behind the Dialog. So, the users should interact the Dialog compulsory before interacting with the remaining content in an application.

While the user clicks the overlay, the action can be handled through the `OnOverlayClick` event. In the following code, it explains the Dialog close action performed while clicking the overlay.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenDialog">Open Modal Dialog</SfButton>
<SfDialog Width="250px" IsModal="true" @bind-Visible="@IsVisible">
  <DialogEvents OnOverlayClick="@OnOverlayclick">
  </DialogEvents>
  <DialogTemplates>
    <Content> This is a modal dialog </Content>
  </DialogTemplates>
</SfDialog>
@code {
private bool IsVisible { get; set; } = true;
private void OpenDialog()
{
this.IsVisible = true;
}
private void OnOverlayclick(MouseEventArgs arg)
{
this.IsVisible = false;
}
}
```

The output will be as follows.



#### Enable header

The Dialog header can be enabled by adding the header content as text or HTML content using the `Header` template of the dialog.

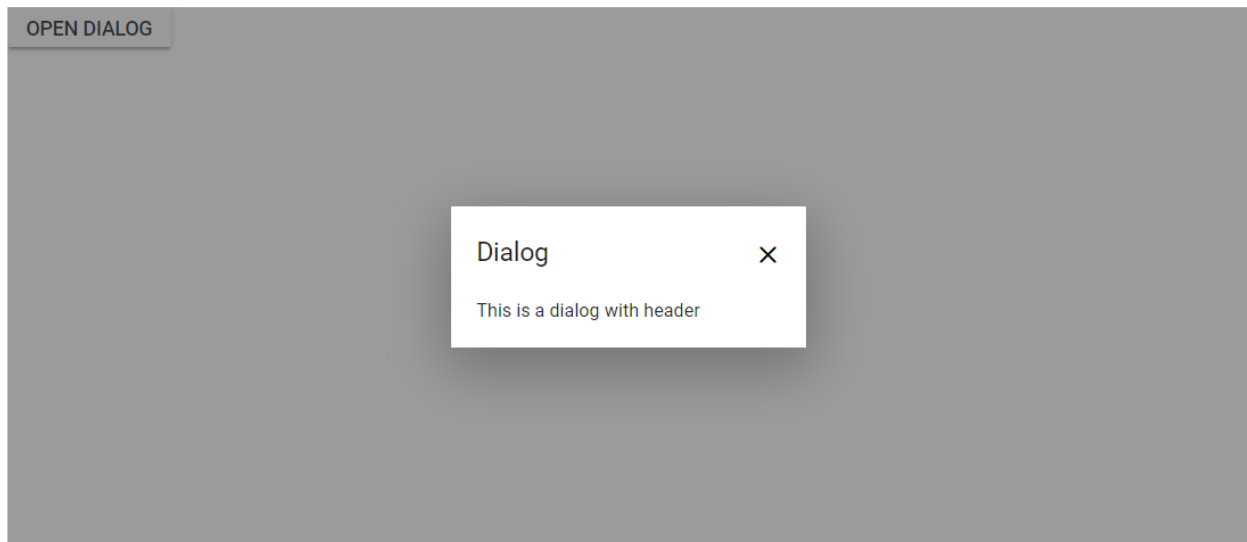
#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
```

```
<SfDialog Width="250px" ShowCloseIcon="true" IsModal="true" @bind-Visible="@IsVisible">
  <DialogTemplates>
    <Header> Dialog </Header>
    <Content> This is a dialog with header </Content>
  </DialogTemplates>
</SfDialog>

@code {
private bool IsVisible { get; set; } = true;
private void OpenDialog()
{
this.IsVisible = true;
}
}
```

The output will be as follows.



### Render Dialog with buttons

By adding the `DialogButtons` can render a Dialog with buttons in Razor page.

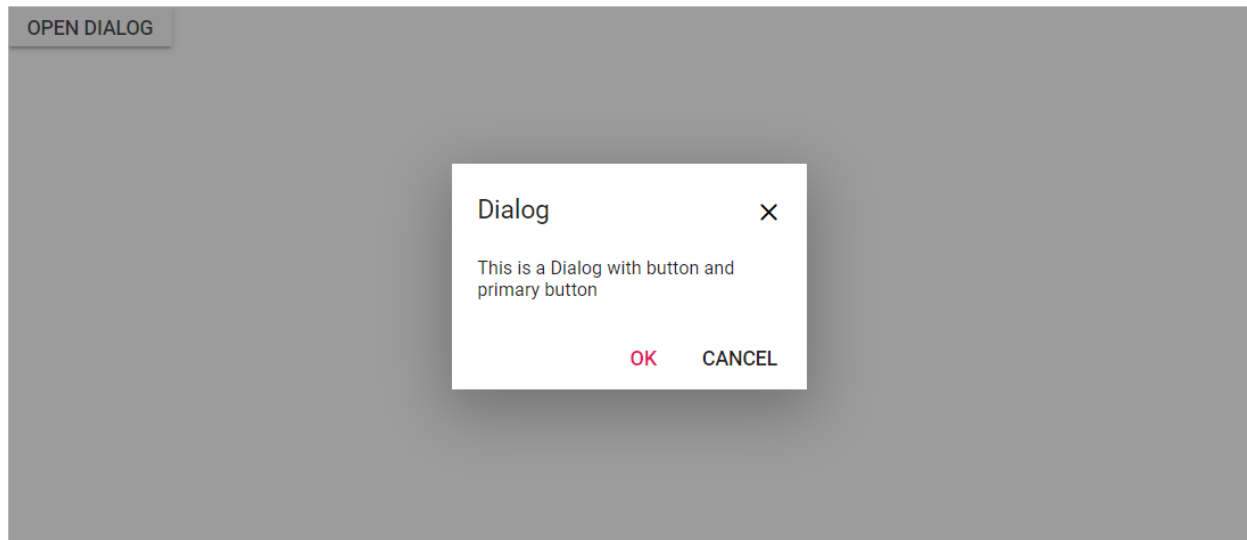
#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
<SfDialog Width="250px" ShowCloseIcon="true" IsModal="true" @bind-Visible="@IsVisible">
  <DialogTemplates>
    <Header> Dialog </Header>
    <Content> This is a Dialog with button and primary button </Content>
  </DialogTemplates>
  <DialogButtons>
    <DialogButton Content="OK" IsPrimary="true" OnClick="@CloseDialog" />
    <DialogButton Content="Cancel" OnClick="@CloseDialog" />
  </DialogButtons>
</SfDialog>

@code {
private bool IsVisible { get; set; } = true;
```

```
private void OpenDialog()
{
    this.IsVisible = true;
}
private void CloseDialog()
{
    this.IsVisible = false;
}
```

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](#)

## Draggable in Blazor Dialog Component

The Dialog supports to **drag** within its target container by grabbing the Dialog header, which allows the user to reposition the Dialog dynamically.

### ASPX-CS

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<div id="target">
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
<SfDialog Target="#target" Width="250px" AllowDragging="true" IsModal="true"
ShowCloseIcon="true" @bind-Visible="@IsVisible">
<DialogTemplates>
<Header> Dialog </Header>
<Content> This is a Dialog with drag enabled </Content>
</DialogTemplates>
<DialogButtons>
<DialogButton Content="OK" IsPrimary="true" OnClick="@CloseDialog" />
```

```

<DialogButton Content="Cancel" OnClick="@CloseDialog" />
</DialogButtons>
</SfDialog>
</div>
<style>
#target {
min-height: 400px;
height: 100%;
position: relative;
}
</style>
@code {
private bool IsVisible { get; set; } = true;
private void OpenDialog()
{
this.IsVisible = true;
}
private void CloseDialog()
{
this.IsVisible = false;
}
}

```

### Resizing in Blazor Dialog Component

The Dialog supports resizing feature. To resize the dialog, we have to select and resize it by using its handle (grip) or hovering on any of the edges or borders of the dialog within the sample container.

The resizable dialog can be created by setting the `EnableResize` property to true, which is used to change the size of a dialog dynamically and view its content with expanded mode. The `ResizeHandles` property can also be configured for all the which directions in which the dialog should be resized. When you configure the target property along with the `EnableResize` property, the dialog can be resized within its specified target container.

#### ASPX-CS

```

@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<div id="target">
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
<SfDialog Target="#target" Width="250px" AllowDragging="true"
ResizeHandles="@dialogResizeDirections" ShowCloseIcon="true" @bind-
Visible="@IsVisible">
<DialogTemplates>
<Header> Dialog </Header>
<Content> This is a Dialog with drag enabled </Content>
</DialogTemplates>
<DialogButtons>
<DialogButton Content="OK" IsPrimary="true" OnClick="@CloseDialog" />
<DialogButton Content="Cancel" OnClick="@CloseDialog" />
</DialogButtons>
</SfDialog>
</div>
<style>
#target {
min-height: 400px;

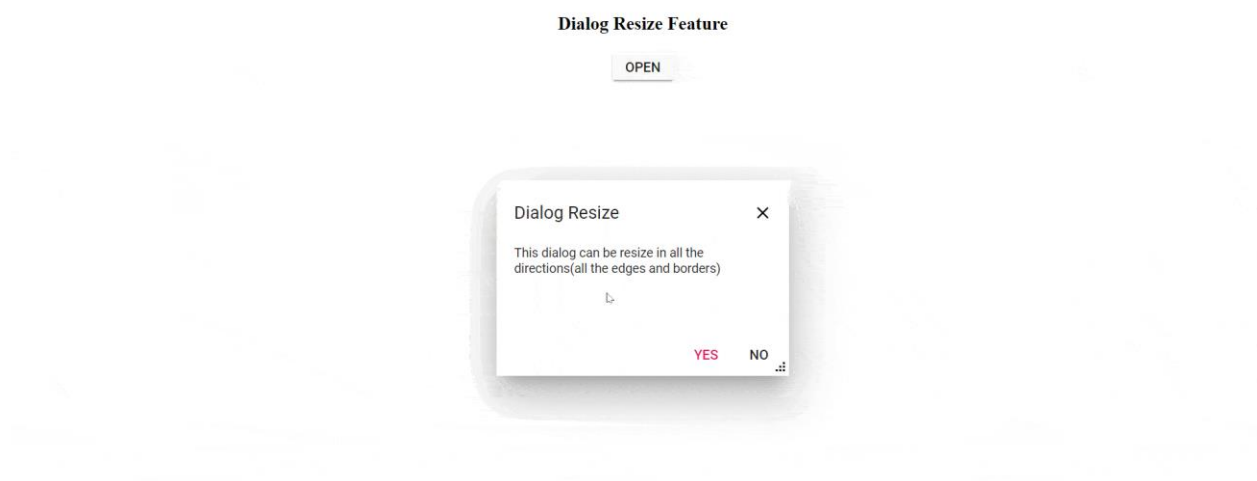
```

```

height: 100%;
position: relative;
}
</style>
@code {
private bool IsVisible { get; set; } = true;
private ResizeDirection[] dialogResizeDirections { get; set; } = new
ResizeDirection[] { ResizeDirection.All };
private void OpenDialog()
{
this.IsVisible = true;
}
private void CloseDialog()
{
this.IsVisible = false;
}
}

```

The output will be as follows.



### Positioning in Blazor Dialog Component

The Dialog can be positioned using the `DialogPositionData` property by providing the X and Y coordinates. It can be positioned inside the target of the container or `<body>` of the element based on the given X and Y values.

For X is: left, center, right (or) any offset value

For Y is: top, center, bottom (or) any offset value

The following code demonstrates the different Dialog positions.

#### ASPX-CS

```

@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<div id="target">
<div>
@if (this.ShowButton)
{
<SfButton Content="Open Dialog" @onclick="@OpenDialog"></SfButton>

```



```

}
</div>
<SfDialog ID="defaultDialog" Target="#target" Width="445px"
ShowCloseIcon="true" @bind-Visible="@Visibility">
<DialogTemplates>
<Header>
<div>Choose a Dialog Position</div>
</Header>
<Content>
<table style="width:405px;" id="poschange">
<tr>
<td>
<SfRadioButton ID="LeftTop" TChecked="string" Name="xy" Label="Left Top"
Value="left top" @bind-Checked="@Checked"
ValueChange="@OnChangeHandler"></SfRadioButton>
</td>
<td>
<SfRadioButton ID="CenterTop" TChecked="string" Name="xy" Label="Center Top"
Value="center top" @bind-Checked="@Checked"
ValueChange="@OnChangeHandler"></SfRadioButton>
</td>
<td>
<SfRadioButton ID="RightTop" TChecked="string" Name="xy" Label="Right Top"
Value="right top" @bind-Checked="@Checked"
ValueChange="@OnChangeHandler"></SfRadioButton>
</td>
</tr>
<tr>
<td>
<SfRadioButton ID="LeftCenter" TChecked="string" Name="xy" Label="Left
Center" Value="left center" @bind-Checked="@Checked"
ValueChange="@OnChangeHandler"></SfRadioButton>
</td>
<td>
<SfRadioButton ID="CenterCenter" TChecked="string" Name="xy" Label="Center
Center" Value="center center" @bind-Checked="@Checked"
ValueChange="@OnChangeHandler"></SfRadioButton>
</td>
<td>
<SfRadioButton ID="RightCenter" TChecked="string" Name="xy" Label="Right
Center" Value="right center" @bind-Checked="@Checked"
ValueChange="@OnChangeHandler"></SfRadioButton>
</td>
</tr>
<tr>
<td>
<SfRadioButton ID="LeftBottom" TChecked="string" Name="xy" Label="Left
Bottom" Value="left bottom" @bind-Checked="@Checked"
ValueChange="@OnChangeHandler"></SfRadioButton>
</td>
<td>
<SfRadioButton ID="CenterBottom" TChecked="string" Name="xy" Label="Center
Bottom" Value="center bottom" @bind-Checked="@Checked"
ValueChange="@OnChangeHandler"></SfRadioButton>
</td>
<td>

```

```

<SfRadioButton ID="RightBottom" TChecked="string" Name="xy" Label="Right
Bottom" Value="right bottom" @bind-Checked="@Checked"
ValueChange="@OnChangeHandler"></SfRadioButton>
</td>
</tr>
</table>
</Content>
<FooterTemplate><span>Position : { X: '@Xvalue', Y: '@Yvalue'
}</span></FooterTemplate>
</DialogTemplates>
<DialogPositionData X="@Xvalue" Y="@Yvalue"></DialogPositionData>
<DialogEvents OnOpen="@BeforeDialogOpen"
Closed="@DialogClosed"></DialogEvents>
</SfDialog>
</div>
<style>
#defaultDialog table,
#defaultDialog th,
#defaultDialog td {
border: 1px solid #D8D8D8;
border-collapse: collapse;
}
#defaultDialog.e-dialog .e-footer-content {
padding: 0px 10px 10px;
text-align: center;
}
#target {
min-height: 450px;
height: 100%;
}
.e-dialog .e-dlg-content {
padding: 10px 16px 10px;
}
.e-radio + label .e-label {
line-height: 18px;
}
td {
padding: 4px;
}
</style>
@code {
private string Xvalue = "center";
private string Yvalue = "center";
private bool Visibility { get; set; } = true;
private bool ShowButton { get; set; } = false;
private string Checked { get; set; } = "center center";
private void BeforeDialogOpen(BeforeOpenEventArgs args)
{
this.ShowButton = false;
}
private void DialogClosed(CloseEventArgs args)
{
this.ShowButton = true;
}
private void OpenDialog()
{
this.Visibility = true;
}
}

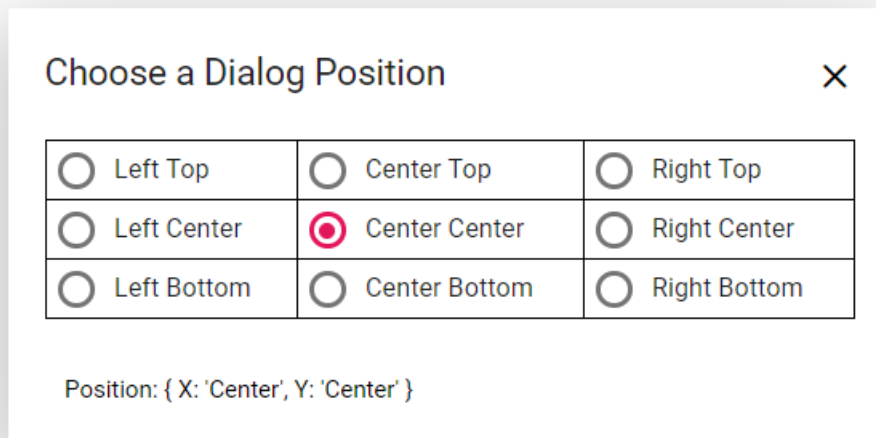
```

```

}
private void OnChangeHandler(ChangeArgs<string> arg)
{
    this.Xvalue = arg.Value.ToString().Split(' ')[0];
    this.Yvalue = arg.Value.ToString().Split(' ')[1];
    this.StateHasChanged();
}
}

```

The output will be as follows.



## Templates in Blazor Dialog Component

In Dialog, the template support is provided to the header and footer sections. So any text or HTML content can be appended in these sections.

### Header

The Dialog header content can be provided through the `Header` property, and it will allow both text and any HTML content as a string. Also in header, close button is provided as built-in support, and this can be enabled through the `ShowCloseIcon` property.

### Footer

The Dialog footer can be enabled by adding built-in `DialogButton` or providing any HTML string through the `FooterTemplate`.

---

The `DialogButton` and `FooterTemplate` properties can't be used at the same time.

---

The following example demonstrates the usage of header and footer template in the Dialog.

### ASPX-CS

```

@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Inputs
<div id="target" class="col-lg-12 control-section" style="height:100%">
<div>
@if (this.ShowButton)

```

```

{
<button class="e-btn" @onclick="@OpenDialog">Open Dialog</button>
}
</div>
<SfDialog Height="75%" Width="435px" Target="#target" ShowCloseIcon="true"
@bind-Visible="Visibility">
<DialogTemplates>
<Header>
<span class="e-avatar template-image e-avatar-xsmall e-avatar-
circle"></span>
<div id="template" title="Nancy" class="e-icon-settings">Nancy</div>
</Header>
<Content>
<div class="dialogContent">
<span class="dialogText">@DialogText</span>
</div>
</Content>
<FooterTemplate>
<SfTextBox ID="inVal" @bind-Value="@TextBoxValue" @ref="TextboxObj"
Type="InputType.Text" Placeholder="Enter your message here!" />
<button id="sendButton" @onclick="@SendBtnClicked" class="e-control e-btn e-
primary" data-ripple="true">Send</button>
</FooterTemplate>
</DialogTemplates>
<DialogEvents OnOpen="@BeforeDialogOpen"
Closed="@DialogClosed"></DialogEvents>
</SfDialog>
</div>
@code{
SfTextBox TextboxObj;
private string TextBoxValue;
private bool Visibility { get; set; } = true;
private bool ShowButton { get; set; } = false;
private string DialogText = "Greetings Nancy! When will you share me the
source files of the project?";
private void BeforeDialogOpen(BeforeOpenEventArgs args)
{
this.ShowButton = false;
}
private void DialogClosed(CloseEventArgs args)
{
this.ShowButton = true;
}
private void OpenDialog()
{
this.Visibility = true;
}
private void SendBtnClicked()
{
if (this.TextboxObj.Value != "")
{
DialogText = this.TextboxObj.Value;
TextBoxValue = "";
this.StateHasChanged();
}
}
}
}

```

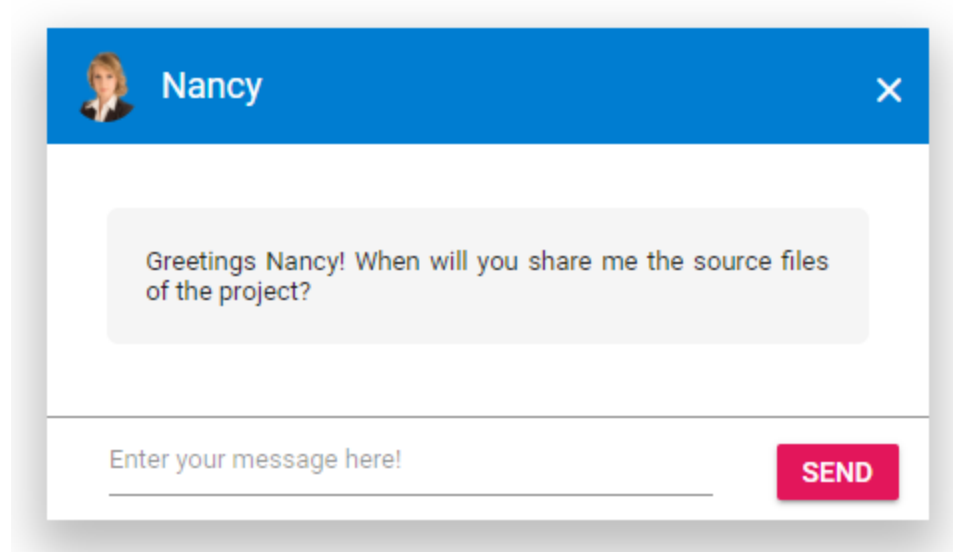
```
<style>
#sendButton {
top: 2px;
position: relative;
}
.e-footer-content .e-input-group {
width: 75%;
float: left;
}
#target {
min-height: 400px;
}
.e-dialog .e-dlg-header-content {
background-color: #3f51b5;
}
.e-dialog .e-dlg-header-content .e-btn.e-dlg-closeicon-btn {
top: 5px;
left: -11px;
}
.e-dialog .e-dlg-header {
position: relative;
}
.e-dialog .e-footer-content {
padding: 15px;
}
.e-dialog .e-dlg-content {
padding: 0;
}
.e-dialog .e-dlg-header-content {
padding: 6px;
}
.e-open-icon::before {
content: "\e782";
}
#template {
display: inline-block;
padding: 0px 10px;
vertical-align: middle;
height: 40px;
line-height: 40px;
}
input {
width: 75%;
float: left;
}
.e-icon-settings.e-icons {
float: left;
position: relative;
left: 14%;
top: -33px;
}
.dialogContent .dialogText {
font-size: 13px;
padding: 5%;
word-wrap: break-word;
border-radius: 6px;
text-align: justify;
}
```

```

font-style: initial;
display: block;
}
.e-dlg-header .e-icon-settings, .e-icon-btn {
color: #fff;
}
.dialogContent .dialogText, .dialogContent .dialogText {
background-color: #f5f5f5;
}
.e-dialog .e-footer-content {
border-top: 0.5px solid rgba(0, 0, 0, 0.42);
}
.dialogContent {
display: block;
font-size: 15px;
word-wrap: break-word;
text-align: center;
font-style: italic;
border-radius: 6px;
padding: 3%;
position: relative;
top: 25px;
}
.bootstrap .dialogContent {
top: 7px;
}
.control-wrapper .e-control.e-dialog {
width: 30%;
}
.e-dialog .e-dlg-header-content .e-icon-dlg-close {
color: #fff;
}
.e-dialog .e-dlg-header-content .e-btn.e-dlg-closeicon-btn:hover,
.e-dialog .e-dlg-header-content .e-btn.e-dlg-closeicon-btn:focus {
background-color: rgba(255,255,255, 0.10);
}
.e-dialog .e-dlg-header-content .e-btn.e-dlg-closeicon-btn:active .e-icon-
dlg-close,
.e-dialog .e-dlg-header-content .e-btn.e-dlg-closeicon-btn:focus .e-icon-
dlg-close,
.e-dialog .e-dlg-header-content .e-btn.e-dlg-closeicon-btn:hover .e-icon-
dlg-close {
color: #fff;
}
.e-dialog .e-dlg-header-content .e-dlg-header .e-avatar.template-image {
background-image:
url("https://ej2.syncfusion.com/demos/src/dialog/images/1.png");
vertical-align: middle;
display: inline-block;
width: 36px;
height: 36px;
}
</style>

```

The output will be as follows.



See Also

- [How to add an icon to dialog buttons](#)
- [How to customize the dialog appearance](#)

### Animation in Blazor Dialog Component

The Dialog can be animated during the open and close actions. Also, users can customize animation's Delay, Duration and Effect by using the DialogAnimationSettings property.

<!-- markdownlint-disable MD033 -->

delay	The Dialog animation will start with the mentioned delay
duration	Specifies the animation duration to complete with one animation cycle
effect	<p>Specifies the animation effects of Dialog open and close actions effect.</p> <p>List of supported animation effects:</p> <p>'Fade'   'FadeZoom'   'FlipLeftDown'   'FlipLeftUp'   'FlipRightDown'   'FlipRightUp'   'FlipXDown'   'FlipXUp'   'FlipYLeft'   'FlipYRight'   'SlideBottom'   'SlideLeft'   'SlideRight'   'SlideTop'   'Zoom'   'None'</p> <p>If the user sets 'Fade' effect, then the Dialog will open with 'FadeIn' effect and close with 'FadeOut' effect</p>

In the following sample, Zoom effect is enabled. So, The Dialog will open with ZoomIn and close with ZoomOut effects.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
```

```
<SfDialog Width="500px" ShowCloseIcon="true" @bind-Visible="@IsVisible">
  <DialogTemplates>
    <Header> Dialog </Header>
    <Content> Dialog enabled with Zoom effect </Content>
  </DialogTemplates>
  <DialogAnimationSettings Effect="@AnimationEffect" Duration=400 />
  <DialogButtons>
    <DialogButton Content="Hide" IsPrimary="true" OnClick="@CloseDialog" />
  </DialogButtons>
</SfDialog>

@code {
  private bool IsVisible { get; set; } = true;
  private DialogEffect AnimationEffect = DialogEffect.Zoom;
  private void OpenDialog()
  {
    this.IsVisible = true;
  }
  private void CloseDialog()
  {
    this.IsVisible = false;
  }
}
```

### Style and appearance in Blazor Dialog Component

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

#### Customizing the dialog header

Use the following CSS to customize the dialog header properties.

##### CSS

```
.e-dialog .e-dlg-header {
  color: green;
  font-size: 20px;
  font-weight: normal;
}
```

#### Customizing the dialog content

Use the following CSS to customize the dialog content properties.

##### CSS

```
.e-dialog .e-dlg-content {
  color: red;
  font-size: 10px;
  font-weight: normal;
  line-height: normal;
}
```

#### Customizing modal dialog overlay

Use the following CSS to customize the modal dialog overlay.

##### CSS



```
.e-dlg-overlay {  
background-color: slategray;  
opacity: 0.6;  
}
```

### Customizing the dialog resize icon

Use the following CSS to customize the dialog resize icon.

#### CSS

```
/* To change the icon content */  
.e-dialog .e-south-east::before, .e-dialog .e-south-west::before {  
content: "\f047";  
}  
/* To set the icon pack */  
.e-dialog .e-resize-handle {  
font: normal normal normal 14px/1 FontAwesome;  
}
```

The above CSS demonstration uses the font awesome icon.

### Customizing the dialog close button

Use the following CSS to customize the dialog close button.

#### CSS

```
/* To specify font size and color */  
.e-dialog .e-btn .e-btn-icon.e-icon-dlg-close {  
font-size: 12px;  
color: red;  
}
```

### Customizing the dialog footer button

Use the following CSS to customize the dialog footer button.

#### CSS

```
/* To specify font color, background color and border color */  
.e-dialog .e-btn.e-flat, .e-css.e-btn.e-flat {  
background-color: transparent;  
border-color: transparent;  
color: blue;  
}
```

## Localization in Blazor Dialog Component

Localization library allows you to localize the default text content of Dialog. In Dialog, the close button's tooltip text alone will be localized based on the culture.

Use **Resource** file to translate the static text of the Dialog. 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 Blazor Server and WebAssembly project for Syncfusion Blazor components.

By using **Locale** property, you can set the culture dynamically in dialog component.

| Locale key | en-US (default) |

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

| Dialog\_Close | Close |

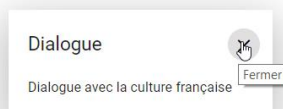
In the following sample, French culture is set to Dialog and change the close button's tooltip text.

### ASPX-CS

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
<SfDialog Width="250px" ShowCloseIcon="true" Locale="fr-CH" @bind-
Visible="@IsVisible">
  <DialogTemplates>
    <Header> Dialogue </Header>
    <Content> Dialogue avec la culture française </Content>
  </DialogTemplates>
</SfDialog>
@code {
private bool IsVisible { get; set; } = true;
private void OpenDialog()
{
this.IsVisible = true;
}
}
```

The output will be as follows.

OPEN DIALOG



### Accessibility in Blazor Dialog Component

The Dialog characterized with complete ARIA Accessibility support which helps to accessible by on-screen readers and other assistive technology devices. This component designed with the reference of the guidelines document given in [WAI ARIA Accessibility Practices](#).

The Dialog component uses the Dialog role and following ARIA properties to its element based on its state.

| Property | Functionalities |

| --- | --- |

| aria-describedby | Indicates the Dialog content description is notified to the users through assistive technologies. |

| aria-labelledby | The Dialog title is notified to the users through assistive technologies. |

| aria-modal | For modal dialog, it's value is true and non-modal dialog its value is false |

| aria-grabbed | Enables the draggable Dialog and starts dragging if its value is true and stops dragging if its value is false. |

### Keyboard interaction

Keyboard interaction of Dialog component has designed based on [WAI-ARIA Practices](#) described for Dialog. User can use the following shortcut keys to interact with the Dialog.

<!-- markdownlint-disable MD033 -->

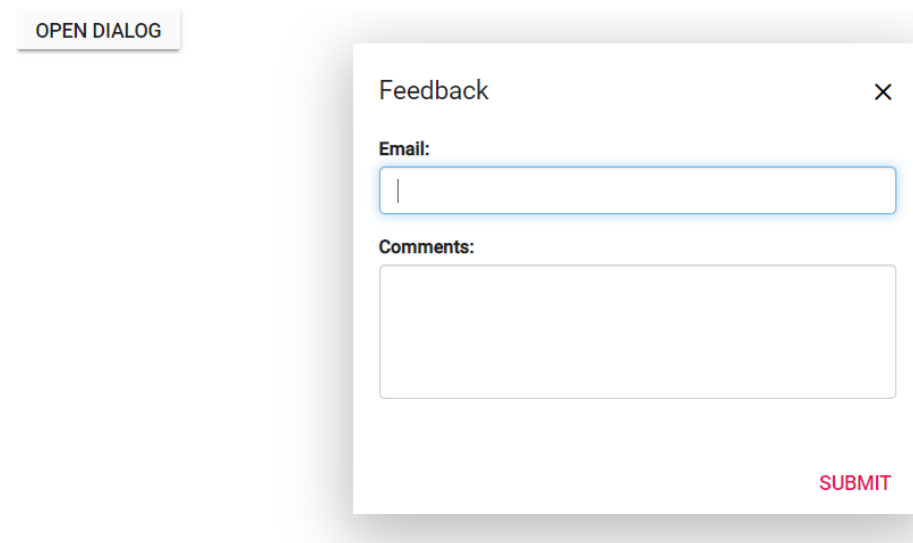
Keyboard shortcuts	Actions
Esc	Closes the Dialog. This functionality can be controlled by using closeOnEscape
Enter	When the Dialog button or any input (except text area) is in focus state, when pressing the Enter key, the click event associated with the primary button or button will trigger. Enter key is not working when the Dialog content contains any text area with initial focus
Ctrl + Enter	When the Dialog content contains text area and it is in focus state, and press the Ctrl + Enter key to call the click event function associated with primary button
Tab	Focus will be changed within the Dialog elements
Shift + Tab	The Focus will be changed previous focusable element within the Dialog elements. When focusing a first focusable element in the Dialog, then press the shift + tab key, it will change the focus to last focusable element

### ASPX-CS

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
<SfDialog Width="400px" Height="358px" ShowCloseIcon="true" @bind-Visible="@IsVisible">
  <DialogTemplates>
    <Header>
      <div>Feedback</div>
    </Header>
    <Content>
      <form>
        <div class='form-group'>
          <label for='email'>Email:</label>
          <input type='email' class='form-control' id='email'>
        </div>
        <div class='form-group'></div>
        <div class='form-group'>
          <label for='comment'>Comments:</label>
```

```
<textarea style='resize: none;' class='form-control' rows='4'
id='comment'></textarea>
</div>
</form>
</Content>
</DialogTemplates>
<DialogButtons>
<DialogButton Content="Submit" IsPrimary="true" OnClick="@CloseDialog" />
</DialogButtons>
</SfDialog>
@code {
private bool IsVisible { get; set; } = true;
private void OpenDialog()
{
this.IsVisible = true;
}
private void CloseDialog()
{
this.IsVisible = false;
}
}
```

The output will be as follows.



See Also

- [Show dialog with fullscreen](#)

## Events in Blazor Dialog Component

This section explains the list of events of the Dialog component which will be triggered for appropriate Dialog actions.

### Created

Created event triggers when the dialog is created.

### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog>
<DialogEvents Created="@CreatedHandler" ></DialogEvents>
</SfDialog>
@code{
public void CreatedHandler(Object args)
{
// Here you can customize your code
}
}
```

### Opened

**Opened** event triggers when a dialog is opened.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog>
<DialogEvents Opened="@OpenedHandler" ></DialogEvents>
</SfDialog>
@code{
public void OpenedHandler(OpenEventArgs args)
{
// Here you can customize your code
}
}
```

### OnOpen

**OnOpen** event triggers when the dialog is being opened.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog>
<DialogEvents OnOpen="@OnOpenHandler" ></DialogEvents>
</SfDialog>
@code{
public void OnOpenHandler(BeforeOpenEventArgs args)
{
// Here you can customize your code
}
}
```

### Closed

**Closed** event triggers after the dialog has been closed.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog>
<DialogEvents Closed="@ClosedHandler" ></DialogEvents>
</SfDialog>
@code{
public void ClosedHandler(CloseEventArgs args)
{
}
```

```
// Here you can customize your code
}
```

### OnClose

**OnClose** event triggers before the dialog is closed.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog>
<DialogEvents OnClose="@OnCloseHandler" ></DialogEvents>
</SfDialog>
@code{
public void OnCloseHandler(BeforeCloseEventArgs args)
{
// Here you can customize your code
}
}
```

### OnDragStart

**OnDragStart** event triggers when the user begins dragging the dialog.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog>
<DialogEvents OnDragStart="@OnDragStartHandler" ></DialogEvents>
</SfDialog>
@code{
public void OnDragStartHandler(DragStartEventArgs args)
{
// Here you can customize your code
}
}
```

### OnDragStop

**OnDragStop** event triggers when the user stop dragging the dialog.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog>
<DialogEvents OnDragStop="@OnDragStopHandler" ></DialogEvents>
</SfDialog>
@code{
public void OnDragStopHandler(DragStopEventArgs args)
{
// Here you can customize your code
}
}
```

### OnDrag

**OnDrag** event triggers when the user drags the dialog.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog>
  <DialogEvents OnDrag="@OnDragHandler" ></DialogEvents>
</SfDialog>
@code{
public void OnDragHandler(DragEventArgs args)
{
  // Here you can customize your code
}
}
```

### OnOverlayModalClick

**OnOverlayModalClick** event triggers when the overlay of dialog is clicked.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog>
  <DialogEvents OnOverlayModalClick ="@OnOverlayModalClickHandler"
  ></DialogEvents>
</SfDialog>
@code{
public void OnOverlayModalClickHandler(OverlayModalClickEventArgs args)
{
  // Here you can customize your code
}
}
```

### OnResizeStart

**OnResizeStart** event triggers when the user begins to resize a dialog.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog>
  <DialogEvents OnResizeStart="@OnResizeStartHandler" ></DialogEvents>
</SfDialog>
@code{
public void OnResizeStartHandler(MouseEventArgs args)
{
  // Here you can customize your code
}
}
```

### Resizing

**Resizing** event triggers when the user resize the dialog.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
```

```
<SfDialog>
<DialogEvents Resizing="@ResizingHandler" ></DialogEvents>
</SfDialog>
@code{
public void ResizingHandler(MouseEventArgs args)
{
// Here you can customize your code
}
}
```

### OnResizeStop

**OnResizeStop** event triggers when the user stop to resize a dialog.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog>
<DialogEvents OnResizeStop="@OnResizeStopHandler" ></DialogEvents>
</SfDialog>
@code{
public void OnResizeStopHandler(MouseEventArgs args)
{
// Here you can customize your code
}
}
```

### Destroyed

**Destroyed** event triggers when the dialog is destroyed.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<SfDialog>
<DialogEvents Destroyed="@DestroyedHandler" ></DialogEvents>
</SfDialog>
@code{
public void DestroyedHandler(Object args)
{
// Here you can customize your code
}
}
```

## How To

### Create nested Dialog in Blazor Dialog Component

A Dialog can be nested within another Dialog. The following sample contains parent and child Dialog (inner Dialog).

#### Step 1:

Create two dialog elements with ID **#dialog** and **#innerDialog**.

#### Step 2:

Initialize the Dialog as mentioned in the below sample.



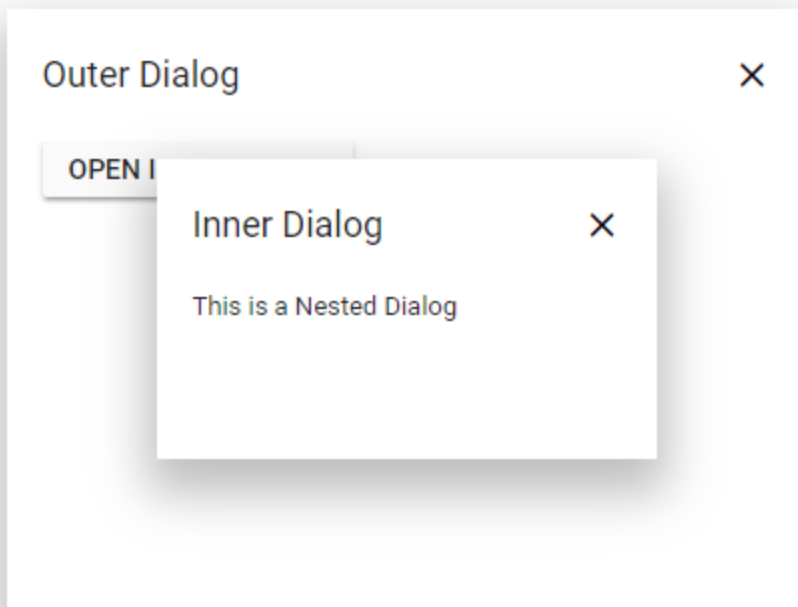
**Step 3:**

Set the inner Dialog target as `#dialog`.

**ASPX-CS**

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenParentDialog">Open Parent Dialog</SfButton>
<SfDialog ID="innerDialog" Width="250px" MinHeight="150px"
ShowCloseIcon="true" Target="@Target" @bind-Visible="@IsVisibleChild">
<DialogTemplates>
<Header> Inner Dialog </Header>
<Content> This is a Nested Dialog </Content>
</DialogTemplates>
<DialogPositionData X="center" Y="center"></DialogPositionData>
</SfDialog>
<SfDialog ID="dialog" Width="400px" Height="300px" ShowCloseIcon="true"
@bind-Visible="@IsVisibleParent">
<DialogTemplates>
<Header> Outer Dialog </Header>
<Content>
<SfButton @onclick="@OpenChildDialog">Open Inner Dialog</SfButton>
</Content>
</DialogTemplates>
<DialogPositionData X="center" Y="center"></DialogPositionData>
</SfDialog>
<style>
.content {
padding: 10px;
}
</style>
@code {
private string Target { get; set; } = "body";
private bool IsVisibleParent { get; set; } = true;
private bool IsVisibleChild { get; set; } = false;
private void OpenParentDialog()
{
this.IsVisibleParent = true;
}
private void OpenChildDialog()
{
this.Target = "#dialog.e-dialog";
this.IsVisibleChild = true;
}
}
```

The output will be as follows.



## Two-way binding using the visible property in Blazor Dialog Component

### *Two-way binding*

The **Visible** property is enabled by default and has two-way binding capabilities in Blazor dialog. To prevent the dialog from showing on-page load, set the property to **false** using the **@bind-Visible** attribute.

Bind the **Visible** property as mentioned below to show/hide the dialog on CheckBox state change.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<div id="target" class="col-lg-12 control-section">
<div class="row">
<SfCheckBox Label="Show/Hide Dialog" @bind-
Checked="@Visibility"></SfCheckBox>
</div>
<SfDialog Width="400px" ShowCloseIcon="true" @bind-Visible="@Visibility">
<DialogTemplates>
<Header> Update Required!!! </Header>
<Content> Would you like to install the latest updates? </Content>
</DialogTemplates>
<DialogButtons>
<DialogButton Content="Update" IsPrimary="true" OnClick="@CloseDialog" />
<DialogButton Content="Later" IsPrimary="false" OnClick="@CloseDialog" />
</DialogButtons>
</SfDialog>
</div>
<style>
#target {
min-height: 300px;
}
</style>
```

```
@code{
private bool Visibility { get; set; } = false;
private void CloseDialog()
{
this.Visibility = false;
}
}
```

### Position the Blazor Dialog in center of the page on scrolling

By default, when you scroll the page/container, Dialog will also scroll along with the page/container. To display the Dialog in the same position without scrolling, refer to the following code sample. Here added e-fixed class to Dialog element by using `CssClass` property and prevent the scrolling.

### ASPX-CS

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<div id="targetContainer" style="height: 900px;">
<b>JavaScript:</b><br />
JavaScript is a high-level, dynamic, untyped, and interpreted programming
language. It has been standardized in the ECMAScript
language specification. Alongside HTML and CSS, it is one of the three
essential technologies of World Wide Web
content production; the majority of websites employ it and it is supported
by all modern Web browsers without
plug-ins. JavaScript is prototype-based with first-class functions, making
it a multi-paradigm language, supporting
object - oriented , imperative, and functional programming styles.
<br /><br />
<b>MVC:</b><br />
Model-view-controller (MVC) is a software architecture pattern which
separates the representation of information from the user's interaction with
it. The model consists of application data, business rules, logic, and
functions. A view can be any output representation of data, such as a chart
or a diagram. Multiple views of the same data are possible, such as a bar
chart for management and a tabular view for accountants. The controller
mediates input, converting it to commands for the model or view. The central
ideas behind MVC are code reusability and in addition to dividing the
application into three kinds of components, the MVC design defines the
interactions between them.
A controller can send commands to its associated view to change the view's
presentation of the model (e.g., by scrolling through a document). It can
also send commands to the model to update the model's state (e.g., editing a
document).
A model notifies its associated views and controllers when there has been a
change in its state. This notification allows the views to produce updated
output, and the controllers to change the available set of commands. A
passive implementation of MVC omits these notifications, because the
application does not require them or the software platform does not support
them.
A view requests from the model the information that it needs to generate an
output representation to the user.
</div>
<SfDialog Width="250px" CssClass="@DialogClass" Target="#targetContainer">
<DialogTemplates>
<Header>Dialog</Header>
```

```

<Content>
<SfButton @onclick='@OnClicked'>Prevent Dialog Scroll</SfButton>
</Content>
</DialogTemplates>
</SfDialog>
<style>
body {
overflow-y: scroll;
}
.e-fixed {
position: fixed;
}
</style>
@code {
private string DialogClass { get; set; }
private void OnClicked()
{
this.DialogClass = "e-fixed";
}
}

```

### Render a dialog header dynamically in Blazor Dialog Component

By default, the dialog is rendered without header. You can update its header dynamically using the **Header** property.

In the following code, the dialog header is rendered on a button click.

#### ASPX-CS

```

@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
<SfButton @onclick="@RenderHeader">Render Dynamic Header</SfButton>
<SfDialog Header="@Header" Width="250px" @bind-Visible="@IsVisible">
<DialogTemplates>
<Content>This is a dialog without header</Content>
</DialogTemplates>
<DialogButtons>
<DialogButton Content="OK" IsPrimary="true" OnClick="@CloseDialog" />
<DialogButton Content="Cancel" OnClick="@CloseDialog" />
</DialogButtons>
</SfDialog>
@code {
private string Header { get; set; }
private bool IsVisible { get; set; } = true;
private void OpenDialog()
{
this.IsVisible = true;
}
private void CloseDialog()
{
this.IsVisible = false;
}
private void RenderHeader()
{
this.Header = "Dynamic Header";
}
}

```

```
}  
}
```

The output will be as follows.

OPEN DIALOG

This is a dialog without header

OK

CANCEL

### Show Dialog with fullscreen in Blazor Dialog Component

You can show the dialog in fullscreen by passing `true` as argument to the dialog `Show` method. By using `Visible` property you can prevent the dialog from showing initially.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups  
@using Syncfusion.Blazor.Buttons  
<SfButton @onclick="@OpenDialog">Open FullScreen Dialog</SfButton>  
<SfDialog @ref="DialogRef" Width="250px" ShowCloseIcon="true"  
Visible="false">  
  <DialogTemplates>  
    <Header> Dialog </Header>  
    <Content> This is a Dialog with button and primary button </Content>  
  </DialogTemplates>  
  <DialogButtons>  
    <DialogButton Content="OK" IsPrimary="true" OnClick="@CloseDialog" />  
    <DialogButton Content="Cancel" OnClick="@CloseDialog" />  
  </DialogButtons>  
</SfDialog>  
@code {  
  SfDialog DialogRef;  
  private async Task OpenDialog()  
  {  
    await this.DialogRef.ShowAsync(true);  
  }  
  private async Task CloseDialog()  
  {  
    await this.DialogRef.HideAsync();  
  }  
}
```

### Display a Dialog with custom position in Blazor Dialog Component

By default, the dialog is displayed in the center of the target container. The dialog position can be set using the `DialogPositionData` property by providing custom `X` and `Y` coordinates. The dialog can be positioned inside the target based on the given `X` and `Y` values.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
<div id="target">
<SfDialog Width="360px" Target="#target" Height="120px">
<DialogTemplates>
<Header> Position-01 </Header>
<Content> The dialog is positioned at {X: 420, Y: 14} coordinates </Content>
</DialogTemplates>
<DialogPositionData X="420" Y="14" />
</SfDialog>
<SfDialog Width="360px" Target="#target" Height="120px">
<DialogTemplates>
<Header> Position-02 </Header>
<Content> The dialog is positioned at {X: 420, Y: 240} coordinates
</Content>
</DialogTemplates>
<DialogPositionData X="420" Y="240" />
</SfDialog>
</div>
<style>
#target {
min-height: 400px;
}
</style>
```

The output will be as follows.



### Prevent closing of modal Dialog in Blazor Dialog Component

You can prevent closing of modal dialog by setting the `OnClose` event argument cancel value to `true`. In the following sample, the dialog is closed when you enter the username value with minimum 4 characters. Otherwise, it will not be closed.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
```

```

<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
<SfDialog ID="dialog" IsModal="true" Width="300px" @bind-
Visible="@IsVisible">
  <DialogTemplates>
    <Header>
      <div id="heading">Sign In</div>
    </Header>
    <Content>
      <div id='input-container'>
        <div class="e-float-input e-input-group">
          <input id="username" type="text" @bind="@UserName" required />
          <span class="e-float-line"></span>
          <label class="e-float-text">Username</label>
        </div>
      </div>
      <div class='form-group'>
        <div class="e-float-input e-input-group">
          <input id="password" type="text" @bind="@Password" required />
          <span class="e-float-line"></span>
          <label class="e-float-text">Password</label>
        </div>
      </div>
      @if (!string.IsNullOrEmpty(this.ErrorMessage))
      {
        <div class="error-container">
          @ErrorMessage
        </div>
      }
    </Content>
  </DialogTemplates>
  <DialogButtons>
    <DialogButton Content="Log In" IsPrimary="true" OnClick="@OnSubmit" />
  </DialogButtons>
  <DialogEvents OnClose="@Validation"></DialogEvents>
</SfDialog>
<style>
#dialog.e-dialog .e-dlg-header-content .e-dlg-header {
text-align: center;
width: 100%;
color: #333;
font-weight: bold;
font-size: 20px;
}
#input-container.e-float-input { /* csslint allow: adjoining-classes */
margin: 17px 0;
}
.e-footer-content {
padding: 20px 0 0;
width: 100%;
}
.e-dialog.e-footer-content.e-btn {
width: 100%;
height: 36px;
}
#heading {
color: #333;
font-weight: bold;

```

```
margin: 0 0 15px;
text-align: center;
font-size: 20px;
}
#dialog.e-dialog .e-footer-content {
padding: 0 18px 18px;
}
#dialog.e-dialog .e-footer-content .e-btn {
margin-left: 0;
}
.error-container {
color: red;
}
</style>
@code {
private string UserName { get; set; } = "";
private string Password { get; set; } = "";
private bool IsVisible { get; set; } = true;
private string ErrorMessage { get; set; } = "";
private void OpenDialog()
{
this.IsVisible = true;
}
private void OnSubmit()
{
this.IsVisible = false;
}
private void Validation(BeforeCloseEventArgs args)
{
if (UserName == "" && Password == "")
{
args.Cancel = true;
this.IsVisible = true;
this.ErrorMessage = "Enter the Username and Password";
}
else if (UserName == "")
{
args.Cancel = true;
this.IsVisible = true;
this.ErrorMessage = "Enter the username";
}
else if (Password == "")
{
args.Cancel = true;
this.IsVisible = true;
this.ErrorMessage = "Enter the password";
}
else if (UserName.Length < 4)
{
args.Cancel = true;
this.IsVisible = true;
this.ErrorMessage = "Username must be minimum 4 characters";
}
else
{
args.Cancel = false;
UserName = "";
}
```

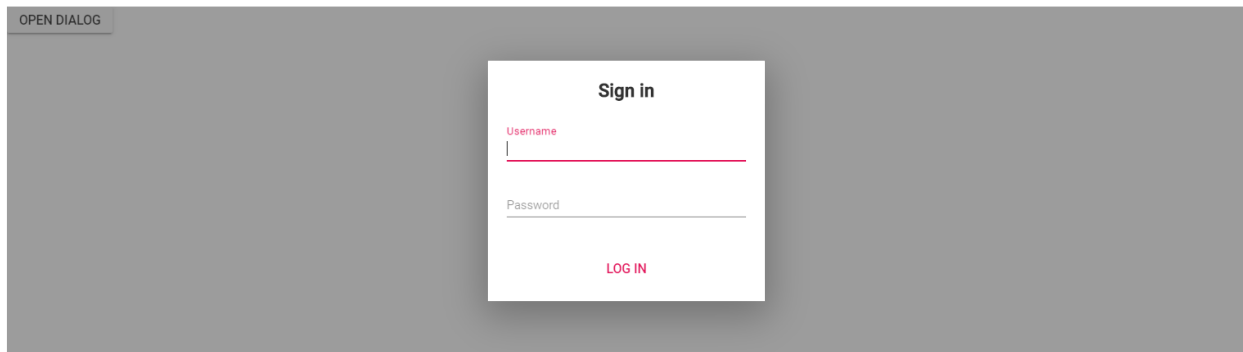


```

Password = "";
}
}
}

```

The output will be as follows.



### Prevent the focus on the first element in Blazor Dialog Component

By default, the Blazor dialog focuses on the first elements of the content area which can be active and focusable. You can prevent this default focusing behavior using the `Opened` event and by enabling the `PreventFocus` argument.

Bind the `Opened` event and enable the `PreventFocus` argument within an event like the below example.

#### ASPX-CS

```

@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
<SfDialog Width="300px" @bind-Visible="@IsVisible">
  <DialogEvents Opened="@onOpen"></DialogEvents>
  <DialogTemplates>
    <Header> Sign In </Header>
    <Content>
      <div class='form-group'>
        <label for='email'>Email:</label>
        <input type='email' class='form-control' id='email'>
      </div>
      <div class='form-group'>
        <label for='comment'>Password:</label>
        <input type='password' class='form-control' id='password'>
      </div>
    </Content>
  </DialogTemplates>
  <DialogButtons>
    <DialogButton Content="OK" IsPrimary="true" OnClick="@CloseDialog" />
    <DialogButton Content="Cancel" OnClick="@CloseDialog" />
  </DialogButtons>
</SfDialog>
@code {
  private bool IsVisible { get; set; } = true;
  private void OpenDialog()
  {
    this.IsVisible = true;
  }
}

```

```

}
private void CloseDialog()
{
    this.IsVisible = false;
}
private void onOpen(OpenEventArgs args)
{
    args.PreventFocus = true;
}
}

```

### Open a Dialog on condition in Blazor Dialog Component

You can prevent opening of the dialog by setting the **OnOpen** event argument cancel value to true. In the following sample, the success dialog is opened when you enter the username value with minimum 4 characters. Otherwise, it will not be opened.

#### ASPX-CS

```

@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<div class="login-form">
<div class='wrap'>
<div id="heading">Sign In</div>
<div id="input-container">
<div class="e-float-input e-input-group">
<input id="username" type="text" @bind-value="@UserName" @onfocus="@OnFocus"
required />
<span class="e-float-line"></span>
<label class="e-float-text">Username</label>
</div>
<div class="e-float-input e-input-group">
<input id="password" type="password" @bind-value="@Password"
@onfocus="@OnFocus" required />
<span class="e-float-line"></span>
<label class="e-float-text">Password</label>
</div>
</div>
<div class="error-container">
@ErrorMessage
</div>
</div>
<div class="button-container">
<SfButton @onclick="@OnSubmit">Log In</SfButton>
</div>
</div>
<SfDialog ID="dialog" IsModal="true" Width="280px" @bind-
Visible="@Visibility">
<DialogTemplates>
<Header> Success </Header>
<Content> Congratulations! Login Success </Content>
</DialogTemplates>
<DialogButtons>
<DialogButton Content="Dismiss" IsPrimary="true" OnClick="@CloseDialog" />

```

```
</DialogButtons>
<DialogEvents OnOpen="@Validation"></DialogEvents>
</SfDialog>
<style>
.wrap {
box-sizing: border-box;
margin: 0 auto;
padding: 20px 30px;
width: 340px;
background: #f7f7f7;
}
#input-container .e-float-input { /* csslint allow: adjoining-classes */
margin: 17px 0;
}
.wrap #input-container .e-control e-btn { /* csslint allow: adjoining-
classes */
margin: 3% 26%;
}
.button-container {
padding: 20px 0 0;
width: 100%;
}
.button-container .e-btn { /* csslint allow: adjoining-classes */
width: 100%;
height: 36px;
}
#heading {
color: #333;
font-weight: bold;
margin: 0 0 15px;
text-align: center;
font-size: 20px;
}
.login-form {
width: 340px;
margin: 50px auto;
}
#dialog.e-dialog .e-footer-content {
text-align: center;
}
.error-container {
color: red;
}
</style>
@code {
private string UserName { get; set; } = "";
private string Password { get; set; } = "";
private bool Visibility { get; set; } = false;
private string ErrorMessage { get; set; } = "";
private void OnSubmit()
{
this.Visibility = true;
}
private void CloseDialog()
{
this.Visibility = false;
}
```

```
private void OnFocus()
{
    this.ErrorMessage = "";
}
private void Validation(BeforeOpenEventArgs args)
{
    if (this.UserName == "" && this.Password == "")
    {
        args.Cancel = true;
        this.Visibility = false;
        this.ErrorMessage = "Enter the Username and Password";
    }
    else if (this.UserName == "")
    {
        args.Cancel = true;
        this.Visibility = false;
        this.ErrorMessage = "Enter the username";
    }
    else if (this.Password == "")
    {
        args.Cancel = true;
        this.Visibility = false;
        this.ErrorMessage = "Enter the password";
    }
    else if (this.UserName.Length < 4)
    {
        args.Cancel = true;
        this.Visibility = false;
        this.ErrorMessage = "Username must be minimum 4 characters";
    }
    else
    {
        args.Cancel = false;
        this.UserName = "";
        this.Password = "";
    }
}
}
```

### Customize the appearance in Blazor Dialog Component

You can customize the dialog appearance by providing dialog template as string or HTML element to the **Content** template property. In the following code block, dialog is customized as error window appearance.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
<SfDialog Width="400px" ShowCloseIcon="true" CloseOnEscape="true" @bind-Visible="@IsVisible">
    <DialogTemplates>
        <Header> File and Folder Rename </Header>
        <Content>
            <div class='msg-wrapper col-lg-12'>
                <span class='e-icons close-icon col-lg-2'></span>
            </div>
        </Content>
    </DialogTemplates>
</SfDialog>
```

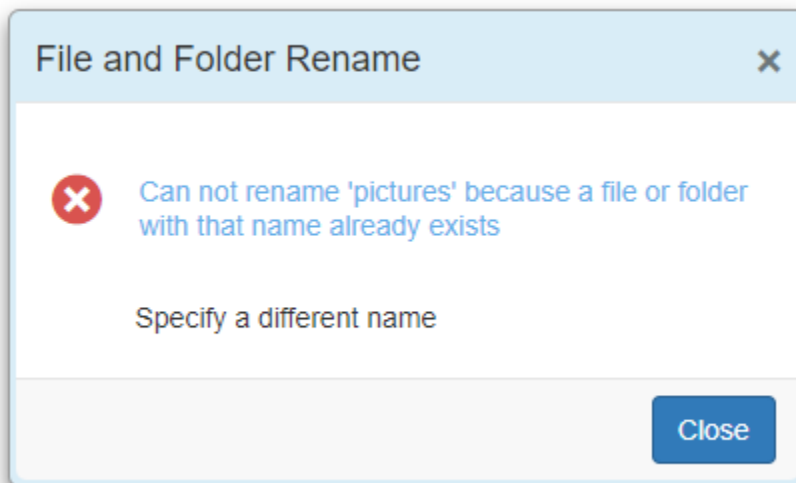
```

<span class='error-msg col-lg-10'>Can not rename 'pictures' because a file
or folder with that name already exists</span>
</div>
<div class='error-detail col-lg-8'>
<span>Specify a different name</span>
</div>
</Content>
</DialogTemplates>
<DialogAnimationSettings Effect="@AnimationEffect" Duration=400 />
<DialogButtons>
<DialogButton Content="Close" IsPrimary="true" OnClick="@CloseDialog" />
</DialogButtons>
</SfDialog>
@code {
private bool IsVisible { get; set; } = true;
private DialogEffect AnimationEffect = DialogEffect.Zoom;
private void OpenDialog()
{
this.IsVisible = true;
}
private void CloseDialog()
{
this.IsVisible = false;
}
}
<style>
.e-btn.e-flat.e-primary, .e-btn.e-flat.e-primary:focus {
background-color: #317ab9;
border-color: #265f91;
color: #fff;
}
.e-btn.e-flat.e-primary:hover, .e-btn.e-flat.e-primary:active {
background-color: #21527d;
border-color: #163854;
color: #fff;
}
.close-icon {
width: 24px;
height: 24px;
position: relative;
display: inline-block;
}
.error-msg {
color: #66afe9;
display: inline-block;
position: relative;
top: -20px;
left: 10px;
}
.error-detail {
position: relative;
left: 56px;
margin: 0 0 21px;
}
.e-icons.close-icon.col-lg-2:before {
content: '\e7e9';
font-size: 26px;
}

```

```
color: #d9534f;
position: relative;
left: 1px;
bottom: 18px;
}
.e-dialog .e-footer-content {
background-color: #f8f8f8;
}
.e-dialog.e-control.e-popup, .e-dialog.e-control.e-popup .e-dlg-header-
content {
background-color: #d9edf7;
}
.e-dialog.e-control.e-popup {
padding: 3px;
}
.e-dialog.e-control .e-dlg-header-content {
padding: 10px;
}
.e-dialog.e-control .e-footer-content {
padding: 8px 12px;
}
.e-dialog.e-control .e-dlg-content {
padding: 15px 0 0;
}
.msg-wrapper.col-lg-12 {
margin-top: 20px;
}
</style>
```

The output will be as follows.



#### Close Blazor Dialog Component when clicking outside of its region

By default, dialog can be closed by pressing **Esc** key and clicking the close icon at the right of dialog header. It can also be closed by clicking outside of the dialog using **Visible** property. Set the **CloseOnEscape** property value to **false** to prevent closing of the dialog when pressing **Esc** key.

In the following code, dialog is closed when clicking outside the dialog area using `Visible` property.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
<div id="target">
<SfDialog Target="#target" Width="300px" IsModal="true" ShowCloseIcon="true"
CloseOnEscape="false" @bind-Visible="@IsVisible">
<DialogEvents OnOverlayClick="@OverlayClick"></DialogEvents>
<DialogTemplates>
<Header> Delete Multiple Items</Header>
<Content> Are you sure you want to permanently delete all of these items?
</Content>
</DialogTemplates>
<DialogButtons>
<DialogButton Content="Yes" IsPrimary="true" OnClick="@CloseDialog" />
<DialogButton Content="No" OnClick="@CloseDialog" />
</DialogButtons>
</SfDialog>
</div>
<style>
#target {
min-height: 450px;
height: 100%;
}
</style>
@code {
private bool IsVisible { get; set; } = true;
private void OpenDialog()
{
this.IsVisible = true;
}
private void OverlayClick(MouseEventArgs args)
{
this.IsVisible = false;
}
}
```

#### Add an icons to Dialog buttons in Blazor Dialog Component

You can add icons to the dialog buttons using the `DialogButton` property or `FooterTemplate` property.

In the following code, dialog footer buttons are customized with icons using `DialogButton` property.

#### ASPX-CS

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
<SfDialog Width="300px" ShowCloseIcon="true" CloseOnEscape="true" @bind-
Visible="@IsVisible">
<DialogTemplates>
<Header>
<div>Delete Multiple Items</div>
</Header>
<Content>
```

```

<div>Are you sure you want to permanently delete all of these items?</div>
</Content>
</DialogTemplates>
<DialogButtons>
<DialogButton Content="Yes" IsPrimary="true" IconCss="e-icons e-ok-icon"
OnClick="@CloseDialog" />
<DialogButton Content="No" IconCss="e-icons e-close-icon"
OnClick="@CloseDialog" />
</DialogButtons>
</SfDialog>
<style>
a, a:hover, .highcontrast #dialog a, .highcontrast #dialog a:hover {
color: inherit;
text-decoration: none;
}
.e-btn-icon.e-icons.e-ok-icon.e-icon-left:before {
content: '\e7ff';
}
.e-btn-icon.e-icons.e-close-icon.e-icon-left:before {
content: '\e825';
}
</style>
@code {
private bool IsVisible { get; set; } = true;
private void OpenDialog()
{
this.IsVisible = true;
}
private void CloseDialog()
{
this.IsVisible = false;
}
}

```

In the following code, dialog footer buttons are customized with icons using `FooterTemplate` template property.

#### ASPX-CS

```

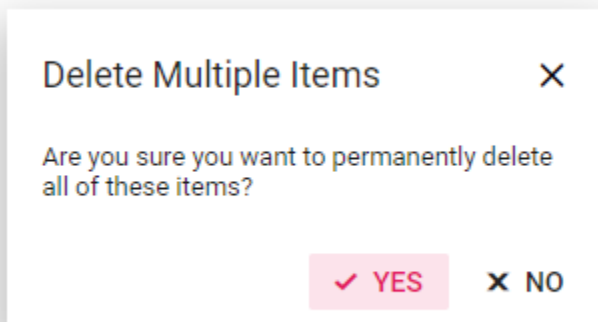
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="@OpenDialog">Open Dialog</SfButton>
<SfDialog Width="300px" ShowCloseIcon="true" CloseOnEscape="true" @bind-
Visible="@IsVisible">
<DialogTemplates>
<Header> Delete Multiple Items </Header>
<Content> Are you sure you want to permanently delete all of these items?
</Content>
<FooterTemplate>
<SfButton CssClass="e-primary e-flat" @onclick="@CloseDialog">
<span class='e-btn-icon e-icons e-ok-icon e-icon-left'></span>Yes
</SfButton>
<SfButton CssClass="e-flat" @onclick="@CloseDialog">
<span class='e-btn-icon e-icons e-close-icon e-icon-left'></span>No
</SfButton>
</FooterTemplate>

```



```
</DialogTemplates>
</SfDialog>
<style>
a, a:hover, .highcontrast #dialog a, .highcontrast #dialog a:hover {
color: inherit;
text-decoration: none;
}
.e-btn-icon.e-icons.e-ok-icon.e-icon-left:before {
content: '\e7ff';
}
.e-btn-icon.e-icons.e-close-icon.e-icon-left:before {
content: '\e825';
}
</style>
@code {
private bool IsVisible { get; set; } = true;
private void OpenDialog()
{
this.IsVisible = true;
}
private void CloseDialog()
{
this.IsVisible = false;
}
}
```

The output will be as follows.



## DocumentEditor

### Getting Started

#### Features in Blazor DocumentEditor Component

The [Blazor Word Processor component](#) for Blazor is used to compose, edit, view, and print Word (DOC, DOCX, and RTF) documents in Blazor applications.

- Composes a document from scratch.
- Opens and edits the Word (DOC, DOCX), RTF, and SFDT (Syncfusion Document Text) format files.
- Saves and exports the documents as DOCX and SDFT at the client-side.

- Saves and exports the documents as PDF files with Syncfusion's DocIO library.
- Prints the document.
- Finds and replaces the text.
- Allows spell checking.
- Contains a rich set of document elements like text, table, lists, inline image, fields, bookmark, hyperlink, page number, header, and footer.
- Contains a rich set of text, paragraph, and table formatting options.
- Creates or updates the table of contents.
- Contains tables.
- Allows Undo and Redo.
- Creates, edits, and applies paragraph and character styles.
- Contains clipboard operations such as Cut, Copy, and Paste (with formatting).
- Preserves the chart present in the opened Word document.
- Restricts editing for certain regions.
- Views the documents in read only mode.
- Allows user interactions through mouse, touch, and keyboard.
- Contains Intuitive UI options through context menu, dialogs, and navigation pane.
- Localizes all the static text to any desired language.

#### *Prerequisites*

- [Visual Studio 2019 Preview](#)
- [.NET Core SDK 3.1.2](#)

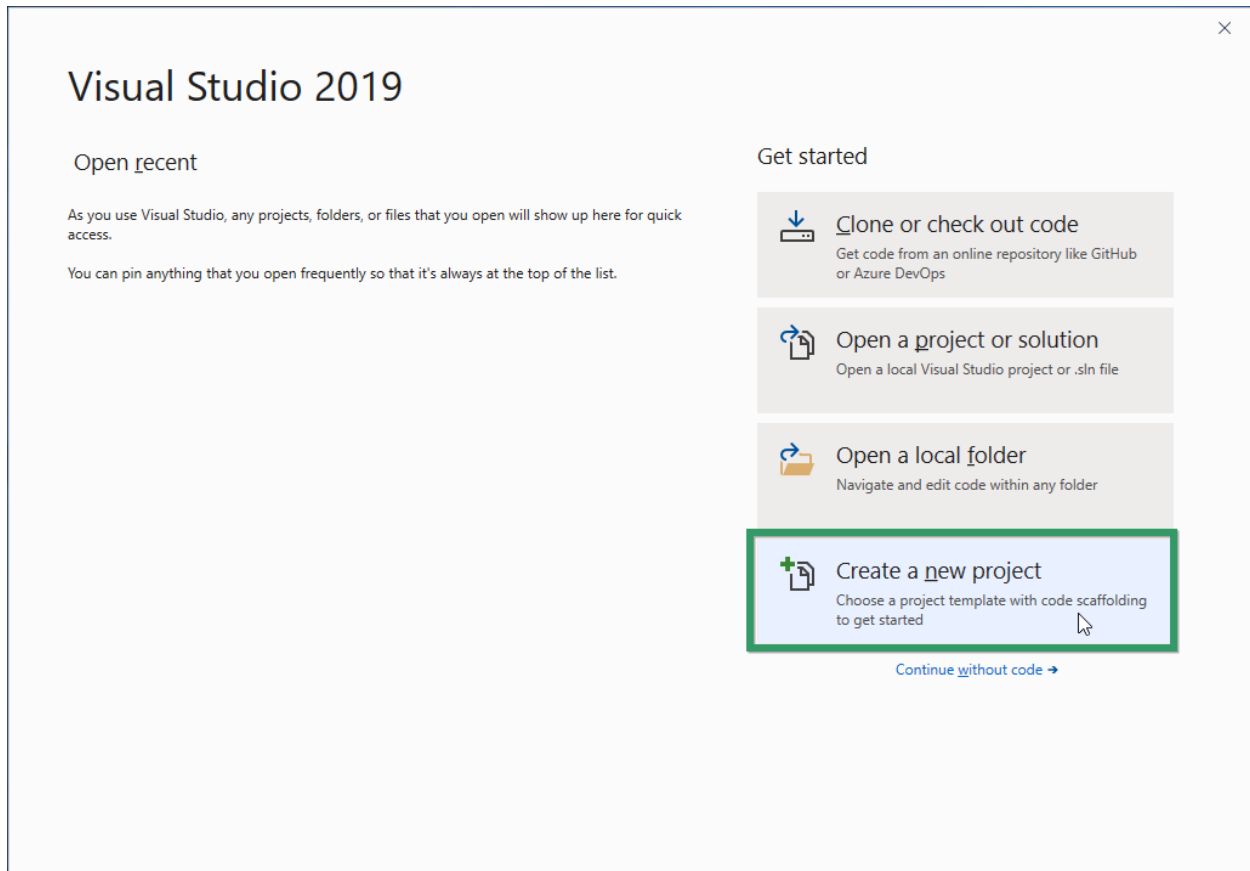
You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

#### Blazor DocumentEditor Component in Server Side App

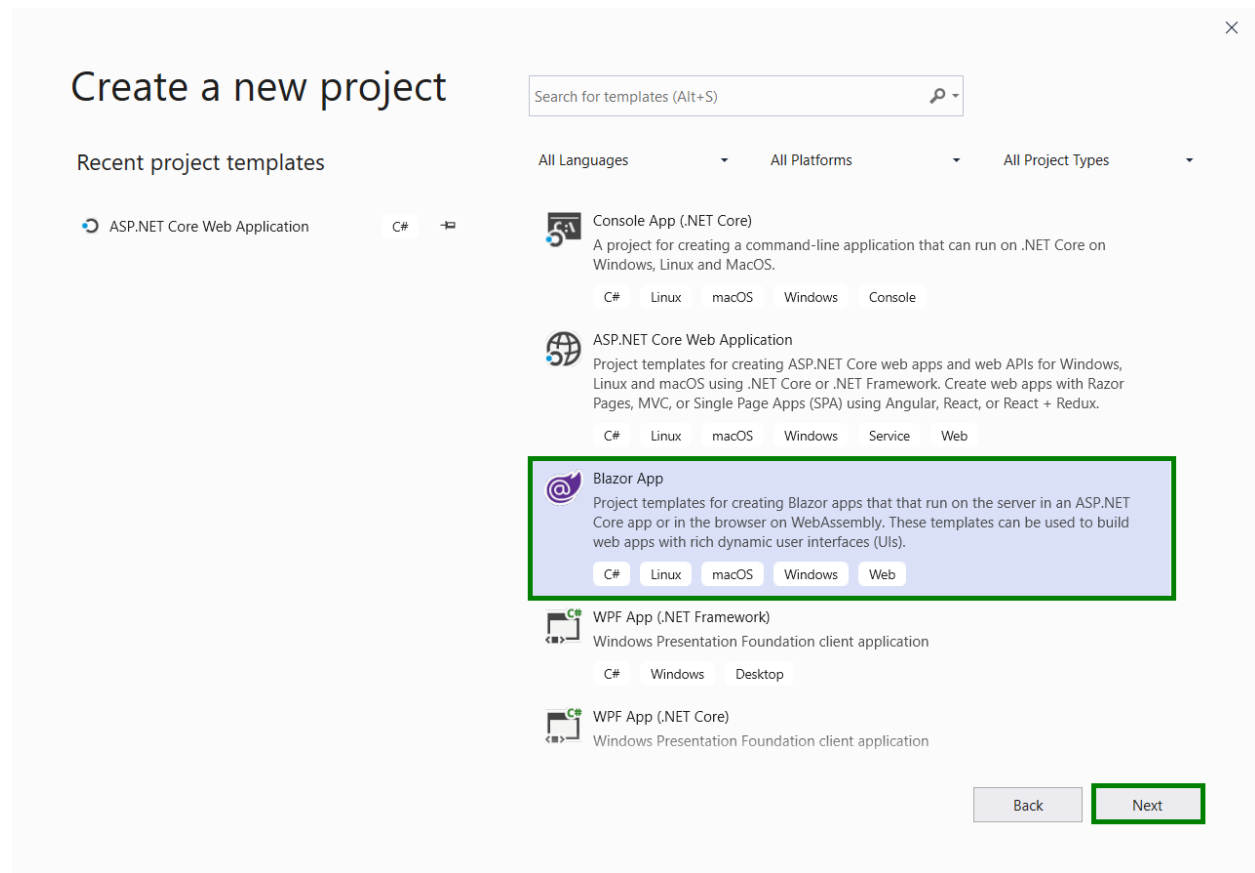
This article provides the step-by-step instructions to integrate the Word processor in Blazor server app using [Visual Studio 2019](#).

Steps to get started with Word processor component for Blazor:

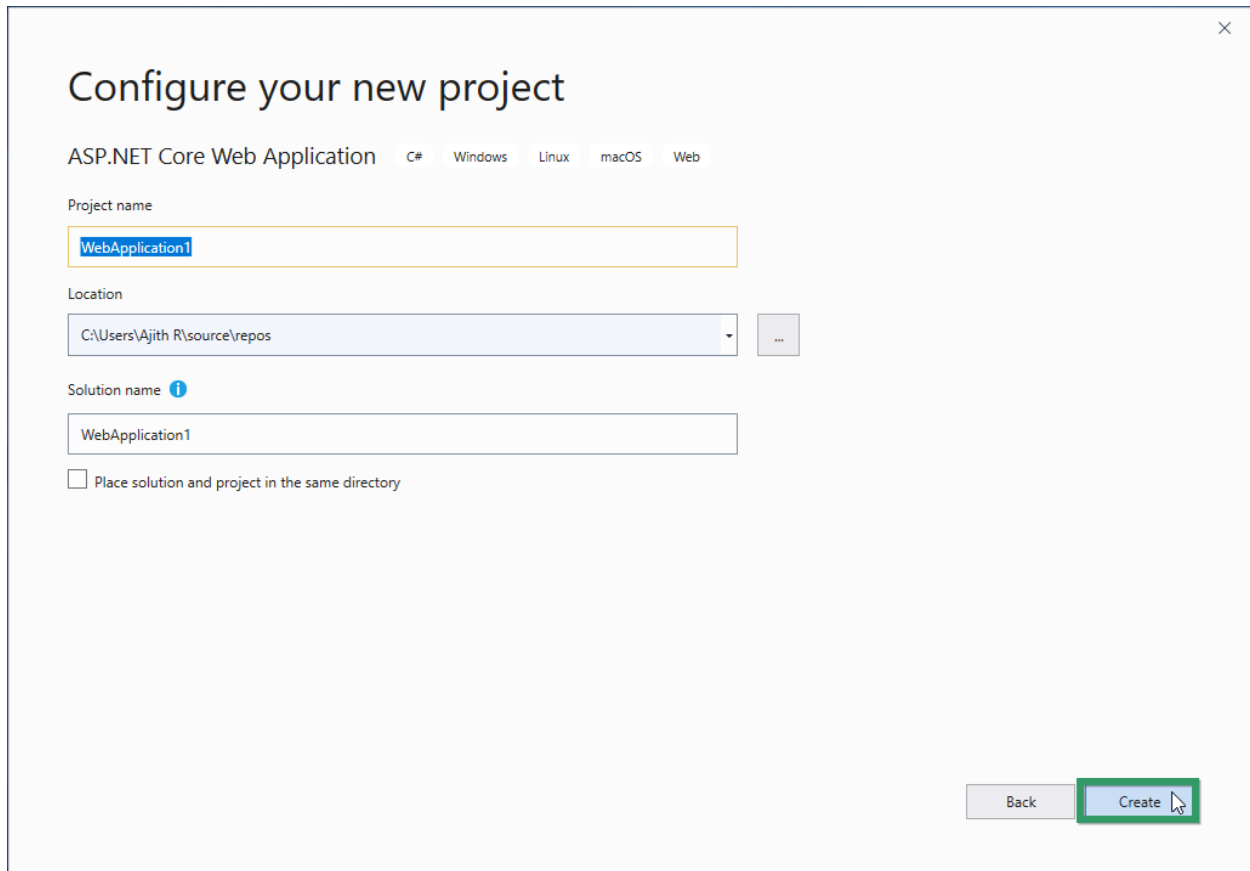
1. Select **Create a new project** from the Visual Studio dashboard.



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



3. In the project configuration window, click the **Create** button to create a new project with the default project configuration.



Configure your new project

ASP.NET Core Web Application C# Windows Linux macOS Web

Project name

WebApplication1

Location

C:\Users\Ajith R\source\repos

Solution name ⓘ

WebApplication1


☐ Place solution and project in the same directory


Back Create

4. Select **Blazor Server App** from the dashboard and click the **Create** button to create a new Blazor Server application. Make sure that **.NET Core** and **ASP.NET Core 3.1** are selected at the top.

## Create a new Blazor app

ASP.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

Author: Microsoft  
 Source: .NET Core 3.1.1

[Get additional project templates](#)


Back


Create

- Install the [Syncfusion.Blazor.WordProcessor](#) NuGet package to the newly created application by using the **NuGet Package Manager**. Right-click the project and select Manage NuGet Packages.
- Search **Syncfusion.Blazor.WordProcessor** keyword in the Browse tab and install [Syncfusion.Blazor.WordProcessor](#) NuGet package in the application.

[Browse](#)
[Installed](#)
[Updates](#)

Syncfusion.Blazor.WordProcessor


☒ Include prerelease


**Syncfusion.Blazor.WordProcessor** by Syncfusion Inc. v18.1.0.36-beta  
 Prerelease Syncfusion Blazor Word Processor Component used for processing Word Document and fetch the details to be used for rendering client-side.

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

### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
```

- Add the Syncfusion Word processor component (a.k.a DocumentEditor) to any webpages (razor) in the `Pages` folder. For example, the DocumentEditorContainer component is added to the `~/Pages/Index.razor` page.

## ASPX-CS

```
<SfDocumentEditorContainer EnableToolbar=true></SfDocumentEditorContainer>
```

9. Add the SyncfusionBlazor service in `ConfigureServices` method of **Startup.cs** file.

## CSHARP

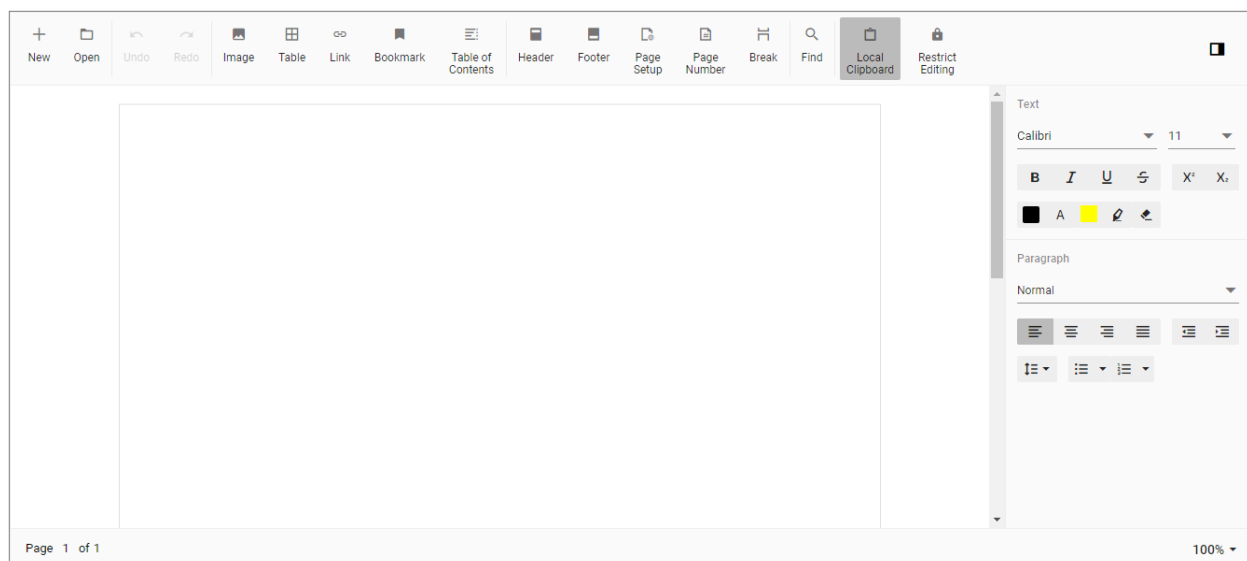
```
public void ConfigureServices(IServiceCollection services) {
    .....
    .....
    services.AddSyncfusionBlazor();
}
```

10. Add the client-side resources through [CDN](#) or from [NuGet](#) package in the `element` of the `~/Pages/_Host.cshtml` page.

## HTML

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

11. Run the application. The Word processor component will be rendered in the web browser.



12. To load an existing document during control initialization, use the following code example, which opens a Word document. Convert it to SFDT and load in the editor.

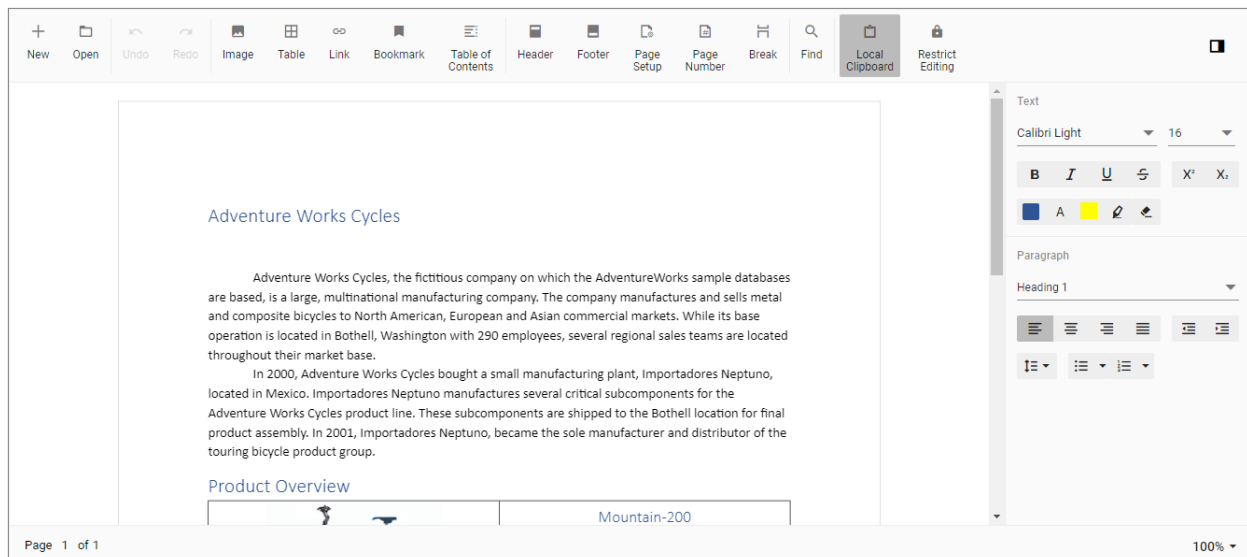
**ASPX-CS**

```

@using System.IO;
@using Syncfusion.Blazor.DocumentEditor;
<SfDocumentEditorContainer @ref="container" EnableToolbar=true>
<DocumentEditorContainerEvents
Created="OnCreated"></DocumentEditorContainerEvents>
</SfDocumentEditorContainer>
@code {
SfDocumentEditorContainer container;
public void OnCreated(object args)
{
string filePath = "wwwroot/data/GettingStarted.docx";
using (FileStream fileStream = new FileStream(filePath,
System.IO.FileMode.Open, System.IO.FileAccess.Read))
{
WordDocument document = WordDocument.Load(fileStream,
ImportFormatType.Docx);
string json = Newtonsoft.Json.JsonConvert.SerializeObject(document);
document.Dispose();
//To observe the memory go down, null out the reference of document
variable.
document = null;
SfDocumentEditor editor = container.DocumentEditor;
editor.Open(json);
//To observe the memory go down, null out the reference of json variable.
json = null;
}
}
}

```

As per the discussion thread [#30064](#), please null out the reference of streams and other instances when they are no longer required. Using this approach you'll observe the memory go down and become stable.





## Blazor DocumentEditor Component in WebAssembly App using Visual Studio

This article provides the step-by-step instructions to integrate the [Blazor Word Processor component](#) in Blazor WebAssembly application using [Visual Studio 2019](#).

Steps to get started with Word processor component for Blazor:

1. Install the essential project templates in the Visual Studio 2019 by running the below command line in the command prompt.

### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
```

9. Add the Syncfusion Word processor component (a.k.a DocumentEditor) to any web pages (razor) in the Pages folder. For example, the DocumentEditorContainer component is added in the ~/Pages/Index.razor page.

### ASPX-CS

```
<SfDocumentEditorContainer EnableToolbar=true></SfDocumentEditorContainer>
```

10. 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();  
        }  
    }  
}
```

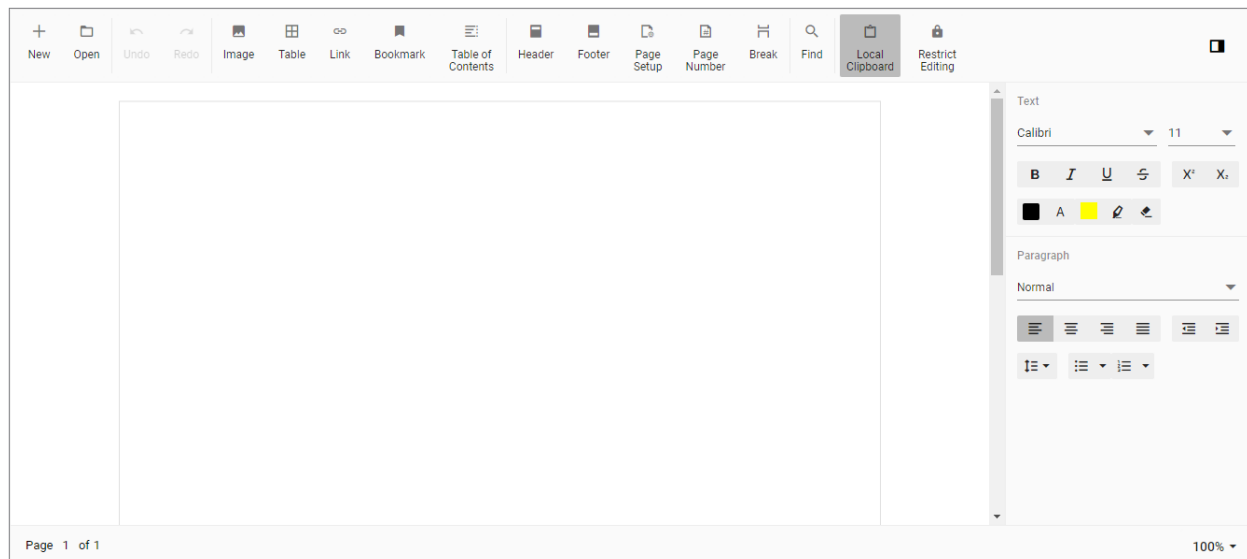
11. Add the Syncfusion bootstrap4 theme in the element of the ~/wwwroot/index.html page.

### HTML

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

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

12. Run the application. The Word processor component will be rendered in the web browser.



You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

## Opening a document in Blazor DocumentEditor Component

You might need to open and view Word documents from various location. In this section, you can find the information about how to open Word documents from URL, cloud, database, and local file system and also how to load a document during control initialization.

### Opening a document from URL

If you have your Word document file in the web, you can open it in [Blazor Word Processor](#) using URL. The following code example explains how to open a Word document file from URL.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
@using System.IO
@using System.Net
@using Newtonsoft.Json
<SfDocumentEditorContainer @ref="container" EnableToolbar=true>
  <DocumentEditorContainerEvents
    Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditorContainer>
@code {
    SfDocumentEditorContainer container;
    string sfdtString;
    protected override void OnInitialized()
    {
```

```
string fileUrl =
"https://www.syncfusion.com/downloads/support/directtrac/general/doc/Getting
_Started1018066633.docx";
WebClient webClient = new WebClient();
byte[] byteArray = webClient.DownloadData(fileUrl);
Stream stream = new MemoryStream(byteArray);
//To observe the memory go down, null out the reference of byteArray
variable.
byteArray = null;
WordDocument document = WordDocument.Load(stream, ImportFormatType.Docx);
stream.Dispose();
//To observe the memory go down, null out the reference of stream variable.
stream = null;
sfdtString = JsonConvert.SerializeObject(document);
document.Dispose();
//To observe the memory go down, null out the reference of document
variable.
document = null;
}
public void OnLoad(object args)
{
SfDocumentEditor editor = container.DocumentEditor;
editor.Open(sfdtString);
//To observe the memory go down, null out the reference of sfdtString
variable.
sfdtString = null;
}
}
```

As per the discussion thread [#30064](#), please null out the reference of streams and other instances when they are no longer required. Using this approach you'll observe the memory go down and become stable.

### Opening a document from Cloud

You can open the Word documents from Cloud storage.

The following code example shows how to open and load the Word document file stored in Azure Blob Storage.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
@using System.IO
@using Microsoft.Azure.Storage
@using Microsoft.Azure.Storage.Blob
@using Newtonsoft.Json
<SfDocumentEditorContainer @ref="container" EnableToolbar=true>
<DocumentEditorContainerEvents
Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditorContainer>
@code {
SfDocumentEditorContainer container;
string sfdtString;
protected override void OnInitialized()
{
//Connection string of storage account
```

```
string connectionString = "Here Place Your Connection string";
//Container name
string containerName = "document";
//File name to be loaded into Syncfusion Document Editor
string fileName = "Getting_Started.docx";
CloudStorageAccount cloudStorageAccount =
CloudStorageAccount.Parse(connectionString);
CloudBlobClient cloudBlobClient =
cloudStorageAccount.CreateCloudBlobClient();
CloudBlobContainer cloudBlobContainer =
cloudBlobClient.GetContainerReference(containerName);
CloudBlockBlob cloudBlockBlob =
cloudBlobContainer.GetBlockBlobReference(fileName);
MemoryStream memoryStream = new MemoryStream();
cloudBlockBlob.DownloadToStream(memoryStream);
WordDocument document = WordDocument.Load(memoryStream,
ImportFormatType.Docx);
memoryStream.Dispose();
//To observe the memory go down, null out the reference of memoryStream
variable.
memoryStream = null;
sfdtString = JsonConvert.SerializeObject(document);
document.Dispose();
//To observe the memory go down, null out the reference of document
variable.
document = null;
}
public void OnLoad(object args)
{
SfDocumentEditor editor = container.DocumentEditor;
editor.Open(sfdtString);
//To observe the memory go down, null out the reference of sfdtString
variable.
sfdtString = null;
}
}
```

The **Microsoft.Azure.Storage.Blob** NuGet package must be installed in your application to use the previous code example.

You can open the Word documents from Azure File Storage using the following code example.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
@using System.IO
@using Microsoft.Azure.Storage
@using Microsoft.Azure.Storage.File
@using Newtonsoft.Json
<SfDocumentEditorContainer @ref="container" EnableToolbar=true>
<DocumentEditorContainerEvents
Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditorContainer>
@code {
SfDocumentEditorContainer container;
string sfdtString;
```

```
protected override void OnInitialized()
{
    //Connection string of storage account
    string connectionString = "Here Place Your Connection string";
    //Container name
    string shareReference = "document";
    //String directoryReference = "document";
    //File name to be loaded into Syncfusion Document Editor
    string fileReference = "Getting_Started.docx";
    CloudStorageAccount cloudStorageAccount =
    CloudStorageAccount.Parse(connectionString);
    CloudFileClient fileClient = cloudStorageAccount.CreateCloudFileClient();
    //Get file share
    CloudFileShare cloudFileShare =
    fileClient.GetShareReference(shareReference);
    if (cloudFileShare.Exists())
    {
        //Get the related directory
        CloudFileDirectory dir = cloudFileShare.GetRootDirectoryReference();
        if (dir.Exists())
        {
            //Get the file reference
            CloudFile file = dir.GetFileReference(fileReference);
            MemoryStream memoryStream = new MemoryStream();
            //Download file to local disk
            file.DownloadToStream(memoryStream);
            WordDocument document = WordDocument.Load(memoryStream,
            ImportFormatType.Docx);
            memoryStream.Dispose();
            //To observe the memory go down, null out the reference of memoryStream
            variable.
            memoryStream = null;
            sfdtString = JsonConvert.SerializeObject(document);
            document.Dispose();
            //To observe the memory go down, null out the reference of document
            variable.
            document = null;
        }
    }
}

public void OnLoad(object args)
{
    if (!String.IsNullOrEmpty(sfdtString))
    {
        SfDocumentEditor editor = container.DocumentEditor;
        editor.Open(sfdtString);
        //To observe the memory go down, null out the reference of sfdtString
        variable.
        sfdtString = null;
    }
}
```

---

The **Microsoft.Azure.Storage.File** NuGet package must be installed in your application to use the previous code example.

---

### Opening a document from database

The following code example shows how to open the Word document file in viewer from SQL Server database.

#### ASPX-CS

```
@using System.IO;
@using Syncfusion.Blazor.DocumentEditor
@using System.Data.SqlClient
@using Newtonsoft.Json
<SfDocumentEditorContainer @ref="container" EnableToolbar=true>
<DocumentEditorContainerEvents
Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditorContainer>
@code {
SfDocumentEditorContainer container;
public void OnLoad(object args)
{
string documentID = "GettingStarted.docx";
string connectionString = "Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=C:\\\\database.mdf;";
SqlConnection connection = new SqlConnection(connectionString);
//Searches for the Word document from the database
string query = "select Data from DocumentsTable where DocumentName = '" +
documentID + "'";
SqlCommand command = new SqlCommand(query, connection);
connection.Open();
SqlDataReader read = command.ExecuteReader();
read.Read();
//Reads the Word document data as byte array from the database
byte[] byteArray = (byte[])read["Data"];
Stream stream = new MemoryStream(byteArray);
//To observe the memory go down, null out the reference of byteArray
variable.
byteArray = null;
WordDocument document = WordDocument.Load(stream, ImportFormatType.Docx);
stream.Dispose();
//To observe the memory go down, null out the reference of stream variable.
stream = null;
string json = JsonConvert.SerializeObject(document);
document.Dispose();
//To observe the memory go down, null out the reference of document
variable.
document = null;
SfDocumentEditor editor = container.DocumentEditor;
editor.Open(json);
//To observe the memory go down, null out the reference of json variable.
json = null;
}
}
```

The **System.Data.SqlClient** package must be installed in your application to use the previous code example. You need to modify the `connectionString` and `query` variable in the previous code example as per the connection string of your database.

### Opening a document from file system

There is an UI option in built-in toolbar to open the Word documents from local file system. If you want to achieve the same functionality when design your own toolbar, you can use the following code example to load and open the Word documents. In this sample, the Syncfusion's Uploader control is used for Blazor.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
@using Syncfusion.Blazor.Inputs
@using System.IO
@using Newtonsoft.Json

<SfUploader>
  <UploaderEvents OnUploadStart="OnSuccess"></UploaderEvents>
  <UploaderAsyncSettings
    SaveUrl="https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save"
    RemoveUrl="https://aspnetmvc.syncfusion.com/services/api/uploadbox/Remove"></UploaderAsyncSettings>
</SfUploader>

<SfDocumentEditorContainer @ref="container"
  EnableToolbar=false></SfDocumentEditorContainer>
@code {
  SfDocumentEditorContainer container;
  public void OnSuccess(UploadingEventArgs action)
  {
    string base64 = action.FileData.RawFile.ToString();
    string fileName = action.FileData.Name;
    string data = base64.Split(',')[1];
    byte[] bytes = Convert.FromBase64String(data);
    using (Stream stream = new MemoryStream(bytes))
    {
      WordDocument document = WordDocument.Load(stream, ImportFormatType.Docx);
      string sfdtString = JsonConvert.SerializeObject(document);
      document.Dispose();
      //To observe the memory go down, null out the reference of document
      variable.
      document = null;
      SfDocumentEditor editor = container.DocumentEditor;
      editor.Open(sfdtString);
      //To observe the memory go down, null out the reference of sfdtString
      variable.
      sfdtString = null;
    }
    action.Cancel = true;
  }
}
```

### Opening a document on control initialization

The Word document can be opened on control initialization, in this sample, the document is opened when the control is initialized.

#### ASPX-CS

```
@using System.IO
@using Syncfusion.Blazor.DocumentEditor
@using Newtonsoft.Json
```

```
<SfDocumentEditorContainer @ref="container" EnableToolbar=true>
<DocumentEditorContainerEvents
Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditorContainer>
@code {
SfDocumentEditorContainer container;
public void OnLoad(object args)
{
string filePath = "wwwroot/data/GettingStarted.docx";
using (FileStream fileStream = new FileStream(filePath,
System.IO.FileMode.Open, System.IO.FileAccess.Read))
{
WordDocument document = WordDocument.Load(fileStream,
ImportFormatType.Docx);
string json = JsonConvert.SerializeObject(document);
document.Dispose();
//To observe the memory go down, null out the reference of document
variable.
document = null;
SfDocumentEditor editor = container.DocumentEditor;
editor.Open(json);
//To observe the memory go down, null out the reference of json variable.
json = null;
}
}
}
```

You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

### Saving document in Blazor DocumentEditor Component

After composing or editing the document, you will need to save the document to the server, database, or local file system.

#### Save document to server

You might need to save the document back to the server. The following code example shows how to save the composed document to server.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
@using System.IO
<button @onclick="OnSave">Save</button>
<SfDocumentEditorContainer @ref="container"
EnableToolbar=true></SfDocumentEditorContainer>
@code {
SfDocumentEditorContainer container;
public async void OnSave()
{
SfDocumentEditor editor = container.DocumentEditor;
string base64Data = await editor.SaveAsBlob(FormatType.Docx);
byte[] data = Convert.FromBase64String(base64Data);
//To observe the memory go down, null out the reference of base64Data
variable.
base64Data = null;
//Word document file stream
```



```
Stream stream = new MemoryStream(data);
//To observe the memory go down, null out the reference of data variable.
data = null;
using (var fileStream = new FileStream(@"wwwroot\data\GettingStarted.docx",
    FileMode.Create, FileAccess.Write))
{
    //Saving the new file in root path of application
    stream.CopyTo(fileStream);
    fileStream.Close();
}
stream.Close();
//To observe the memory go down, null out the reference of stream variable.
stream = null;
}
}
```

### Save document to database

If you have plenty of documents stored in database and you want to save the composed or updated document back to the database, use the following code example.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
@using System.Data.SqlClient
<button onclick="OnSave">Save</button>
<SfDocumentEditorContainer @ref="container"
    EnableToolbar="true"></SfDocumentEditorContainer>
@code {
    SfDocumentEditorContainer container;
    public async void OnSave()
    {
        string documentID = "Getting_Started.docx";
        SfDocumentEditor editor = container.DocumentEditor;
        string base64Data = await editor.SaveAsBlob(FormatType.Docx);
        byte[] data = Convert.FromBase64String(base64Data);
        //To observe the memory go down, null out the reference of base64Data
        variable.
        base64Data = null;
        string connectionString = "Data
        Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=C:\\database.mdf;";
        string queryStmt = "Update DocumentsTable SET Data = @Content where
        DocumentName = '" + documentID + "'";
        using (SqlConnection con = new SqlConnection(connectionString))
        using (SqlCommand cmd = new SqlCommand(queryStmt, con))
        {
            SqlParameter param = cmd.Parameters.Add("@Content",
            System.Data.SqlDbType.VarBinary);
            param.Value = data;
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
            //To observe the memory go down, null out the reference of data variable.
            data = null;
        }
    }
}
```

### Download document as a copy

You can also save or download the document in local file system.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
<button @onclick="OnDownload">Download</button>
<SfDocumentEditorContainer @ref="container"
EnableToolbar="true"></SfDocumentEditorContainer>
@code {
SfDocumentEditorContainer container;
public void OnDownload()
{
SfDocumentEditor editor = container.DocumentEditor;
editor.Save("sample", FormatType.Docx);
}
}
```

### Clipboard in Blazor DocumentEditor Component

[Blazor Word Processor](#) component (a.k.a Document Editor) provides built-in support for clipboard operations such as cut, copy, and paste. You can perform the clipboard operations using keyboard shortcuts, touch, and keyboard interactions. Also, the same functionalities can be invoked programmatically.

There is a built-in clipboard (local clipboard) with this Word processor component, which allows the users to perform cut, copy, and paste faster. If you want to use system clipboard instead of local clipboard, turn off the local clipboard by setting the `EnableLocalPaste` to false.

If you need to copy or paste the contents from other applications, use system clipboard. To copy or paste the contents within the Word processor component, use local clipboard.

Let's see how to invoke each clipboard operations using code.

#### Copy

You can copy the selected contents using the `Copy` method as shown in the following code example.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
<button @onclick="CopyClick">Copy</button>
<SfDocumentEditorContainer @ref="container"
EnableToolbar=true></SfDocumentEditorContainer>
@code {
SfDocumentEditorContainer container;
protected void CopyClick(object args)
{
container.DocumentEditor.Selection.Copy();
}
}
```

#### Cut

You can cut the selected content using the "Cut" method as shown in the following code example.

### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
<button @onclick="CutClick">Cut</button>
<SfDocumentEditorContainer @ref="container"
EnableToolbar=true></SfDocumentEditorContainer>
@code {
SfDocumentEditorContainer container;
protected void CutClick(object args)
{
container.DocumentEditor.Editor.Cut();
}
}
```

### Paste

#### Local paste

The following code example shows how to perform the paste operation from the local clipboard.

### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
<button @onclick="PasteClick">Paste</button>
<SfDocumentEditorContainer @ref="container" EnableToolbar=true
EnableLocalPaste=true></SfDocumentEditorContainer>
@code {
SfDocumentEditorContainer container;
protected void PasteClick(object args)
{
container.DocumentEditor.Editor.Paste();
}
}
```

You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

### Undo and redo in Blazor DocumentEditor Component

[Blazor Document Editor](#) tracks the history of all editing actions done in the document, which allows undo and redo functionality.

#### Enable or disable history

Inject the **EditorHistory** module in your application to provide history preservation functionality for **DocumentEditor**. Refer to the following code example.

### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
<SfDocumentEditor @ref="documentEditor" IsReadOnly=false EnableEditor=true
EnableSelection=true>
<DocumentEditorContainerEvents
Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditor>
@code {
SfDocumentEditor documentEditor;
protected void OnLoad(object args)
{
}
```

```
documentEditor.EnableEditorHistory = true;
}
```

You can enable or disable history preservation for a document editor instance any time using the `EnableEditorHistory` property. Refer to the following sample code.

#### **CSHARP**

```
documentEditor.EnableEditorHistory = true;
```

#### Undo and redo

You can perform undo and redo by `CTRL+Z` and `CTRL+Y` keyboard shortcuts. Document editor exposes API to do it programmatically.

To undo the last editing operation in document editor, refer to the following sample code.

#### **CSHARP**

```
documentEditor.EditorHistory.Undo();
```

To redo the last undone action, refer to the following code example.

#### **CSHARP**

```
documentEditor.EditorHistory.Redo();
```

#### Stack size

History of editing actions will be maintained in stack, so that the last item will be reverted first. By default, document editor limits the size of undo and redo stacks to 500 each respectively. However, you can customize this limit. Refer to the following sample code.

#### **CSHARP**

```
documentEditor.EditorHistory.SetRedoLimit(400);
documentEditor.EditorHistory.SetUndoLimit(400);
```

You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

#### Images in Blazor DocumentEditor Component

[Blazor Document Editor](#) supports common raster format images like PNG, BMP, JPEG, and GIF. You can insert an image file or online image in the document using the `InsertImage()` method.

#### **CSHARP**

```
documentEditor.Editor.InsertImage("<<base64String>>");
```

Image files will be internally converted to base64 string. Whereas, online images are preserved as URL.

### Image resizing

Document editor provides built-in image resizer that can be injected into your application based on the requirements. This allows you to resize the image by dragging the resizing points using mouse or touch interactions. This resizer appears as follows.



### Changing size

Document editor expose API to get or resize the selected image. Refer to the following sample code.

#### CSHARP

```
int height = await documentEditor.Selection.ImageFormat.GetHeight();  
int width = await documentEditor.Selection.ImageFormat.GetWidth();  
documentEditor.Selection.ImageFormat.Resize(width + 10, height + 10);
```

You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

### Tables in Blazor DocumentEditor Component

Tables are an efficient way to present information. Document editor can display and edit the tables. You can select and edit tables through keyboard, mouse, or touch interactions. [Blazor Document Editor](#) exposes a rich set of APIs to perform these operations programmatically.

#### Create a table

You can create and insert a table at cursor position by specifying the required number of rows and columns.

Refer to the following sample code.

#### CSHARP

```
documentEditor.Editor.InsertTable(3, 3);
```

The maximum size of row and column is limited to 32767 and 63 respectively.

#### Insert rows

You can add a row (or several rows) above or below the row at cursor position by using the `InsertRow` method. This method accepts the following parameters:

Parameter	Type	Description
-----------	------	-------------

left(optional) | boolean| This is optional and if omitted, it takes the value as false and inserts to the right of column at cursor position.

count(optional) | number | This is optional and if omitted, it takes the value as 1.

Refer to the following sample code.

#### **CSHARP**

```
//Insert a column to the right of the column at cursor position.
documentEditor.Editor.InsertColumn();
//Insert a column to the left of the column at cursor position.
documentEditor.Editor.InsertColumn(false);
//Insert two columns to the left of the column at cursor position.
documentEditor.Editor.InsertColumn(false, 2);
```

#### *Select an entire table*

If the cursor position is inside a table, you can select the entire table by using the following sample code.

#### **CSHARP**

```
documentEditor.Selection.SelectTable();
```

#### *Select row*

You can select the entire row at cursor position by using the following sample code.

#### **CSHARP**

```
documentEditor.Selection.SelectRow();
```

If current selection spans across cells of different rows, all these rows will be selected.

#### *Select column*

You can select the entire column at cursor position by using the following sample code.

#### **CSHARP**

```
documentEditor.Selection.SelectColumn();
```

If current selection spans across cells of different columns, all these columns will be selected.

#### *Select cell*

You can select the cell at cursor position by using the following sample code.

#### **CSHARP**

```
documentEditor.Selection.SelectCell();
```

#### *Delete table*

Document editor allows you to delete the entire table. You can use the `DeleteTable()` method of editor instance, if selection is in table. Refer to the following sample code.

#### **CSHARP**

```
documentEditor.Editor.DeleteTable();
```

### Delete row

Document editor allows you to delete the selected number of rows. You can use the `DeleteRow()` method of editor instance to delete the selected number of rows, if selection is in table. Refer to the following sample code.

#### CSHARP

```
documentEditor.Editor.DeleteRow();
```

### Delete column

Document editor allows you to delete the selected number of columns. You can use the `DeleteColumn()` method of editor instance to delete the selected number of columns, if selection is in table. Refer to the following sample code.

#### CSHARP

```
documentEditor.Editor.DeleteColumn();
```

### Merge cells

You can merge cells vertically, horizontally, or combination of both to a single cell. To vertically merge the cells, the columns within selection should be even in left and right directions. To horizontally merge the cells, the rows within selection should be even in top and bottom direction.

Refer to the following sample code.

#### CSHARP

```
documentEditor.Editor.MergeCells();
```

You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

## Table of contents in Blazor DocumentEditor Component

The table of contents in a document is same as the list of chapters at the beginning of a book. It lists each heading in the document and the page number, where that heading starts with various options to customize the appearance.

### Inserting table of contents

[Blazor Document Editor](#) exposes an API to insert table of contents at cursor position programmatically. You can specify the settings for table of contents explicitly. Otherwise, the default settings will be applied.

`TableOfContentsSettings` contain the following properties:

- **startLevel**: Specifies the start level for constructing table of contents.
- **endLevel**: Specifies the end level for constructing table of contents.
- **includeHyperlink**: Specifies whether the link for headings is included.
- **includePageNumber**: Specified whether the page number of the headings is included.
- **rightAlign**: Specifies whether the page number is right aligned.
- **tabLeader**: Specifies the tab leader styles such as none, dot, hyphen, and underscore.

- **includeOutlineLevels:** Specifies whether the outline levels are included.

The following code illustrates how to insert table of content in document editor.

### CSHARP

```
TableOfContentsSettings tableOfContentsSettings = new
TableOfContentsSettings();
tableOfContentsSettings.StartLevel = 1;
tableOfContentsSettings.EndLevel = 3;
tableOfContentsSettings.IncludeHyperlink = true;
tableOfContentsSettings.IncludePageNumber = true;
tableOfContentsSettings.RightAlign = true;
documentEditor.Editor.InsertTableOfContents(tableOfContentsSettings);
```

### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor;
<SfDocumentEditor @ref="documentEditor" IsReadOnly=false
EnableEditorHistory=true EnableEditor=true EnableSelection=true>
<DocumentEditorContainerEvents
Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditor>
@code {
SfDocumentEditor documentEditor;
protected void OnLoad(object args)
{
string sfdt =
"{\"sections\": [{\"blocks\": [{\"paragraphFormat\": {\"styleName\": \"Heading 1\"}, \"inlines\": [{\"text\": \"Headin\", \"name\": \"_GoBack\", \"bookmarkType\": 0}, {\"text\": \"g1\"}]}], {\"paragraphFormat\": {\"styleName\": \"Heading 2\"}, \"inlines\": [{\"text\": \"Heading2\"}]}], {\"paragraphFormat\": {\"styleName\": \"Heading 3\"}, \"inlines\": [{\"text\": \"Heading3\"}]}], {\"paragraphFormat\": {\"styleName\": \"Heading 4\"}, \"inlines\": [{\"text\": \"Heading4\"}]}], {\"paragraphFormat\": {\"styleName\": \"Heading 5\"}, \"inlines\": [{\"text\": \"Heading5\"}]}], {\"paragraphFormat\": {\"styleName\": \"Heading 6\"}, \"inlines\": [{\"text\": \"Heading6\"}]}], {\"paragraphFormat\": {\"styleName\": \"Normal\"}, \"inlines\": [{\"text\": \"Normal\"}]}], \"headersFooters\": {, \"sectionFormat\": {\"headerDistance\": 36.0, \"footerDistance\": 36.0, \"pageWidth\": 612.0, \"pageHeight\": 792.0, \"leftMargin\": 72.0, \"rightMargin\": 72.0, \"topMargin\": 72.0, \"bottomMargin\": 72.0, \"differentFirstPage\": false, \"differentOddAndEvenPages\": false}}, \"characterFormat\": {\"fontSize\": 11.0, \"fontFamily\": \"Calibri\", \"paragraphFormat\": {\"afterSpacing\": 8.0, \"lineSpacing\": 1.0791666507720947, \"lineSpacingType\": \"Multiple\"}, \"background\": {\"color\": \"#FFFFFF\"}, \"styles\": [{\"type\": \"Paragraph\", \"name\": \"Normal\", \"next\": \"Normal\", \"type\": \"Paragraph\", \"name\": \"Heading 1\", \"basedOn\": \"Normal\", \"next\": \"Normal\", \"link\": \"Heading 1 Char\", \"characterFormat\": {\"fontSize\": 16.0, \"fontFamily\": \"Calibri Light\", \"fontColor\": \"#2F5496FF\"}, \"paragraphFormat\": {\"beforeSpacing\": 12.0, \"afterSpacing\": 0.0, \"outlineLevel\": \"Level1\"}}, {\"type\": \"Paragraph\", \"name\": \"Heading 2\", \"basedOn\": \"Normal\", \"next\": \"Normal\", \"link\": \"Heading 2
```



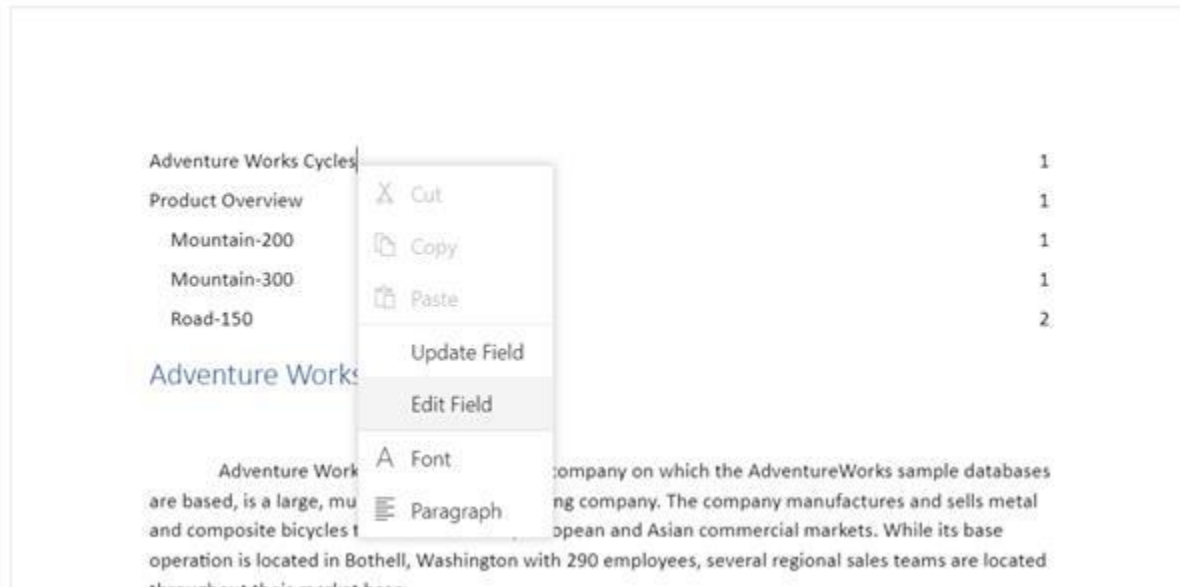
```

Char\","characterFormat\":{\\"fontSize\\":13.0,\\\"fontFamily\\":\\"Calibri
Light\\",\\\"fontColor\\":\\"#2F5496FF\\"},\\\"paragraphFormat\\":{\\"beforeSpacing\\":
2.0,\\\"afterSpacing\\":0.0,\\\"outlineLevel\\":\\"Level2\\"}},{\\"type\\":\\"Paragraph
\\",\\\"name\\":\\"Heading
3\\",\\\"basedOn\\":\\"Normal\\",\\\"next\\":\\"Normal\\",\\\"link\\":\\"Heading 3
Char\\",\\\"characterFormat\\":{\\"fontSize\\":12.0,\\\"fontFamily\\":\\"Calibri
Light\\",\\\"fontColor\\":\\"#1F3763FF\\"},\\\"paragraphFormat\\":{\\"beforeSpacing\\":
2.0,\\\"afterSpacing\\":0.0,\\\"outlineLevel\\":\\"Level3\\"}},{\\"type\\":\\"Paragraph
\\",\\\"name\\":\\"Heading
4\\",\\\"basedOn\\":\\"Normal\\",\\\"next\\":\\"Normal\\",\\\"link\\":\\"Heading 4
Char\\",\\\"characterFormat\\":{\\"italic\\":true,\\\"fontFamily\\":\\"Calibri
Light\\",\\\"fontColor\\":\\"#2F5496FF\\"},\\\"paragraphFormat\\":{\\"beforeSpacing\\":
2.0,\\\"afterSpacing\\":0.0,\\\"outlineLevel\\":\\"Level4\\"}},{\\"type\\":\\"Paragraph
\\",\\\"name\\":\\"Heading
5\\",\\\"basedOn\\":\\"Normal\\",\\\"next\\":\\"Normal\\",\\\"link\\":\\"Heading 5
Char\\",\\\"characterFormat\\":{\\"fontFamily\\":\\"Calibri
Light\\",\\\"fontColor\\":\\"#2F5496FF\\"},\\\"paragraphFormat\\":{\\"beforeSpacing\\":
2.0,\\\"afterSpacing\\":0.0,\\\"outlineLevel\\":\\"Level5\\"}},{\\"type\\":\\"Paragraph
\\",\\\"name\\":\\"Heading
6\\",\\\"basedOn\\":\\"Normal\\",\\\"next\\":\\"Normal\\",\\\"link\\":\\"Heading 6
Char\\",\\\"characterFormat\\":{\\"fontFamily\\":\\"Calibri
Light\\",\\\"fontColor\\":\\"#1F3763FF\\"},\\\"paragraphFormat\\":{\\"beforeSpacing\\":
2.0,\\\"afterSpacing\\":0.0,\\\"outlineLevel\\":\\"Level6\\"}},{\\"type\\":\\"Character
\\",\\\"name\\":\\"Default Paragraph
Font\\"},{\\"type\\":\\"Character\\",\\\"name\\":\\"Heading 1
Char\\",\\\"basedOn\\":\\"Default Paragraph
Font\\",\\\"characterFormat\\":{\\"fontSize\\":16.0,\\\"fontFamily\\":\\"Calibri
Light\\",\\\"fontColor\\":\\"#2F5496FF\\"}},{\\"type\\":\\"Character\\",\\\"name\\":\\"Hea
ding 2 Char\\",\\\"basedOn\\":\\"Default Paragraph
Font\\",\\\"characterFormat\\":{\\"fontSize\\":13.0,\\\"fontFamily\\":\\"Calibri
Light\\",\\\"fontColor\\":\\"#2F5496FF\\"}},{\\"type\\":\\"Character\\",\\\"name\\":\\"Hea
ding 3 Char\\",\\\"basedOn\\":\\"Default Paragraph
Font\\",\\\"characterFormat\\":{\\"fontSize\\":12.0,\\\"fontFamily\\":\\"Calibri
Light\\",\\\"fontColor\\":\\"#1F3763FF\\"}},{\\"type\\":\\"Character\\",\\\"name\\":\\"Hea
ding 4 Char\\",\\\"basedOn\\":\\"Default Paragraph
Font\\",\\\"characterFormat\\":{\\"italic\\":true,\\\"fontFamily\\":\\"Calibri
Light\\",\\\"fontColor\\":\\"#2F5496FF\\"}},{\\"type\\":\\"Character\\",\\\"name\\":\\"Hea
ding 5 Char\\",\\\"basedOn\\":\\"Default Paragraph
Font\\",\\\"characterFormat\\":{\\"fontFamily\\":\\"Calibri
Light\\",\\\"fontColor\\":\\"#2F5496FF\\"}},{\\"type\\":\\"Character\\",\\\"name\\":\\"Hea
ding 6 Char\\",\\\"basedOn\\":\\"Default Paragraph
Font\\",\\\"characterFormat\\":{\\"fontFamily\\":\\"Calibri
Light\\",\\\"fontColor\\":\\"#1F3763FF\\"}}}}";
documentEditor.Open(sfddt);
TableOfContentsSettings tableOfContentsSettings = new
TableOfContentsSettings();
tableOfContentsSettings.StartLevel = 1;
tableOfContentsSettings.EndLevel = 3;
tableOfContentsSettings.IncludeHyperlink = true;
tableOfContentsSettings.IncludePageNumber = true;
tableOfContentsSettings.RightAlign = true;
documentEditor.Editor.InsertTableOfContents(tableOfContentsSettings);
}
}

```

### Update or edit table of contents

You can update or edit the table of contents using the built-in context menu shown up by right-clicking it. Refer to the following screenshot.



- **Update Field:** Updates the headings in table of contents with same settings by searching the entire document.
- **Edit Field:** Opens the built-in table of contents dialog and allows you to modify its settings.

You can also do it programmatically by using the exposed API. Refer to the following sample code.

#### CSHARP

```
documenteditor.Open(''); /*Open any existing document*/
TableOfContentsSettings tableOfContentsSettings = new
TableOfContentsSettings();
tableOfContentsSettings.StartLevel = 1;
tableOfContentsSettings.EndLevel = 3;
tableOfContentsSettings.IncludeHyperlink = true;
tableOfContentsSettings.IncludePageNumber = true;
tableOfContentsSettings.RightAlign = true;
documentEditor.Editor.InsertTableOfContents(tableOfContentsSettings);
```

Same method is used for inserting, updating, and editing table of contents. This will work based on the current element at cursor position and the optional settings parameter. If table of contents is present at cursor position, the update operation will be done based on the optional settings parameter. Otherwise, the insert operation will be done.

You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

### Headers and Footers in Blazor DocumentEditor Component

[Blazor Document Editor](#) supports headers and footers in its document. Each section in the document can have the following types of headers and footers:

- First page: Used only on the first page of the section.
- Even pages: Used on all even numbered pages in the section.
- Default: Used on all pages of the section, where first or even pages are not applicable or not specified.

You can define this by setting format properties of the corresponding section using the following sample code.

#### **CSHARP**

```
//Defines whether different header footer is required for first page of the section
documentEditor.Selection.SectionFormat.SetDifferentFirstPage(true);
//Defines whether different header footer is required for odd and even pages in the section
documentEditor.Selection.SectionFormat.SetDifferentOddAndEvenPages(true);
```

#### Go to header footer region

Double click in header or footer region to move the selection into it. You can also do this by using the following code.

#### **CSHARP**

```
documentEditor.Selection.GoToHeader();
documentEditor.Selection.GoToFooter();
```

#### Header and footer distance

You can define the distance of header region content from the top of the page. Refer to the following sample code.

#### **CSHARP**

```
documentEditor.Selection.SectionFormat.SetHeaderDistance(36);
```

Same way, you can define the distance of footer region content from the bottom of the page. Refer to the following sample code.

#### **CSHARP**

```
documentEditor.Selection.SectionFormat.SetFooterDistance(36);
```

#### Close header footer region

Move the selection to the document body from header or footer region by double clicking or tapping the document area. You can also perform this by using the following sample code.

#### **CSHARP**

```
documentEditor.Selection.CloseHeaderFooter();
```

You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

## Working with Text Formatting in Blazor DocumentEditor Component

[Blazor Document Editor](#) supports several formatting options for text like bold, italic, font color, highlight color, and more. This section describes how to modify the formatting for selected text in detail.

### Bold

The bold formatting for selected text can be get or set by using the following sample code.

#### CSHARP

```
//Gets the value for bold formatting of selected text.
bool bold = await documentEditor.Selection.CharacterFormat.GetBold();
//Sets bold formatting for selected text.
documentEditor.Selection.CharacterFormat.SetBold(true);
```

You can toggle the bold formatting based on existing value at selection. Refer to the following sample code.

#### CSHARP

```
documentEditor.Editor.ToggleBold();
```

### Italic

The Italic formatting for selected text can be get or set by using the following sample code.

#### CSHARP

```
documentEditor.Selection.CharacterFormat.SetItalic(true);
```

You can toggle the Italic formatting based on existing value at selection. Refer to the following sample code.

#### CSHARP

```
documentEditor.Editor.ToggleItalic();
```

### Underline property

The underline style for selected text can be get or set by using the following sample code.

#### CSHARP

```
documentEditor.Editor.ToggleUnderline(Underline.Single);
```

You can toggle the underline style of selected text based on existing value at selection by specifying a value. Refer to the following sample code.

#### CSHARP

```
documentEditor.Editor.ToggleUnderline('Single');
```

### Strikethrough property

The strikethrough style for selected text can be get or set by using the following sample code.

#### CSHARP

```
documentEditor.Editor.ToggleStrikethrough(Strikethrough.SingleStrike);
```

You can toggle the strikethrough style of selected text based on existing value at selection by specifying a value. Refer to the following sample code.

#### JAVASCRIPT

```
documentEditor.Editor.ToggleStrikethrough(Strikethrough.SingleStrike);
```

#### Superscript property

The selected text can be made superscript by using the following sample code.

#### CSHARP

```
documentEditor.Selection.CharacterFormat.SetBaselineAlignment(BaselineAlignm
ent.Superscript);
```

Toggle the selected text as superscript or normal using the following sample code.

#### JAVASCRIPT

```
documentEditor.Editor.ToggleSuperscript();
```

#### Subscript property

The selected text can be made subscript by using the following sample code.

#### CSHARP

```
documentEditor.Selection.CharacterFormat.SetBaselineAlignment(BaselineAlignm
ent.Subscript);
```

Toggle the selected text as subscript or normal using the following sample code.

#### CSHARP

```
documentEditor.Editor.ToggleSubscript();
```

You can make a subscript or superscript text as normal using the following code.

#### CSHARP

```
documentEditor.Selection.CharacterFormat.SetBaselineAlignment(BaselineAlignm
ent.Normal);
```

#### Size

The size of selected text can be get or set using the following code.

#### CSHARP

```
documentEditor.Selection.CharacterFormat.SetFontSize(32);
```

#### Color

The color of selected text can be get or set using the following code.

**CSHARP**

```
documentEditor.Selection.CharacterFormat.SetFontColor("Pink");  
documentEditor.Selection.CharacterFormat.SetFontColor("FFC0CB");
```

**Font**

The font style of selected text can be get or set using the following sample code.

**JAVASCRIPT**

```
documentEditor.Selection.CharacterFormat.SetFontFamily("Arial");
```

**Highlight color**

The highlight color of the selected text can be get or set using the following sample code.

**CSHARP**

```
documentEditor.Selection.CharacterFormat.SetHighlightColor(HighlightColor.Pink);
```

You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

**Working with Paragraph Formatting in Blazor DocumentEditor Component**

[Blazor Document Editor](#) supports various paragraph formatting options such as text alignment, indentation, paragraph spacing, and more.

**Indentation**

You can modify the left or right indentation of selected paragraphs using the following sample code.

**CSHARP**

```
documentEditor.Selection.ParagraphFormat.SetLeftIndent(24);  
documentEditor.Selection.ParagraphFormat.SetRightIndent(24);
```

**Special indentation**

You can define special indent for first line of the paragraph using the following sample code.

**CSHARP**

```
documentEditor.Selection.ParagraphFormat.SetFirstLineIndent(24);
```

**Increase indent**

You can increase the left indent of selected paragraphs by a factor of 36 points using the following sample code.

**CSHARP**

```
documentEditor.Editor.IncreaseIndent();
```

**Decrease indent**

You can decrease the left indent of selected paragraphs by a factor of 36 points using the following sample code.

**CSHARP**

```
documentEditor.Editor.DecreaseIndent();
```

**Text alignment**

You can get or set the text alignment of selected paragraphs using the following sample code.

**CSHARP**

```
documentEditor.Selection.ParagraphFormat.SetTextAlignment(TextAlignment.Center);
```

You can toggle the text alignment of selected paragraphs by specifying a value using the following sample code.

**CSHARP**

```
documentEditor.Editor.ToggleTextAlignment(TextAlignment.Center);
```

**Line spacing and its type**

You can define the line spacing and its type for selected paragraphs using the following sample code.

**CSHARP**

```
documentEditor.Selection.ParagraphFormat.SetLineSpacingType(LineSpacingType.AtLeast);  
documentEditor.Selection.ParagraphFormat.SetLineSpacing(6);
```

**Paragraph spacing**

You can define the spacing before or after the paragraph by using the following sample code.

**CSHARP**

```
documentEditor.Selection.ParagraphFormat.SetBeforeSpacing(24);  
documentEditor.Selection.ParagraphFormat.SetAfterSpacing(24);
```

You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

**Working with Styles in Blazor DocumentEditor Component**

Styles are useful for applying a set of formatting consistently throughout the document. In document editor, styles are created and added to a document programmatically or via the built-in Styles dialog.

**Styles definition overview**

A Style in document editor should have the following properties:

- **name:** Name of the style. All styles in a document have a unique name, which is used as an identifier when applying the style.
- **type:** Specifies the document elements that the style will target. For example, paragraph or character.
- **next:** Specifies that the current style inherits the style set to this property. This is how hierarchical styles are defined.

- **link**: Provides a relation between the paragraph and character style.
- **characterFormat**: Specifies the properties of paragraph and character style.
- **paragraphFormat**: Specifies the properties of paragraph style.
- **basedOn**: Specifies that the current style inherits the style set to this property. This is how hierarchical styles are defined. It can be optional.

---

The style type should match the inherited style type. For example, it is not possible to have a character style inherit a paragraph style.

---

#### Default style

The default style for span and paragraph properties is normal. It internally inherits the default style of the document loaded or document editor component.

#### Style hierarchy

Each style initially checks its local value for the property that is being evaluated and turns to the style it is based on. If no local value is found, it turns to its default style. Style inheritance of different styles are listed as follows,

##### Character style

Character styles are based only on other character styles. The inheritance is: Character properties are inherited from the base character style.

##### Paragraph style

Paragraph styles are based on other paragraph styles or on linked styles. When a paragraph style is based on another paragraph style, the inheritance of the properties is as follows:

- Paragraph properties are inherited from the base paragraph style.
- Span properties are inherited from the base paragraph style.

When a paragraph style is based on a linked style, the inheritance of the properties is as follows:

- Paragraph properties are inherited from the paragraph style part in its base linked style.
- Span properties are inherited from the span style part in its base linked style.

##### Linked style

Linked styles are composite styles and their components are paragraph and character styles with link between them. To apply paragraph properties, take the properties from the linked paragraph style. Similarly, to apply character properties, take the properties from linked character style. Linked styles are based on other linked styles or on paragraph styles.

When a linked style is based on a paragraph style, the hierarchy of the properties is as follows:

- Paragraph properties are inherited from the 'basedOn' paragraph style.
- Character properties are inherited from the 'basedOn' paragraph style.

When a linked style is based on another linked style, the hierarchy of the properties is as follows:

- Paragraph properties are inherited from the paragraph style part in its base linked style.
- Span properties are inherited from the span style part in its base linked style.



## Defining new styles

New Styles are defined and added to the style collection of the document. In this way, they will be discovered by the default UI and applied to the parts of a document.

### Defining a character style

The following example shows how to programmatically create a character style.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor;
<SfDocumentEditor @ref="documentEditor" IsReadOnly=false EnableEditor=true
EnableSelection=true>
<DocumentEditorContainerEvents
Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditor>
@code {
SfDocumentEditor documentEditor;
protected void OnLoad(object args)
{
string styleJson = "{\"type\":\"Character\", \"name\":\"New
CharacterStyle\", \"basedOn\":\"Default Paragraph
Font\", \"characterFormat\":{\"fontSize\":16.0, \"fontFamily\":\"Calibri
Light\", \"fontColor\":\"#2F5496\", \"bold\":true, \"italic\":true, \"underline\
\": \"Single\"}}";
documentEditor.Editor.CreateStyle(styleJson);
documentEditor.Editor.ApplyStyle("New CharacterStyle");
}
}
```

### Defining a paragraph style

The following example shows how to programmatically create a paragraph style.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor;
<SfDocumentEditor @ref="documentEditor" IsReadOnly=false EnableEditor=true
EnableSelection=true>
<DocumentEditorContainerEvents
Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditor>
@code {
SfDocumentEditor documentEditor;
protected void OnLoad(object args)
{
string styleJson = "{\"type\":\"Paragraph\", \"name\":\"New
ParagraphStyle\", \"basedOn\":\"Normal\", \"characterFormat\":{\"fontSize\":16
.0, \"fontFamily\":\"Calibri
Light\", \"fontColor\":\"#2F5496\", \"bold\":true, \"italic\":true, \"underline\
\": \"Single\"}, \"paragraphFormat\":{\"leftIndent\":0.0, \"rightIndent\":0.0, \"
firstLineIndent\":0.0, \"beforeSpacing\":12.0, \"afterSpacing\":0.0, \"lineSpac
ing\":1.0791666507720947, \"lineSpacingType\":\"Multiple\", \"textAlignment\":
\"Left\", \"outlineLevel\":\"Level1\"}}";
documentEditor.Editor.CreateStyle(styleJson);
documentEditor.Editor.ApplyStyle("New ParagraphStyle");
}
}
```

## Defining a linked style

The following example shows how to programmatically create linked style.

### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor;
<SfDocumentEditor @ref="documentEditor" IsReadOnly=false EnableEditor=true
EnableSelection=true>
<DocumentEditorContainerEvents
Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditor>
@code {
SfDocumentEditor documentEditor;
protected void OnLoad(object args)
{
string styleJson = "{ \"type\": \"Paragraph\", \"name\": \"New
Linked\", \"basedOn\": \"Normal\", \"next\": \"Normal\", \"link\": \"New Linked
Char\", \"characterFormat\": { \"fontSize\": 16.0, \"fontFamily\": \"Calibri
Light\", \"fontColor\": \"#2F5496\" }, \"paragraphFormat\": { \"leftIndent\":
0.0, \"rightIndent\": 0.0, \"firstLineIndent\": 0.0, \"beforeSpacing\": 12.0, \"aft
erSpacing\": 0.0, \"lineSpacing\": 1.0791666507720947, \"lineSpacingType\": \"Mul
tiple\", \"textAlignment\": \"Left\", \"outlineLevel\": \"Level1\" } }";
documentEditor.Editor.CreateStyle(styleJson);
documentEditor.Editor.ApplyStyle("New Linked");
}
}
```

## Applying a style

The styles are applied using the **applyStyle** method of **editorModule**, the parameter should be passed is the **Name** of the Style.

The styles of the **Character** type is applied to the currently selected part of the document. If there is no selection, the values that will be applied to the word at caret position. The styles of **Paragraph** type follow the same logic and are applied to all paragraphs in the selection or the current paragraph.

When there is no selection, styles of **Linked** type will change the values of the paragraph, and apply both the Paragraph and Character properties. When there is selection, Linked Style changes only the character properties of the selected text.

For example, the following line will apply the "New Linked" to the current paragraph.

### CSHARP

```
documentEditor.Editor.ApplyStyle("New Linked");
//Clear direct formatting and apply the specified style
documentEditor.Editor.ApplyStyle("New Linked", true);
```

You can refer to our [Blazor Word Processor](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Word Processor example](#) to know how to render and configure the document editor.

## Working with Lists in Blazor DocumentEditor Component

[Blazor Document Editor](#) supports both the single-level and multilevel lists. Lists are used to organize data as step-by-step instructions in documents for easy understanding of key points. You can apply list to the paragraph either using supported APIs.

### Create bullet list

Bullets are usually used for unordered lists. To apply bulleted list for selected paragraphs, use `ApplyBullet(bullet, fontFamily)` method of `Editor` instance.

Parameter	Type	Description
-----------	------	-------------

-----	----	-----
-------	------	-------

Bullet	string	Bullet character.
--------	--------	-------------------

fontFamily	string	Bullet font family.
------------	--------	---------------------

Refer to the following sample code.

#### CSHARP

```
documentEditor.Editor.ApplyBullet("\uf0b7", "Symbol");
```

### Create numbered list

Numbered lists are usually used for ordered lists. To apply numbered list for selected paragraphs, use `ApplyNumbering(numberFormat,listLevelPattern)` method of `Editor` instance.

Parameter	Type	Description
-----------	------	-------------

-----	----	-----
-------	------	-------

numberFormat	string	“%n” representations in ‘numberFormat’ parameter will be replaced by respective list level’s value. “%1” will be displayed as “1”
--------------	--------	---

listLevelPattern(optional)	string	Default value is 'Arabic'.
----------------------------	--------	----------------------------

Refer to the following example.

#### CSHARP

```
documentEditor.Editor.ApplyNumbering("%1", ListLevelPattern.UpRoman);
```

### Clear list

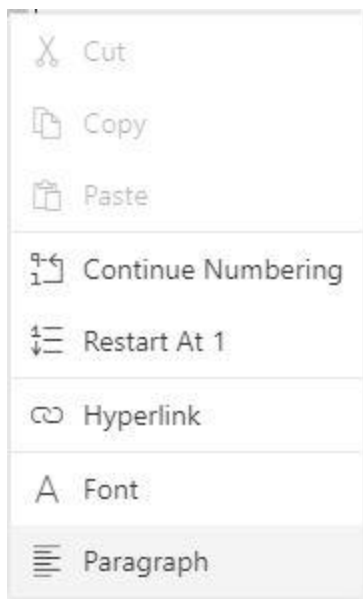
You can also clear the list formatting applied for selected paragraphs. Refer to the following sample code.

#### CSHARP

```
documentEditor.Editor.ClearList();
```

### Editing numbered list

Document editor restarts the numbering or continue numbering for a numbered list. These options are found in the built-in context menu, if the list value is selected. Refer to the following screenshot.



### Working with Table Formatting in Blazor DocumentEditor Component

[Blazor Document Editor](#) customizes the formatting of table, or table cells such as table width, cell margins, cell spacing, background color, and table alignment. This section describes how to customize these formatting for selected cells, rows, or table in detail.

#### Cell margins

You can customize the cell margins by using the following sample code.

##### CSHARP

```
//To change the left margin  
documentEditor.Selection.CellFormat.SetLeftMargin(5);  
//To change the right margin  
documentEditor.Selection.CellFormat.SetRightMargin(5);  
//To change the top margin  
documentEditor.Selection.CellFormat.SetTopMargin(5);  
//To change the bottom margin  
documentEditor.Selection.CellFormat.SetBottomMargin(5);
```

You can also define the default cell margins for a table. If the specific cell margin value is not defined explicitly in the cell formatting, the corresponding value will be retrieved from default cells margin of the table. Refer to the following sample code.

##### CSHARP

```
//To change the left margin  
documentEditor.Selection.TableFormat.SetLeftMargin(5);  
//To change the right margin  
documentEditor.Selection.TableFormat.SetRightMargin(5);  
//To change the top margin  
documentEditor.Selection.TableFormat.SetTopMargin(5);  
//To change the bottom margin  
documentEditor.Selection.TableFormat.SetBottomMargin(5);
```

### Background color

You can explicitly set the background color of selected cells using the following sample code.

#### CSHARP

```
documentEditor.Selection.CellFormat.SetBackground("#E0E0E0");
```

Refer to the following sample code to customize the background color of the table.

#### CSHARP

```
documentEditor.Selection.TableFormat.SetBackground("#E0E0E0");
```

### Cell spacing

Refer to the following sample code to customize the spacing between each cell in a table.

#### CSHARP

```
documentEditor.Selection.TableFormat.SetCellSpacing(2);
```

### Cell vertical alignment

The content is aligned within a table cell to **Top**, **Center**, or **Bottom**. You can customize this property of selected cells. Refer to the following sample code.

#### CSHARP

```
documentEditor.Selection.CellFormat.SetVerticalAlignment(CellVerticalAlignme  
nt.Bottom);
```

### Table alignment

The tables are aligned in document editor to **Left**, **Right**, or **Center**. Refer to the following sample code.

#### CSHARP

```
documentEditor.Selection.TableFormat.SetTableAlignment(TableAlignment.Center  
);
```

### Cell width

Set the desired width of table cells that will be considered when the table is layouted. Refer to the following sample code.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor;
<SfDocumentEditor @ref="documentEditor" IsReadOnly=false EnableEditor=true
EnableSelection=true>
<DocumentEditorContainerEvents
Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditor>
@code {
SfDocumentEditor documentEditor;
protected void OnLoad(object args)
{
documentEditor.Editor.InsertTable(2, 2);
}
```

```
documentEditor.Selection.CellFormat.SetPreferredWidth(100);
documentEditor.Selection.CellFormat.SetPreferredWidthType(WidthType.Point);
}
```

### Table width

You can set the desired width of a table in **Point** or **Percent** type. Refer to the following sample code.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
<SfDocumentEditor @ref="documentEditor" IsReadOnly=false EnableEditor=true
EnableSelection=true>
<DocumentEditorContainerEvents
Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditor>
@code {
SfDocumentEditor documentEditor;
protected void OnLoad(object args)
{
documentEditor.Editor.InsertTable(2, 2);
documentEditor.Selection.TableFormat.SetPreferredWidth(300);
documentEditor.Selection.TableFormat.SetPreferredWidthType(WidthType.Point);
}
}
```

### Working with row formatting

Document editor allows various row formatting such as height and repeat header.

#### Row height

You can customize the height of a table row as **Auto**, **AtLeast**, or **Exactly**. Refer to the following sample code.

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
<SfDocumentEditor @ref="documentEditor" IsReadOnly=false
EnableEditorHistory=true EnableEditor=true EnableSelection=true>
<DocumentEditorContainerEvents
Created="OnLoad"></DocumentEditorContainerEvents>
</SfDocumentEditor>
@code {
SfDocumentEditor documentEditor;
protected void OnLoad(object args)
{
documentEditor.Editor.InsertTable(2, 2);
documentEditor.Selection.RowFormat.SetHeight(20);
documentEditor.Selection.RowFormat.SetHeightType(HeightType.Exactly);
}
}
```

### Header row

The header row describes the content of a table. A table can optionally have a header row. Only the first row of a table can be the header row. If the cursor position is at first row of the table, then you can define whether it as header row or not, using the following sample code.

#### CSHARP

```
documentEditor.Selection.RowFormat.SetIsHeader(true);
```

### Allow row break across pages

This property is valid if a table row does not fit in the current page during table layout. It defines whether a table row can be allowed to break. If the value is false, the entire row will be moved to the start of next page. You can modify this property for selected rows using the following sample code.

#### CSHARP

```
documentEditor.Selection.RowFormat.SetAllowBreakAcrossPages(false);
```

You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

## Working with Section Formatting in Blazor DocumentEditor Component

[Blazor Document Editor](#) supports various section formatting such as page size, page margins, and more.

### Page size

You can get or set the size of a section at cursor position by using the following sample code.

#### CSHARP

```
documentEditor.Selection.SectionFormat.SetPageWidth(500);  
documentEditor.Selection.SectionFormat.SetPageHeight(600);
```

You can change the orientation of the page by swapping the values of page width and height respectively.

### Page margins

Left and right page margin defines the gap between the document content from left and right side of the page respectively. Top and bottom page margins defines the gap between the document content from header and footer of the page respectively.

Refer to the following sample code.

#### CSHARP

```
documentEditor.Selection.SectionFormat.SetLeftMargin(10);  
documentEditor.Selection.SectionFormat.SetRightMargin(10);  
documentEditor.Selection.SectionFormat.SetTopMargin(10);  
documentEditor.Selection.SectionFormat.SetBottomMargin(10);
```

### Header distance

You can define the distance of header content from the top of the page by using the following sample code.

## CSHARP

```
documentEditor.Selection.SectionFormat.SetHeaderDistance (72) ;
```

### Footer distance

You can define the distance of footer content from the bottom of the page by using the following sample code.

## CSHARP

```
documentEditor.Selection.SectionFormat.SetFooterDistance (72) ;
```

You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

## Find and Replace in Blazor DocumentEditor Component

Documents can be long, and you might need to search through the text to find and select specific words, text sequences, sentences, or paragraphs and then replace them with the desired content. This Word processor (DocumentEditor) provides a built-in navigation pane like Microsoft Word on the left of the editor.

You can use the search box at the top of this navigation pane to find all the instances of a specific word or phrase in the document. When you enter a word in the search box and perform search, it highlights all the occurrences of those words in the document and displays them below the search options in the pane. When you click to a search result, the cursor or selection move directly to that location in the document.

You can open the navigation pane using the **Ctrl+F** shortcut key and close using the **ESC** key.

### Show or hide navigation pane

You can programmatically toggle the visibility of navigation pane using the **ShowOptionsPane()** method.

## ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
<button @onclick="OpenOptionsPane">OpenOptionsPane</button>
<SfDocumentEditorContainer @ref="container"
EnableToolbar=true></SfDocumentEditorContainer>
@code {
SfDocumentEditorContainer container;
protected void OpenOptionsPane(object args)
{
container.DocumentEditor.ShowOptionsPane();
}
}
```

### Search

You can invoke the search or find text functionality programmatically using the **FindAll()** method. Also, you can customize the search operation with options such as “**match case**” and “**whole words only**”. The following code example explains how to perform text search without any search options.

## ASPX-CS



```
@using Syncfusion.Blazor.DocumentEditor
<button @onclick="FindAll">Find All</button>
<SfDocumentEditorContainer @ref="container"
EnableToolbar=true></SfDocumentEditorContainer>
@code {
SfDocumentEditorContainer container;
protected void FindAll(object args)
{
container.DocumentEditor.Search.FindAll("Some text", FindOption.None);
}
}
```

You can refer to our [Blazor Word Processor](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Word Processor example](#) to know how to render and configure the document editor.

## Accessibility in Blazor DocumentEditor Component

### Keyboard Shortcuts

#### Text formatting

The following table lists the default keyboard shortcuts in document editor for formatting text:

Key combination	Description
----- -----	
Ctrl + B	Toggles the bold property of selected text.
Ctrl + I	Toggles the italic property of selected text.
Ctrl + U	Toggles the underline property of selected text.
Ctrl + +	Toggles the subscript formatting of selected text.
Ctrl + Shift + +	Toggles the superscript formatting of selected contents.
Ctrl + }	Increases the actual font size of selected text by one point.
Ctrl + {	Decreases the actual font size of selected text by one point.

#### Paragraph formatting

The following table lists the default keyboard shortcuts for formatting the paragraph:

Key combination	Description
----- -----	
Ctrl + E	Selected paragraphs are center aligned.
Ctrl + J	Selected paragraphs are justified.
Ctrl + L	Selected paragraphs are left aligned.
Ctrl + R	Selected paragraphs are right aligned.
Ctrl + 1	Single line spacing is applied for selected paragraphs.
Ctrl + 5	1.5 line spacing is applied for selected paragraphs.
Ctrl + 2	Double spacing is applied for selected paragraphs.

|Ctrl + 0 | No spacing is applied before the selected paragraphs. |

|Ctrl + M | Increases the left indent of selected paragraphs by a factor of 36 points. |

|Ctrl + Shift + M | Decreases the left indent of selected paragraphs by a factor of 36 points. |

#### *Clipboard*

Key Combination	Description
-----	-----
Ctrl + C	Copies selected contents to the clipboard.
Ctrl + V	Pastes plain text content from the clipboard.
Ctrl + X	Moves selected content to the clipboard.

#### *Keyboard shortcut to navigate around the document*

Key Combination	Description
-----	-----
Left arrow	Moves the cursor position one character to the left.
Right arrow	Moves the cursor position one character to the right.
Down arrow	Moves the cursor position down one line.
Up arrow	Moves the cursor position up one line.
Ctrl + Left arrow	Moves the cursor position one word to the left.
Ctrl + Right arrow	Moves the cursor position one word to the right.
Ctrl + Up arrow	Moves the cursor position one paragraph up.
Ctrl + Down arrow	Moves the cursor position one paragraph down.
Tab (in table)	Moves the cursor position one cell to the right.
Shift + Tab (in table)	Moves the cursor position one cell to the left.
Home	Moves the cursor position to the start of a line.
End	Moves the cursor position to the end of a line.
Page up	Moves the cursor position one screen up.
Page down	Moves the cursor position one screen down.
Ctrl + Home	Moves the cursor position to the start of a document.
Ctrl + End	Moves the cursor position to the end of a document.

#### *Keyboard shortcut to extend selection*

Key Combination	Description
-----	-----
Shift + Left arrow	Extends selection one character to the left.
Shift + Right arrow	Extends selection one character to the right.
Shift + Down arrow	Extends selection one line downward.

|Shift + Up arrow| Extends selection one line upward. |

|Shift + Home| Extends selection to the start of a line. |

|Shift + End| Extends Selection to the end of a line. |

|Ctrl + A| Extends selection to the entire document. |

|Ctrl + Shift + Left arrow| Extends selection one word to the left. |

|Ctrl + Shift + Right arrow| Extends selection one word to the right. |

|Ctrl + Shift + Down arrow| Extends selection to the end of a paragraph. |

|Ctrl + Shift + Up arrow| Extends selection to the start of a paragraph. |

|Ctrl + Shift + Home| Extends selection to the start of a document. |

|Ctrl + Shift + End| Extends selection to the end of a document. |

#### Find and Replace

Key Combination	Description
-----	-----
Ctrl + F	Opens options pane.
Ctrl + H	Opens replace tab in options pane.

#### Print document

Key Combination	Description
-----	-----
Ctrl + P	Prints the document.

#### Edit Operation

Key Combination	Description
-----	-----
Backspace	Deletes one character to the left.
Delete	Deletes one character to the right.
Ctrl + Z	Undo last performed action.
Ctrl + Y	Redo last undo action.

#### Insert special characters

Key Combination	Description
-----	-----
Ctrl + Enter	Inserts page break.
Shift + Enter	Inserts line break.

#### Dialog

Key Combination	Description
-----	-----
Ctrl + F	Opens options pane.

|Ctrl + D| Opens font dialog.|

|Ctrl + K| Opens hyperlink dialog.|

You can refer to our [Blazor Word Processor](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Word Processor example](#) to know how to render and configure the document editor.

## Restrict editing in Blazor DocumentEditor Component

This [Blazor Word Processor control](#) (DocumentEditor) provides 2 types of restriction for editing:

- Read-only: Allows all the users to view the document contents but not make changes to it.
- Allows changes to certain portion of the document: Allows the users to edit to certain portion of the document.

### Read only

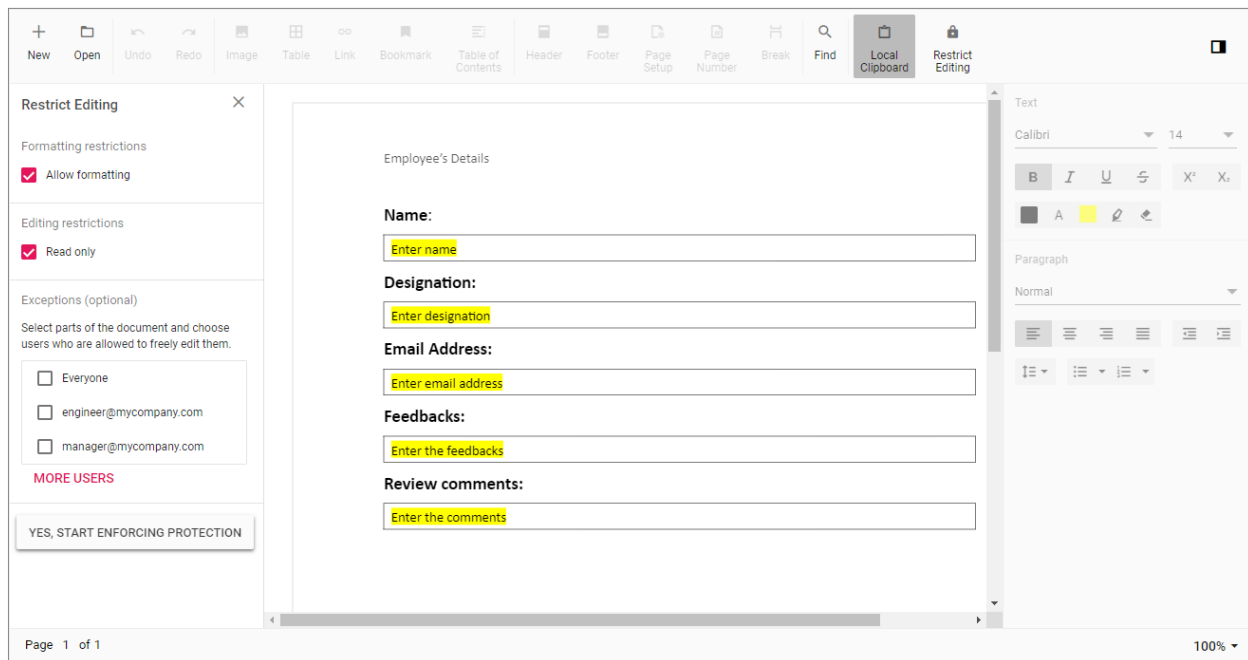
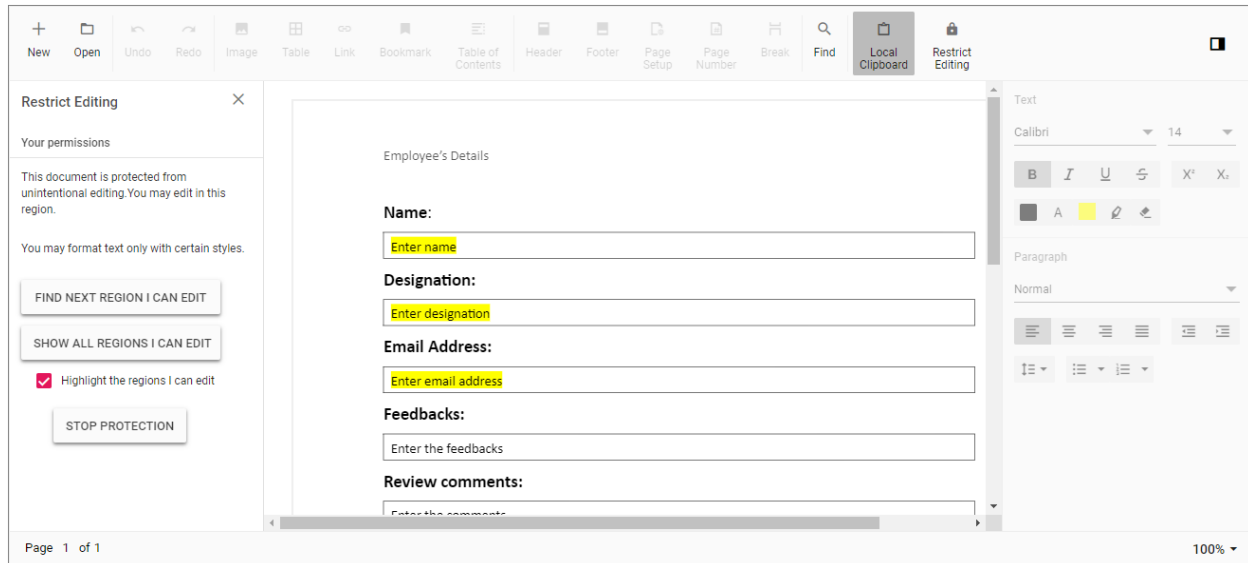
The following code example shows how to restrict or protect editing for the entire content (show as read-only).

#### ASPX-CS

```
@using Syncfusion.Blazor.DocumentEditor
<button @onclick="ReadOnly">Read Only</button>
<SfDocumentEditorContainer @ref="container"
EnableToolbar=true></SfDocumentEditorContainer>
@code {
    SfDocumentEditorContainer container;
    public void ReadOnly(object args)
    {
        container.RestrictEditing = true;
    }
}
```

### Allows changes to certain portion of the document

Also, at some situations, you might need to allow changes for a certain portion of the document alone. Microsoft Word allows you to [make changes to parts of a Word document](#). Likewise, the document editor control allows the users to make changes to certain parts of a document using similar user interface.



You can also explore our [Blazor Word Processor](#) example to know how to render and configure the document editor.

## Dropdown Menu

<!-- markdownlint-disable MD024 -->

### Getting Started with Blazor Dropdown Menu Component

This section briefly explains about how to include Dropdown 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.

### 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.

---

Please ensure to check the **Include prerelease** option.

---

#### 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.

#### C#

```
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 Dropdown Menu component to the application

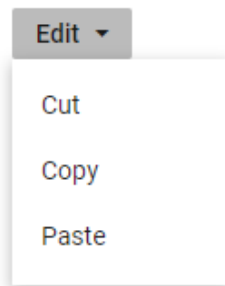
Now, add the Syncfusion Blazor Dropdown Menu component in `razor` page in the `Pages` folder. For example, the Dropdown Menu component is added in the `~/Pages/Index.razor` page.

#### ASPX-CS

```
<SfDropDownButton Content="Edit">  
<DropDownMenuItems>  
<DropDownMenuItem Text="Cut"></DropDownMenuItem>  
<DropDownMenuItem Text="Copy"></DropDownMenuItem>  
<DropDownMenuItem Text="Paste"></DropDownMenuItem>  
</DropDownMenuItems>  
</SfDropDownButton>
```

#### Run the application

After successful compilation of your application, simply press F5 to run the application. The Blazor Dropdown 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](#)

## Events in Blazor Dropdown Menu Component

You can define the dropdown menu event using on [DropDownButtonEvents](#) in component. The value of event is treated as an event handler. The event specific data will be available in event arguments.

### List of events supported

We have provided the following event support to the DropDownButton component. The different event argument types for each event are,

- OnOpen - BeforeOpenCloseMenuEventArgs
- Opened - OpenCloseMenuEventArgs
- ItemSelected - MenuEventArgs
- OnClose – BeforeOpenCloseMenuEventArgs
- OnItemRender – MenuEventArgs
- Closed – OpenCloseMenuEventArgs

### How to bind event to Dropdown Menu

Above defined events are bind the Dropdown Menu component. Here, we have explained about the sample code snippets of Dropdown Menu.

#### ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfDropDownButton Content="Profile">
  <DropDownButtonEvents Created="Created" OnOpen="OnOpen" Opened="Opened"
  ItemSelected="ItemSelected" OnClose="OnClose" OnItemRender="ItemRender"
  Closed="Closed">
  </DropDownButtonEvents>
  <DropDownMenuItem>
    <DropDownMenuItem Text="Dashboard" Id="Dashboard"></DropDownMenuItem>
    <DropDownMenuItem Text="Notifications"
    Id="Notifications"></DropDownMenuItem>
    <DropDownMenuItem Text="User Settings" Id="UserSettings"></DropDownMenuItem>
```



```

<DropDownMenuItem Text="Log Out" Id="Logout"></DropDownMenuItem>
</DropDownMenuItems>
</SfDropDownButton>
@code {
private void Created()
{
}
private void OnOpen(BeforeOpenCloseMenuEventArgs args)
{
}
private void Opened(OpenCloseMenuEventArgs args)
{
}
private void ItemSelected(MenuEventArgs args)
{
}
private void OnClose(BeforeOpenCloseMenuEventArgs args)
{
}
private void ItemRender(MenuEventArgs args)
{
}
private void Closed(OpenCloseMenuEventArgs args)
{
}
}

```

## Icons in Blazor Dropdown Menu Component

### Dropdown Menu icons

Dropdown Menu can have an icon to provide the visual representation of the action. To place the icon on a Dropdown Menu, set the [IconCss](#) property to `e-icons` with the required icon CSS. By default, the icon is positioned to the left side of the Dropdown Menu. You can customize the icon's position using the [IconPosition](#) property.

In the following example, the Dropdown Menu with default iconPosition and iconPosition as `Top` is showcased:

### ASPX-CS

```

@using Syncfusion.Blazor.SplitButtons
<SfDropDownButton IconCss="e-icons e-message" content="Message">
<DropDownMenuItems>
<DropDownMenuItem Text="Edit"></DropDownMenuItem>
<DropDownMenuItem Text="Delete"></DropDownMenuItem>
<DropDownMenuItem Text="Mark as Read"></DropDownMenuItem>
<DropDownMenuItem Text="Like Message"></DropDownMenuItem>
</DropDownMenuItems>
</SfDropDownButton>
<SfDropDownButton IconCss="e-icons e-message"
IconPosition="SplitButtonIconPosition.Top" Content="Message">
<DropDownMenuItems>
<DropDownMenuItem Text="Edit"></DropDownMenuItem>
<DropDownMenuItem Text="Delete"></DropDownMenuItem>
<DropDownMenuItem Text="Mark as Read"></DropDownMenuItem>
<DropDownMenuItem Text="Like Message"></DropDownMenuItem>

```

```

</DropDownMenuItems>
</SfDropDownButton>
<style>
.e-message::before {
content: '\e7cb';
}
</style>

```

Output be like



You can also use third party icons on the Dropdown Menu using the [IconCss](#) property.

#### Vertical button

Vertical button in Dropdown Menu can be achieved by adding `e-vertical` class using [CssClass](#) property.

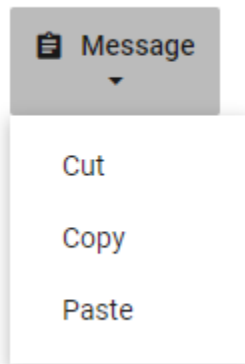
#### ASPX-CS

```

@using Syncfusion.Blazor.SplitButtons
<SfDropDownButton IconCss="e-icons e-message" CssClass="e-vertical"
Content="Message">
<DropDownMenuItems>
<DropDownMenuItem Text="Cut"></DropDownMenuItem>
<DropDownMenuItem Text="Copy"></DropDownMenuItem>
<DropDownMenuItem Text="Paste"></DropDownMenuItem>
</DropDownMenuItems>
</SfDropDownButton>
<style>
.e-message::before {
content: '\e7cb';
}
</style>

```

Output be like



See Also

- [Dropdown popup with icons](#)
- [Customized icon size](#)

## Popup Items in Blazor Dropdown Menu Component

### Icons

The popup action item have an icon or image to provide visual representation of the action. To place the icon on a popup item, set the [IconCss](#) property to `e-icons` with the required icon CSS. By default, the icon is positioned to the left side of the popup action item.

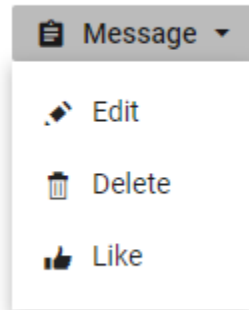
In the following sample, the icons for edit, delete, mark as read and like message menu items are added using the `IconCss` property.

### ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfDropDownButton Content="Message" IconCss="e-icons e-message">
  <DropDownMenuItems>
    <DropDownMenuItem IconCss="e-icons e-edit" Text="Edit"></DropDownMenuItem>
    <DropDownMenuItem IconCss="e-icons e-delete"
      Text="Delete"></DropDownMenuItem>
    <DropDownMenuItem IconCss="e-icons e-like" Text="Like"></DropDownMenuItem>
  </DropDownMenuItems>
</SfDropDownButton>
<style>
.e-message::before {
content: '\e7cb';
}
.e-edit::before {
content: '\e78f';
}
.e-delete::before {
content: '\e773';
}
.e-like::before {
content: '\e682';
}
```

```
</style>
```

Output be like



### Separator

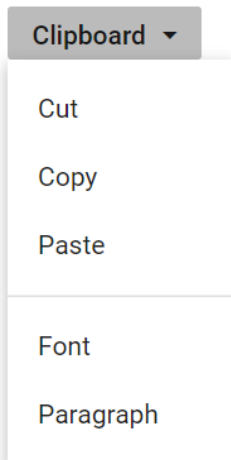
The Separators are the horizontal lines that are used to separate the popup items. You cannot select the separators. You can enable separators to group the popup items using the [Separator](#) property.

In the following sample, cut, copy, and paste popup items are grouped using the separator property:

### ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfDropDownButton Content="Clipboard">
  <DropDownMenuItems>
    <DropDownMenuItem Text="Cut"></DropDownMenuItem>
    <DropDownMenuItem Text="Copy"></DropDownMenuItem>
    <DropDownMenuItem Text="Paste"></DropDownMenuItem>
    <DropDownMenuItem Separator=true></DropDownMenuItem>
    <DropDownMenuItem Text="Font"></DropDownMenuItem>
    <DropDownMenuItem Text="Paragraph"></DropDownMenuItem>
  </DropDownMenuItems>
</SfDropDownButton>
```

Output be like



### Navigations

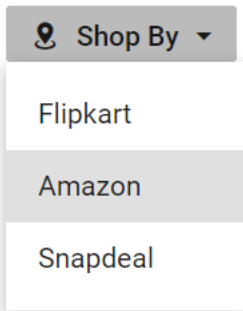
Actions in Dropdown Menu can be used to navigate to the other web page when action item is clicked. This can be achieved by providing link to the action item using [url](#) property.

In the following sample, navigation URL for Flipkart, Amazon, and Snapdeal action items are added using the url property:

#### ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfDropDownButton IconCss="e-icons e-shopping" Content="Shop By">
  <DropDownMenuItems>
    <DropDownMenuItem Text="Flipkart"
      Url="https://www.google.co.in/search?q=flipkart"></DropDownMenuItem>
    <DropDownMenuItem Text="Amazon"
      Url="https://www.google.co.in/search?q=amazon"></DropDownMenuItem>
    <DropDownMenuItem Text="Snapdeal"
      Url="https://www.google.co.in/search?q=snapdeal"></DropDownMenuItem>
  </DropDownMenuItems>
</SfDropDownButton>
<style>
.e-shopping::before {
  content: '\e7d0';
}
</style>
```

Output be like



## Template

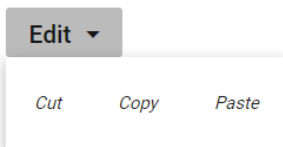
### Item Templating

Popup items can be customized using the `CssClass` property. We have customize the items using CSS style.

### ASPX-CS

```
<SfDropDownButton Content="Edit" CssClass="custom">
  <DropDownMenuItems>
    <DropDownMenuItem Text="Cut"></DropDownMenuItem>
    <DropDownMenuItem Text="Copy"></DropDownMenuItem>
    <DropDownMenuItem Text="Paste"></DropDownMenuItem>
  </DropDownMenuItems>
</SfDropDownButton>
<style>
.custom li {
float: left;
font-size: 10px;
padding-left: 50px;
font-style: oblique;
}
</style>
```

Output be like



## Accessibility in Blazor Dropdown Menu Component

### ARIA attributes

The web accessibility makes web content and web applications more accessible for people with disabilities. Mostly it helps in dynamic content change and development of advanced user interface controls with AJAX, HTML, JavaScript, and related technologies.

Dropdown Menu provides built-in compliance with WAI-ARIA specifications. WAI-ARIA support is achieved through the attributes like `aria-expanded`, `aria-owns` and `aria-haspopup` applied for action item in

Dropdown Menu. It helps by providing information about the widget for assistive technology in the screen readers. Dropdown Menu component contains the `Menu` role and `MenuItem` role.

| Properties | Functionality |

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

| `menu` | Specified for an Dropdown Menu element. |

| `menuItem` | Specified for an action items. |

| `aria-haspopup` | Indicates the availability and type of interactive dropdown popup element. |

| `aria-expanded` | Indicates whether the current state of the dropdown popup can be expanded or collapsed. |

| `aria-owns` | Identifies elements to define a visual, functional, or contextual parent or child relationship between DOM(Document Object Model) elements where the hierarchy cannot be used to represent the relationship. |

#### Keyboard interaction

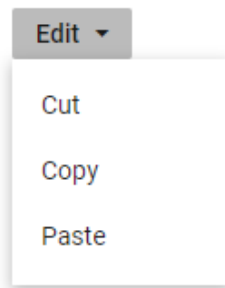
<!-- markdownlint-disable MD033 -->

Keyboard shortcuts	Actions
Esc	Closes the popup.
Enter	Opens the popup, or activates the highlighted item and closes the popup.
Space	Opens the popup.
Up	Navigates up or to the previous action item.
Down	Navigates down or to the next action item.
Alt + Up Arrow	Closes the popup.
Alt + Down Arrow	Opens the popup

#### ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfDropDownButton Content="Edit">
  <DropDownMenuItems>
    <DropDownMenuItem Text="Cut"></DropDownMenuItem>
    <DropDownMenuItem Text="Copy"></DropDownMenuItem>
    <DropDownMenuItem Text="Paste"></DropDownMenuItem>
  </DropDownMenuItems>
</SfDropDownButton>
```

Output be like



## Styles and Appearances in Blazor Dropdown Menu Component

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

|CSS Class | Purpose of Class|

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

|.e-dropdown-btn|To customize the dropdown button. |

|.e-dropdown-btn:hover|To customize the dropdown button on hover. |

|.e-dropdown-btn.e-active|To customize the dropdown button on active. |

|.e-dropdown-popup|To customize the dropdown button pop up. |

|.e-dropdown-popup ul .e-item:hover|To customize the dropdown button pop up items on hover. |

|.e-dropdown-popup ul .e-item:active|To customize the dropdown button pop up items on active. |

## How To

### Change caret icon in Blazor Dropdown Menu Component

Dropdown arrow can be customized on popup open and close. It can be handled in [OnOpen](#) and [OnClose](#) event.

In the following example, the up arrow is updated on popup close and down arrow is updated on popup open using [OnOpen](#) and [OnClose](#) event by adding and removing `e-caret-up` class.

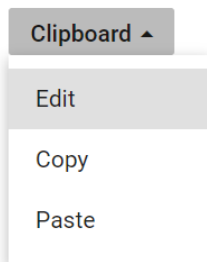
### ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfDropDownButton CssClass="@IconCss" Content="Clipboard">
  <DropDownButtonEvents OnOpen="beforeOpen"
    OnClose="beforeClose"></DropDownButtonEvents>
  <DropDownMenuItems>
    <DropDownMenuItem Text="Edit"></DropDownMenuItem>
    <DropDownMenuItem Text="Copy"></DropDownMenuItem>
    <DropDownMenuItem Text="Paste"></DropDownMenuItem>
  </DropDownMenuItems>
</SfDropDownButton>
@code {
  public string IconCss = "";
```



```
private void beforeOpen(BeforeOpenCloseMenuEventArgs args)
{
    this.IconCss = "e-caret-up";
    this.StateHasChanged();
}
private void beforeClose(BeforeOpenCloseMenuEventArgs args)
{
    this.IconCss = "";
    this.StateHasChanged();
}
}
<style>
.e-caret {
transform: rotate(0deg);
transition: transform 200ms ease-in-out;
}
.e-caret-up .e-caret {
transform: rotate(180deg);
}
</style>
```

Output be like



### Create Blazor Dropdown Menu with Rounded Corner

Dropdown Menu with rounded corner can be achieved by adding `border-radius` CSS property to button element.

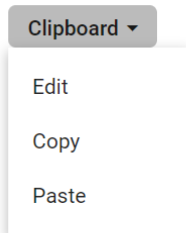
In the following example, `e-round-corner` class is defined with `5px border-radius` property and added that class to button element using `CssClass` property.

#### ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfDropDownButton CssClass="e-round-corner" Content="Clipboard">
  <DropDownMenuItems>
    <DropDownMenuItem Text="Edit"></DropDownMenuItem>
    <DropDownMenuItem Text="Copy"></DropDownMenuItem>
    <DropDownMenuItem Text="Paste"></DropDownMenuItem>
  </DropDownMenuItems>
</SfDropDownButton>
<style>
.e-round-corner {
border-radius: 5px;
```

```
}
</style>
```

Output be like



### Create right-to-left Blazor Dropdown Menu Component

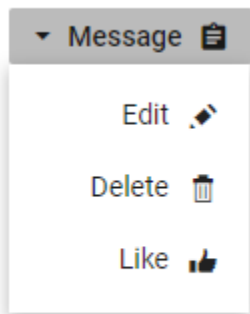
Dropdown Menu component has RTL support. This can be achieved by setting [EnableRtl](#) as true.

The following example illustrates how to enable right-to-left support in Dropdown Menu component.

#### ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfDropDownButton Content="Message" IconCss="e-icons e-message"
EnableRtl="true">
  <DropDownMenuItems>
    <DropDownMenuItem IconCss="e-icons e-edit" Text="Edit"></DropDownMenuItem>
    <DropDownMenuItem IconCss="e-icons e-delete"
Text="Delete"></DropDownMenuItem>
    <DropDownMenuItem IconCss="e-icons e-like" Text="Like"></DropDownMenuItem>
  </DropDownMenuItems>
</SfDropDownButton>
<style>
.e-message::before {
content: '\e7cb';
}
.e-edit::before {
content: '\e78f';
}
.e-delete::before {
content: '\e773';
}
.e-like::before {
content: '\e682';
}
</style>
```

Output be like



### Customize icon and width in Blazor Dropdown Menu Component

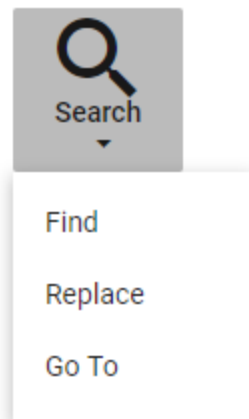
Width of the Dropdown Menu can be customized by setting required width to the dropdown element.

The following UI can be achieved by setting [IconPosition](#) as `Top`, width as `85px` and size of the font icon as `40px` by adding `e-custom` class.

#### ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfdDropDownButton IconCss="e-icons e-search" CssClass="e-custom"
IconPosition = "SplitButtonIconPosition.Top" Content="Search">
  <DropDownMenuItems>
    <DropDownMenuItem Text="Find"></DropDownMenuItem>
    <DropDownMenuItem Text="Replace"></DropDownMenuItem>
    <DropDownMenuItem Text="Go To"></DropDownMenuItem>
  </DropDownMenuItems>
</SfdDropDownButton>
<style>
.e-search::before {
content: '\e724';
}
.e-dropdown-btn.e-custom {
width: 85px;
}
.e-dropdown-btn.e-custom .e-search::before {
font-size: 40px;
}
</style>
```

Output be like



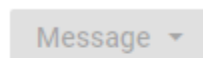
### Disable a Dropdown Menu in Blazor Dropdown Menu Component

Dropdown Menu component can be enabled/disabled by giving [Disabled](#) property. To disable Dropdown Menu component, the disabled property can be set as `true`.

#### ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfDropDownButton Disabled="true" Content="Message">
  <DropDownMenuItems>
    <DropDownMenuItem Text="Edit"></DropDownMenuItem>
    <DropDownMenuItem Text="Delete"></DropDownMenuItem>
    <DropDownMenuItem Text="Mark as Read"></DropDownMenuItem>
    <DropDownMenuItem Text="Like Message"></DropDownMenuItem>
  </DropDownMenuItems>
</SfDropDownButton>
```

Output be like



### Group dropdown listview items in Blazor Dropdown Menu Component

Header in popup items is possible in Dropdown Menu by templating entire popup with ListView. Create ListView with [ID](#) listview and provide it as a `PopupContent` for Dropdown Menu.

In the following example, ListView element is given as `PopupContent` to Dropdown Menu and header can be achieved by [GroupBy](#) property.

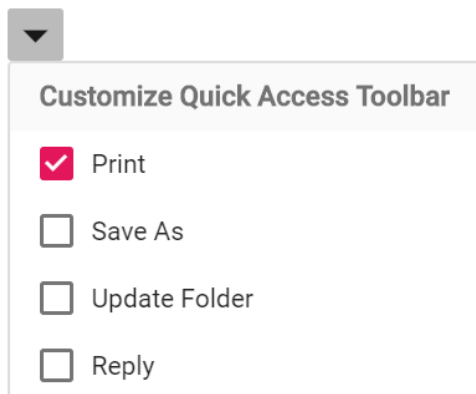
**ASPX-CS**

```

<SfDropDownButton CssClass="e-caret-hide" IconCss="e-icons e-down">
  <PopupContent>
    <SfListView ID="listview" DataSource="@Data" ShowCheckBox="true">
      <ListViewFieldSettings Text="Text" GroupBy="Category"
        TValue="ListData"></ListViewFieldSettings>
    </SfListView>
  </PopupContent>
</SfDropDownButton>
@code {
public List<ListData> Data = new List<ListData>{
  new ListData{ Class = "data", Text = "Print", Id = "data1", Category =
    "Customize Quick Access Toolbar" },
  new ListData{ Class = "data", Text = "Save As", Id = "data2", Category =
    "Customize Quick Access Toolbar" },
  new ListData{ Class = "data", Text = "Update Folder", Id = "data3", Category =
    "Customize Quick Access Toolbar"},
  new ListData{ Class = "data", Text = "Reply", Id = "data4", Category =
    "Customize Quick Access Toolbar" }
};
public class ListData
{
  public string Text { get; set; }
  public string Id { get; set; }
  public string Class { get; set; }
  public string Category { get; set; }
}
}

```

Output be like



### Hide dropdown arrow in Blazor Dropdown Menu Component

You can hide the dropdown arrow from the Dropdown Menu by adding class `e-caret-hide` to Dropdown Menu element using [CssClass](#) property.

**ASPX-CS**

```

@using Syncfusion.Blazor.SplitButtons
<SfDropDownButton CssClass="e-caret-hide" Content="Message">
  <DropDownMenuItems>

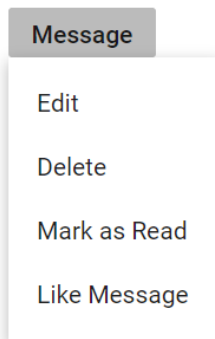
```

```

<DropDownMenuItem Text="Edit"></DropDownMenuItem>
<DropDownMenuItem Text="Delete"></DropDownMenuItem>
<DropDownMenuItem Text="Mark as Read"></DropDownMenuItem>
<DropDownMenuItem Text="Like Message"></DropDownMenuItem>
</DropDownMenuItems>
</SfDropDownButton>

```

Output be like



Open a dialog on popup item click in Blazor Dropdown Menu Component

This section explains about how to open a dialog on Dropdown Menu popup item click. This can be achieved by handling dialog open in [ItemSelected](#) event of the Dropdown Menu.

In the following example, Dialog will open while selecting **Other Folder...** item.

#### ASPX-CS

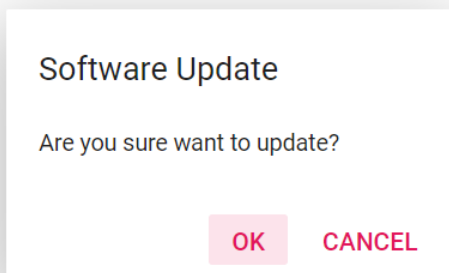
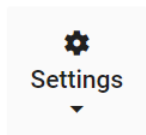
```

@using Syncfusion.Blazor.SplitButtons
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
<SfDropDownButton Content="Settings" IconCss="e-icons e-setting-icon"
  CssClass="e-vertical" IconPosition="SplitButtonIconPosition.Top">
  <DropDownButtonEvents ItemSelected="select"></DropDownButtonEvents>
  <DropDownMenuItems>
    <DropDownMenuItem Text="Help"></DropDownMenuItem>
    <DropDownMenuItem Text="About"></DropDownMenuItem>
    <DropDownMenuItem Text="Update"></DropDownMenuItem>
  </DropDownMenuItems>
</SfDropDownButton>
<SfDialog Content="@Content" Header="@Header" Width="250px" Height="150px"
  Visible="false" @ref="DialogObj" >
  <DialogPositionData X="300" Y="200"></DialogPositionData>
  <DialogButtons>
    <DialogButton OnClick="@click">
    <DialogButton Content="OK" IsPrimary="true"></DialogButton>
  </DialogButtons>
  <DialogButton OnClick="@click">
  <DialogButton Content="Cancel"></DialogButton>
</DialogButtons>
</SfDialog>
@code {

```

```
SfDialog DialogObj;  
public string Content = "Are you sure want to update?";  
public string Header = "Software Update";  
private void click(object args)  
{  
    DialogObj.Hide();  
}  
private void select(MenuEventArgs args)  
{  
    if (args.Item.Text == "Update")  
    {  
        DialogObj.Show();  
    }  
}  
  
<style>  
.e-setting-icon::before {  
    content: '\e679';  
}  
</style>
```

Output be like



#### Position popup open in Blazor Dropdown Menu Component

Popup open position can be changed according to the requirement. We have set the Popup open position using `CssClass` property as `custom` in `Top` and `Left` for the popup element.

In the following example, the `Top` position of the popup element.

#### C#

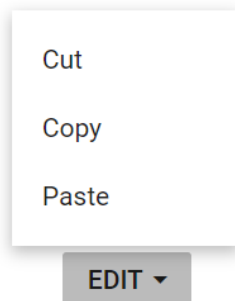
```
@using Syncfusion.Blazor.SplitButtons  
<SfDropDownButton Content="EDIT" CssClass="custom">  
<DropDownMenuItems>  
<DropDownMenuItem Text="Cut"></DropDownMenuItem>  
<DropDownMenuItem Text="Copy"></DropDownMenuItem>  
<DropDownMenuItem Text="Paste"></DropDownMenuItem>  
</DropDownMenuItems>
```

```

</SfDropDownButton>
<style>
.custom.e-dropdown-popup {
margin: -122px -27px;
}
</style>

```

Output be like



### Create Dropdown List in Popup of Blazor Dropdown Menu Component

We have render the `DropDownList` component in `DropDownMenu` popup using `PopupContent` property.

In the following example, render `DropDownList` component in popup.

#### ASPX-CS

```

@using Syncfusion.Blazor.SplitButtons
@using Syncfusion.Blazor.DropDowns
<SfDropDownButton CssClass="e-caret-hide" Content="Games">
  <PopupContent>
    <div id="dropDownFilterExpenses">
      <SfDropDownList TValue="string" TItem="GameFields" PopupHeight="230px"
        Placeholder="Select a game" DataSource="@Games">
        <DropDownListFieldSettings Text="Text"
          Value="ID"></DropDownListFieldSettings>
        <DropDownListEvents Opened="OpenPopup" OnClose="beforeClose" TValue="string"
          TItem="GameFields"></DropDownListEvents>
      </SfDropDownList>
    </div>
  </PopupContent>
  <ChildContent>
    <DropDownButtonEvents OnClose="popupClose"></DropDownButtonEvents>
  </ChildContent>
</SfDropDownButton>
@code {
private bool ispopup = false;
private void popupClose(BeforeOpenCloseMenuEventArgs args)
{
args.Cancel = this.ispopup;
}
private void OpenPopup(PopupEventArgs args)
{
this.ispopup = true;
}
}

```

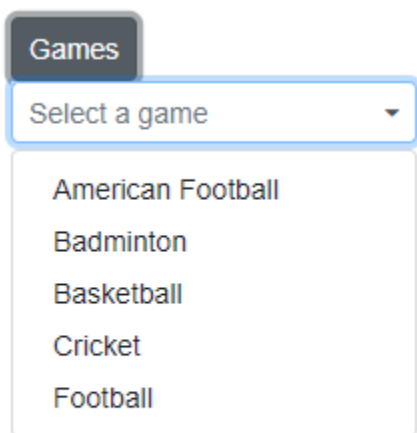


```

}
private void beforeClose(PopupEventArgs args)
{
    this.ispopUp = 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" }
};
}
<style>
.e-dropdown-popup.e-transparent div {
display: none;
}
</style>

```

Output be like



### Add and Remove Items in Blazor Dropdown Menu Component

Dropdown Menu component can be dynamically add or remove items using `AddItems`, `RemoveItems` method.

The following example illustrates how to add and remove items in Dropdown Menu component.

#### ASPX-CS

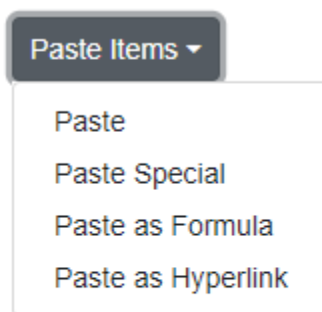
```

@using Syncfusion.Blazor.SplitButtons
@using Syncfusion.Blazor.Buttons
<SfDropDownButton Content="Paste Items" @ref="DropDownbuttonRef">

```

```
<DropDownMenuItems>
<DropDownMenuItem Text="Paste"></DropDownMenuItem>
</DropDownMenuItems>
</SfDropDownButton>
<div>
<SfButton Content="Additem" IsPrimary="true" @onclick="addItem"></SfButton>
<SfButton Content="Removeitem" IsPrimary="true"
@onclick="removeItem"></SfButton>
</div>
@code {
SfDropDownButton DropDownbuttonRef;
private void addItem()
{
DropDownbuttonRef.AddItems(dropdownbtnItems);
}
private void removeItem()
{
DropDownbuttonRef.RemoveItems(removeItems);
}
public List<DropDownMenuItem> dropdownbtnItems = new List<DropDownMenuItem>
{
new DropDownMenuItem{ Text="Paste Special" },
new DropDownMenuItem{ Text="Paste as Formula" },
new DropDownMenuItem{ Text="Paste as Hyperlink" }
};
public List<string> removeItems = new List<string>()
{
"Paste"
};
}
```

Output be like



## DropDown List

### Getting Started with Blazor DropDown List Component

This section briefly explains how to include a **DropDownList** 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 DropDownList component, you can check on this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=bavTLtVAn6I"%}

### Importing Syncfusion Blazor component in the application

- Install **Syncfusion.Blazor.DropDowns** NuGet package to the application by using the **NuGet Package Manager**.

---

Please ensure to check the **Include prerelease** option for our Beta release.

---

- 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 DropDownList component to the application

To initialize the DropDownList component add the below code to your **Index.razor** view page which is present under **~/Pages** folder.

#### ASPX-CS

```
<SfDropDownList TValue="string" TItem="string" Placeholder="Select a
game"></SfDropDownList>
```

Output be like below

Select a game ▼

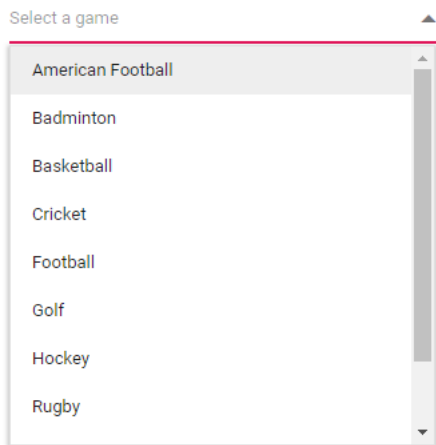
### Binding data source

After initialization, populate the DropDownList with data using the [DataSource](#) property. Here, an array of object values is passed to the DropDownList component. **Item** specifies the type of the Datasource in DropDownList.

#### ASPX-CS

```
<SfDropDownList TValue="string" TItem="Games" Placeholder="Select a game"
DataSource="@LocalData">
  <DropDownListFieldSettings Value="ID"
  Text="Text"></DropDownListFieldSettings>
</SfDropDownList>
@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 DropDownList 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.

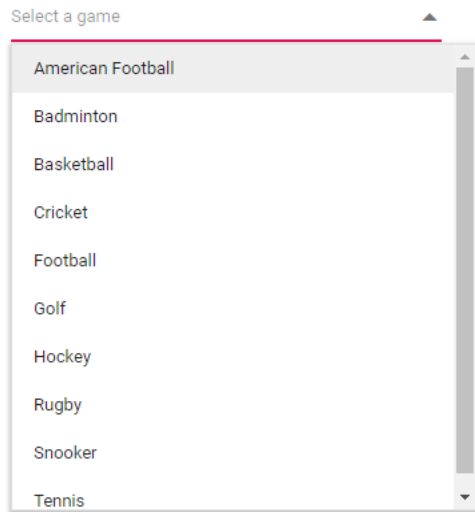
In the following sample, popup list's width and height are configured.

### ASPX-CS

```
<SfDropDownList TValue="string" TItem="Games" PopupHeight="350px"
PopupWidth="350px" Placeholder="Select a game" DataSource="@LocalData">
  <DropDownListFieldSettings Value="ID"
  Text="Text"></DropDownListFieldSettings>
</SfDropDownList>
```

```
</SfDropDownList>
@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 DropDown List Component

Data binding can be achieved by using the **bind-Value** attribute and it supports string, int, Enum and bool types. If component value has been changed, it will affect the all places where you bind the variable for the **bind-value** attribute.

### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<p>DropDownList value is:<strong>@DropVal</strong></p>
<SfDropDownList TValue="string" Placeholder="e.g. Australia"
TItem="Countries" @bind-Value="@DropVal" DataSource="@Country">
<DropDownListFieldSettings Value="Name"></DropDownListFieldSettings>
</SfDropDownList>
@code {
public string DropVal;
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
<SfAutoComplete TValue="string" Placeholder="e.g. Australia"
TItem="Countries" @bind-Index="@ddlIndex" DataSource="@Country">
<AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
@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 DropDown List Component

The DropDownList loads the data either from local data sources or remote data services using the [DataSource](#) property. It supports the data type of `array` or [DataManager](#).

The DropDownList also supports different kinds of data services such as OData, OData V4, and Web API, and data formats such as XML, JSON, and JSONP with the help of [DataManager](#) adaptors.

Fields	Type	Description
Text	string	Specifies the display text of each list item.
Value	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.

---

When binding complex data to the DropDownList, fields should be mapped correctly. Otherwise, the selected item remains undefined.

---

### Binding local data

Local data can be represented in two ways as described below.

#### Array of JSON data

The DropDownList can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [Fields](#) property.

In the following example, **Name** column from complex data have been mapped to the **Value** field.

### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" TItem="Countries" Placeholder="e.g.
Australia" DataSource="@Country">
  <DropDownListFieldSettings Text="Name"
  Value="Code"></DropDownListFieldSettings>
</SfDropDownList>

@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" },
    }
}
```

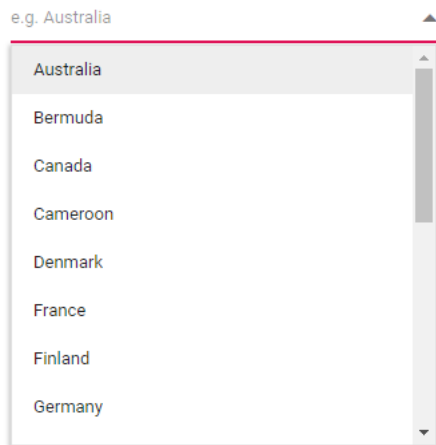


```

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 DropDownList can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [Fields](#) property.

In the following example, `Code.ID` column and `Country.CountryID` column from complex data have been mapped to the `Value` field and `Text` field, respectively.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" TItem="Complex" Placeholder="e.g. Select a
country" DataSource="@LocalData">
  <DropDownListFieldSettings Text="Country.CountryID"
  Value="Code.ID"></DropDownListFieldSettings>
</SfDropDownList>
@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; }
}
}

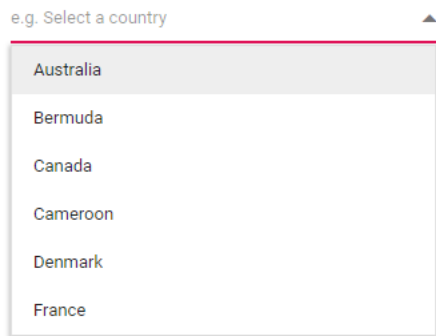
```

```

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 DropDownList supports retrieval of data from remote data services with the help of [DataManager](#) control. The [Query](#) property is used to fetch data from the database and bind it to the DropDownList.

The following sample displays the first 6 contacts from **Customers** table of the **Northwind** Data Service.

### ASPX-CS

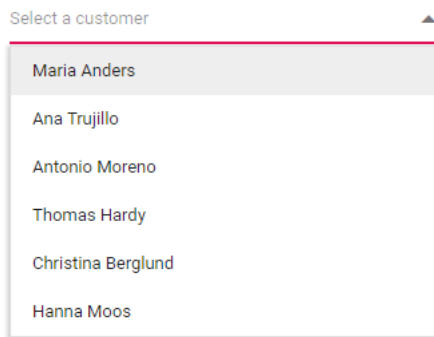
```

@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" TItem="OrderDetails" Placeholder="Select a customer" Query="@Query">
    <SfDataManager
        Url="https://services.odata.org/V4/Northwind/Northwind.svc/Orders"
        Adaptor="Syncfusion.Blazor.Adaptors.ODataV4Adaptor"
        CrossDomain=true></SfDataManager>
    <DropDownListFieldSettings Text="CustomerID" Value="OrderID"></DropDownListFieldSettings>
</SfDropDownList>
@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.



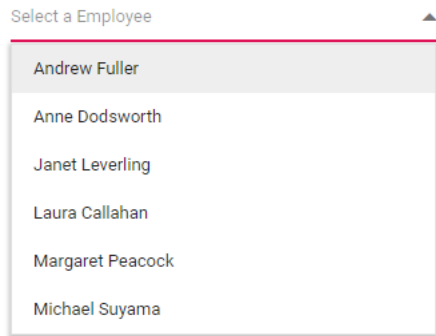
#### Web API Adaptor

Use the **WebApiAdaptor** to bind DropDownList with Web API created using OData.

#### ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" TItem="EmployeeData" Placeholder="Select a
Employee" Query="@Query">
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Employees" Adaptor="Syncfusion.Blazor.Adaptors.WebApiAdaptor"
CrossDomain=true></SfDataManager>
<DropDownListFieldSettings Text="FirstName"
Value="EmployeeID"></DropDownListFieldSettings>
</SfDropDownList>
@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 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 DropDownList component.

For implementing custom data binding in DropDownList, 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 DropDownList 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

```
<SfDropDownList TValue="string" TItem="Orders">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor) "
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<DropDownListFieldSettings Value="CustomerID"></DropDownListFieldSettings>
</SfDropDownList>
@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 DropDownList 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

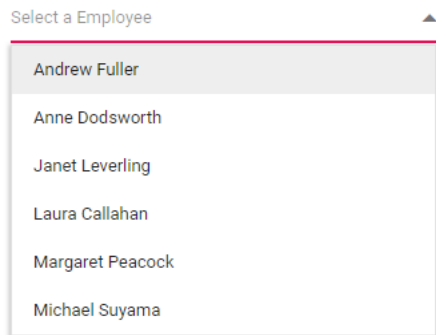
```
<SfDropDownList TValue="string" TItem="EmployeeData" Placeholder="Select a
Employee" Query="@Query">
```

```

<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Employees" Offline=true
Adaptor="Syncfusion.Blazor.Adaptors.WebApiAdaptor"
CrossDomain=true></SfDataManager>
<DropDownListFieldSettings Text="FirstName"
Value="EmployeeID"></DropDownListFieldSettings>
</SfDropDownList>
@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.



### Enum data binding

You can bind enum data to DropDownList component. The following code helps you get a string value from the enumeration data.

### ASPX-CS

```

@using Syncfusion.Blazor.DropDowns;
<SfDropDownList TValue="Values" TItem="string" Placeholder="e.g. Australia"
DataSource="@EnumValues" @bind-Value="@ddlVal">
</SfDropDownList>
@code{
public string[] EnumValues = Enum.GetNames(typeof(Values));
public Values ddlVal { get; set; } = Values.Canada;
public enum Values
{
Australia,
Bermuda,
Canada,
Denmark,
India,
US
}
}

```

The output will shown as follows,



#### ValueTuple data binding

You can bind [ValueTuple](#) data to the DropDownList component. The following code helps you to get a string value from the enumeration data by using [ValueTuple](#).

#### CSHARP

```
@using Syncfusion.Blazor.DropDowns;
<SfDropDownList TItem="(DayOfWeek, string)" Width="250px" TValue="DayOfWeek"
DataSource="@ (Enum.GetValues<DayOfWeek>().Select(e => (e, e.ToString()))) ">
<DropDownListFieldSettings Value="Item1" Text="Item2" />
</SfDropDownList>
```

The output will shown as follows,



#### Binding ExpandoObject

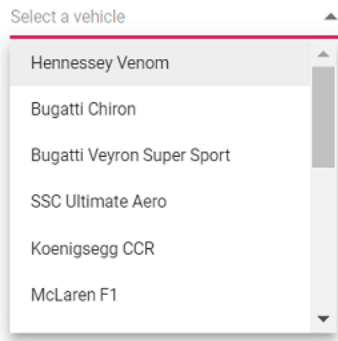
You can bind [ExpandoObject](#) data to the DropDownList component. The following example [ExpandoObject](#) is bound to the collection of vehicles data.

#### CSHARP

```
@using Syncfusion.Blazor.DropDowns
@using System.Dynamic
<SfDropDownList TItem="ExpandoObject" TValue="string" PopupHeight="230px"
Placeholder="Select a vehicle" DataSource="@VehicleData" >
<DropDownListFieldSettings Text="Text"
Value="ID"></DropDownListFieldSettings>
</SfDropDownList>
@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 DropDownList component. The following example `DynamicObject` is bound to the collection of customers data.

#### CSHARP

```
@using Syncfusion.Blazor.DropDowns
@using System.Dynamic
<SfDropDownList TValue="string" Titem="DynamicDictionary"
Placeholder="Select a name" DataSource="@Orders">
<DropDownListFieldSettings Text="CustomerName"
Value="CustomerName"></DropDownListFieldSettings>
</SfDropDownList>
@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);
}
}
```

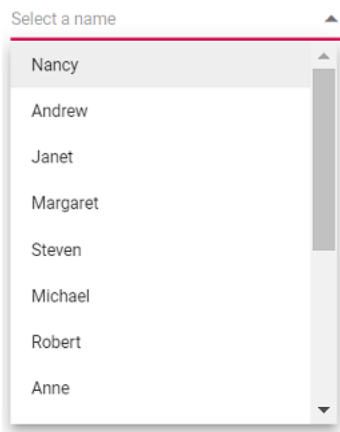


```

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 DropDownList component. The following example **Observable Data** is bound to a collection of colors data.

#### CSHARP

```

@using Syncfusion.Blazor.DropDowns
@using System.Collections.ObjectModel;
<SfDropDownList TValue="string" TItem="Colors" PopupHeight="230px"
Placeholder="Select a color" DataSource="@ColorsData">
<DropDownListFieldSettings Text="Color"
Value="Code"></DropDownListFieldSettings>
</SfDropDownList>
@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" },
    }
}

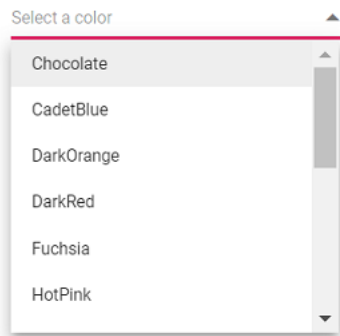
```

```

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 DropDownList 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 DropDownList 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 DropDownList component using Web API adaptor

Now you can configure the DropDownList 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
<SfDropDownList TValue="string" TItem="Order" Placeholder="Select a
Country">
<SfDataManager Url="api/Default" Adaptor="Adaptors.WebApiAdaptor"
CrossDomain="true"></SfDataManager>
<DropDownListFieldSettings Text="ShipCountry"
Value="OrderID"></DropDownListFieldSettings>
</SfDropDownList>
@code{
public class Order
{
    public int? OrderID { get; set; }
    public string ShipCountry { get; set; }
}

```

```
}
```

## Grouping in Blazor DropDown List Component

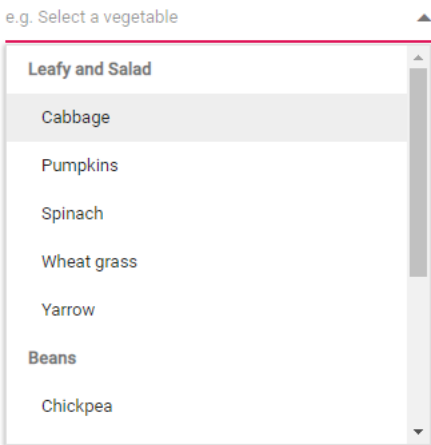
The DropDownList supports wrapping nested elements into a group based on different categories. The category of each list item can be mapped through the [GroupBy](#) field in the data table. The group header is displayed both as inline and fixed headers. The fixed group header content is updated dynamically 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.DropDowns
<SfDropDownList TValue="string" TItem="Vegetables" Placeholder="e.g. Select
a vegetable" DataSource="@LocalData">
<DropDownListFieldSettings GroupBy="Category"
Value="Vegetable"></DropDownListFieldSettings>
</SfDropDownList>
@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 DropDown List Component

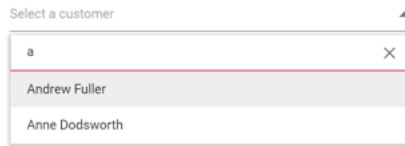
The DropDownList has built-in support to filter data items when [AllowFiltering](#) is enabled. The filter operation starts as soon as you start typing characters in the search box.

#### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" TItem="Countries" Placeholder="Select a
country" AllowFiltering="true" DataSource="@Country">
<DropDownListFieldSettings Text="Name"
Value="Code"></DropDownListFieldSettings>
</SfDropDownList>
@code{
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
private 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.



### Custom Filtering

The DropDownList 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
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" @ref="ddlObj" TItem="Countries"
Placeholder="e.g. Australia" DataSource="@Country" AllowFiltering="true">
  <DropDownListFieldSettings Text="Name"
  Value="Code"></DropDownListFieldSettings>
  <DropDownListEvents TValue="string" TItem="Countries"
  Filtering="OnFilter"></DropDownListEvents>
</SfDropDownList>
@code {
SfDropDownList<string, Countries> ddlObj { 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 ddlObj.FilterAsync(Country1, query);  
}  
}
```

## Templates in Blazor DropDown List Component

The DropDownList 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 DropDownList can be customized with the help of [ItemTemplate](#) property.

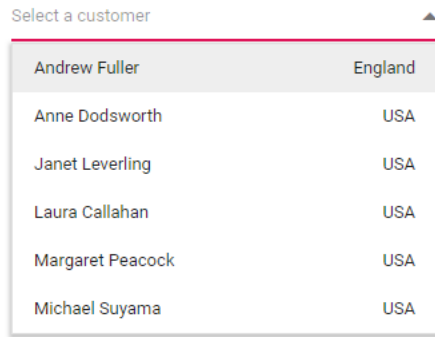
In the following sample, each list item is split into two columns to display relevant data.

### ASPX-CS

```
@using Syncfusion.Blazor.Data  
@using Syncfusion.Blazor.DropDowns  
<SfDropDownList TValue="string" TItem="EmployeeData" Placeholder="Select a  
customer" Query="@Query">  
  <DropDownListTemplates TItem="EmployeeData">  
    <ItemTemplate>  
      <span><span class='name'>@((context as EmployeeData).FirstName)</span><span  
class='country'>@((context as EmployeeData).Country)</span></span>  
    </ItemTemplate>  
  </DropDownListTemplates>  
  <SfDataManager Url="https://ej2services.syncfusion.com/production/web-  
services/api/Employees" Adaptor="Syncfusion.Blazor.Adaptors.WebApiAdaptor"  
CrossDomain="true"></SfDataManager>  
  <DropDownListFieldSettings Text="FirstName"  
Value="Country"></DropDownListFieldSettings>  
</SfDropDownList>  
@code {  
  public class EmployeeData  
  {  
    public string FirstName { get; set; }  
    public string Country { get; set; }  
  }  
  public EmployeeData Data = new EmployeeData();  
  public Query Query = new Query();  
}  
<style>  
  .country {  
    right: 15px;  
    position: absolute;  
  }  
</style>
```

The output will be as follows.





### Value template

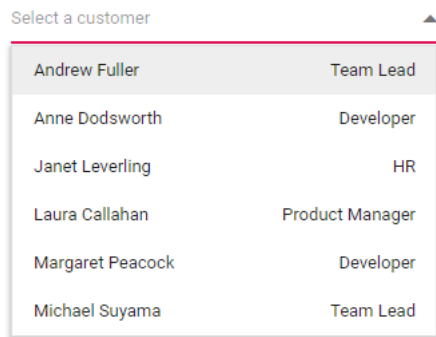
The currently selected value that is displayed by default on the DropDownList input element can be customized using the [ValueTemplate](#) property.

In the following sample, the selected value is displayed as a combined text of both **FirstName** and **Designation** in the DropDownList input, which is separated by a hyphen.

### ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" TItem="EmployeeData" Placeholder="Select a
customer" Query="@Query">
  <DropDownListTemplates TItem="EmployeeData">
    <ItemTemplate>
      <span><span class='name'>@((context as EmployeeData).FirstName)</span><span
class='destination'>@((context as EmployeeData).Designation)</span></span>
    </ItemTemplate>
    <ValueTemplate>
      <span>@((context as EmployeeData).FirstName) - @((context as
EmployeeData).Designation)</span>
    </ValueTemplate>
  </DropDownListTemplates>
  <SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Employees" Adaptor="Syncfusion.Blazor.Adaptors.WebApiAdaptor"
CrossDomain="true"></SfDataManager>
  <DropDownListFieldSettings Text="FirstName"
Value="Designation"></DropDownListFieldSettings>
</SfDropDownList>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
public string Designation { get; set; }
}
public Query Query = new Query();
}
<style>
.destination {
right: 15px;
position: absolute;
}
</style>
```

The output will be as follows.



### Group template

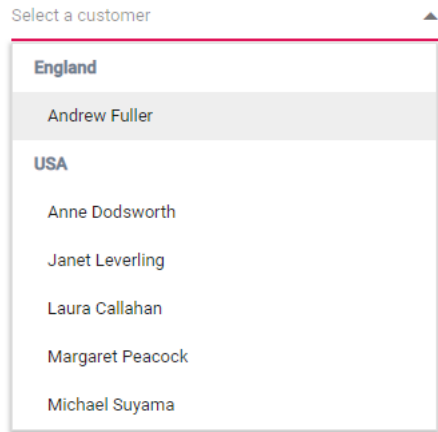
The group header title under which appropriate sub-items are categorized can also be customized with the help of [GroupTemplate](#) property. This template is common for both inline and floating group header template.

In the following sample, employees are grouped according to their city.

### ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" TItem="EmployeeData" Placeholder="Select a
customer" Query="@Query">
<DropDownListTemplates TItem="EmployeeData">
<GroupTemplate>
<span class="group">@(context.Text)</span>
</GroupTemplate>
</DropDownListTemplates>
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Employees" Adaptor="Syncfusion.Blazor.Adaptors.WebApiAdaptor"
CrossDomain=true></SfDataManager>
<DropDownListFieldSettings Value="FirstName"
GroupBy="Country"></DropDownListFieldSettings>
</SfDropDownList>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
public string Country { get; set; }
}
public Query Query = new Query();
}
<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 DropDownList, and any custom element can be placed as a header element using the [HeaderTemplate](#) property.

In the following sample, the list items and its headers are designed and displayed as two columns similar to multiple columns of the grid.

### ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" TItem="EmployeeData" Placeholder="Select a
customer" Query="@Query">
<DropDownListTemplates 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>
</DropDownListTemplates>
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Employees" Adaptor="Syncfusion.Blazor.Adaptors.WebApiAdaptor"
CrossDomain=true></SfDataManager>
<DropDownListFieldSettings Value="Country"
Text="FirstName"></DropDownListFieldSettings>
</SfDropDownList>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
public string Country { get; set; }
}
public Query Query = new Query();
}
<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;
}
.head .city {
text-indent: 10px;
}
</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 DropDownList has options to show a footer element at the bottom of the list items in the popup list. Here, you can place any custom element as a footer element using the [FooterTemplate](#) property.

In the following sample, footer element displays the total number of list items present in the DropDownList.

### ASPX-CS

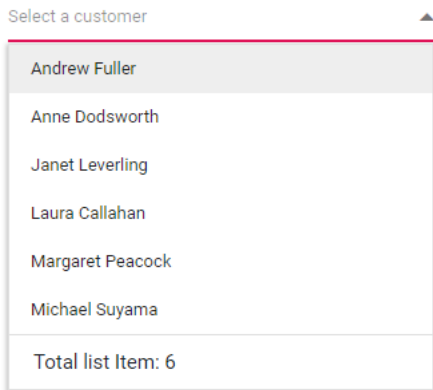
```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" TItem="EmployeeData" Query="@Query"
Placeholder="Select a customer">
<DropDownListTemplates TItem="EmployeeData">
<FooterTemplate>
<span class='footer'>Total list Item: 6 </span>
</FooterTemplate>
</DropDownListTemplates>
```

```

<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Employees" Adaptor="Syncfusion.Blazor.Adaptors.WebApiAdaptor"
CrossDomain=true></SfDataManager>
<DropDownListFieldSettings Value="Country"
Text="FirstName"></DropDownListFieldSettings>
</SfDropDownList>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
}
public Query Query = new Query();
}
<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 DropDownList is provided with support to custom design the popup list content when no data is found and no matches found on search with the help of [NoRecordsTemplate](#) property.

In the following sample, popup list content displays the notification of no data available.

### ASPX-CS

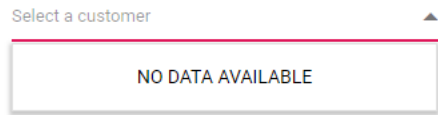
```

@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" TItem="Countries" Placeholder="Select a
customer" DataSource="@Country">
<DropDownListTemplates TItem="Countries">
<NoRecordsTemplate>
<span class='norecord'> NO DATA AVAILABLE</span>
</NoRecordsTemplate>
</DropDownListTemplates>
</SfDropDownList>

```

```
@code {
public class Countries { }
List<Countries> Country = new List<Countries> { };
}
```

The output will be as follows.



### Action failure template

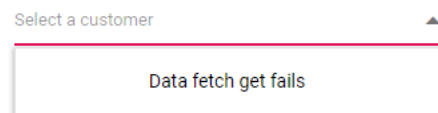
There is also an option to custom design the popup list content when the data fetch request fails at the remote server. This can be achieved using the [ActionFailureTemplate](#) property.

In the following sample, when the data fetch request fails, the DropDownList displays the notification.

### ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" TItem="EmployeeData" Placeholder="Select a
customer" Query="@Query">
  <DropDownListTemplates TItem="EmployeeData">
    <ActionFailureTemplate>
      <span class='norecord'>Data fetch get fails </span>
    </ActionFailureTemplate>
  </DropDownListTemplates>
</SfDropDownList>
<SfDataManager
  Url="https://services.odata.org/V4/Northwind/Northwind.svcs/Employees"
  Adaptor="Syncfusion.Blazor.Adaptors.ODataV4Adaptor"
  CrossDomain=true></SfDataManager>
<DropDownListFieldSettings Value="Country"
  Text="FirstName"></DropDownListFieldSettings>
</SfDropDownList>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
}
public Query Query = new Query();
}
```

The output will be as follows,



### Style and appearance in Blazor DropDown List Component

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

### Customizing the appearance of wrapper element

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

#### CSS

```
.e-ddl.e-input-group.e-control-wrapper .e-input {  
  font-size: 20px;  
  font-family: emoji;  
  color: #ab3243;  
  background: #32a5ab;  
}
```

### Customizing the dropdown icon's color

Use the following CSS to customize the dropdown icon's color.

#### CSS

```
.e-ddl .e-input-group-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 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 DropDown List Component

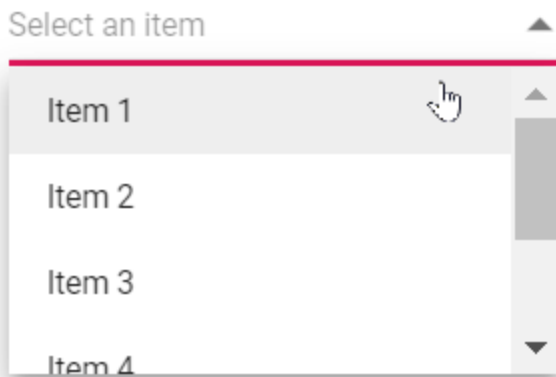
The DropDownList has been provided virtualization to improve the UI performance for a large amount of data when [EnableVirtualization](#) is true. This feature doesn't render out the entire data source on initial component rendering. It loads the N number of items in the popup on initial rendering and 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
<SfDropDownList TValue="string" TItem="Record" Placeholder="Select an item"
DataSource="@Records" Query="@LocalDataQuery" PopupHeight="130px"
EnableVirtualization="true">
<DropDownListFieldSettings Text="Text"
Value="ID"></DropDownListFieldSettings>
</SfDropDownList>
@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,



## Localization in Blazor DropDown List 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

### 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 DropDownList 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 DropDownList 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)
------	----------------------------

---	---
-----	-----

DropDownList_ActionFailureTemplate	Die Anfrage ist fehlgeschlagen
------------------------------------	--------------------------------

DropDownList_NoRecordsTemplate	Keine Aufzeichnungen gefunden
--------------------------------	-------------------------------

- Finally, Specify the culture for DropDownList using **locale** property.

### ASPX-CS

```

@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" TItem="Games" Placeholder="Select a game"
Locale="de" AllowFiltering="true" DataSource="@LocalData">
    <DropDownListFieldSettings Value="ID"
    Text="Text"></DropDownListFieldSettings>
</SfDropDownList>
@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" },
    };
}

```

### 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 DropDownList.

Locale key	en-US (default)
------------	-----------------

-----	-----
-------	-------

| NoRecordsTemplate | No records found

| ActionFailureTemplate | The request failed

The following steps explain how to render the DropDownList 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()
                };
            });
            ....
            ....
        }
    }
}
```

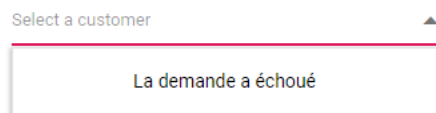
- To download the locale definition of Blazor components, use this [link](#).
- After downloading the **blazor-locale** package, copy the **blazor-locale** folder with required local definition file into **wwwroot** folder.
- By default, the **blazor-locale** package contains the localized text for static 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 DropDownList and no data is loaded. Hence, the [NoRecordsTemplate](#) property displays its text in French culture initially, and if the sample is run offline, the [ActionFailureTemplate](#) property displays its text appropriately.

### ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
@inject HttpClient Http;
<SfDropDownList TValue="string" TItem="EmployeeData" Query="@Query"
Placeholder="Select a customer" AllowFiltering="true" Locale="fr"
PopupHeight="220px">
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Employees" Adaptor="Syncfusion.Blazor.Adaptors.WebApiAdaptor"
CrossDomain=true></SfDataManager>
<DropDownListFieldSettings Value="Country"
Text="FirstName"></DropDownListFieldSettings>
</SfDropDownList>
@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.



### Accessibility in Blazor DropDown List Component

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

#### ARIA attributes

The DropDownList component uses the Listbox role, and each list item has an option role. The following ARIA attributes denote the DropDownList state.

| **Properties** | **Functionalities** |

| --- | --- |

| aria-haspopup | Indicates whether the DropDownList input element has a popup list or not. |

| aria-expanded | Indicates whether the popup list has expanded or not. |

| aria-selected | Indicates the selected option. |

| aria-readonly | Indicates the readonly state of the DropDownList element. |

| aria-disabled | Indicates whether the DropDownList component is in a disabled state or not. |

| aria-activedescendent | This attribute holds the ID of the active list item to focus its descendant child element. |

| aria-owns | This attribute contains the ID of the popup list to indicate popup as a child element. |

### Keyboard interaction

You can use the following key shortcuts to access the DropDownList without interruptions:

| **Keyboard shortcuts** | **Actions** |

| --- | --- |

| Arrow Down | Selects the first item in the DropDownList when no item 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 when it is in open state, the popup list closes. Otherwise, toggles the popup list. |

| Tab | Focuses on the next TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |

| Shift + tab | Focuses on the previous TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |

| Alt + Down | Opens the popup list. |

| Alt + Up | Closes the popup list. |

| Esc(Escape) | Closes the popup list when it is in an open state and the currently selected item remains the same. |

| Home | Selects the first item. |

| End | Selects the last item. |

### Events in Blazor DropDown List Component

This section explains the list of events of the DropDown List component which will be triggered for appropriate DropDown List 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
}
}
```

### ValueChange

**ValueChange** event triggers when the DropDown List value is changed.

### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TItem="GameFields" TValue="string" DataSource="@Games">
<DropDownListEvents TItem="GameFields" TValue="string"
ValueChange="@ValueChangeHandler" ></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 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
<SfDropDownList TItem="GameFields" TValue="string" DataSource="@Games">
  <DropDownListEvents TItem="GameFields" TValue="string"
    Closed="@CloseHandler" ></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 CloseHandler(ClosedEventArgs args)
{
// Here you can customize your code
}
}
```

### Created

**Created** event triggers when the component is created.

#### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TItem="GameFields" TValue="string" DataSource="@Games">
  <DropDownListEvents TItem="GameFields" TValue="string"
    Created="@CreatedHandler" ></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 CreatedHandler(Object args)
{
    // Here you can customize your code
}
}
```

### Destroyed

**Destroyed** event triggers when the component is destroyed.

#### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TItem="GameFields" TValue="string" DataSource="@Games">
    <DropDownListEvents TItem="GameFields" TValue="string"
        Destroyed="@DestroyedHandler" ></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 DestroyedHandler(Object args)
    {
        // Here you can customize your code
    }
}
```

### Focus

**Focus** event triggers when the input gets focus.

#### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TItem="GameFields" TValue="string" DataSource="@Games">
    <DropDownListEvents TItem="GameFields" TValue="string" Focus="@FocusHandler"
    ></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 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
<SfDropDownList TItem="GameFields" TValue="string" DataSource="@Games">
<DropDownListEvents TItem="GameFields" TValue="string"
OnOpen="@OnOpenHandler" ></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 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
<SfDropDownList TItem="GameFields" TValue="string" DataSource="@Games">
<DropDownListEvents TItem="GameFields" TValue="string"
OnClose="@OnCloseHandler" ></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 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
<SfDropDownList TItem="GameFields" TValue="string" DataSource="@Games">
<DropDownListEvents TItem="GameFields" TValue="string"
DataBound="@DataBoundHandler" ></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 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
<SfDropDownList TItem="GameFields" TValue="string" AllowFiltering="true"
DataSource="@Games">
```

```

<DropDownListEvents TItem="GameFields" TValue="string"
Filtering="@Filteringhandler" ></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 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
<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
}
}

```

### 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
<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"
OnActionFailure="@OnActionFailurehandler"></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 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
<SfDropDownList TItem="GameFields" TValue="string" DataSource="@Games">
<DropDownListEvents TItem="GameFields" TValue="string"
OnValueSelect="@OnValueSelecthandler" ></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 OnValueSelecthandler(SelectEventArgs<GameFields> args)
{
    // Here you can customize your code
}
}

```

### Opened

**Opened** event triggers when the popup opens.

### ASPX-CS

```

@using Syncfusion.Blazor.DropDowns
<SfDropDownList TItem="GameFields" TValue="string" DataSource="@Games">
    <DropDownListEvents TItem="GameFields" TValue="string"
        Opened="@Openedhandler" ></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 Openedhandler(PopupEventArgs args)
    {
        // Here you can customize your code
    }
}

```

DropDown List 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 DropDown List 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

```

@using Syncfusion.Blazor.DropDowns;
@using Syncfusion.Blazor.Popups;

```

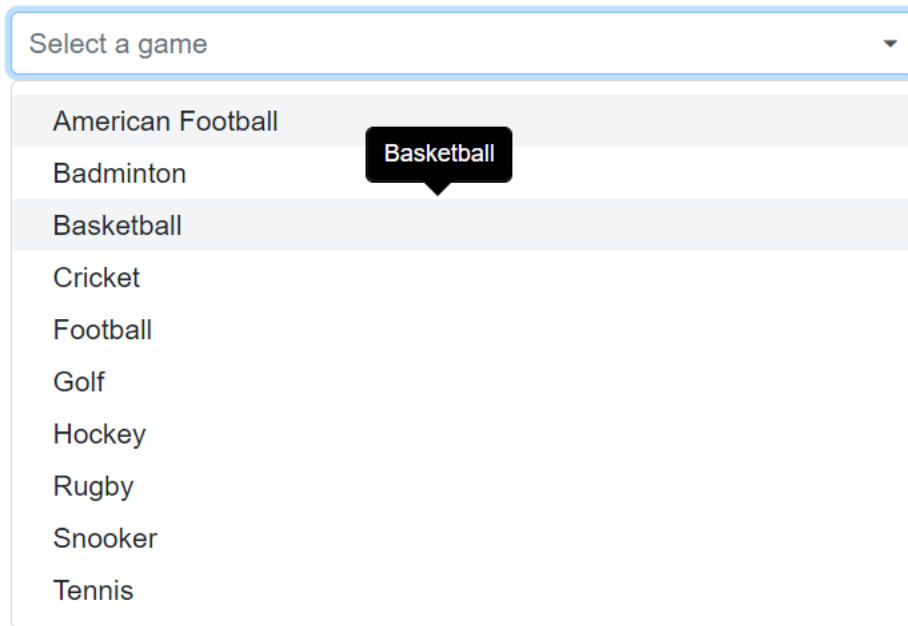


```

<SfTooltip @ref="TooltipObj" ID="Tooltip" Target=".e-list-item
.name[title]">
</SfTooltip>
<SfDropDownList TItem="GameFields" TValue="string" Placeholder="Select a
game" DataSource="@Games">
<DropDownListFieldSettings Text="Text"
Value="ID"></DropDownListFieldSettings>
<DropDownListTemplates TItem="GameFields">
<ItemTemplate>
<div class="name" title="@((context as GameFields).Text)"> @((context as
GameFields).Text) </div>
</ItemTemplate>
</DropDownListTemplates>
<DropDownListEvents TValue="string" TItem="GameFields" Opened="OnOpen"
OnClose="OnClose"></DropDownListEvents>
</SfDropDownList>
@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.



## FileManager

<!-- markdownlint-disable MD024 -->

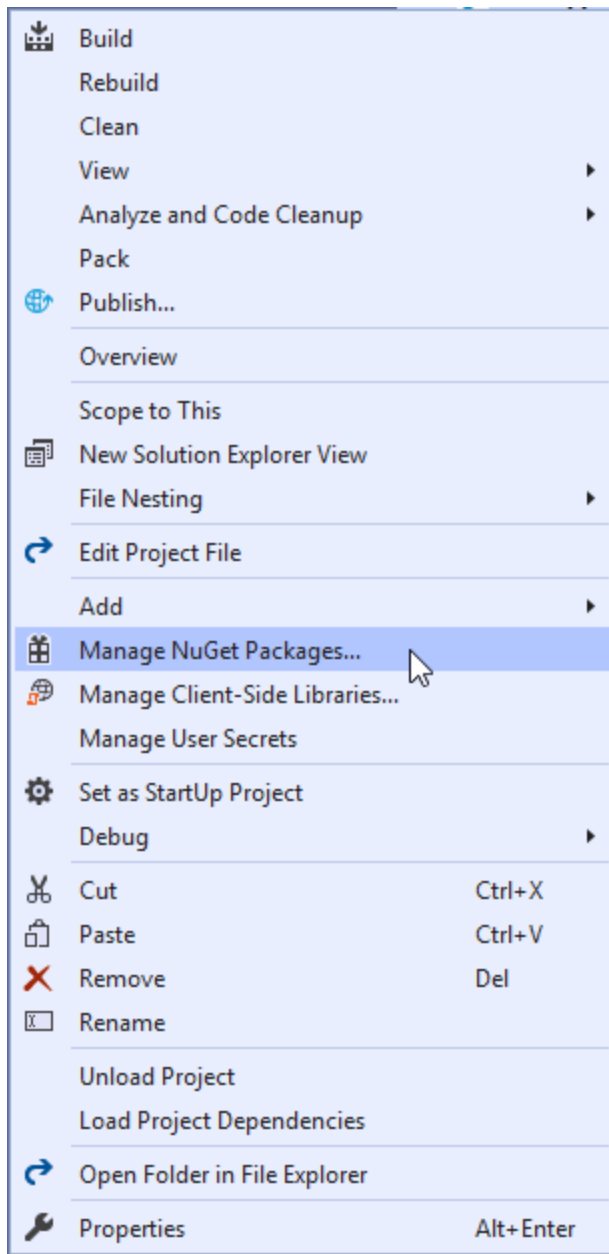
### Getting Started with Blazor FileManager Component

This section briefly explains how to include a **File Manager** 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.

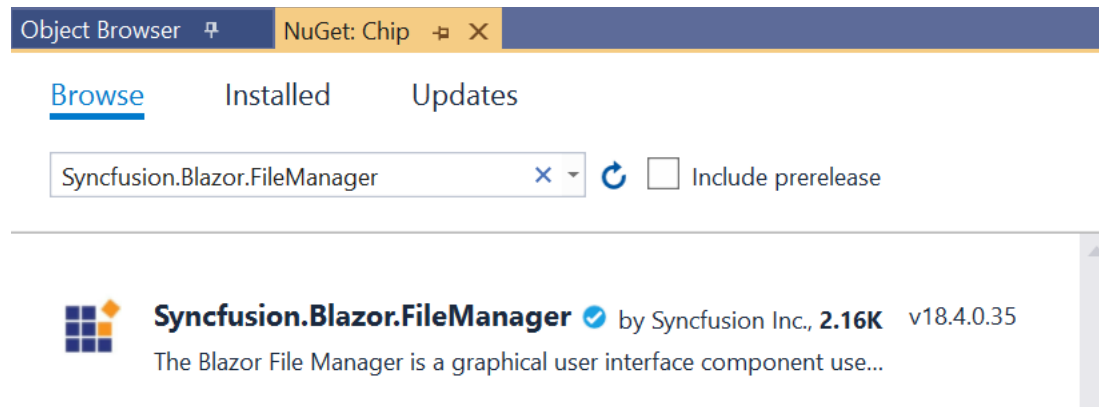
Importing Syncfusion Blazor component in the application

*Using Syncfusion.Blazor NuGet Package [New standard]*

1. Install **Syncfusion.Blazor.FileManager** 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.FileManager keyword in the Browse tab and install Syncfusion.Blazor.FileManager NuGet package in the application.



3. Once the installation process is completed, the Syncfusion Blazor FileManager 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.

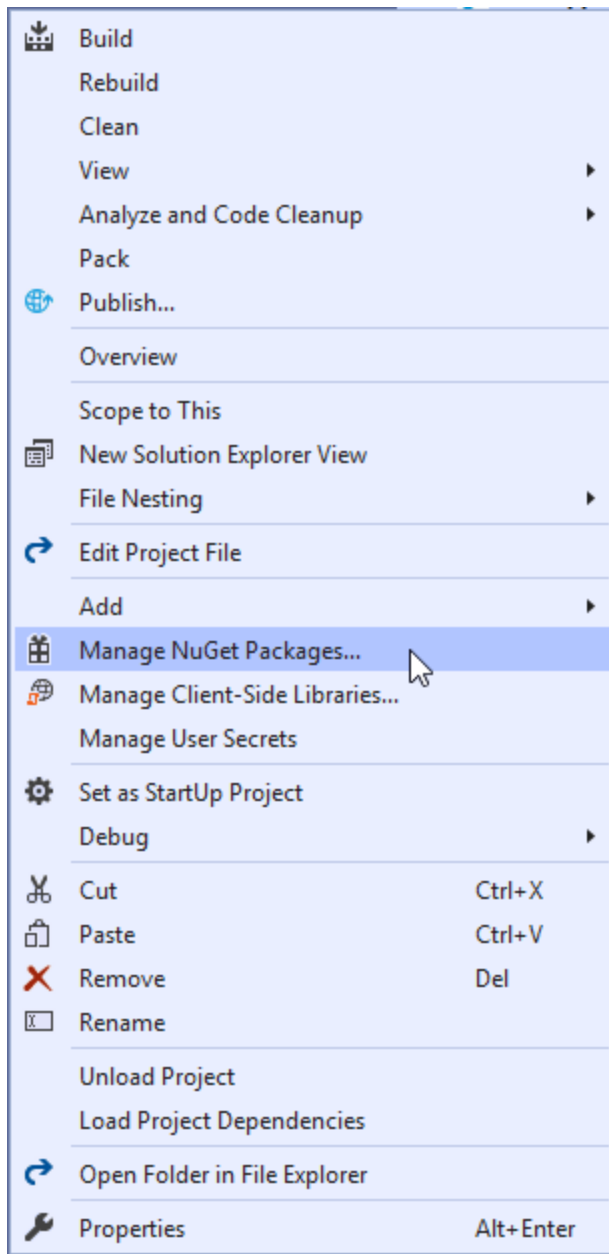
#### HTML

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

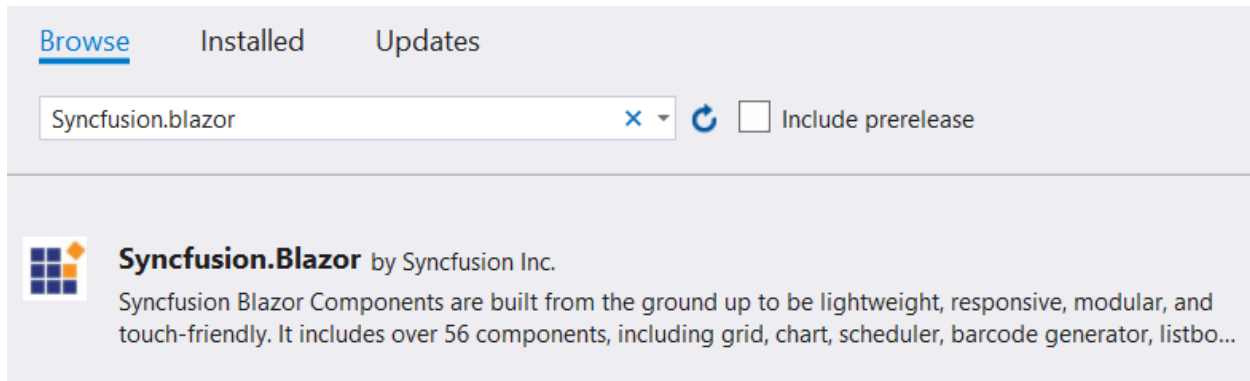
**Warning:** If you prefer the above new standard (individual NuGet packages), then skip this section. Using both old and new standards in the same application will throw ambiguous compilation errors.

#### *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.

#### CSHARP

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

Adding File Manager component namespace to the application

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

#### ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.FileManager
```

Initialize the File Manager component

The File Manager can be rendered with local service, which sends ajax request. The Ajax request will be sent to the server, which then processes the request and sends back the response.

To render the File Manager with local service, refer to the following code snippet.

#### ASPX-CS

```
@using Syncfusion.Blazor.FileManager
<SfFileManager TValue="FileManagerDirectoryContent">
<FileManagerAjaxSettings Url="/api/SampleData/FileOperations">
```

```
</FileManagerAjaxSettings>
</SfFileManager>
```

### Initialize the service in controller

File Manager supports the basic file actions like Read, Delete, Copy, Move, Get Details, Search, Rename, and Create New Folder.

To initialize a local service, create a new folder name with **Controllers** inside the server part of the project. Then, create a new file **SampleDataController** with extension **.cs** inside the **Controllers** folder and add the following code in that file.

[Controllers/SampleDataController.cs]

### CSHARP

```
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Http.Features;
//File Manager's base functions are available in the below namespace
using Syncfusion.EJ2.FileManager.Base;
//File Manager's operations are available in the below namespace
using Syncfusion.EJ2.FileManager.PhysicalFileProvider;
using Newtonsoft.Json;
using System.Linq;
using System.Threading.Tasks;
namespace filemanager.Server.Controllers
{
    [Route("api/[controller]")]
    public class SampleDataController : Controller
    {
        public PhysicalFileProvider operation;
        public string basePath;
        string root = "wwwroot\\Files";
        [Obsolete]
        public SampleDataController(IHostingEnvironment hostingEnvironment)
        {
            this.basePath = hostingEnvironment.ContentRootPath;
            this.operation = new PhysicalFileProvider();
            this.operation.RootFolder(this.basePath + "\\\" + this.root); // It denotes
            // in which files and folders are available.
        }
        // Processing the File Manager operations
        [Route("FileOperations")]
        public object FileOperations([FromBody] FileManagerDirectoryContent args)
        {
            switch (args.Action)
            {
                // Add your custom action here
                case "read":
                    // Path - Current path; ShowHiddenItems - Boolean value to show/hide hidden
                    // items
                    return this.operation.ToCamelCase(this.operation.GetFiles(args.Path,
                    args.ShowHiddenItems));
            }
        }
    }
}
```



```

case "delete":
    // Path - Current path where of the folder to be deleted; Names - Name of
    // the files to be deleted
    return this.operation.ToCamelCase(this.operation.Delete(args.Path,
    args.Names));
case "copy":
    // Path - Path from where the file was copied; TargetPath - Path where the
    // file/folder is to be copied; RenameFiles - Files with same name in the
    // copied location that is confirmed for renaming; TargetData - Data of the
    // copied file
    return this.operation.ToCamelCase(this.operation.Copy(args.Path,
    args.TargetPath, args.Names, args.RenameFiles, args.TargetData));
case "move":
    // Path - Path from where the file was cut; TargetPath - Path where the
    // file/folder is to be moved; RenameFiles - Files with same name in the moved
    // location that is confirmed for renaming; TargetData - Data of the moved file
    return this.operation.ToCamelCase(this.operation.Move(args.Path,
    args.TargetPath, args.Names, args.RenameFiles, args.TargetData));
case "details":
    // Path - Current path where details of file/folder is requested; Name -
    // Names of the requested folders
    return this.operation.ToCamelCase(this.operation.Details(args.Path,
    args.Names));
case "create":
    // Path - Current path where the folder is to be created; Name - Name of the
    // new folder
    return this.operation.ToCamelCase(this.operation.Create(args.Path,
    args.Name));
case "search":
    // Path - Current path where the search is performed; SearchString - String
    // typed in the searchbox; CaseSensitive - Boolean value which specifies
    // whether the search must be casesensitive
    return this.operation.ToCamelCase(this.operation.Search(args.Path,
    args.SearchString, args.ShowHiddenItems, args.CaseSensitive));
case "rename":
    // Path - Current path of the renamed file; Name - Old file name; NewName -
    // New file name
    return this.operation.ToCamelCase(this.operation.Rename(args.Path,
    args.Name, args.NewName));
}
return null;
}
}
}

```

To access the above File Operations, you need some model class files that have file operations methods. So, create **Models** folder in server part of the application and download the **PhysicalFileProvider.cs** and **Base** folder from the [this](#) link in the Models folder.

Add your required files and folders under the **wwwroot\Files** directory.

For Server-side application, Add the following code in your **Startup.cs** file.

#### CSHARP

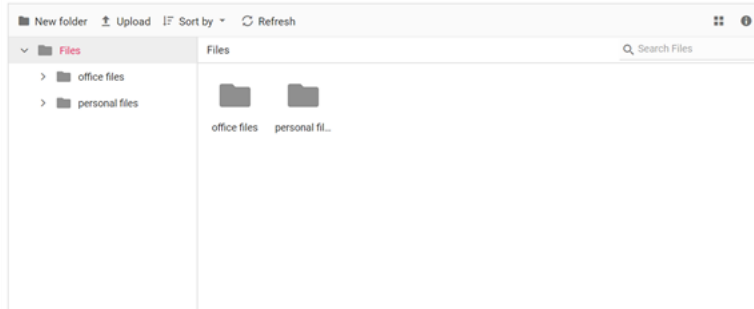
```
app.UseEndpoints(endpoints =>
```

```
{
    ....
    ....
    endpoints.MapControllers();
};
```

### Run the application

After successful compilation of your application, simply press **F5** to run the application.

The FileManager will be in the default web browser with local service as shown in the following image.



### File download support

To perform the download operation, initialize the **DownloadUrl** property in a FileManagerAjaxSettings.

#### CSHARP

```
@using Syncfusion.Blazor.FileManager
<SfFileManager TValue="FileManagerDirectoryContent">
<FileManagerAjaxSettings Url="/api/SampleData/FileOperations"
DownloadUrl="/api/SampleData/Download">
</FileManagerAjaxSettings>
</SfFileManager>
```

Initialize the **Download** FileOperation in Controller part with the following code snippet.

[Controllers/SampleDataController.cs]

#### CSHARP

```
namespace filemanager.Server.Controllers
{
    [Route("api/[controller]")]
    public class SampleDataController : Controller
    {
        // Processing the Download operation
        [Route("Download")]
        public IActionResult Download(string downloadInput)
        {
            //Invoking download operation with the required paramaters
            // path - Current path where the file is downloaded; Names - Files to be
            // downloaded;
            FileManagerDirectoryContent args =
            JsonConvert.DeserializeObject<FileManagerDirectoryContent>(downloadInput);
            return operation.Download(args.Path, args.Names);
        }
    }
}
```

```
}  
}
```

### File upload support

To perform the upload operation, initialize the **UploadUrl** property in a FileManagerAjaxSettings.

#### CSHARP

```
@using Syncfusion.Blazor.FileManager  
<SfFileManager TValue="FileManagerDirectoryContent">  
<FileManagerAjaxSettings Url="/api/SampleData/FileOperations"  
UploadUrl="/api/SampleData/Upload">  
</FileManagerAjaxSettings>  
</SfFileManager>
```

Initialize the **Upload** File Operation in Controller part with the following code snippet.

[Controllers/SampleDataController.cs]

#### CSHARP

```
namespace filemanager.Server.Controllers  
{  
    [Route("api/[controller]")]  
    public class SampleDataController : Controller  
    {  
        // Processing the Upload operation  
        [Route("Upload")]  
        public IActionResult Upload(string path, IList<IFormFile> uploadFiles,  
            string action)  
        {  
            //Invoking upload operation with the required paramaters  
            // path - Current path where the file is to uploaded; uploadFiles - Files to  
            // be uploaded; action - name of the operation(upload)  
            FileManagerResponse uploadResponse;  
            uploadResponse = operation.Upload(path, uploadFiles, action, null);  
            if (uploadResponse.Error != null)  
            {  
                Response.Clear();  
                Response.ContentType = "application/json; charset=utf-8";  
                Response.StatusCode = Convert.ToInt32(uploadResponse.Error.Code);  
                Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =  
                    uploadResponse.Error.Message;  
            }  
            return Content("");  
        }  
    }  
}
```

### Image preview support

To perform image preview support in the File Manager component, initialize the **GetImageUrl** property in a FileManagerAjaxSettings.

#### CSHARP

```
@using Syncfusion.Blazor.FileManager
<SfFileManager TValue="FileManagerDirectoryContent">
<FileManagerAjaxSettings Url="/api/SampleData/FileOperations"
GetImageUrl="/api/SampleData/GetImage">
</FileManagerAjaxSettings>
</SfFileManager>
```

Initialize the **GetImage** File Operation in Controller part with the following code snippet.

[Controllers/SampleDataController.cs]

### CSHARP

```
namespace filemanager.Server.Controllers
{
    [Route("api/[controller]")]
    public class SampleDataController : Controller
    {
        // Processing the GetImage operation
        [Route("GetImage")]
        public IActionResult GetImage(FileManagerDirectoryContent args)
        {
            //Invoking GetImage operation with the required paramaters
            // path - Current path of the image file; Id - Image file id;
            return this.operation.GetImage(args.Path, args.Id, false, null, null);
        }
    }
}
```

The following output will demonstrate the image preview of File Manager.



Sample application

Refer to the following sample link, which is preconfigured with above steps.

[File Manager with local service](#)

See Also

[Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)

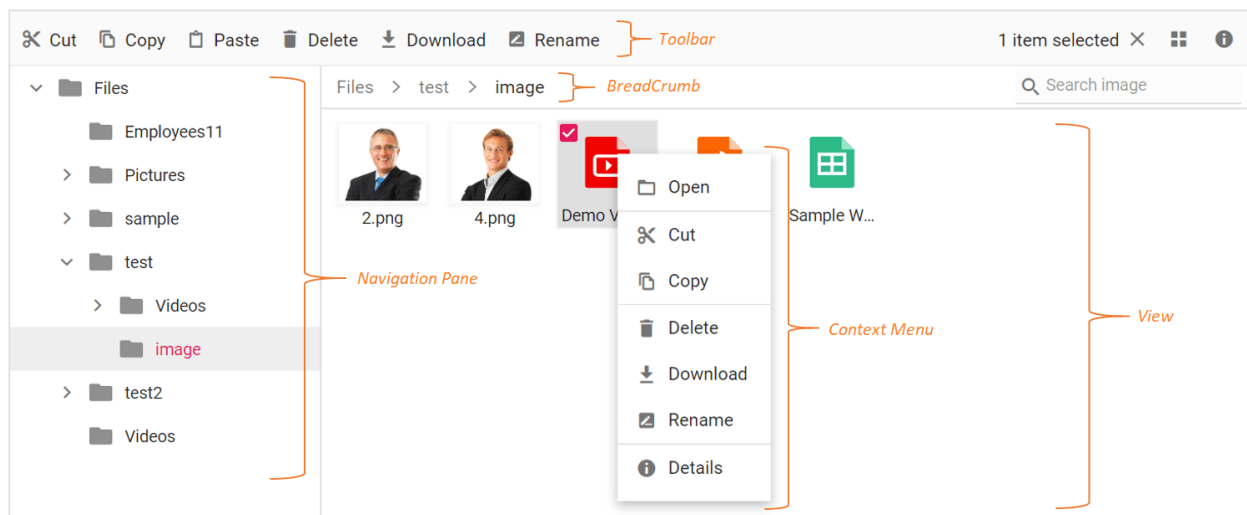
[Getting Started with Syncfusion Blazor for Client-Side in Visual Studio 2019](#)

[Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

## User Interface in Blazor FileManager Component

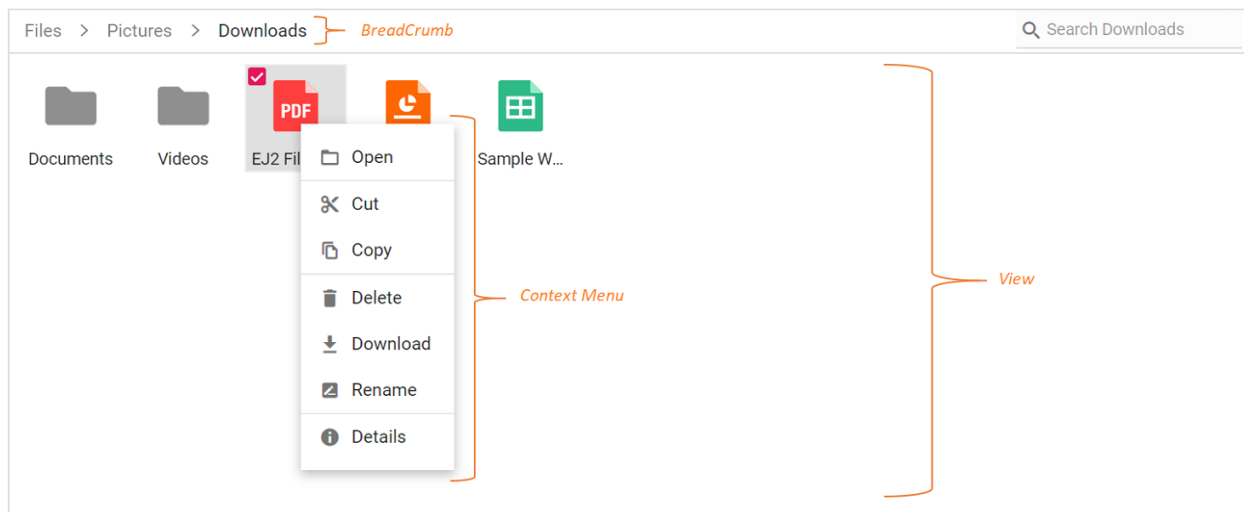
The file manager UI is comprised of several sections like view, toolbar, breadcrumb, context menu, and so on. The UI of the file manager is enhanced with **Details View** for browsing files and folders in a grid, **Navigation Pane** for folder navigation, and **Toolbar** for file operations. The file manager with all feature have the following sections in its UI.

- [Toolbar](#) (For direct access to file operations)
- [Navigation Pane](#) (For easy navigation between folders)
- [Breadcrumb](#) (For parent folder navigations)
- [View](#) (For browsing files and folders using large icon view or details view)
- [Context Menu](#) (For accessing file operations)



The basic file manager is a light weight component with all the basic functions. The basic file manager have the following sections in its UI to browse files and folders and manage them with file operations.

- [Breadcrumb](#) (For parent folder navigations)
- [View](#) (Large Icons view for browsing files and folders)
- [Context Menu](#) (For accessing file operations)

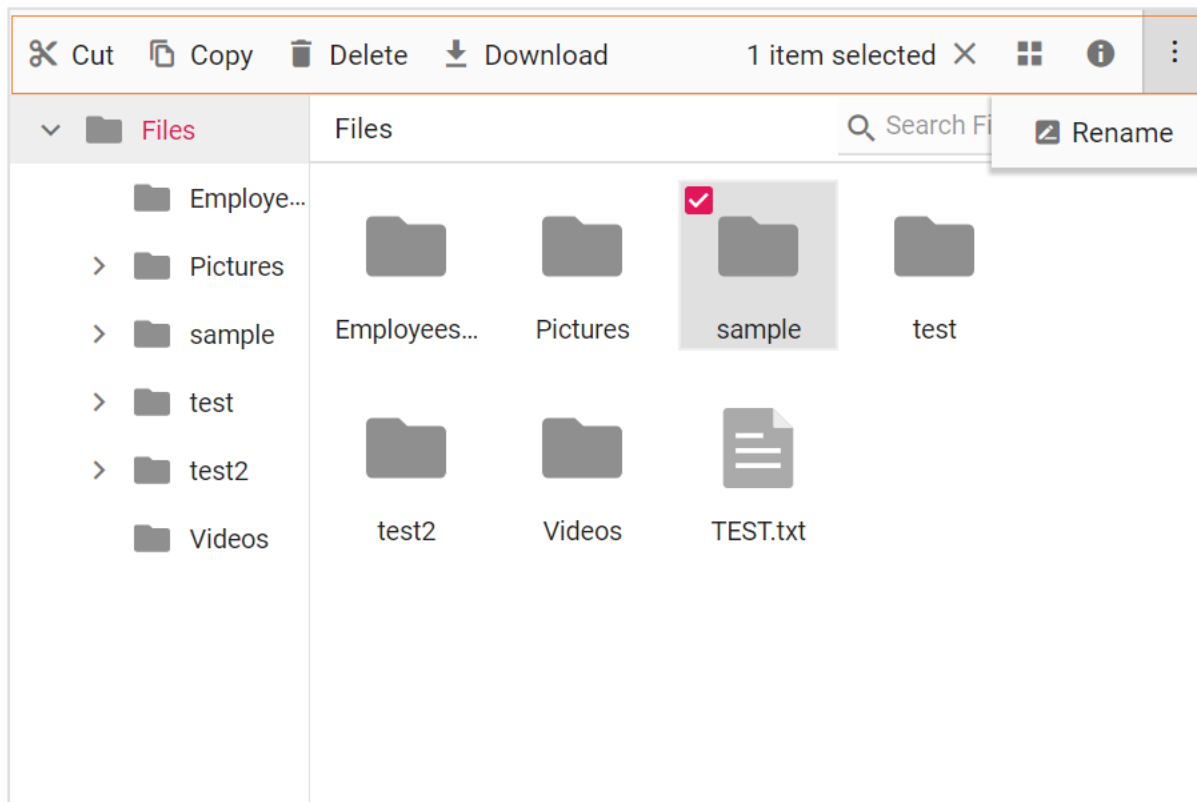


### Toolbar

The **Toolbar** provides easy access to the file operations using different buttons and it is presented at the top of the file manager.

If the toolbar items exceed the size of the toolbar, then the exceeding toolbar size will be moved to toolbar popup with a dropdown button at the end of toolbar.

Refer [Toolbar](#) section in file operations to know more about the buttons present in toolbar.



### Files and folders navigation

The file manager provides navigation between files and folders using the following two options.

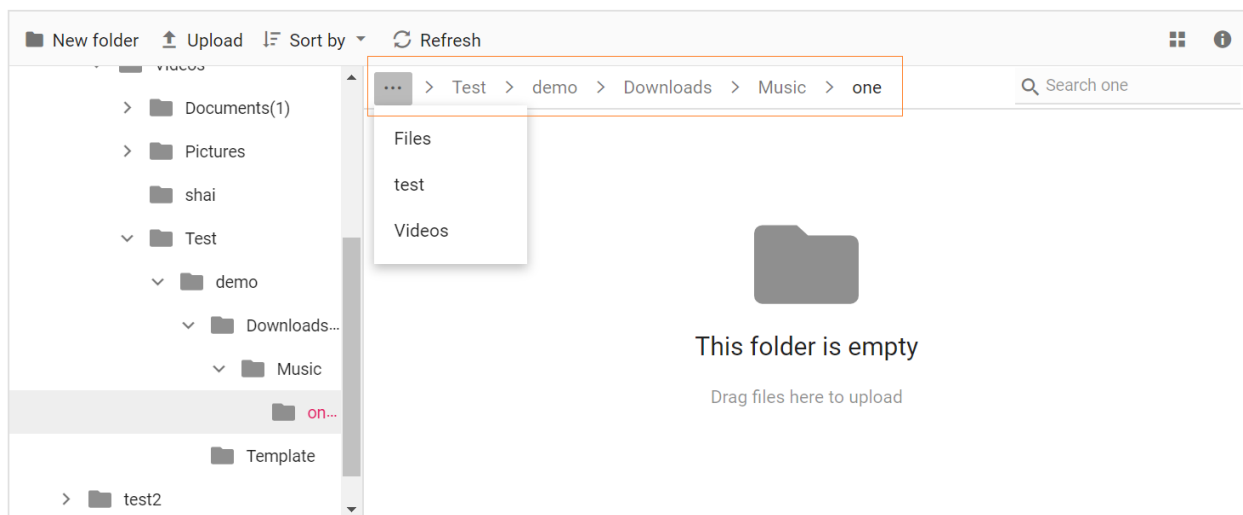
- [Navigation Pane](#)
- [Breadcrumb](#)

### Navigation pane

The navigation pane displays the folder hierarchy of the file system and provides easy navigation to the desired folder. Using `NavigationPaneSettings` minimum and maximum width of the navigation pane can be changed. The navigation pane can be shown or hidden using the `Visible` option in the `NavigationPaneSettings`.

### BreadCrumb

The file manager provides breadcrumb for navigating to the parent folders. The breadcrumb the in file manager is responsible for resizing. Whenever the path length exceeds the breadcrumb length, a dropdown button will be added at the starting of the breadcrumb to hold the parent folders adjacent to root.



### View

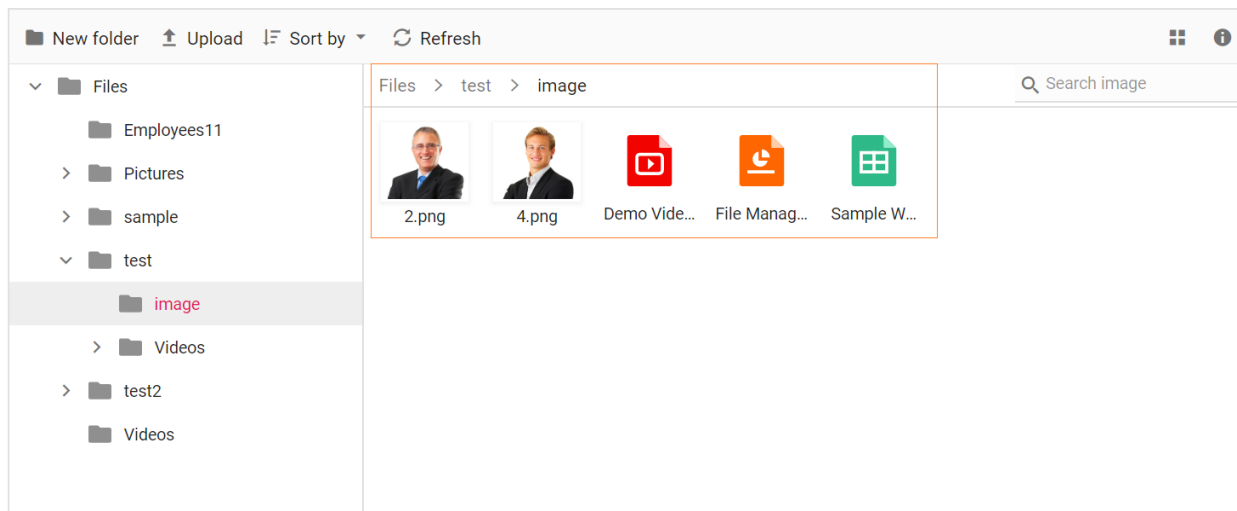
View is the section where the files and folders are displayed for the user to browse. The file manager has two types of views to display the files and folders.

- [Large Icons View](#)
- [Details View](#)

The `Large Icons View` is the default starting view in the file manager. The view can be changed by using the `Toolbar` view button or by using the view menu in `Context Menu`. The `View` API can also be used to change the initial view of the file manager.

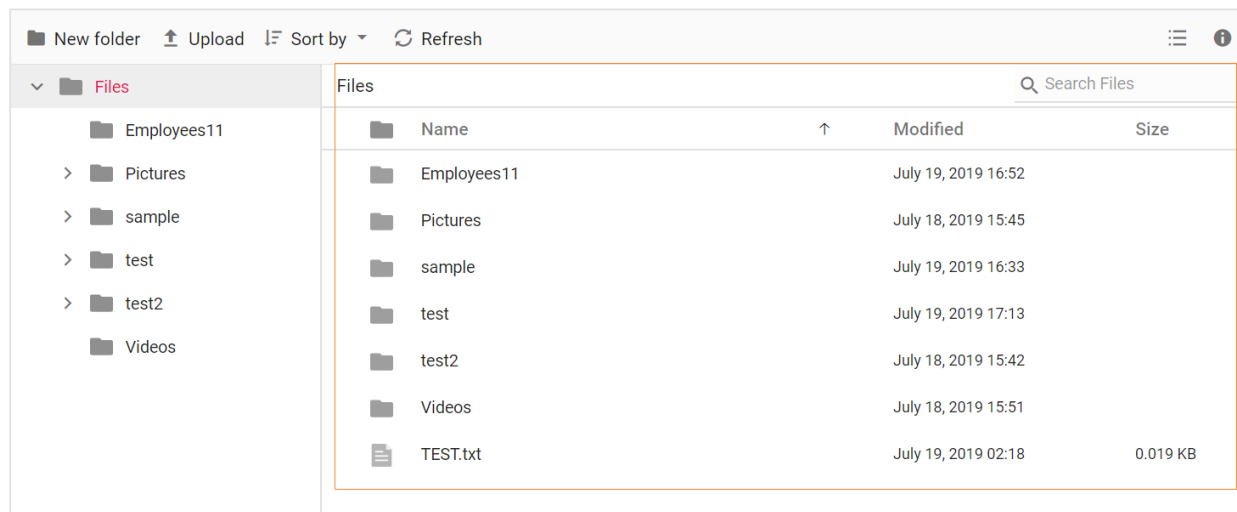
### Large icons view

In the large icons view, the thumbnail icons will be shown in a larger size, which displays the data in a form that best suits their content. For image and video type files, a **preview** will be displayed. Extension thumbnails will be displayed for other type files.



### Details view

In the details view, the files are displayed in a sorted list order. This file list comprises of several columns of information about the files such as **Name**, **Date Modified**, **Type**, and **Size**. Each file has its own small icon representing the file type. Additional columns can be added using `DetailsViewSettings` API. The details view allows you to perform sorting using column header.



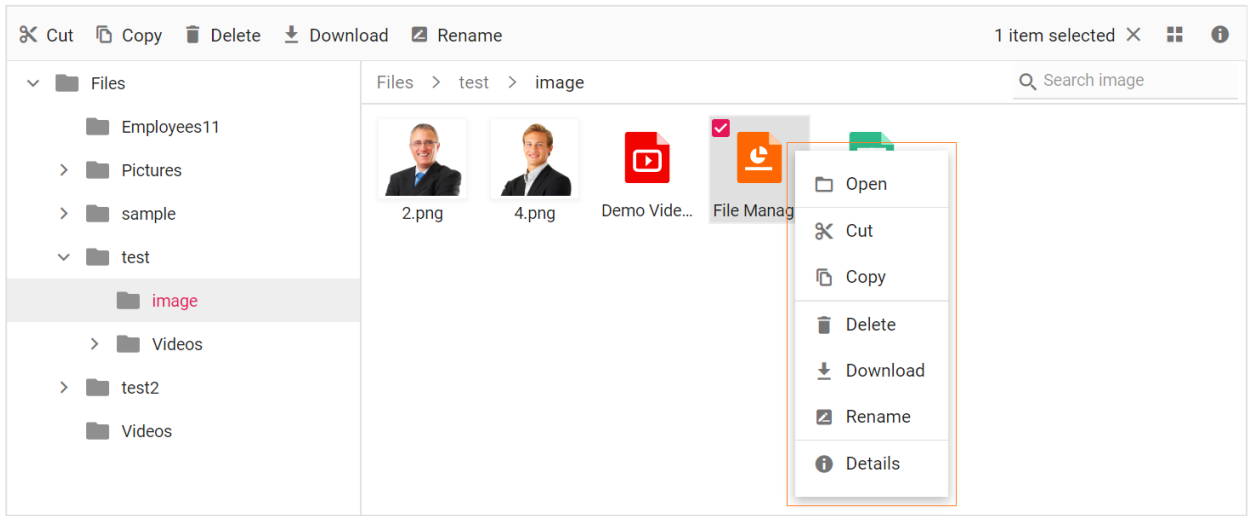
### Context menu

The context menu appears on user interaction such as right-click. The file manager is provided with context menu support to perform list of file operations with the files and folders. Context menu appears with varying menu items based on the targets such as file, folder (including navigation pane folders), and layout (empty area in view).

Context menu can be customized using the `ContextMenuSettings`, `MenuOpened`, and `OnMenuClick` events.

Refer [Context Menu](#) section in file operations to know more about the menu items present in context menu.





File Operations in Blazor FileManager Component

The file manager component is used to browse, manage, and organize the files and folders in a file system through a web application. All basic file operations like creating a new folder, uploading and downloading of files in the file system, and deleting and renaming of existing files and folders are available in the file manager component. Additionally, previewing of image files is also provided in the file manager component.

The following table represents the basic operations available in the file manager and their corresponding functions.

Operation Name	Function
----	----
read	Read the details of files or folders available in the given path from the file system, to display the files for the user to browse the content.
create	Creates a new folder in the current path of the file system.
delete	Removes the file or folder from the file server.
rename	Rename the selected file or folder in the file system.
search	Searches for items matching the search string in the current and child directories.
details	Gets the detail of the selected item(s) from the file server.
copy	Copy the selected file or folder in the file system.
move	Cut the selected file or folder in the file server.
upload	Upload files to the current path or directory in the file system.
download	Downloads the file from the server and the multiple files can be downloaded as ZIP files.

The *CreateFolder*, *Remove*, and *Rename* actions will be reflected in the file manager only after the successful response from the server.

File operation request and response Parameters

The default parameters available in file operation request from the file manager and the corresponding response parameters required by the file manager are listed as follows.

### Read

The following table represents the request parameters of *read* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	read	Name of the file operation.
path	String	-	Relative path from which the data has to be read.
showHiddenItems	Boolean	-	Defines show or hide the hidden items.
data	FileManagerDirectoryContent	-	Details about the current path (directory).

*\*Refer [File request and response contents](#) for the contents of data\*.*

*Example:*

#### C#

```
{
  action: "read",
  path: "/",
  showHiddenItems: false,
  data: []
}
```

The following table represents the response parameters of *read* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Details of files and folders present in given path or directory.
error	ErrorDetails	-	Error Details

*\*Refer [File request and response contents](#) for the contents of cwd, files, and error\*.*

*Example:*

#### C#

```
{
  cwd:
  {
    name: "Download",
    size: 0,
    dateModified: "2019-02-28T03:48:19.8319708+00:00",
    dateCreated: "2019-02-27T17:36:15.812193+00:00",
    hasChild: false,
    isFile: false,
    type: "",
    filterPath: "\\Download\\"
  },
  files: [
    {
      name: "Sample Work Sheet.xlsx",
```

```

size:6172,
dateModified:"2019-02-27T17:23:50.9651206+00:00",
dateCreated:"2019-02-27T17:36:15.8151955+00:00",
hasChild:false,
isFile:true,
type:".xlsx",
filterPath:"\\Download\\"
}
],
error:null,
details:null
}

```

### Create

The following table represents the request parameters of *create* operations.

Parameter	Type	Default	Explanation
action	String	create	Name of the file operation.
path	String	-	Relative path in which the folder has to be created.
name	String	-	Name of the folder to be created.
data	FileManagerDirectoryContent	-	Details about the current path (directory).

*\*Refer [File request and response contents](#) for the contents of data\**

**Example:**

### CSHARP

```

{
  action: "create",
  data: [
    {
      dateCreated: "2019-02-27T17:36:15.6571949+00:00",
      dateModified: "2019-03-12T10:17:31.8505975+00:00",
      filterPath: "\",
      hasChild: true,
      isFile: false,
      name: files,
      nodeId: "fe_tree",
      size: 0,
      type: ""
    }
  ],
  name: "Hello",
  path: "/"
}

```

The following table represents the response parameters of *create* operations.

Parameter	Type	Default	Explanation

|files|FileManagerDirectoryContent[]|-|Details of the created folder|

|error|ErrorDetails|-|Error Details|

*\*Refer [File request and response contents](#) for the contents of files and error\*.*

*Example:*

#### **CSHARP**

```
{
  cwd: null,
  files: [
    {
      dateCreated: "2019-03-15T10:25:05.3596171+00:00",
      dateModified: "2019-03-15T10:25:05.3596171+00:00",
      filterPath: null,
      hasChild: false,
      isFile: false,
      name: "New",
      size: 0,
      type: ""
    }
  ],
  details: null,
  error: null
}
```

#### *Rename*

The following table represents the request parameters of *rename* operations.

Parameter	Type	Default	Explanation
-----------	------	---------	-------------

----	----	----	----
------	------	------	------

action	String	rename	Name of the file operation.
--------	--------	--------	-----------------------------

path	String	-	Relative path in which the item is located.
------	--------	---	---

name	String	-	Current name of the item to be renamed.
------	--------	---	---

newname	String	-	New name for the item.
---------	--------	---	------------------------

data	FileManagerDirectoryContent	-	Details of the item to be renamed.
------	-----------------------------	---	------------------------------------

*\*Refer [File request and response contents](#) for the contents of data\*.*

*Example:*

#### **CSHARP**

```
{
  action: "rename",
  data: [
    {
      dateCreated: "2019-03-20T05:22:34.621Z",
      dateModified: "2019-03-20T08:45:56.000Z",
      filterPath: "\\Pictures\\Nature\\",
      hasChild: false,
      iconClass: "e-fe-image",
    }
  ]
}
```

```

isFile: true,
name: "seaviews.jpg",
size: 95866,
type: ".jpg"
},
newname: "seaview.jpg",
name: "seaviews.jpg",
path: "/Pictures/Nature/"
}

```

The following table represents the response parameters of *rename* operations.

Parameter	Type	Default	Explanation
files	FileManagerDirectoryContent[]	-	Details of the renamed item.
error	ErrorDetails	-	Error Details

*\*Refer [File request and response contents](#) for the contents of files and error\*.*

Example:

#### CSHARP

```

{
  cwd: null,
  files: [
    {
      name: "seaview.jpg",
      size: 95866,
      dateModified: "2019-03-20T08:45:56+00:00",
      dateCreated: "2019-03-20T05:22:34.6214847+00:00",
      hasChild: false,
      isFile: true,
      type: ".jpg",
      filterPath: "\\Pictures\\Nature\\seaview.jpg"
    }
  ],
  error: null,
  details: null
}

```

#### Delete

The following table represents the request parameters of *delete* operations.

Parameter	Type	Default	Explanation
action	String	delete	Name of the file operation.
path	String	-	Relative path where the items to be deleted are located.
names	String[]	-	List of the items to be deleted.
data	FileManagerDirectoryContent	-	Details of the item to be deleted.

*\*Refer [File request and response contents](#) for the contents of data\*.*

*Example:*

#### **CSHARP**

```
{
  action: "delete",
  path: "/Hello/",
  names: ["New"],
  data: []
}
```

The following table represents the response parameters of *delete* operations.

Parameter	Type	Default	Explanation
files	FileManagerDirectoryContent[]	-	Details about the deleted item(s).
error	ErrorDetails	-	Error Details

*\*Refer [File request and response contents](#) for the contents of files and error\*.*

*Example:*

#### **CSHARP**

```
{
  cwd: null,
  details: null,
  error: null,
  files: [
    {
      dateCreated: "2019-03-15T10:13:30.346309+00:00",
      dateModified: "2019-03-15T10:13:30.346309+00:00",
      filterPath: "\\Hello\\folder",
      hasChild: true,
      isFile: false,
      name: "folder",
      size: 0,
      type: ""
    }
  ]
}
```

#### *Details*

The following table represents the request parameters of *details* operations.

Parameter	Type	Default	Explanation
action	String	details	Name of the file operation.
path	String	-	Relative path where the items are located.
names	String[]	-	List of the items to get details.

|data|FileManagerDirectoryContent|-|Details of the selected item.|

*\*Refer [File request and response contents](#) for the contents of data\*.*

*Example:*

#### **CSHARP**

```
{
  action: "details",
  path: "/FileContents/",
  names: ["All Files"],
  data: []
}
```

The following table represents the response parameters of *details* operations.

|Parameter|Type|Default|Explanation|

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

|details|FileManagerDirectoryContent|-|Details of the requested item(s).|

|error|ErrorDetails|-|Error Details|

*\*Refer [File request and response contents](#) for the contents of details and error\*.*

*Example:*

#### **CSHARP**

```
{
  cwd:null,
  files:null,
  error:null,
  details:
  {
    name:"All Files",
    location:"\\Files\\FileContents\\All Files",
    isFile:false,
    size:"679.8 KB",
    created:"3/8/2019 10:18:37 AM",
    modified:"3/8/2019 10:18:39 AM",
    multipleFiles:false
  }
}
```

### [Search](#)

The following table represents the request parameters of *search* operations.

|Parameter|Type|Default|Explanation|

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

|action|String|search|Name of the file operation.|

|path|String|-|Relative path to the directory where the files should be searched.|

|showHiddenItems|Boolean|-|Defines show or hide the hidden items.|

caseSensitive	Boolean	-	Defines search is case sensitive or not.
searchString	String	-	String to be searched in the directory.
data	FileManagerDirectoryContent	-	Details of the searched item.

*Example:*

#### **CSHARP**

```
{
  action: "search",
  path: "/",
  searchString: "*nature*",
  showHiddenItems: false,
  caseSensitive: false,
  data: []
}
```

The following table represents the response parameters of *search* operations.

Parameter	Type	Default	Explanation
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Files and folders in the searched directory that matches the search input.
error	ErrorDetails	-	Error Details

*\*Refer [File request and response contents](#) for the contents of cwd, files and error.\**

*Example:*

#### **CSHARP**

```
{
  cwd:
  {
    name:files,
    size:0,
    dateModified:"2019-03-15T10:07:00.8658158+00:00",
    dateCreated:"2019-02-27T17:36:15.6571949+00:00",
    hasChild:true,
    isFile:false,
    type:"",
    filterPath:"\\",
  },
  files:[
    {
      name:"Nature",
      size:0,
      dateModified:"2019-03-08T10:18:42.9937708+00:00",
      dateCreated:"2019-03-08T10:18:42.5907729+00:00",
      hasChild:true,
      isFile:false,
      type:"",
      filterPath:"\\FileContents\\Nature"
    }
  ]
}
```



```

}
],
error: null,
details: null
}

```

### Copy

The following table represents the request parameters of *copy* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	copy	Name of the file operation.
path	String	-	Relative path to the directory where the files should be copied.
names	String[]	-	List of files to be copied.
targetPath	String	-	Relative path where the items to be pasted are located.
data	FileManagerDirectoryContent	-	Details of the copied item.
renameFiles	String[]	-	Details of the renamed item.

*Example:*

### CSHARP

```

{
  action: "copy",
  path: "/",
  names: ["6.png"],
  renameFiles: ["6.png"],
  targetPath: "/Videos/"
}

```

The following table represents the response parameters of *copy* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Details of copied files or folders
error	ErrorDetails	-	Error Details

*\*Refer [File request and response contents](#) for the contents of cwd, files and error.\**

*Example:*

### CSHARP

```

{
  cwd: null,
  files: [
    {
      path: null,
      action: null,

```

```

newName: null,
names: null,
name: "justin.mp4",
size: 0,
previousName: "album.mp4",
dateModified: "2019-06-21T06:58:32+00:00",
dateCreated: "2019-06-24T04:22:14.6245618+00:00",
hasChild: false,
isFile: true,
type: ".mp4",
id: null,
filterPath: "\\\"
}
],
error: null,
details: null
}

```

### Move

The following table represents the request parameters of *move* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	move	Name of the file operation.
path	String	-	Relative path to the directory where the files should be copied.
names	String[]	-	List of files to be moved.
targetPath	String	-	Relative path where the items to be pasted are located.
data	FileManagerDirectoryContent	-	Details of the moved item.
renameFiles	String[]	-	Details of the renamed item.

*Example:*

### CSHARP

```

{
  action: "move",
  path: "/",
  names: ["6.png"],
  renameFiles: ["6.png"],
  targetPath: "/Videos/"
}

```

The following table represents the response parameters of *copy* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Details of cut files or folders

|error|ErrorDetails|-|Error Details|

\*Refer [File request and response contents](#) for the contents of cwd, files and error.\*

Example:

#### CSHARP

```
{
  cwd:null,
  files:[
    {
      path:null,
      action:null,
      newName:null,
      names:null,
      name:"justin biber.mp4",
      size:0,
      previousName:"justin biber.mp4",
      dateModified:"2019-06-21T06:58:32+00:00",
      dateCreated:"2019-06-24T04:26:49.2690476+00:00",
      hasChild:false,
      isFile:true,
      type:".mp4",
      id:null,
      filterPath:"\\Videos\\"
    }
  ],
  error:null,
  details:null
}
```

#### Upload

The following table represents the request parameters of *Upload* operations.

Parameter	Type	Default	Explanation
-----------	------	---------	-------------

----	----	----	----
------	------	------	------

action	String	Save	Name of the file operation.
--------	--------	------	-----------------------------

path	String	-	Relative path to the location where the file has to be uploaded.
------	--------	---	--

uploadFiles	IList<IFormFile>	-	File that are uploaded.
-------------	------------------	---	-------------------------

Example:

#### CSHARP

```
uploadFiles: (binary),
path: /,
action: Save,
data: {
  path:null,
  action:null,
  newName:null,
  names:null,
  name:"Downloads",
  size:0,
```

```

previousName:null,
dateModified:"2019-07-22T11:23:46.7153977 00:00",
dateCreated:"2019-07-22T11:26:13.9047229 00:00",
hasChild:false,
isFile:false,
type:"",
id:null,
filterPath:"\\",
targetPath:null,
renameFiles:null,
uploadFiles:null,
caseSensitive:false,
searchString:null,
showHiddenItems:false,
_fm_iconClass:null,
_fm_id:"fe_tree_1",
_fm_pId:null,
_fm_selected:false,
_fm_icon:null,
data:null,
targetData:null,
permission:null
}

```

The upload response is an empty string.

#### Download

The following table represents the request parameters of *download* operations.

Parameter	Type	Default	Explanation
action	String	download	Name of the file operation
path	String	-	Relative path to location where the files to download are present.
names	String[]	-	Name list of the items to be downloaded.
data	FileManagerDirectoryContent	-	Details of the download item.

*Example:*

#### C#

```

{
  action:"download",
  path:"/",
  names:["1.png"],
  data:[
    {
      path:null,
      action:null,
      newName:null,
      names:null,
      name:"1.png",
      size:49792,
      previousName:null,

```

```

dateModified:"2019-07-22T12:15:45.0972405+00:00",
dateCreated:"2019-07-22T12:15:45.0816042+00:00",
hasChild:false,
isFile:true,
type:".png",
id:null,
filterPath:"\\",
targetPath:null,
renameFiles:null,
uploadFiles:null,
caseSensitive:false,
searchString:null,
showHiddenItems:false,
_fm_iconClass:"e-fe-image",
_fm_id:null,
_fm_pId:null,
_fm_selected:false,
_fm_icon:null,
data:null,
targetData:null,
permission:null,
_fm_created:"2019-07-22T12:15:45.081Z",
_fm_modified:"2019-07-22T12:15:45.097Z",
_fm_imageUrl:"https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage?path=/1.png",
_fm_imageAttr:
{
  alt:"1.png"
},
_fm_htmlAttr:
{
  class:"e-large-icon",
  title:"1.png"
}
}
]
}

```

Downloads the requested items from the file server in response.

#### [GetImage](#)

The following table represents the request parameters of *GetImage* operations.

Parameter	Type	Default	Explanation
path	String	-	Relative path to the image file

Return the image as a file stream in response.

The request from the file manager can be customized using the **OnSend** event. Additional information can be passed to the file manager in file operation response and can be used in customization.

#### [File request and response contents](#)

The following table represents the contents of *data*, *cwd*, and *files* in the file manager request and response.

Parameter	Type	Default	Explanation
----	----	----	----
name	String	-	File name
dateCreated	String	-	Date in which file was created (UTC Date string).
dateModified	String	-	Date in which file was last modified (UTC Date string).
filterPath	String	-	Relative path to the file or folder.
hasChild	Boolean	-	Defines this folder has any child folder or not.
isFile	Boolean	-	Say whether the item is file or folder.
size	Number	-	File size
type	String	-	File extension

The following table represents the contents of *error* in the file manager request and response.

Parameter	Type	Default	Explanation
----	----	----	----
code	String	-	Error code
message	String	-	Error message
fileExists	String[]	-	List of duplicate file names

The following table represents the contents of *details* in the file manager request and response.

Parameter	Type	Default	Explanation
----	----	----	----
name	String	-	File name
dateCreated	String	-	Date in which file was created (UTC Date string).
dateModified	String	-	Date in which file was last modified (UTC Date string).
filterPath	String	-	Relative path to the file or folder.
hasChild	Boolean	-	Defines this folder has any child folder or not.
isFile	Boolean	-	Say whether the item is file or folder.
size	Number	-	File size
type	String	-	File extension
multipleFiles	Boolean	-	Say whether the details are about single file or multiple files.

### Action Buttons

The file manager has several menu buttons to access the file operations. The list of menu buttons available in the file manager is given in the following table.

Menu Button	Behaviour
----	----
SortBy	Opens the sub menu to choose the sorting order and sorting parameter.

|View| Opens the sub menu to choose the View.|

|Open| Navigates to the selected folder. Opens the preview for image files.|

|Refresh| Initiates the read operation for the current directory and displays the updated directory content.|

|NewFolder| Opens the new folder dialog box to receive the name for the new folder.|

|Rename| Opens the rename dialog box to receive the new name for the selected item.|

|Delete| Opens the delete dialog box to confirm the removal of the selected items from the file system.|

|Upload| Opens the upload box to select the items to upload to the file system.|

|Download| Downloads the selected item(s).|

|Details| Get details about the selected items and display them in details dialog box.|

|SelectAll| Selects all the files and folders displayed in the view section.|

The action menu buttons are present in the toolbar and context menu. The toolbar contains the buttons based on the selected items count, while the context menu will appear with a list based on the target.

#### Toolbar

The toolbar can be divided into two sections as right and left. Whenever the toolbar buttons exceed the size, the buttons present in the left section of the toolbar will be moved to the toolbar popup.

The following table provides the toolbar buttons that appear based on the selection.

<!-- markdownlint-disable MD033 -->

Selected Items Count	Left section	Right section
0 (none of the item )	<i>SortBy</i> Refresh <i>NewFolder</i> Upload	<i>View</i> Details
1 (single item selected)	<i>Delete</i> Download* Rename	<i>Selected items count</i> View* Details
>1 (multiple selection)	<i>Delete</i> Download	<i>Selected items count</i> View* Details

#### Context menu

The following table provides the default context menu item and the corresponding target areas.

<!-- markdownlint-disable MD033 -->

Menu Name	Menu Items	Target
Layout	<i>SortBy</i> View <i>Refresh</i> <i>NewFolder</i> Upload Details* Select all	<i>Empty space in the view section (details view and large icon view area)</i> . Empty folder content.
Folders	<i>Open</i> Delete <i>Rename</i> Downloads* Details	* Folders in treeview, details view, and large icon view.
Files	<i>Open</i> Delete <i>Rename</i> Downloads* Details	* Files in details view and large icon view.

## Multiple File Selection in Blazor FileManager Component

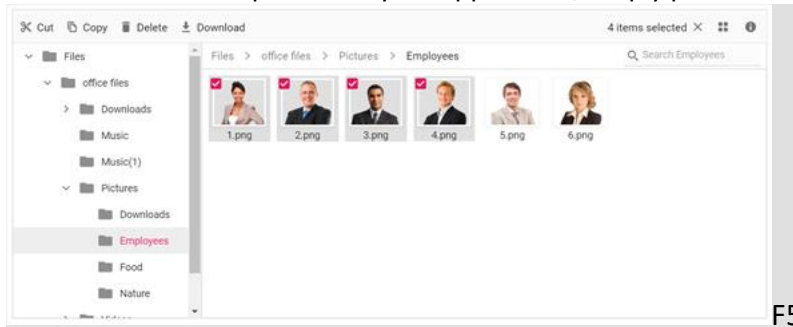
The file manager allows you to select multiple files by enabling the `AllowMultiSelection` property (enabled by default). The multiple selection can be done by pressing the `Ctrl` key or `Shift` key and selecting the files. The check box can also be used to do multiple selection. `Ctrl + A` can be used to select all files in the current directory. The `FileSelected` event will be triggered when the items of file manager control is selected or unselected.

### ASPX-CS

```
@using Syncfusion.Blazor.FileManager
<SfFileManager AllowMultiSelection="true"
TValue="FileManagerDirectoryContent">
  <FileManagerAjaxSettings Url="/api/SampleData/FileOperations"
UploadUrl="/api/SampleData/Upload"
DownloadUrl="/api/SampleData/Download"
GetImageUrl="/api/SampleData/GetImage">
  </FileManagerAjaxSettings>
</SfFileManager>
```

### Output

After successful compilation of your application, simply press



F5 to run the application.

Output be like the below.

![Blazor FileManager with Multiple Selection](images/blazor-filemanager-multi-selection.png)

## Drag and Drop in Blazor FileManager Component

The file manager allows files and folders to be moved within the file system by drag and dropping them. This support can be enabled or disabled using the [AllowDragAndDrop](#) property of the file manager.

The events which trigger when using drag and drop functionality are listed below.

- `OnFileDragStart` - Triggers when the file/folder dragging is started.
- `OnFileDragStop` - Triggers when the file/folder is about to be dropped at the target.
- `FileDropped` - Triggers when the file/folder is dropped.

### ASPX-CS

```
@using Syncfusion.Blazor.FileManager
<SfFileManager AllowDragAndDrop="true" TValue="FileManagerDirectoryContent">
  <FileManagerAjaxSettings Url="/api/SampleData/FileOperations"
UploadUrl="/api/SampleData/Upload"
DownloadUrl="/api/SampleData/Download">
```

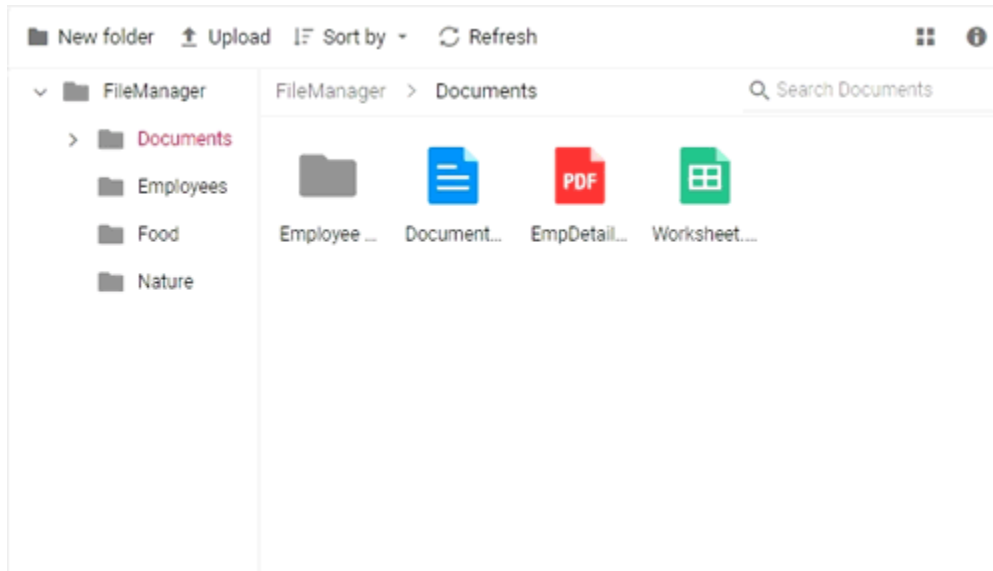


```
GetImageUrl="/api/SampleData/GetImage">
</FileManagerAjaxSettings>
</SfFileManager>
```

### Output

After successful compilation of your application, simply press **F5** to run the application.

Output be like the below.



### File System Providers in Blazor FileManager Component

The file system provider allows the File Manager component to manage the files and folders in a physical or cloud-based file system. It provides the methods for performing various file actions like creating a new folder, copying and moving of files or folders, deleting, uploading, and downloading the files or folders in the file system.

The following file providers are added in Syncfusion Blazor File Manager component.

- [ASP.NET Core file system provider](#)
- [ASP.NET MVC 5 file system provider](#)
- [ASP.NET Core Azure cloud file system Provider](#)
- [ASP.NET Core MVC 5 Azure cloud file system Provider](#)
- [ASP.NET Core Amazon S3 cloud file provider](#)
- [ASP.NET MVC Amazon S3 cloud file provider](#)
- [File Transfer Protocol file system provider](#)
- [SQL database file system provider](#)
- [NodeJS file system provider](#)
- [Google Drive file system provider](#)
- [Firebase Real time Database file system provider](#)

### ASP.NET Core file system provider

The ASP.NET Core file system provider allows the users to access and manage the physical file system. To get started, we need to clone the [EJ2.ASP.NET Core File Provider](#) using the following command.

### ASPX-CS

```
@* Initializing File Manager with ASP.NET Core service *@
@* Replace the hosted port number in the place of "{port}" *@
<SfFileManager TValue="FileManagerDirectoryContent">
  <FileManagerAjaxSettings
    Url="http://localhost:{port}/api/FileManager/FileOperations"
    UploadUrl="http://localhost:{port}/api/FileManager/Upload"
    DownloadUrl="http://localhost:{port}/api/FileManager/Download"
    GetImageUrl="http://localhost:{port}/api/FileManager/GetImage">
  </FileManagerAjaxSettings>
</SfFileManager>
```

To learn more about file actions that can be performed with ASP.NET Core file system provider, refer to this [link](#)

#### ASP.NET MVC 5 file system provider

The ASP.NET MVC5 file system provider allows the users to access and manage the physical file system. To get started, we need to clone the [EJ2.ASP.NET MVC File Provider](#) using the following command.

### CSHARP

```
git clone https://github.com/SyncfusionExamples/ej2-aspmvc-file-provider
ej2-aspmvc-file-provider
cd ej2-aspmvc-file-provider
```

After cloning, just open the project in Visual Studio and restore the NuGet packages. Now, we need to set the root directory of the physical file system in the FileManager controller using the Root Folder method.

After setting the root directory of the file system, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the ajaxSettings property of the FileManager component to the appropriate controller methods allows to manage the files in the physical file system.

### ASPX-CS

```
@*Initializing File Manager with ASP.NET MVC service*@
@* Replace the hosted port number in the place of "{port}" *@
<SfFileManager TValue="FileManagerDirectoryContent">
  <FileManagerAjaxSettings
    Url="http://localhost:{port}/FileManager/FileOperations"
    UploadUrl="http://localhost:{port}/FileManager/Upload"
    DownloadUrl="http://localhost:{port}/FileManager/Download"
    GetImageUrl="http://localhost:{port}/FileManager/GetImage">
  </FileManagerAjaxSettings>
</SfFileManager>
```

To learn more about file actions that can be performed with ASP.NET MVC 5 file system provider, refer to this [link](#)

#### ASP.NET Core Azure cloud file system provider

The Azure file system provider allows the users to access and manage the blobs in the Azure blob storage. To get started, we need to clone the [EJ2.ASP.NET Core Azure File Provider](#) using the following command

### CSHARP

```
void RegisterAzure(string accountName, string accountKey, string blobName)
```

Then, set the blob container and the root blob directory by passing the corresponding URLs as parameters in the `setBlobContainer` method as shown below.

### CSHARP

```
void setBlobContainer(string blobPath, string filePath)
```

Also, assign the same *blobPath* URL and *filePath* URL in [AzureFileOperations](#) and [AzureUpload](#) methods in the `FileManager` controller to determine the original path of the Azure blob.

After setting the blob container references, just build and run the project. Now, the project will be hosted in `http://localhost:{port}:{port}` and just mapping the `ajaxSettings` property of the `FileManager` component to the appropriate controller methods allows to manage the Azure blob storage.

### ASPX-CS

```
@*Initializing File Manager with Azure service.*@
@* Replace the hosted port number in the place of "{port}" *@
<SfFileManager TValue="FileManagerDirectoryContent">
  <FileManagerAjaxSettings
    Url="http://localhost:{port}/api/AzureProvider/AzureFileOperations"
    UploadUrl="http://localhost:{port}/api/AzureProvider/AzureUpload"
    DownloadUrl="http://localhost:{port}/api/AzureProvider/AzureDownload"
    GetImageUrl="http://localhost:{port}/api/AzureProvider/AzureGetImage">
  </FileManagerAjaxSettings>
</SfFileManager>
```

**NuGet:** Additionally, we have created a [NuGet](#) package of **ASP.NET Core Azure file system provider**.

Please, use the following command to install the NuGet package in an application.

### CSHARP

```
void RegisterAzure(string accountName, string accountKey, string blobName)
```

Then, set the blob container and the root blob directory by passing the corresponding URLs as parameters in the `setBlobContainer` method as shown below.

### CSHARP

```
void setBlobContainer(string blobPath, string filePath)
```

Also, assign the same *blobPath* URL and *filePath* URL in [AzureFileOperations](#) and [AzureUpload](#) methods in the `FileManager` controller to determine the original path of the Azure blob.

After setting the blob container references, just build and run the project. Now, the project will be hosted in `http://localhost:{port}:{port}` and just mapping the `ajaxSettings` property of the `FileManager` component to the appropriate controller methods allows to manage the Azure blob storage.

### ASPX-CS

```
@*Initializing File Manager with Azure service.*@
@* Replace the hosted port number in the place of "{port}" *@
<SfFileManager TValue="FileManagerDirectoryContent">
  <FileManagerAjaxSettings
    Url="http://localhost:{port}/AzureProvider/AzureFileOperations"
    UploadUrl="http://localhost:{port}/AzureProvider/AzureUpload"
    DownloadUrl="http://localhost:{port}/AzureProvider/AzureDownload"
    GetImageUrl="http://localhost:{port}/AzureProvider/AzureGetImage">
  </FileManagerAjaxSettings>
</SfFileManager>
```

To learn more about file actions that can be performed with ASP.NET MVC Azure cloud file system provider, refer to this [link](#)

#### ASP.NET Core Amazon S3 cloud file provider

In ASP.NET Core, Amazon **S3** (*Simple Storage Service*) cloud file provider allows the users to access and manage a server hosted file system as collection of objects stored in the Amazon S3 Bucket. To get started, clone the [EJ2.ASP.NET Core Amazon S3 File Provider](#) using the following command

### CSHARP

```
void RegisterAmazonS3(string bucketName, string awsAccessKeyId, string
awsSecretAccessKey, string bucketRegion)
```

After registering the Amazon client account details, just build and run the project. Now, the project will be hosted in `http://localhost:{port}:{port}` and just mapping the `ajaxSettings` property of the FileManager component to the appropriate controller methods allows to manage the Amazon **S3** (*Simple Storage Service*) bucket's objects storage.

### ASPX-CS

```
@*Initializing File Manager with ASP.NET Core Amazon service*@
@* Replace the hosted port number in the place of "{port}" *@
<SfFileManager TValue="FileManagerDirectoryContent">
  <FileManagerAjaxSettings
    Url="http://localhost:{port}/api/AmazonS3Provider/AmazonS3FileOperations"
    UploadUrl="http://localhost:{port}/api/AmazonS3Provider/AmazonS3Upload"
    DownloadUrl="http://localhost:{port}/api/AmazonS3Provider/AmazonS3Download"
    GetImageUrl="http://localhost:{port}/api/AmazonS3Provider/AmazonS3GetImage">
  </FileManagerAjaxSettings>
</SfFileManager>
```

To learn more about the file actions that can be performed with Amazon S3 Cloud File provider, refer to this [link](#)

#### ASP.NET MVC Amazon S3 cloud file provider

In ASP.NET MVC, Amazon **S3** (*Simple Storage Service*) cloud file provider allows the users to access and manage a server hosted files as collection of objects stored in the Amazon S3 Bucket. To get started, clone the [EJ2.ASP.NET MVC File Provider](#) using the following command

### CSHARP

```
void RegisterAmazonS3(string bucketName, string awsAccessKeyId, string
awsSecretAccessKey, string bucketRegion)
```

After registering the Amazon client account details, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the **ajaxSettings** property of the FileManager component to the appropriate controller methods allows to manage the Amazon **S3** (*Simple Storage Service*) bucket's objects storage.

### ASPX-CS

```
@*Initializing File Manager with ASP.NET MVC Amazon service*@
@* Replace the hosted port number in the place of "{port}" *@
<SfFileManager>
<FileManagerAjaxSettings
Url="http://localhost:{port}/FileManager/FileOperations"
UploadUrl="http://localhost:{port}/FileManager/Upload"
DownloadUrl="http://localhost:{port}/FileManager/Download"
GetImageUrl="http://localhost:{port}/FileManager/GetImage">
</FileManagerAjaxSettings>
</SfFileManager>
```

To learn more about the file actions that can be performed with ASP.NET MVC Amazon S3 Cloud File Provider, refer to this [link](#)

### File Transfer Protocol file system provider

In ASP.NET Core, File Transfer Protocol file system provider allows the users to access to the hosted file system as collection of objects stored in the file storage using File Transfer Protocol. To get started, clone the [EJ2.ASP.NET Core FTP File Provider](#) using the following command

### CSHARP

```
void SetFTPConnection(string hostName, string userName, string password)
```

After registering the File Transfer Protocol details, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the **ajaxSettings** property of the FileManager component to the appropriate controller methods allows you to manage the FTP's objects storage.

### ASPX-CS

```
@*Initializing File Manager with File Transfer Protocol service*@
@* Replace the hosted port number in the place of "{port}" *@
<SfFileManager TValue="FileManagerDirectoryContent">
<FileManagerAjaxSettings
Url="http://localhost:{port}/api/FTPProvider/FTPFileOperations"
UploadUrl="http://localhost:{port}/api/FTPProvider/FTPUpload"
DownloadUrl="http://localhost:{port}/api/FTPProvider/FTPDownload"
GetImageUrl="http://localhost:{port}/api/FTPProvider/FTPGetImage">
</FileManagerAjaxSettings>
</SfFileManager>
```

To learn more about the file actions that can be performed with File Transfer Protocol file system provider, refer to this [link](#)

### SQL database file system provider

The SQL database file system provider allows the users to manage the file system being maintained in a SQL database table. Unlike the other file system providers, the SQL database file system provider works on ID basis. Here, each file and folder have a unique ID based on which all the file operations will be performed. To get started, we need to clone the [EJ2.ASP.NET Core SQL Server Database File Provider](#) using the following command.

#### JSON

```
<add name="FileExplorerConnection" connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\FileManager.mdf;Inte
grated Security=True;Trusted Connection=true" />
```

After cloning, just open the project in Visual Studio and restore the NuGet packages. To establish the SQL server connection with the database file (for eg: FileManager.mdf), we need to specify the connection string in the web config file as shown below.

#### JSON

```
<add name="FileExplorerConnection" connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\FileManager.mdf;Inte
grated Security=True;Trusted Connection=true" />
```

Then, make an entry for the connection string in `appsettings.json` file as shown below.

#### JSON

```
"ConnectionStrings": {
  "FileManagerConnection": "Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\App_Data\\F
ileManager.mdf;Integrated Security=True;Connect Timeout=30"
}
```

Now, to configure the database connection, we need to set the connection name, table name and root folder ID value by passing these values to the SetSQLConnection method.

#### C#

```
void SetSQLConnection(string name, string tableName, string tableID)\
```

Refer to this [FileManager.mdf](#), to learn about the pre-defined file system SQL database for the Blazor File Manager.

After configuring the connection, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the `ajaxSettings` property of the FileManager component to the appropriate controller methods allows to manage the files in the SQL database table.

#### ASPX-CS

```
@*Initializing File Manager with SQL database file system service*@
@* Replace the hosted port number in the place of "{port}" *@
<SfFileManager TValue="FileManagerDirectoryContent">
  <FileManagerAjaxSettings
    Url="http://localhost:{port}/api/SQLProvider/SQLFileOperations"
```

```
UploadUrl="http://localhost:{port}/api/SQLProvider/SQLUpload"
DownloadUrl="http://localhost:{port}/api/SQLProvider/SQLDownload"
GetImageUrl="http://localhost:{port}/api/SQLProvider/SQLGetImage">
</FileManagerAjaxSettings>
</SfFileManager>
```

To learn more about file actions that can be performed with SQL database file system provider, refer to this [link](#)

### Google Drive file system provider

The Google Drive file system provider allows the users to manage the files and folders in a Google Drive account. The Google Drive file system provider works on id basis where each file and folder have a unique ID. To get started, we need to clone the [EJ2.ASP.NET Core Google Drive File Provider](#) using the following command.

#### ASPX-CS

```
@*Initializing File Manager with Google Drive file system service.*@
@* Replace the hosted port number in the place of "{port}" *@
<SfFileManager TValue="FileManagerDirectoryContent">
<FileManagerAjaxSettings
Url="http://localhost:{port}/api/GoogleDriveProvider/GoogleDriveFileOperations"
UploadUrl="http://localhost:{port}/api/GoogleDriveProvider/GoogleDriveUpload"
DownloadUrl="http://localhost:{port}/api/GoogleDriveProvider/GoogleDriveDownload"
GetImageUrl="http://localhost:{port}/api/GoogleDriveProvider/GoogleDriveGetImage">
</FileManagerAjaxSettings>
</SfFileManager>
```

To learn more about file actions that can be performed with Google drive file system provider, refer to this [link](#)

### Node.js file system provider

The Node.js file system provider allows the users to manage the files and folders in a physical file system. It provides methods for performing all basic file operations like creating a folder, copy, move, delete, and download files and folders in the file system. You can use of the Node.js file system provider either by installing the [EJ2 FileManager node file system](#) package or by cloning the [file system provider](#) from the GitHub.

#### Using EJ2 FileManager node filesystem package

- Install the EJ2 FileManager node filesystem package by running the below command.

#### ASPX-CS

```
@*Initializing File Manager with NodeJS service *@
@* Replace the hosted port number in the place of "{port}" *@
<SfFileManager TValue="FileManagerDirectoryContent">
<FileManagerAjaxSettings Url="http://localhost:{port}/"
UploadUrl="http://localhost:{port}/Upload"
```

```
DownloadUrl="http://localhost:{port}/Download"  
GetImageUrl="http://localhost:{port}/GetImage">  
</FileManagerAjaxSettings>  
</SfFileManager>
```

To learn more about file actions that can be performed with Node.js file system provider, refer to this [link](#)

### Firebase file system provider

The [Firebase Real time Database](#) file system provider in **ASP.NET Core** provides the efficient way to store the File Manager file system in a cloud database as JSON representation.

#### *Generate Secret access key from service account*

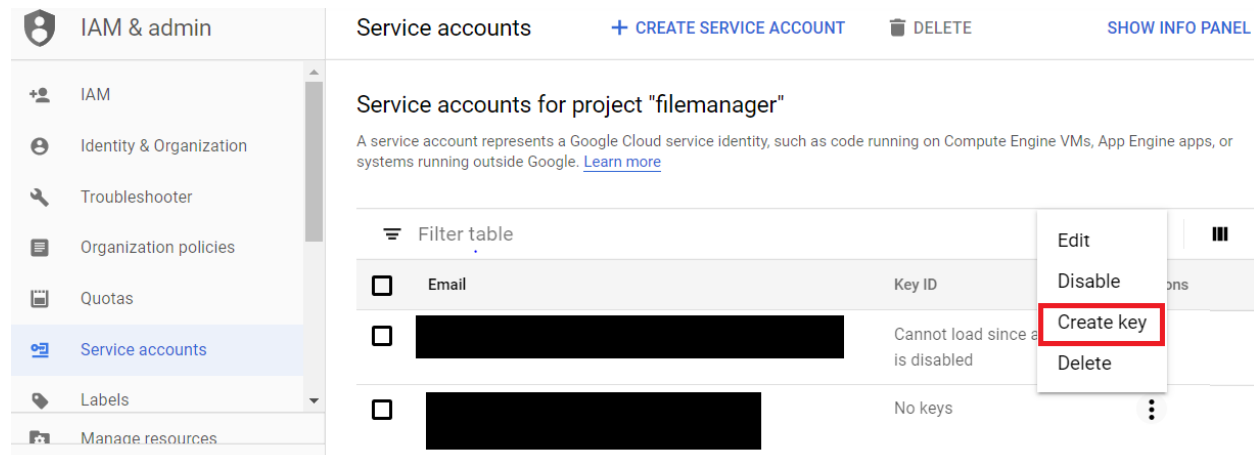
Follow the given steps to generate the secret access key:

- Click this [link](#) to Firebase console and navigate to the project settings.
- And then, navigate to the **Service Accounts** tab in the window.
- In the new dialog window, click the **Other service account** option to navigate to the Google service accounts console to generate the secret key.

The screenshot shows the Firebase console interface. On the left is a sidebar with navigation options like 'Project Overview', 'Develop', 'Quality', 'Analytics', and 'Grow'. The main area is titled 'Settings' with tabs for 'General', 'Cloud Messaging', 'Integrations', 'Service accounts', 'Data privacy', and 'Users and permissions'. The 'Service accounts' tab is active. It displays 'Legacy credentials' (Database secrets) and 'Other service accounts' (6 service accounts from Google Cloud Platform). A red box highlights the 'Other service accounts' section, and a red arrow points to the text: 'Click on this option to navigate to the Google service accounts site (https://console.cloud.google.com/iam-admin/serviceaccounts) to generate secret key for your project to enable authentication.' The right side of the page shows the 'Firebase Admin SDK' configuration, including the service account email and a code snippet for initializing the SDK in Node.js.

- Now, open the Firebase service project from the Google services console, and generate a Secret key.





- After generating the secret key, replace secret key JSON in the `access_key.json` file in the Firebase Real time Database provider project to enable authentication for performing read and write operations.

To interpolate with the Firebase Real time Database, create a project under Firebase Real time Database, and then enable the **read** and **write** permissions to access the Firebase Database by specifying the rules within the authentication tab of the Firebase project as demonstrated in the following code snippet.

By default, rules of a Firebase project will be **false**. To read and write the data, configure the **Rules** as given in the following code snippet in the *Rules* tab in the Firebase Real time Database project.

### JSON

```
{
  /* Visit https://firebase.google.com/docs/database/security to learn more
  about security rules. */
  "rules": {
    ".read": "auth!=null",
    ".write": "auth!=null"
  }
}
```

Then, create a root node and add children to the root node. Refer to the following code snippet for the structure of JSON.

### JSON

```
{
  "Files" : [ {
    "caseSensitive" : false,
    "dateCreated" : "8/22/2019 5:17:55 PM",
    "dateModified" : "8/22/2019 5:17:55 PM",
    "filterId" : "0/",
    "filterPath" : "/",
    "hasChild" : false,
    "id" : "5",
    "isFile" : false,
    "isRoot" : true,
  }
]
```

```

    "name" : "Music",
    "parentId" : "0",
    "selected" : false,
    "showHiddenItems" : false,
    "size" : 0,
    "type" : "folder"
  },
  {
    "caseSensitive" : false,
    "dateCreated" : "8/22/2019 5:18:03 PM",
    "dateModified" : "8/22/2019 5:18:03 PM",
    "filterId" : "0/",
    "filterPath" : "/",
    "hasChild" : false,
    "id" : "6",
    "isFile" : false,
    "isRoot" : true,
    "name" : "videos",
    "parentId" : "0",
    "selected" : false,
    "showHiddenItems" : false,
    "size" : 0,
    "type" : ""
  }
]
}

```

Here, the `Files` denotes the `rootNode` and the subsequent object refers to the children of the root node. `rootNode` will be taken as the root folder of the file system loaded which will be loaded in File Manager component.

After that, clone the [EJ2.ASP.NET Core Firebase Real Time Database File Provider](#) and just open the project in Visual Studio and restore the NuGet package.

Register the Firebase Real time Database by assigning *Firebase Real time Database REST API link*, *rootNode*, and *serviceAccountKeyPath* parameters in the `RegisterFirebaseRealtimeDB` method of class `FirebaseRealtimeDBFileProvider` in controller part of the ASP.NET Core application.

### ASPX-CS

```

@*Initializing File Manager with Firebase Realtime Database service*@
@* Replace the hosted port number in the place of "{port}" *@
<SfFileManager TValue="FileManagerDirectoryContent">
  <FileManagerAjaxSettings
    Url="http://localhost:{port}/api/FirebaseProvider/FirebaseRealtimeFileOperations"
    UploadUrl="http://localhost:{port}/api/FirebaseProvider/FirebaseRealtimeUpload"
    DownloadUrl="http://localhost:{port}/api/FirebaseProvider/FirebaseRealtimeDownload"
    GetImageUrl="http://localhost:{port}/api/FirebaseProvider/FirebaseRealtimeGetImage">
  </FileManagerAjaxSettings>
</SfFileManager>

```

## Accessibility in Blazor FileManager Component

The File Manager component has been designed with keeping the WAI-ARIA specifications in mind, and applying the WAI-ARIA roles, states, and properties along with keyboard support. This component is characterized by complete keyboard interaction support and ARIA accessibility support, which makes navigation easy for people who use assistive technologies (AT) or for users who completely rely on keyboard navigation.

### ARIA attributes

The following ARIA Attributes denote the state of File Manager.

#### | Property | Functionalities |

| --- | --- |

| aria-disabled | Indicates whether the File Manager component is in disabled state. |

| aria-haspopup | Indicates whether the Toolbar element has a suggestion list. |

| aria-orientation | Indicates whether the File Manager element is oriented horizontally or vertically. |

| aria-expanded | Indicates whether the Treeview node has been expanded. |

| aria-owns | Contains the ID of the suggestion list to indicate popup as a child element. |

| aria-activedescendent | Holds the ID of the active list item to focus its descendant child element. |

| aria-level | Specifies the level of the element in Treeview Structure. |

| aria-selected | Indicates whether a particular node is in selected state. |

| aria-placeholder | Represents a hint (word or phrase) to the user about what to enter in the text field  
|

| aria-label | Defines a string that labels the current element. |

| aria-checked | Indicates whether the checkbox is in checked state. |

| aria-labelledby | Labels the dialog. Often, the value of the aria-labelledby attribute will be the id of the element, which is used to title the dialog |

| aria-describedby | Describes the contents of the dialog. |

| aria-modal | Indicates whether an element is a modal when display. |

| aria-colcount | Specifies the number of columns in full table. |

| aria-colindexnt | Defines the number of columns within a grid. |

| aria-rowspan | Defines the number of rows a cell spanned within a grid. |

| aria-colspan | Defines the number of columns a cell spanned within a grid. |

| aria-sort | Indicates whether items in the grid or table are sorted in ascending or descending order. |

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

| aria-busy | This attribute is set to false when grid content is loaded. |

| aria-multiselectable | Defines more than one item has been selected. |

### Keyboard interaction

You can use the following key shortcuts to access the File Manager without interruptions.

| **Keyboard shortcuts** | **Actions** |

| --- | --- |

| Page Down | Scrolls down to the next folder or file and selects the first item when files are loaded. |

| Page Up | Scrolls up to previous folder and select the first item when files are loaded. |

| Enter | Selects the focused item and navigate through the child elements. |

| Tab | Focuses on the first element of toolbar and navigates through the next tab indexed element. |

| Esc(Escape) | Closes the image when it is in open state. |

| Alt+N | Creates a new folder dialog. |

| F5 | Refresh the file manager element. |

| Ctrl+Shift+1 | Changes the file manager layout to Grid view. |

| Ctrl+Shift+2 | Changes the file manager layout to Details view. |

### How To

#### Adding Custom Item To Context Menu in Blazor FileManager Component

The context menu can be customized using the `ContextMenuSettings`, `MenuOpened`, and `OnMenuClick` events.

The following example shows adding a custom item in the context menu. The `ContextMenuSettings` is used to add new menu item. The `MenuOpened` event is used to add the icon to the new menu item. The `OnMenuClick` event is used to add an event handler to the new menu item.

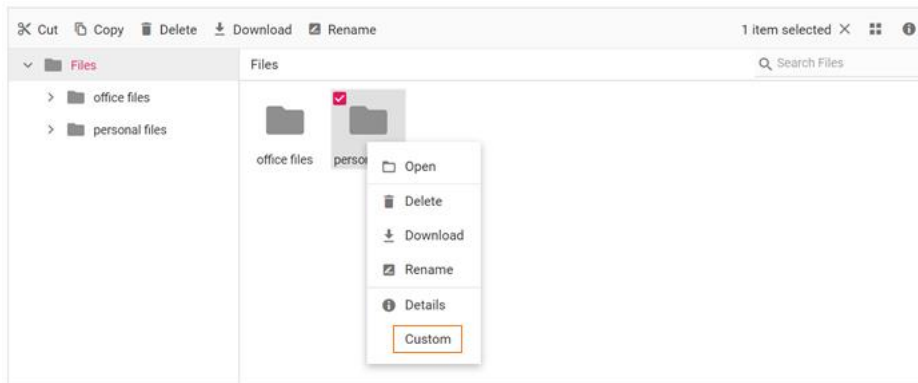
#### ASPX-CS

```
@using Syncfusion.Blazor.FileManager
<SfFileManager TValue="FileManagerDirectoryContent">
  <FileManagerAjaxSettings Url="/api/SampleData/FileOperations"
    UploadUrl="/api/SampleData/Upload"
    DownloadUrl="/api/SampleData/Download"
    GetImageUrl="/api/SampleData/GetImage">
  </FileManagerAjaxSettings>
  <FileManagerContextMenuSettings File="@Items"
    Folder="@Items"></FileManagerContextMenuSettings>
</SfFileManager>
@code {
  public string[] Items = new string[] { "Open", "|", "Delete", "Download",
    "Rename", "|", "Details", "Custom" };
}
```

#### Run the application

After successful compilation of your application, simply press **F5** to run the application.

Output be like the below.



### Adding Custom Item To Toolbar in Blazor FileManager Component

The toolbar items can be customized using the `ToolbarSettings` API and `ToolbarItemClicked` events.

The following example shows adding a custom item in the toolbar. The new toolbar button is added using `ToolbarSettings`. The event is used to add an `ToolbarItemClicked` event handler to the new toolbar button.

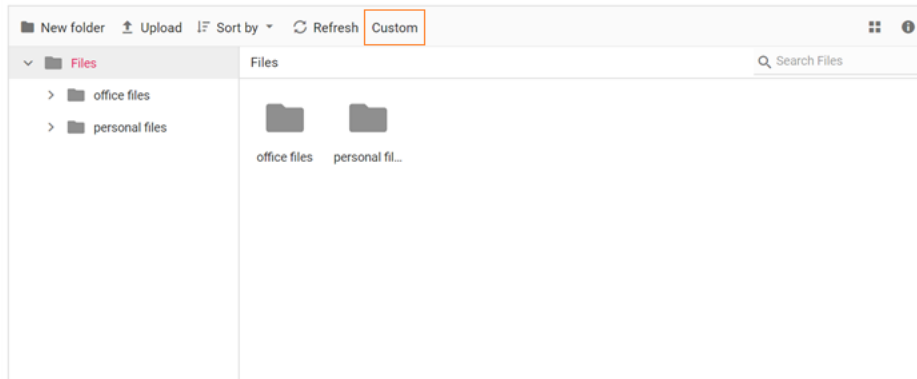
#### ASPX-CS

```
@using Syncfusion.Blazor.FileManager
<SfFileManager TValue="FileManagerDirectoryContent">
  <FileManagerAjaxSettings Url="/api/SampleData/FileOperations"
    UploadUrl="/api/SampleData/Upload"
    DownloadUrl="/api/SampleData/Download"
    GetImageUrl="/api/SampleData/GetImage">
  </FileManagerAjaxSettings>
  <FileManagerToolbarSettings Items="@Items"></FileManagerToolbarSettings>
</SfFileManager>
@code {
  public string[] Items = new string[] { "NewFolder", "Upload", "Delete",
    "Download", "Rename", "SortBy", "Refresh", "Selection", "View", "Details",
    "Custom" };
}
```

#### Run the application

After successful compilation of your application, simply press **F5** to run the application.

Output be like the below.



## File Upload

### Getting Started with Blazor File Upload Component

This section briefly explains about how to include a [Blazor File Upload](#) Component in your Blazor Server-Side and Client-Side application. You can refer to our Getting Started with [Blazor Server-Side File Upload](#) and [Blazor WebAssembly File Upload](#) documentation pages for configuration specifications.

#### Importing Syncfusion Blazor component in the application

- Install `Syncfusion.Blazor.Inputs` NuGet package to the application by using the `NuGet Package Manager`.

---

Please ensure to check the `Include prerelease` option for our Beta release.

---

- 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.Inputs` packages.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
```

### 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 uploader component to the application

To initialize the uploader component add the below code to your `Index.razor` view page which is present under `~/Pages` folder.

#### ASPX-CS

```
<SfUploader></SfUploader>
```

### Run the application

After successful compilation of your application, press **F5** to run the application.

The output will be as follows.



### Without server-side API endpoint

You can upload the files and folders in the Blazor application without specifying the server-side API endpoint using [AsyncSettings](#).

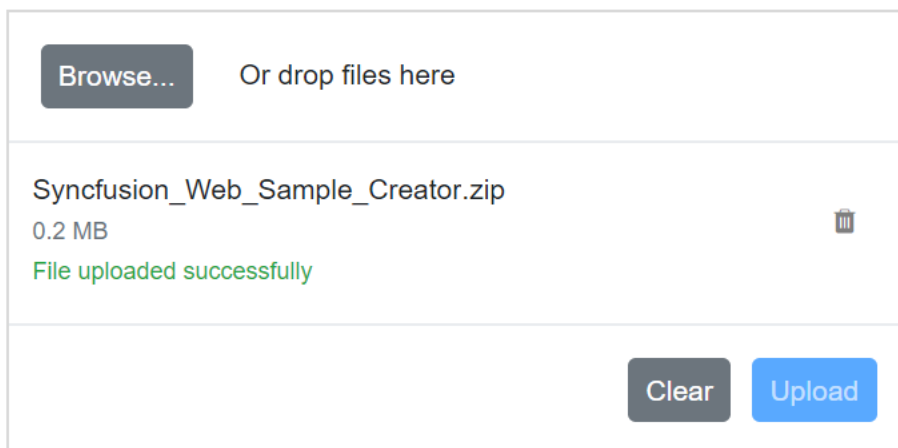
### Save and Remove actions

You can get the uploaded files as file stream in the [ValueChange](#) event argument. Now, you can write the save handler inside ValueChange event to save the files to desired location. Please find the save action code on below.

### ASPX-CS

```
@using System.IO
<SfUploader AutoUpload="false">
  <UploaderEvents ValueChange="OnChange"></UploaderEvents>
</SfUploader>
@code {
  private void OnChange(UploadChangeEventArgs args)
  {
    foreach (var file in args.Files)
    {
      var path = @"path" + file.FileInfo.Name;
      FileStream filestream = new FileStream(path, FileMode.Create,
      FileAccess.Write);
      file.Stream.WriteTo(filestream);
      filestream.Close();
      file.Stream.Close();
    }
  }
}
```

The output will be as follows.



While clicking on the remove icon in the file list, you will get the [OnRemove](#) event with removing file name as argument. So, you can write the remove handler inside OnRemove event to remove the particular file from desired location. Please find the remove action code on below.



### CSHARP

```
Private void onRemove(RemovingEventArgs args)
{
    foreach(var removeFile in args.FilesData)
    {
        if (File.Exists(Path.Combine(@"rootPath", removeFile.Name)))
        {
            File.Delete(Path.Combine(@"rootPath", removeFile.Name))
        }
    }
}
```

#### With server-side API endpoint

The upload process requires save and remove action URL to manage the upload process in the server.

---

\* The save action is necessary to handle the upload operation.

\* The remove action is optional, one can handle the removed files from server.

---

The save action handler upload the files that needs to be specified in the [SaveUrl](#) property.

The save handler receives the submitted files and manages the save process in server. After uploading the files to server location, the color of the selected file name changes to green and the remove icon is changed as bin icon.

The remove action is optional. The remove action handler removes the files that needs to be specified in the [RemoveUrl](#) property.

### CSHARP

```
[Route("api/[controller]")]
private IHostingEnvironment hostingEnv;
public SampleDataController(IHostingEnvironment env)
{
    this.hostingEnv = env;
}
[HttpPost("[action]")]
public void Save(IList<IFormFile> UploadFiles)
{
    long size = 0;
    try
    {
        foreach (var file in UploadFiles)
        {
            var filename = ContentDispositionHeaderValue
                .Parse(file.ContentDisposition)
                .FileName
                .Trim('"');
            filename = hostingEnv.ContentRootPath + @"\" + filename;
            size += (int)file.Length;
            if (!System.IO.File.Exists(filename))
            {
                using (FileStream fs = System.IO.File.Create(filename))
                {
                    file.CopyTo(fs);
                    fs.Flush();
                }
            }
        }
    }
}
```

```

    }
    }
    }
    }
    catch (Exception e)
    {
        Response.Clear();
        Response.StatusCode = 204;
        Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
        "File failed to upload";
        Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
        e.Message;
    }
    }
    [HttpPost("[action]")]
    public void Remove(IList<IFormFile> UploadFiles)
    {
        try
        {
            var filename = hostingEnv.ContentRootPath + $"{@"\{UploadFiles[0].FileName}";
            if (System.IO.File.Exists(filename))
            {
                System.IO.File.Delete(filename);
            }
        }
        catch (Exception e)
        {
            Response.Clear();
            Response.StatusCode = 200;
            Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
            "File removed successfully";
            Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
            e.Message;
        }
    }
}

```

**ASPX-CS**

```

<SfUploader ID="UploadFiles">
  <UploaderAsyncSettings SaveUrl="api/SampleData/Save"
  RemoveUrl="api/SampleData/Remove"></UploaderAsyncSettings>
</SfUploader>

```

**Configure allowed file types**

You can allow the specific files alone to upload using the [AllowedExtensions](#) property. The extension can be represented as collection by comma separators. The uploader component filters the selected or dropped files to match against the specified file types and processes the upload operation. The validation happens when you specify value to inline attribute to accept the original input element.

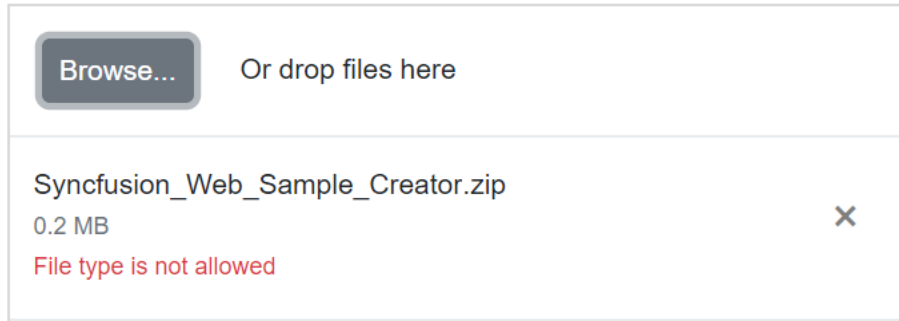
**ASPX-CS**

```

<SfUploader AllowedExtensions=".doc, docx, .xls, xlsx"></SfUploader>

```

The output will be as follows,



You can also explore our [Blazor File Upload example](#) to understand how to browse the files which you want to upload to the server.

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](#)
- [Getting Started with Syncfusion File Upload in Blazor WebAssembly using Visual Studio 2019](#)

### Asynchronous Upload in Blazor File Upload Component

The uploader component allows you to upload the files asynchronously. The upload process requires save and remove action URL to manage the upload process in the server.

- The save action is necessary to handle the upload operation.
- The remove action is optional, one can handle the removed files from server.

The name attribute must match the name of a parameter in the POST method. For more information, refer [here](#). The name attribute is automatically generated from the control's ID property. If the name attribute differs from the ID property, then you can use the `htmlAttributes` property to set the name attribute directly to the input element. For more information refer [here](#).

The file can be uploaded automatically or manually. For more information, you can refer to the [Auto Upload](#) section from the documentation.

#### Multiple file upload

By Default, the uploader component allows you to select and upload multiple files simultaneously. The selected files are organized in a list for every file selection until you clear it by clicking clear button that is shown in footer. You can add the multiple attributes to original input element of file by enabling the multiple file selection. The following example explains about [AllowMultiple](#) file upload settings.

`SaveUrl` and `RemoveUrl` action explained in this [link](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader ID="UploadFiles">
  <UploaderAsyncSettings SaveUrl="api/SampleData/Save"
    RemoveUrl="api/SampleData/Remove">
  </UploaderAsyncSettings>
</SfUploader>
```

### Single file upload

You can select and upload a single file by disabling the [AllowMultiple](#) file selection property. The file list item is removed for every selection and it always maintain a single file to upload. You can remove the multiple attributes form the original input element of file by enabling the single file upload property.

The following example explains about single file upload settings.

**SaveUrl** and **RemoveUrl** action explained in this [link](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader ID="UploadFiles" AllowMultiple=false>
  <UploaderAsyncSettings SaveUrl="api/SampleData/Save"
    RemoveUrl="api/SampleData/Remove">
  </UploaderAsyncSettings>
</SfUploader>
```

### Auto upload

By default, the uploader processes the files to upload once the files are selected and added in upload queue. To upload manually, disable the [AutoUpload](#) property. When you disable this property, you can use the action buttons to call upload all or clear all actions manually. You can change those buttons text using the [Buttons](#) property in Uploader component.

**SaveUrl** and **RemoveUrl** action explained in this [link](#).

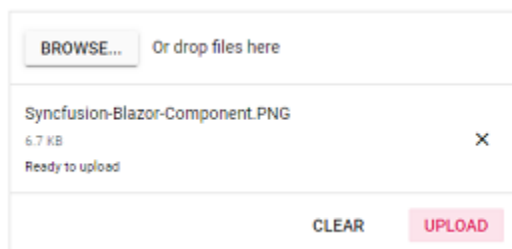
#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader ID="UploadFiles" AllowMultiple=false AutoUpload=false>
  <UploaderAsyncSettings SaveUrl="api/SampleData/Save"
    RemoveUrl="api/SampleData/Remove">
  </UploaderAsyncSettings>
</SfUploader>
```

The auto upload output will be as follows.



The auto upload false output will be as follows.



### Sequential upload

By default, the uploader component process multiple files to upload simultaneously. When you enable the [SequentialUpload](#) property, the selected files will process sequentially (one after the other) to the server. If the file uploaded successfully or failed, the next file will upload automatically in this sequential upload. This feature helps to reduce the upload traffic and reduce the failure of file upload.

`SaveUrl` and `RemoveUrl` action explained in this [link](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader ID="UploadFiles" SequentialUpload=true AutoUpload=false>
  <UploaderAsyncSettings SaveUrl="api/SampleData/Save"
  RemoveUrl="api/SampleData/Remove">
  </UploaderAsyncSettings>
</SfUploader>
```

### Preloaded files

The uploader component allows you to pre load the list of files that are uploaded in the server. The preloaded files are useful to view and remove the files from server that can be achieved by the [Files](#) property. By default, the files are configured with uploaded successfully state on rendering file list. The following properties are mandatory to configure the preloaded files:

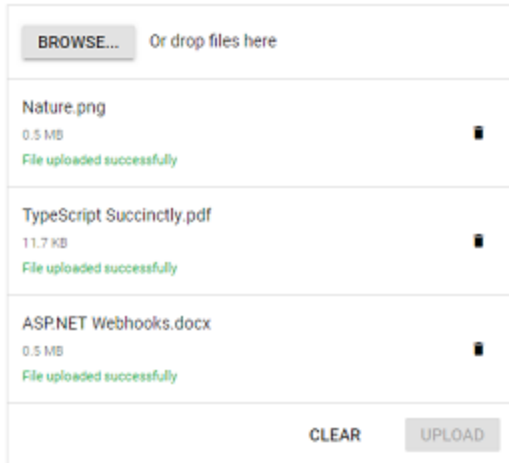
- Name
- Size
- Type

`SaveUrl` and `RemoveUrl` action explained in this [link](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader ID="UploadFiles" AutoUpload=false>
  <UploaderAsyncSettings SaveUrl="api/SampleData/Save"
  RemoveUrl="api/SampleData/Remove">
  </UploaderAsyncSettings>
  <UploaderFiles>
    <UploaderUploadedFiles Name="Nature" Size=500000
    Type=".png"></UploaderUploadedFiles>
    <UploaderUploadedFiles Name="TypeScript Succinctly" Size=12000
    Type=".pdf"></UploaderUploadedFiles>
    <UploaderUploadedFiles Name="ASP.NET Webhooks" Size=500000
    Type=".docx"></UploaderUploadedFiles>
  </UploaderFiles>
</SfUploader>
```

The output will be as follows.



### Chunk Upload in Blazor File Upload Component

The Uploader sends the large file split into small chunks and transmits to the server using AJAX. You can also pause, resume, and retry the failed chunk file.

---

The chunk upload works in asynchronous upload only.

To enable the chunk upload, set the size to [ChunkSize](#) option of the upload and it receives the value in bytes.

### Save and remove action for Blazor (ASP.NET Core hosted) application

The Uploader sends the large file split into small chunks and transmits to the server using AJAX. You can also pause, resume, and retry the failed chunk file.

### CSHARP

```
[Route("api/[controller]")]
private IHostingEnvironment hostingEnv;
public SampleDataController(IHostingEnvironment env)
{
    this.hostingEnv = env;
}
[HttpPost("[action]")]
public void Save(IList<IFormFile> chunkFile, IList<IFormFile> UploadFiles)
{
    long size = 0;
    try
    {
        foreach (var file in chunkFile)
        {
            var filename = ContentDispositionHeaderValue
                .Parse(file.ContentDisposition)
                .FileName
                .Trim('"');
            filename = hostingEnv.ContentRootPath + $"\\{filename}";
            size += file.Length;
            if (!System.IO.File.Exists(filename))
            {
                using (FileStream fs = System.IO.File.Create(filename))
                {
                    file.CopyTo(fs);
                }
            }
        }
    }
}
```

```

fs.Flush();
}
}
else
{
    using (FileStream fs = System.IO.File.Open(filename, FileMode.Append))
    {
        file.CopyTo(fs);
        fs.Flush();
    }
}
}
}
catch (Exception e)
{
    Response.Clear();
    Response.StatusCode = 204;
    Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
    "File failed to upload";
    Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
    e.Message;
}
}
[HttpPost("[action]")]
public void Remove(IList<IFormFile> UploadFiles)
{
    try
    {
        var filename = hostingEnv.ContentRootPath + $"{@"\{UploadFiles[0].FileName}";
        if (System.IO.File.Exists(filename))
        {
            System.IO.File.Delete(filename);
        }
    }
    catch (Exception e)
    {
        Response.Clear();
        Response.StatusCode = 200;
        Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
        "File removed successfully";
        Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
        e.Message;
    }
}

```

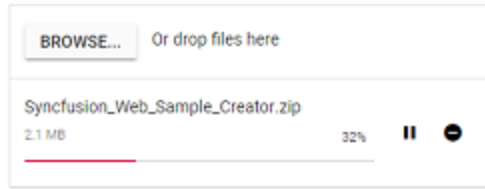
### ASPX-CS

```

@using Syncfusion.Blazor.Inputs
<SfUploader ID="UploadFiles">
    <UploaderAsyncSettings SaveUrl="api/SampleData/Save"
    RemoveUrl="api/SampleData/Remove"
    ChunkSize="500000"></UploaderAsyncSettings>
</SfUploader>

```

The output will be as follows.



The chunk upload functionality separates the selected files into blobs of the data or chunks. These chunks are transmitted to the server using an AJAX request. The chunks are sent in **sequential** order, and the next chunk can be sent to the server according to the success of the previous chunk. If any one of the chunk failed, then the remaining chunk cannot be sent to the server. The [ChunkSuccess](#) or [ChunkFailure](#) event will be triggered when the chunk is sent to the server successfully or failed. If all the chunks are sent to the server successfully, the uploader success event is triggered.

Chunk upload will work when the selected file size is greater than the specified chunk size. otherwise, it upload the files normally.

#### Save action configuration in server-side blazor

The uploader save action configuration in server-side blazor application, using MVC via `UseMvcWithDefaultRoute` in ASP.NET Core 3.0 and `services.AddMvc(option => option.EnableEndpointRouting = false).SetCompatibilityVersion(CompatibilityVersion.Version30)` on `IServiceCollection` requires an explicit opt-in inside **Startup.cs** page. This is required because MVC must know whether it can rely on the authorization and CORS Middle ware during initialization.

#### CSHARP

```
using Microsoft.AspNetCore.Mvc;
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(option => option.EnableEndpointRouting =
false).SetCompatibilityVersion(CompatibilityVersion.Version_3_0);
    services.AddRazorPages();
    services.AddServerSideBlazor();
    services.AddSingleton<WeatherForecastService>();
}
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseRouting();
    app.UseMvcWithDefaultRoute();
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapBlazorHub<App>(selector: "app");
        endpoints.MapFallbackToPage("/_Host");
    }
    );
}
```



```
} } ;  
}
```

### Additional configurations

To modify the chunk upload, the following options can be used.

- **RetryAfterDelay:** If error occurs while sending any chunk request from JavaScript, hold the operation for 500 milliseconds (by default), and retry the operation using chunk. This can be achieved by using the [AsyncSettings](#) property. You can modify the holding time interval in milliseconds.
- **RetryCount:** Specifies the number of retry actions performed when the file fails to upload. By default, retry action is performed 3 times. If the file fails to upload continuously, the request is

aborted and the uploader [Failure](#) event will trigger.

The following sample specifies the chunk upload delay with 3000 milliseconds and the retry count is 5. The failure event is triggered as the wrong saveUrl is used.

`SaveUrl` and `RemoveUrl` action explained in this [link](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs  
<SfUploader ID="UploadFiles">  
  <UploaderAsyncSettings SaveUrl="api/SampleData/Save"  
    RemoveUrl="api/SampleData/Remove" ChunkSize=500000 RetryCount=5  
    RetryAfterDelay =3000>  
</UploaderAsyncSettings>  
</SfUploader>
```

### Resumable upload

Allows you to resume an upload operation after a network failure or manually interrupts (pause) the upload. You can perform pause and resume upload actions using public methods (pause and resume) and UI interaction. The pause icon is enabled after the upload begins.

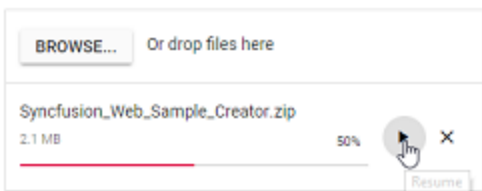
This pause and resume features available only when the chunk upload is enabled.

`SaveUrl` and `RemoveUrl` action explained in this [link](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs  
<SfUploader ID="UploadFiles">  
  <UploaderAsyncSettings SaveUrl="api/SampleData/Save"  
    RemoveUrl="api/SampleData/Remove" ChunkSize=500000>  
</UploaderAsyncSettings>  
</SfUploader>
```

The output will be as follows.



### Cancel upload

The uploader component allows you to cancel the uploading file. This can be achieved by clicking the cancel icon or using the `Cancel` method. The `Canceling` event will be fired whenever the file upload request is canceled. While canceling the upload request, the partially uploaded file is removed from the server.

When the request fails, the pause icon is changed to retry icon. By clicking the retry icon, sends the failed chunk request again to the server and upload started from where it is failed. You can retry the canceled upload request again using retry UI or `Retry` methods. But, if you retry this, the file upload action again starts from initial.

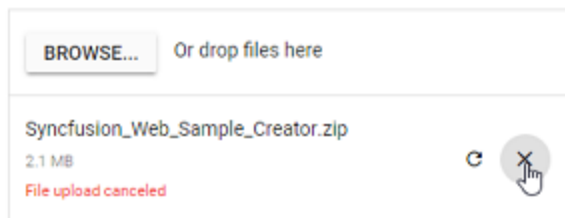
The following example explains about chunk upload with cancel support.

`SaveUrl` and `RemoveUrl` action explained in this [link](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader ID="UploadFiles">
  <UploaderAsyncSettings SaveUrl="api/SampleData/Save"
    RemoveUrl="api/SampleData/Remove" ChunkSize=500000>
  </UploaderAsyncSettings>
</SfUploader>
```

The output will be as follows.



The retry action has different working behavior for chunk upload and default upload.

- \* Chunk upload: Retries to upload the failed request where it is failed previously.

- \* Default upload: Retries to upload the failed file again from initial.

## Localization in Blazor File Upload 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

### 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 Uploader 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 Uploader 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 blazorInputs
{
    public class SampleLocalizer : ISyncfusionStringLocalizer
    {
        public string Get(string key)
        {
            return this.Manager.GetString(key);
        }
        public System.Resources.ResourceManager Manager
        {
            get
            {
                return blazorInputs.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)
------	----------------------------

---	---
-----	-----

Uploader_Abort	Abbrechen
----------------	-----------

Uploader_Browse	Durchsuche...
-----------------	---------------

Uploader_Cancel	Stornieren
-----------------	------------

Uploader_Clear	klar
----------------	------

Uploader\_Delete	Datei löschen
Uploader\_DropFilesHint	Oder legen Sie Dateien hier ab
Uploader\_FileUploadCancel	Datei-Upload abgebrochen
Uploader\_InProgress	Hochladen
Uploader\_InvalidFileType	Dateityp ist nicht erlaubt
Uploader\_InvalidMaxFileSize	Dateigröße ist zu groß
Uploader\_invalidMinFileSize	Dateigröße ist zu klein
Uploader\_Pause	Pause
Uploader\_PauseUpload	Datei-Upload angehalten
Uploader\_ReadyToUploadMessage	Bereit zum Hochladen
Uploader\_Remove	Entfernen
Uploader\_RemovedFailedMessage	Datei kann nicht entfernt werden
Uploader\_RemovedSuccessMessage	Datei erfolgreich entfernt
Uploader\_Resume	Fortsetzen
Uploader\_Retry	Wiederholen
Uploader\_Upload	Hochladen
Uploader\_UploadFailedMessage	Datei konnte nicht hochgeladen werden
Uploader\_UploadSuccessMessage	Datei erfolgreich hochgeladen

- Finally, Specify the culture for Uploader using `locale` property.

### ASPX-CS

```

@using Syncfusion.Blazor.Inputs
<SfUploader ID="UploadFiles" Locale="de">
  <UploaderAsyncSettings SaveUrl="api/SampleData/Save"
    RemoveUrl="api/SampleData/Remove"></UploaderAsyncSettings>
</SfUploader>
  
```

### Blazor WebAssembly

The Localization library allows you to localize static text content of the Uploader according to the culture currently assigned to the Uploader.

Locale key	en-US (default)
Browse	Browse...
Clear	Clear
Upload	Upload
dropFilesHint	Or drop files here

| invalidMaxFileSize | File size is too large  
| invalidMinFileSize | File size is too small  
| invalidFileType | File type is not allowed  
| uploadFailedMessage | File failed to upload  
| uploadSuccessMessage | File uploaded successfully  
| removedSuccessMessage | File removed successfully  
| removedFailedMessage | Unable to remove file  
| inProgress | Uploading  
| readyToUploadMessage | Ready to upload  
| abort | Abort  
| remove | Remove  
| cancel | Cancel  
| delete | Delete file  
| pauseUpload | File upload paused  
| pause | Pause  
| resume | Resume  
| retry | Retry  
| fileUploadCancel | File upload canceled

The following steps explain how to render the Uploader 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"),
                }
            })
        }
    }
}
```

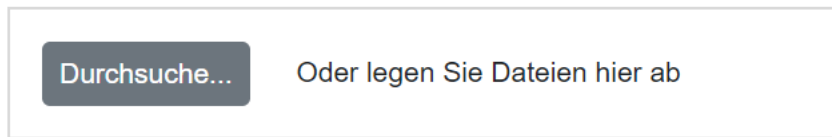
```
};
// Set the default culture
options.DefaultRequestCulture = new
Microsoft.AspNetCore.Localization.RequestCulture("de");
options.SupportedCultures = supportedCultures;
options.SupportedUICultures = supportedCultures;
options.RequestCultureProviders = new
List<Microsoft.AspNetCore.Localization.IRequestCultureProvider>() {
new Microsoft.AspNetCore.Localization.QueryStringRequestCultureProvider()
};
});
....
....
}
}
}
```

- Download the required locale packages to render the Blazor Uploader 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.Inputs
@inject HttpClient Http
<SfUploader ID="UploadFiles" Locale="de">
  <UploaderAsyncSettings
    SaveUrl="https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save"
    RemoveUrl="https://aspnetmvc.syncfusion.com/services/api/uploadbox/Remove"><
  /UploaderAsyncSettings>
</SfUploader>
@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.



## File Source in Blazor File Upload Component

### Directory upload

The [Blazor File Upload](#) component allows you to upload all files in the folders to server by using the [DirectoryUpload](#) property. When this property is enabled, the uploader component processes the files by iterating through the files and sub-directories in a directory. It allows you to select only folders instead of files to upload.

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader ID="UploadFiles" AutoUpload=false DirectoryUpload=true>
  <UploaderAsyncSettings SaveUrl="api/SampleData/Save"
    RemoveUrl="api/SampleData/Remove"></UploaderAsyncSettings>
</SfUploader>
```

### Save action configuration in server-side blazor

The uploader save action configuration in server-side blazor application, using MVC via `UseMvcWithDefaultRoute` in ASP.NET Core 3.0 and `services.AddMvc(option => option.EnableEndpointRouting = false).SetCompatibilityVersion(CompatibilityVersion.Version30)` on `IServiceCollection` requires an explicit opt-in inside **Startup.cs** page. This is required because MVC must know whether it can rely on the authorization and CORS Middle ware during initialization.

### CSHARP

```
using Microsoft.AspNetCore.Mvc;
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(option => option.EnableEndpointRouting =
        false).SetCompatibilityVersion(CompatibilityVersion.Version_3_0);
    services.AddRazorPages();
    services.AddServerSideBlazor();
    services.AddSingleton<WeatherForecastService>();
}
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }
}
```



```
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseMvcWithDefaultRoute();
app.UseEndpoints(endpoints =>
{
    endpoints.MapBlazorHub<App>(selector: "app");
    endpoints.MapFallbackToPage("/_Host");
});
}
```

*Server-side configuration for save the files of folders*

#### **CSHARP**

```
private IHostingEnvironment hostingEnv;
public SampleDataController(IHostingEnvironment env)
{
    this.hostingEnv = env;
}
[HttpPost("[action]")]
public void Save(IList<IFormFile> chunkFile, IList<IFormFile> UploadFiles)
{
    long size = 0;
    try
    {
        foreach (var file in UploadFiles)
        {
            var filename = ContentDispositionHeaderValue
                .Parse(file.ContentDisposition)
                .FileName
                .Trim('"');
            var folders = filename.Split('/');
            var uploaderFilePath = hostingEnv.ContentRootPath;
            // for Directory upload
            if (folders.Length > 1)
            {
                for (var i = 0; i < folders.Length - 1; i++)
                {
                    var newFolder = uploaderFilePath + $"{folders[i]}";
                    Directory.CreateDirectory(newFolder);
                    uploaderFilePath = newFolder;
                    filename = folders[i + 1];
                }
            }
            filename = uploaderFilePath + $"{filename}";
            size += file.Length;
            if (!System.IO.File.Exists(filename))
            {
                using (FileStream fs = System.IO.File.Create(filename))
                {
                    file.CopyTo(fs);
                    fs.Flush();
                }
            }
        }
    }
}
```

```

catch (Exception e)
{
    Response.Clear();
    Response.StatusCode = 204;
    Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
    "File failed to upload";
    Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
    e.Message;
}
}
[HttpPost("[action]")]
public void Remove(IList<IFormFile> UploadFiles)
{
    try
    {
        var filename = hostingEnv.ContentRootPath + $"{@"\{UploadFiles[0].FileName}";
        if (System.IO.File.Exists(filename))
        {
            System.IO.File.Delete(filename);
        }
    }
    catch (Exception e)
    {
        Response.Clear();
        Response.StatusCode = 200;
        Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
        "File removed successfully";
        Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
        e.Message;
    }
}

```

### Drag and drop

The uploader component allows you to drag and drop the files to upload. You can drag the files from file explorer and drop into the drop area. By default, the uploader component act as drop area element. The drop area gets highlighted when you drag the files over drop area.

### Custom drop area

The uploader component allows you to set external target element as drop area using the [DropArea](#) property. The element can be represented as HTML element or element's ID.

**SaveUrl** and **RemoveUrl** action explained in this [link](#).

### ASPX-CS

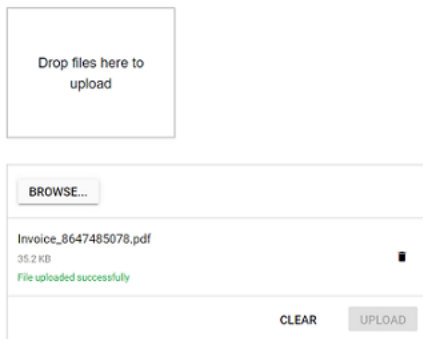
```

@using Syncfusion.Blazor.Inputs
<div ID="DropArea">
    Drop files here to upload
</div>
<SfUploader ID="UploadFiles" AutoUpload=false DropArea="#DropArea">
    <UploaderAsyncSettings SaveUrl="api/SampleData/Save"
    RemoveUrl="api/SampleData/Remove"></UploaderAsyncSettings>
</SfUploader>
<style>
#DropArea {
padding: 50px 25px;

```

```
margin: 30px auto;
border: 1px solid #c3c3c3;
text-align: center;
width: 20%;
display: inline-flex;
}
.e-file-select,
.e-file-drop {
display: none;
}
body .e-upload-drag-hover {
outline: 2px dashed brown;
}
#uploadfile {
width: 60%;
display: inline-flex;
margin-left: 5%;
}
</style>
```

The output will be as follows.



You can also explore our [Blazor File Upload example](#) to understand how to browse the files which you want to upload to the server.

## Validation in Blazor File Upload Component

The uploader component validate the selected files size and extension using the [AllowedExtensions](#), [MinFileSize](#) and [MaxFileSize](#) properties. The files can be validated before uploading to the server and can be ignored on uploading. Also, you can validate the files by setting the HTML attributes to the original input element. The validation process occurs on drag-and-drop the files also.

### File type

You can allow the specific files alone to upload using the [AllowedExtensions](#) property. The extension can be represented as collection by comma separators. The uploader component filters the selected or dropped files to match against the specified file types and processes the upload operation. The validation happens when you specify value to inline attribute to accept the original input element.

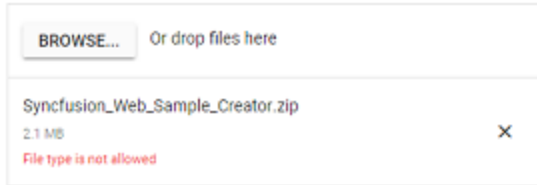
**SaveUrl** and **RemoveUrl** action explained in this [link](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader ID="UploadFiles" AllowedExtensions=".doc, .docx, .xls, .xlsx">
```

```
<UploaderAsyncSettings SaveUrl="api/SampleData/Save"
RemoveUrl="api/SampleData/Remove">
</UploaderAsyncSettings>
</SfUploader>
```

The output will be as follows,



### File size

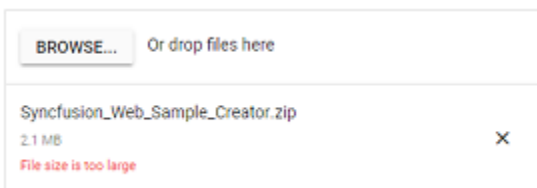
The uploader component allows you to validate the files based on its size. The validation helps to restrict uploading large files or empty files to the server. The size can be represented in **bytes**. By default, the uploader component allows you to upload **minimum file size** as 0 byte and **maximum file size** as 28.4 MB using the [MinFileSize](#) and [MaxFileSize](#) properties.

**SaveUrl** and **RemoveUrl** action explained in this [link](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader ID="UploadFiles" AllowedExtensions=".doc, .docx, .xls, .xlsx"
MinFileSize=10000 MaxFileSize=1000000>
<UploaderAsyncSettings SaveUrl="api/SampleData/Save"
RemoveUrl="api/SampleData/Remove">
</UploaderAsyncSettings>
</SfUploader>
```

The output will be as follows,



## Events in Blazor File Upload Component

This section explains the list of events of the File Upload component which will be triggered for appropriate File Upload actions.

### BeforeRemove

**BeforeRemove** event triggers on remove the uploaded file. The event used to get confirm before remove the file from server.

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
<UploaderEvents BeforeRemove="@BeforeRemovehandler"></UploaderEvents>
```

```
</SfUploader>
@code {
private void BeforeRemovehandler(BeforeRemoveEventArgs args)
{
// Here you can customize your code
}
}
```

### BeforeUpload

**BeforeUpload** event triggers when the upload process before. This event is used to add additional parameter with upload request.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
<UploaderEvents BeforeUpload="@BeforeUploadHandler"></UploaderEvents>
</SfUploader>
@code {
private void BeforeUploadHandler(BeforeUploadEventArgs args)
{
// Here you can customize your code
}
}
```

### Created

**Created** event triggers when the component is created.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
<UploaderEvents Created="@CreatedHandler"></UploaderEvents>
</SfUploader>
@code {
private void CreatedHandler(Object args)
{
// Here you can customize your code
}
}
```

### FileSelected

**FileSelected** event triggers after selecting or dropping the files by adding the files in upload queue.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
<UploaderEvents FileSelected="@FileSelectedHandler"></UploaderEvents>
</SfUploader>
@code {
private void FileSelectedHandler(SelectedEventArgs args)
{
// Here you can customize your code
}
}
```

```
}
```

### OnActionComplete

**OnActionComplete** event triggers after all the selected files has processed to upload successfully or failed to server.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents
    OnActionComplete="@OnActionCompleteHandler"></UploaderEvents>
  </SfUploader>
@code {
  private void OnActionCompleteHandler(ActionCompleteEventArgs args)
  {
    // Here you can customize your code
  }
}
```

### OnCancel

**OnCancel** event fires if cancel the chunk file uploading.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents OnCancel="@OnCancelHandler"></UploaderEvents>
</SfUploader>
@code {
  private void OnCancelHandler(CancelEventArgs args)
  {
    // Here you can customize your code
  }
}
```

### OnChunkFailure

**OnChunkFailure** event fires if the chunk file failed to upload.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents OnChunkFailure="@OnChunkFailureHandler"></UploaderEvents>
</SfUploader>
@code {
  private void OnChunkFailureHandler(FailureEventArgs args)
  {
    // Here you can customize your code
  }
}
```

### OnChunkFailed

**OnChunkFailed** event fires if the chunk file failed to upload.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents OnChunkFailed="@OnChunkFailedHandler"></UploaderEvents>
</SfUploader>
@code {
private void OnChunkFailedHandler(FailureEventArgs args)
{
  // Here you can customize your code
}
}
```

### OnChunkSuccess

**OnChunkSuccess** event fires when the chunk file uploaded successfully.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents OnChunkSuccess="@OnChunkSuccessHandler"></UploaderEvents>
</SfUploader>
@code {
private void OnChunkSuccessHandler(SuccessEventArgs args)
{
  // Here you can customize your code
}
}
```

### OnChunkUploadStart

**OnChunkUploadStart** event fires when every chunk upload process gets started. This event is used to add additional parameter with upload request.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents
    OnChunkUploadStart="@OnChunkUploadStartHandler"></UploaderEvents>
</SfUploader>
@code {
private void OnChunkUploadStartHandler(UploadingEventArgs args)
{
  // Here you can customize your code
}
}
```

### OnClear

**OnClear** event triggers before clearing the items in file list when clicking "clear".

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents OnClear="@OnClearHandler"></UploaderEvents>
</SfUploader>
@code {
private void OnClearHandler(ClearingEventArgs args)
{
  // Here you can customize your code
}
}
```

### OnFailure

**OnFailure** event triggers when the AJAX request fails on uploading or removing files.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents OnFailure="@OnFailureHandler"></UploaderEvents>
</SfUploader>
@code {
private void OnFailureHandler(FailureEventArgs args)
{
  // Here you can customize your code
}
}
```

### OnFailed

**OnFailed** event triggers when the AJAX request fails on uploading or removing files.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents OnFailed="@OnFailedHandler"></UploaderEvents>
</SfUploader>
@code {
private void OnFailedHandler(FailureEventArgs args)
{
  // Here you can customize your code
}
}
```

### OnFileListRender

**OnFileListRender** event triggers before rendering each file item from the file list in a page. It helps to customize specific file item structure.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents
    OnFileListRender="@OnFileListRenderHandler"></UploaderEvents>
</SfUploader>
@code {
```



```
private void OnFileListRenderHandler(FileListRenderingEventArgs args)
{
    // Here you can customize your code
}
```

### OnRemove

**OnRemove** event triggers on removing the uploaded file. The event used to get confirm before removing the file from server.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
<UploaderEvents OnRemove="@OnRemoveHandler"></UploaderEvents>
</SfUploader>
@code {
private void OnRemoveHandler(RemovingEventArgs args)
{
    // Here you can customize your code
}
```

### OnResume

**OnResume** event fires if resume the paused chunk file upload.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
<UploaderEvents OnResume="@OnResumeHandler"></UploaderEvents>
</SfUploader>
@code {
private void OnResumeHandler(PauseResumeEventArgs args)
{
    // Here you can customize your code
}
```

### OnUploadStart

**OnUploadStart** event triggers when the upload process gets started. This event is used to add additional parameter with upload request.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
<UploaderEvents OnUploadStart="@OnUploadStartHandler"></UploaderEvents>
</SfUploader>
@code {
private void OnUploadStartHandler(UploadingEventArgs args)
{
    // Here you can customize your code
}
```

```
}
```

### Paused

**Paused** event fires if pause the chunk file uploading.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents Paused="@PausedHandler"></UploaderEvents>
</SfUploader>
@code {
private void PausedHandler(PauseResumeEventArgs args)
{
  // Here you can customize your code
}
}
```

### Progressing

**Progressing** event triggers when uploading a file to the server using the AJAX request.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents Progressing="@ProgressingHandler"></UploaderEvents>
</SfUploader>
@code {
private void ProgressingHandler(Syncfusion.Blazor.Inputs.ProgressEventArgs args)
{
  // Here you can customize your code
}
}
```

### Success

**Success** event triggers when the AJAX request gets success on uploading files or removing files.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
  <UploaderEvents Success="@SuccessHandler"></UploaderEvents>
</SfUploader>
@code {
private void SuccessHandler(SuccessEventArgs args)
{
  // Here you can customize your code
}
}
```

### ValueChange

**ValueChange** event triggers when changes occur in uploaded file list by selecting or dropping files.

**ASPX-CS**

```
@using Syncfusion.Blazor.Inputs
<SfUploader>
<UploaderEvents ValueChange="@ValueChangeHandler"></UploaderEvents>
</SfUploader>
@code {
private void ValueChangeHandler(UploadChangeEventArgs args)
{
// Here you can customize your code
}
}
```

File Upload 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****Blazor File Upload Component in WebAssembly App using Visual Studio**

This article provides a step-by-step instructions to configure Syncfusion Blazor File Upload 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.

**ASPX-CS**

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Inputs
```

5. 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();
}
}
}
```

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();
}
}
}
```

### *Adding uploader component to the application*

To initialize the uploader component add the below code to your Index.razor view page which is present under ~/Pages folder.

### ASPX-CS

```
<SfUploader></SfUploader>
```

### Run the application

After successful compilation of your application, press **F5** to run the application.

The output will be as follows.



### Without server-side API endpoint

You can upload the files and files of folders in the Blazor application without specifying the server-side API endpoint using [AsyncSettings](#).

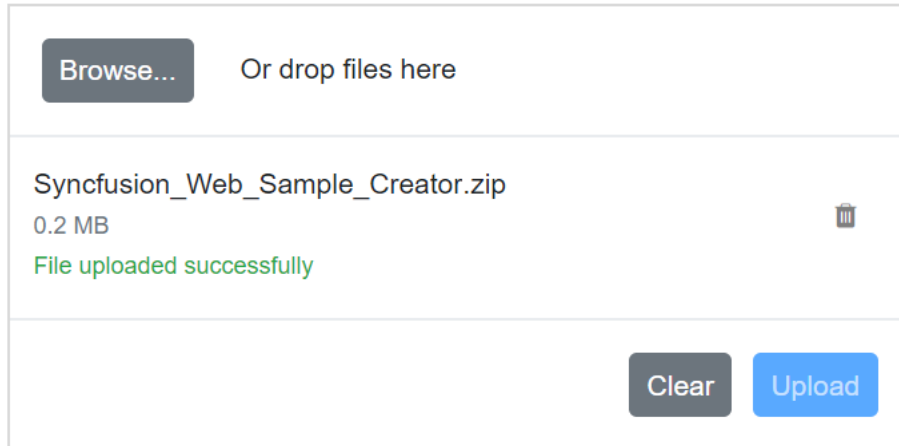
### Save and Remove actions

You can get the uploaded files as file stream in the [ValueChange](#) event argument. Now, you can write the save handler inside ValueChange event to save the files to desired location. Please find the save action code on below.

### ASPX-CS

```
<SfUploader AutoUpload="false">
  <UploaderEvents ValueChange="OnChange"></UploaderEvents>
</SfUploader>
@code {
    private void OnChange(UploadChangeEventArgs args)
    {
        foreach (var file in args.Files)
        {
            var path = @"path" + file.FileInfo.Name;
            FileStream filestream = new FileStream(path, FileMode.Create,
            FileAccess.Write);
            file.Stream.WriteTo(filestream);
            filestream.Close();
            file.Stream.Close();
        }
    }
}
```

The output will be as follows.



While clicking on the remove icon in the file list, you will get the [OnRemove](#) event with removing file name as argument. So, you can write the remove handler inside OnRemove event to remove the particular file from desired location. Please find the remove action code on below.

### CSHARP

```
Private void onRemove(RemovingEventArgs args)
{
    foreach(var removeFile in args.FilesData)
    {
        if (File.Exists(Path.Combine(@"rootPath", removeFile.Name)))
        {
            File.Delete(Path.Combine(@"rootPath", removeFile.Name))
        }
    }
}
```

#### *With server-side API endpoint*

The upload process requires save and remove action URL to manage the upload process in the server.

\* The save action is necessary to handle the upload operation.

\* The remove action is optional, one can handle the removed files from server.

The save action handler upload the files that needs to be specified in the [SaveUrl](#) property.

The save handler receives the submitted files and manages the save process in server. After uploading the files to server location, the color of the selected file name changes to green and the remove icon is changed as bin icon.

The remove action is optional. The remove action handler removes the files that needs to be specified in the [RemoveUrl](#) property.

### CSHARP

```
[Route("api/[controller]")]
private IHostingEnvironment hostingEnv;
public SampleDataController(IHostingEnvironment env)
{
    this.hostingEnv = env;
}
```

```
[HttpPost("[action]")]
public void Save(IList<IFormFile> UploadFiles)
{
    long size = 0;
    try
    {
        foreach (var file in UploadFiles)
        {
            var filename = ContentDispositionHeaderValue
                .Parse(file.ContentDisposition)
                .FileName
                .Trim('"');
            filename = hostingEnv.ContentRootPath + $"{@"\{filename}"}";
            size += (int)file.Length;
            if (!System.IO.File.Exists(filename))
            {
                using (FileStream fs = System.IO.File.Create(filename))
                {
                    file.CopyTo(fs);
                    fs.Flush();
                }
            }
        }
    }
    catch (Exception e)
    {
        Response.Clear();
        Response.StatusCode = 204;
        Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
            "File failed to upload";
        Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
            e.Message;
    }
}

[HttpPost("[action]")]
public void Remove(IList<IFormFile> UploadFiles)
{
    try
    {
        var filename = hostingEnv.ContentRootPath + $"{@"\{UploadFiles[0].FileName}"}";
        if (System.IO.File.Exists(filename))
        {
            System.IO.File.Delete(filename);
        }
    }
    catch (Exception e)
    {
        Response.Clear();
        Response.StatusCode = 200;
        Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
            "File removed successfully";
        Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
            e.Message;
    }
}
```

### ASPX-CS

```
<SfUploader ID="UploadFiles">
  <UploaderAsyncSettings SaveUrl="api/SampleData/Save"
    RemoveUrl="api/SampleData/Remove"></UploaderAsyncSettings>
</SfUploader>
```

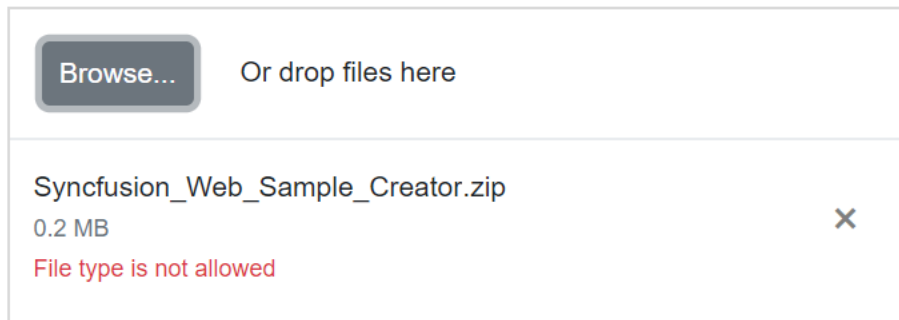
#### Configure allowed file types

You can allow the specific files alone to upload using the [AllowedExtensions](#) property. The extension can be represented as collection by comma separators. The uploader component filters the selected or dropped files to match against the specified file types and processes the upload operation. The validation happens when you specify value to inline attribute to accept the original input element.

### ASPX-CS

```
<SfUploader AllowedExtensions=".doc, docx, .xls, xls"></SfUploader>
```

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](#)

#### Adding html attributes in Blazor File Upload Component

You can add the additional HTML attributes such as disabled, value, name, and more to the element using the [HtmlAttributes](#) property. If you configure both the property and equivalent HTML attribute, then the component will consider the property value.

The following example demonstrates how to set attributes in the HtmlAttributes property in the Uploader.

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfUploader HtmlAttributes="@htmlattribute">
  <UploaderAsyncSettings
    SaveUrl="https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save"
    RemoveUrl="https://aspnetmvc.syncfusion.com/services/api/uploadbox/Remove"></UploaderAsyncSettings>
</SfUploader>
@code{
```



```
Dictionary<string, object> htmlattribute = new Dictionary<string, object>()
{
    { "name", "file-uploader" },
    { "disabled", "true" }
};
}
```

## Gantt Chart

### Getting Started with Blazor Gantt Chart Component

This section briefly explains about how to include a [Blazor Gantt Chart](#) Component in your Blazor Server-Side and Client-Side application. You can refer to our Getting Started with [Blazor Server-Side](#) and [Blazor WebAssembly](#) documentation pages for configuration specifications.

#### Importing Syncfusion Blazor component in the application

1. Install **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](#) packages in the **HEAD** element of the `~/Pages/_Host.cshtml` page.

#### HTML

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

For Internet Explorer 11 kindly refer the poly fills. 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.Gantt** packages.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
```

#### 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 **ConfigureServices** function as follows.

## CSHARP

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

Adding Gantt Chart component to the application

Now, add the Syncfusion [Blazor Gantt Chart component](#) in any web page (razor) in the **Pages** folder. For example, the Gantt Chart component is added in the **~/Pages/Index.razor** page.

## ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt TValue="TaskData">
</SfGantt>
@code{
    public class TaskData
    {
        public int TaskId { get; set; }
        public string TaskName { get; set; }
        public DateTime StartDate { get; set; }
        public DateTime EndDate { get; set; }
        public string Duration { get; set; }
        public int Progress { get; set; }
        public List<TaskData> SubTasks { get; set; }
    }
}
```

Binding Gantt Chart with Data

Bind data with the Gantt Chart component by using the **DataSource** property. It accepts an list objects or the **DataManager** instance.

## ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
</SfGantt>
@code{
    public List<TaskData> TaskCollection { get; set; }
    protected override void OnInitialized()
    {
        this.TaskCollection = GetTaskCollection();
    }
}
```

```
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}

public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 50,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 50,
                }
            })
        }
    };
    return Tasks;
}
```

### Mapping Task Fields

The data source fields that are required to render the tasks are mapped to the Gantt Chart component using the `GanttTaskFields` property.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
    <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
    EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
    </GanttTaskFields>
</SfGantt>
@code{
    public List<TaskData> TaskCollection { get; set; }
    protected override void OnInitialized()
    {
        this.TaskCollection = GetTaskCollection();
    }
}
```

```
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}

public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 50,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 50,
                }
            })
        }
    };
    return Tasks;
}
```

### Defining Columns

Gantt Chart has an option to define columns as an array. You can customize the Gantt Chart columns using the following properties:

- **Field**: Maps the data source fields to the columns.
- **HeaderText**: Changes the title of columns.
- **TextAlign**: Changes the alignment of columns. By default, columns will be left aligned. To change the columns to right align, set **TextAlign** to right.
- **Format**: Formats the number and date values to standard or custom formats. Here, it is defined for the conversion of numeric values to currency.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" HeaderText="Task ID" TextAlign="TextAlign.Right"
Width="100"></GanttColumn>
<GanttColumn Field="TaskName" HeaderText="Task Name"
Width="250"></GanttColumn>
<GanttColumn Field="StartDate" HeaderText="Start Date"
Width="250"></GanttColumn>
<GanttColumn Field="Duration" HeaderText="Duration"
Width="250"></GanttColumn>
<GanttColumn Field="Progress" HeaderText="Progress" Format="@NumberFormat"
Width="250"></GanttColumn>
</GanttColumns>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
public string NumberFormat = "C";
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50,
}
}
}
}
}

```

```

    }
  })
}
};
return Tasks;
}
}

```

For further details regarding Columns, Please refer [here](#)

### Enable Editing

The editing feature enables you to edit the tasks in the Gantt Chart component. It can be enabled by using the `EditSettings.AllowEditing` and `EditSettings.AllowTaskbarEditing` properties.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttEditSettings AllowEditing="true"
  Mode="Syncfusion.Blazor.Gantt.EditMode.Auto"
  AllowTaskbarEditing="true"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50,
},

```

```
new TaskData() {
    TaskId = 3,
    TaskName = "Perform soil test",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "4",
    Progress = 50,
}
}))
}
};
return Tasks;
}
}
```

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

You can find the full information regarding Editing from [here](#)

### Enable Filtering

The filtering feature enables you to view the reduced amount of records based on filter criteria. Gantt Chart provides the menu filtering support for each column. It can be enabled by setting the **AllowFiltering** property to **true**. Filtering feature can also be customized using the **FilterSettings** property.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowFiltering="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
```

```

TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 2,
        TaskName = "Identify Site location",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "4",
        Progress = 50,
    },
    new TaskData() {
        TaskId = 3,
        TaskName = "Perform soil test",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "4",
        Progress = 50,
    }
})
};
return Tasks;
}
}

```

You can find the full information regarding Filtering from [here](#)

### Enable Sorting

The sorting feature enables you to order the records. It can be enabled by setting the `AllowSorting` property to `true`. The sorting feature can be customized using the `SortSettings` property.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowSorting="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
}

```



```
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 50,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 50,
                }
            })
        }
    };
    return Tasks;
}
```

You can find the full information regarding Sorting from [here](#)

### Enabling Predecessors or Task Relationships

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

- Start to Start (SS): You cannot start a task until the dependent task starts.
- Start to Finish (SF): You cannot finish a task until the dependent task finishes.
- Finish to Start (FS): You cannot start a task until the dependent task completes.
- Finish to Finish (FF): You cannot finish a task until the dependent task completes.

You can show the relationship in tasks by using the **Dependency** property as shown in the following code example.

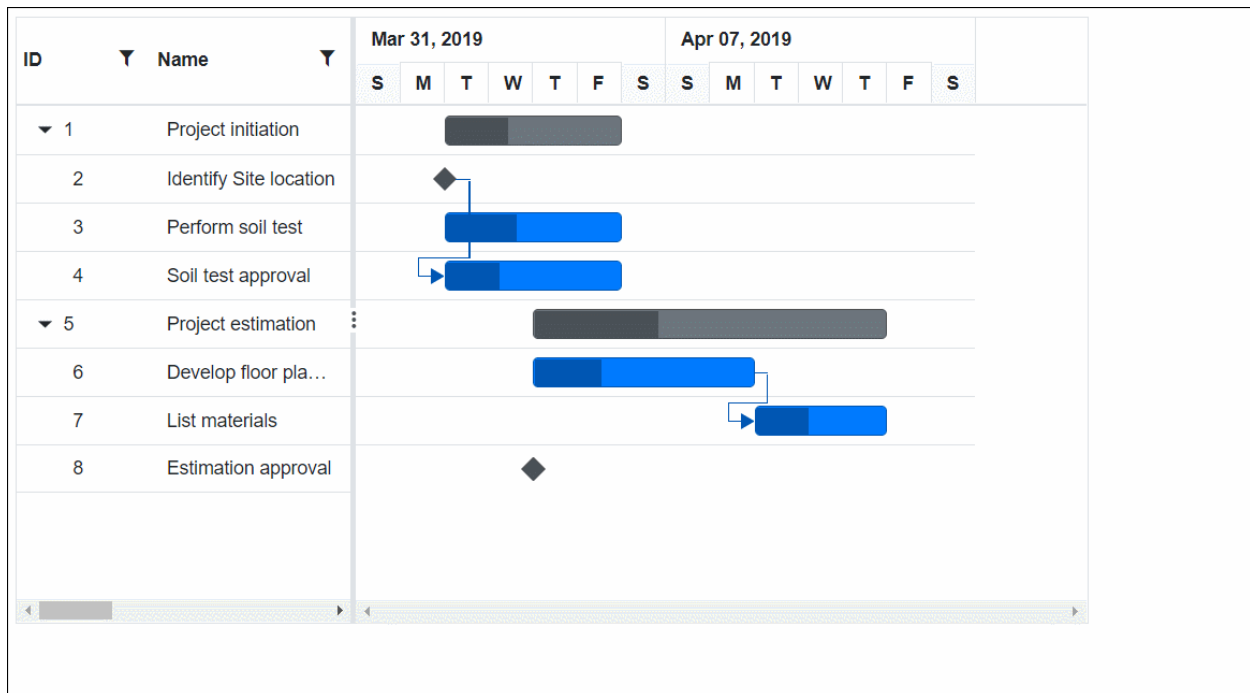
### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
    <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
    EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks"
    Dependency="Predecessor">
    </GanttTaskFields>
</SfGantt>
```

```
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public string Predecessor { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50,
Predecessor = "2"
}
})
}
};
return Tasks;
}
}
```

You can find the full information regarding Predecessors from [here](#)

The following image represents Gantt with Editing, Sorting, Filtering and Predecessors.



You can also explore our [Blazor Gantt Chart example](#) that shows 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 2019](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

### Data Binding in Blazor Gantt Chart Component

The [Blazor Gantt Chart](#) 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 gantt component.

### List Binding

To bind list binding to the gantt component, 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.

### *Hierarchical data Binding*

The **Child** property is used to map the child records in hierarchical data. The following code example shows how to bind the hierarchical list data into the Gantt Chart component.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
  List<TaskData> Tasks = new List<TaskData>() {
    new TaskData() {
      TaskId = 1,
      TaskName = "Project initiation",
      StartDate = new DateTime(2019, 04, 02),
      EndDate = new DateTime(2019, 04, 21),
      SubTasks = (new List<TaskData> () {
        new TaskData() {
          TaskId = 2,
          TaskName = "Identify Site location",
          StartDate = new DateTime(2019, 04, 02),
          Duration = "0",
          Progress = 30,
        },
        new TaskData() {
          TaskId = 3,
          TaskName = "Perform soil test",
          StartDate = new DateTime(2019, 04, 02),
          Duration = "4",
          Progress = 40,
        },
        new TaskData() {
          TaskId = 4,
          TaskName = "Soil test approval",
          StartDate = new DateTime(2019, 04, 02),
          Duration = "0",
          Progress = 30
        }
      })
    }
  }
}
```

```
},
}))
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
});
return Tasks;
}
```

---

\* Indent/Outdent is not supported for Hierarchy Data.

\* ExpandCollapse State maintenance is not supported for Hierarchy Data.

\* Row Drag and Drop feature is not supported for Hierarchy Data.

---

#### *Self-Referential / Flat Data Binding*

The Gantt Chart component can be bound with self-referential data by mapping the data source field values to the **Id** and **ParentID** properties.

- ID field: This field contains unique values used to identify each individual task and it is mapped to the **Id** property.
- Parent ID field: This field contains values that indicate parent tasks and it is mapped to the **ParentID** property.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Gantt
```

```
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress"
  ParentID="ParentId">
</GanttTaskFields>
</SfGantt>
```

```
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
ParentId = 1
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
```

```

StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
ParentId = 5
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
ParentId = 5
}
};
return Tasks;
}
}

```

### DynamicObject binding

Gantt Chart 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 bind data to the gantt chart as list of **DynamicObject**.

**DynamicObject** can be bound to gantt chart by assigning to the [DataSource](#) property. Gantt Chart 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 render and perform data operations, editing etc., while using DynamicObject.

---

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using System.Dynamic
<SfGantt DataSource="@GanttDynamicData" Height="500px" Width="100%"
HighlightWeekends="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
Progress="Progress" Duration="Duration"
ParentID="ParentId"></GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" HeaderText="Task ID"
TextAlign="Syncfusion.Blazor.Grids.TextAlign.Right"
Width="100"></GanttColumn>

```

```

<GanttColumn Field="TaskName" HeaderText="Task Name"
Width="250"></GanttColumn>
<GanttColumn Field="StartDate" HeaderText="Start Date"
Width="250"></GanttColumn>
<GanttColumn Field="Duration" HeaderText="Duration"
Width="250"></GanttColumn>
<GanttColumn Field="Progress" HeaderText="Progress" Format="@NumberFormat"
Width="250"></GanttColumn>
</GanttColumns>
<GanttEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" AllowTaskbarEditing="true"
ShowDeleteConfirmDialog="true"></GanttEditSettings>
</SfGantt>
@code {
SfGantt<DynamicDictionary> GanttChart;
public string NumberFormat = "C";
public static List<DynamicDictionary> Data = new List<DynamicDictionary>();
public List<DynamicDictionary> GanttDynamicData { get; set; }
public static int ParentRecordID { get; set; }
public static int ChildRecordID { get; set; }
protected override void OnInitialized()
{
this.GanttDynamicData = GetData().ToList();
}
public static List<DynamicDictionary> GetData()
{
Data.Clear();
ParentRecordID = 0;
ChildRecordID = 0;
for (var i = 1; i <= 10; i++)
{
Random ran = new Random();
DateTime start = new DateTime(2021, 01, 07);
int range = (DateTime.Today - start).Days;
DateTime startingDate = start.AddDays(ran.Next(range));
dynamic ParentRecord = new DynamicDictionary();
ParentRecord.TaskId = ++ParentRecordID;
ParentRecord.TaskName = "Parent Task " + i;
ParentRecord.StartDate = startingDate;
ParentRecord.Progress = ran.Next(10, 100);
ParentRecord.Duration = ParentRecordID % 2 == 0 ? (32).ToString() :
(76).ToString();
ParentRecord.ParentId = null;
Data.Add(ParentRecord);
AddChildRecords(ParentRecordID);
}
return Data;
}
public static void AddChildRecords(int ParentId)
{
for (var i = 1; i < 4; i++)
{
Random ran = new Random();
DateTime start = new DateTime(2021, 01, 07);
int range = (DateTime.Today - start).Days;
DateTime startingDate = start.AddDays(ran.Next(range));
dynamic ChildRecord = new DynamicDictionary();

```



```

ChildRecord.TaskId = ++ParentRecordID;
ChildRecord.TaskName = "Child Task " + ++ChildRecordID;
ChildRecord.StartDate = startingDate;
ChildRecord.Progress = ran.Next(10, 100);
ChildRecord.Duration = ParentRecordID % 3 == 0 ? (64).ToString() :
(98).ToString();
ChildRecord.ParentId = ParentId;
Data.Add(ChildRecord);
}
}
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;
    }
}
}

```

### ExpandoObject Binding

Gantt 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 Gantt as list of ExpandoObject.

ExpandoObject can be bound to Gantt by assigning to the **DataSource** property. Gantt can also perform all kind of supported data operations and editing in ExpandoObject.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt TValue="ExpandoObject" DataSource="@TreeData" @ref="Gantt"
Height="450px" Width="700px">
    <GanttTaskFields Id="TaskID" Name="TaskName" StartDate="StartDate"
Duration="Duration"
Progress="Progress" ParentID="ParentID">
    </GanttTaskFields>
    <GanttEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" AllowTaskbarEditing="true"></GanttEditSettings>
</SfGantt>
@code {
    SfGantt<ExpandoObject> Gantt;
    public List<ExpandoObject> TreeData { get; set; }
    protected override void OnInitialized()
    {

```

```

this.TreeData = GetData().ToList();
}
public static List<ExpandoObject> Data = new List<ExpandoObject>();
public static int ParentRecordID { get; set; }
public static int ChildRecordID { get; set; }
public static List<ExpandoObject> GetData()
{
    Data.Clear();
    ParentRecordID = 0;
    ChildRecordID = 0;
    for (var i = 1; i <= 60; i++)
    {
        Random ran = new Random();
        DateTime start = new DateTime(2020, 06, 07);
        int range = (DateTime.Today - start).Days;
        DateTime startingDate = start.AddDays(ran.Next(range));
        dynamic ParentRecord = new ExpandoObject();
        ParentRecord.TaskID = ++ParentRecordID;
        ParentRecord.TaskName = "Parent Task " + i;
        ParentRecord.StartDate = startingDate;
        ParentRecord.Progress = ran.Next(10, 100);
        ParentRecord.Duration = ParentRecordID % 2 == 0 ? (32).ToString() :
        (76).ToString();
        ParentRecord.ParentID = null;
        Data.Add(ParentRecord);
        AddChildRecords(ParentRecordID);
    }
    return Data;
}
public static void AddChildRecords(int ParentId)
{
    for (var i = 1; i < 4; i++)
    {
        Random ran = new Random();
        DateTime start = new DateTime(2020, 06, 07);
        int range = (DateTime.Today - start).Days;
        DateTime startingDate = start.AddDays(ran.Next(range));
        dynamic ChildRecord = new ExpandoObject();
        ChildRecord.TaskID = ++ParentRecordID;
        ChildRecord.TaskName = "Child Task " + ++ChildRecordID;
        ChildRecord.StartDate = startingDate;
        ChildRecord.Progress = ran.Next(10, 100);
        ChildRecord.Duration = ParentRecordID % 3 == 0 ? (64).ToString() :
        (98).ToString();
        ChildRecord.ParentID = ParentId;
        Data.Add(ChildRecord);
    }
}
public class ExpandoObject
{
    public int TaskID { get; set; }
    public string TaskName { get; set; }
    public DateTime? StartDate { get; set; }
    public DateTime? EndDate { get; set; }
    public int Progress { get; set; }
    public string Duration { get; set; }
    public int? ParentID { get; set; }
}

```

```
}
}
```

Here, we have provided list of reserved properties and the purpose used in Gantt Chart. We recommend to avoid these reserved properties for Internal purpose(To get rid of conflicts).

Reserved keywords | Purpose

ganttProperties | Specifies the task data details

TaskMode | Specifies the mode of task

childRecords | Specifies the childRecords of a parentData

hasChildRecords | Specifies whether the record contains child records

expanded | Specifies whether the child records are expanded

parentRecord | Specifies the parentItem of childRecords

index | Specifies the index of current record

level | Specifies the hierarchy level of record

uniqueID | Specifies the unique ID of a record

parentUniqueID | Specifies the parent Unique ID of a record

checkboxState | Specifies the checkbox state of a record

### Remote Data

To bind remote data to Gantt 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 Gantt component.

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

### Web API

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

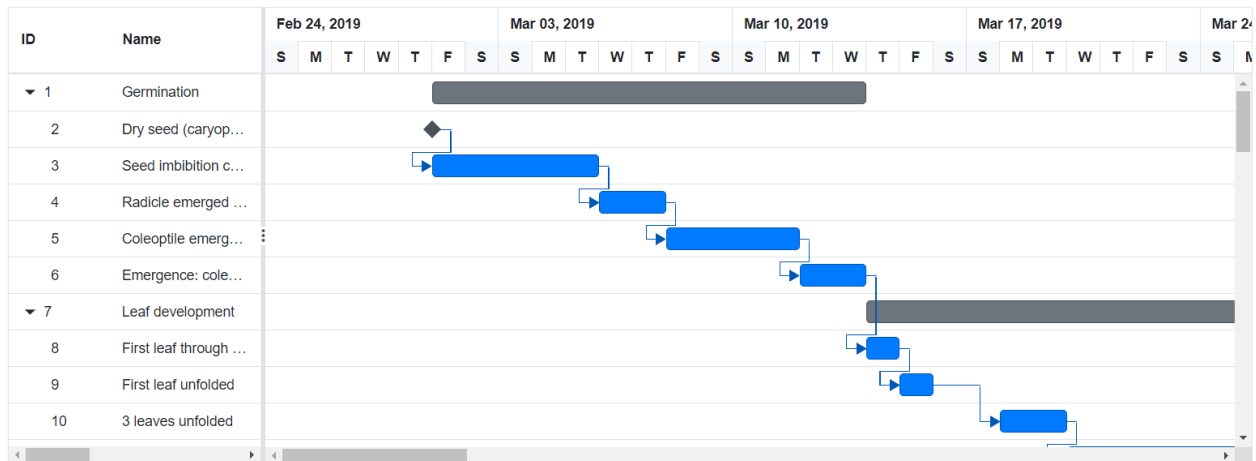
### ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Data
<SfGantt TValue="GanttRemoteData" Height="450px">
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/GanttData" Adaptor="Adaptors.WebApiAdaptor"
CrossDomain="true"></SfDataManager>
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
Duration="Duration" Progress="Progress" Dependency="Predecessor"
Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public class GanttRemoteData
```

```

{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public string Predecessor { get; set; }
    public List<GanttRemoteData> SubTasks { get; set; }
}

```



### *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.Gantt
<SfGantt TValue="TaskData" Height="450px" Width="700px" Query=@GanttQuery>
<SfDataManager Url="/Home/UrlDatasource"
Adaptor="Adaptors.UrlAdaptor"></SfDataManager>
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration"
Progress="Progress" ParentID="ParentId">
</GanttTaskFields>
<GanttEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" AllowTaskbarEditing="true"></GanttEditSettings>
</SfGantt>
@code{
    public string ParamValue = "true";
    public Query GanttQuery { get; set; }
    protected override void OnInitialized() {
        GanttQuery = new Query().AddParams("ej2gantt", ParamValue);
    }
    public class TaskData
    {

```

```
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime? StartDate { get; set; }
public DateTime? EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
}
```

### 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.Gantt
<SfGantt TValue="TaskData" Height="450px" Width="700px">
  <SfDataManager Url="https://some.com/invalidUrl"
  Adaptor="Adaptors.UrlAdaptor"></SfDataManager>
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration"
  Progress="Progress" ParentID="ParentId">
  </GanttTaskFields>
  <GanttEvents TValue="TaskData"
  OnActionFailure="ActionFailure"></GanttEvents>
</SfGantt>
<style>
.error {
color: red;
}
</style>
@code{
public string ErrorDetails = "";
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime? StartDate { get; set; }
public DateTime? EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public void ActionFailure(FailureEventArgs args)
{
this.ErrorDetails = "Server exception: 404 Not found";
StateHasChanged();
}
}
```

## Custom Binding in Blazor Gantt Chart 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 Gantt component.

For implementing custom data binding in Gantt, 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 Gantt 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.Gantt;
@using Syncfusion.Blazor.Data;
@using Syncfusion.Blazor;
<SfGantt TValue="TaskData" Height="450px" Width="1000px">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor)"
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GanttTaskFields Id="TaskID" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Progress="Progress" Duration="Duration"
ParentID="ParentID">
</GanttTaskFields>
</SfGantt>
@code{
public static List<TaskData> GanttData { get; set; }
public static List<TaskData> gantt = new List<TaskData>();
public class TaskData
{
public int? TaskID { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public int Progress { get; set; }
public string Duration { get; set; }
public int? ParentID { get; set; }
public TaskData() { }
}
public static List<TaskData> GetGantt()
{
if (gantt.Count == 0)
{
int root = -1;
for (var t = 1; t <= 8; t++)
{
Random ran = new Random();

```

```

string math = (ran.Next() % 3) == 0 ? "High" : (ran.Next() % 2) == 0 ?
"Release Breaker" : "Critical";
root++;
int rootItem = gantt.Count + root + 1;
gant.Add(new TaskData() { TaskID = rootItem, TaskName = "Parent Task " +
rootItem.ToString(), StartDate = new DateTime(1992, 06, 07), EndDate = new
DateTime(1994, 08, 25), Progress = (ran.Next(10, 100)), ParentID = null,
Duration = (ran.Next(1, 50)).ToString() });
int parent = gantt.Count;
for (var c = 0; c < 3; c++)
{
    root++;
    string val = ((parent + c + 1) % 3 == 0) ? "Low" : "Critical";
    int parn = parent + c + 1;
    int iD = gantt.Count + root + 1;
    gant.Add(new TaskData() { TaskID = iD, TaskName = "Child Task " +
iD.ToString(), StartDate = new DateTime(1992, 06, 07), EndDate = new
DateTime(1994, 08, 25), Progress = (ran.Next(10, 100)), ParentID = rootItem,
Duration = (ran.Next(1, 50)).ToString() });
    if (((parent + c + 1) % 3) == 0)
    {
        int immParent = gantt.Count;
        for (var s = 0; s <= 1; s++)
        {
            root++;
            gant.Add(new TaskData() { TaskID = gantt.Count + root + 1, TaskName = "Sub
Task " + (gant.Count + root + 1).ToString(), StartDate = new DateTime(1992,
06, 07), EndDate = new DateTime(1994, 08, 25), Progress = (ran.Next(10,
100)), ParentID = iD, Duration = (ran.Next(1, 50)).ToString() });
        }
    }
}
return gantt;
}
// 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<TaskData> DataSource = GanttData;
        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<TaskData>().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;
}
}
protected override void OnInitialized()
{
GanttData = GetGantt().ToList();
}
}

```

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.

<br /> 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<TaskDataAccessLayer>();
services.AddScoped<CustomAdaptor>();
services.AddScoped<ServiceClass>();
}

```

The following sample code demonstrates injecting service into Custom Adaptor,

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt;
@using Syncfusion.Blazor.Data;
@using Syncfusion.Blazor;
<SfGantt TValue="TaskData" Height="450px" Width="1000px">

```

```

<SfDataManager AdaptorInstance="@typeof(CustomAdaptor)"
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GanttTaskFields Id="TaskID" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Progress="Progress" Duration="Duration"
ParentID="ParentID">
</GanttTaskFields>
</SfGantt>
@code{
public static List<TaskData> GanttData { get; set; }
public static List<TaskData> gantt = new List<TaskData>();
public class TaskData
{
public int? TaskID { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public int Progress { get; set; }
public string Duration { get; set; }
public int? ParentID { get; set; }
public TaskData() { }
}
public static List<TaskData> GetGantt()
{
if (gantt.Count == 0)
{
int root = -1;
for (var t = 1; t <= 8; t++)
{
Random ran = new Random();
string math = (ran.Next() % 3) == 0 ? "High" : (ran.Next() % 2) == 0 ?
"Release Breaker" : "Critical";
root++;
int rootItem = gantt.Count + root + 1;
gantt.Add(new TaskData() { TaskID = rootItem, TaskName = "Parent Task " +
rootItem.ToString(), StartDate = new DateTime(1992, 06, 07), EndDate = new
DateTime(1994, 08, 25), Progress = (ran.Next(10, 100)), ParentID = null,
Duration = (ran.Next(1, 50)).ToString() });
int parent = gantt.Count;
for (var c = 0; c < 3; c++)
{
root++;
string val = ((parent + c + 1) % 3 == 0) ? "Low" : "Critical";
int parn = parent + c + 1;
int iD = gantt.Count + root + 1;
gantt.Add(new TaskData() { TaskID = iD, TaskName = "Child Task " +
iD.ToString(), StartDate = new DateTime(1992, 06, 07), EndDate = new
DateTime(1994, 08, 25), Progress = (ran.Next(10, 100)), ParentID = rootItem,
Duration = (ran.Next(1, 50)).ToString() });
if (((parent + c + 1) % 3) == 0)
{
int immParent = gantt.Count;
for (var s = 0; s <= 1; s++)
{
root++;
gantt.Add(new TaskData() { TaskID = gantt.Count + root + 1, TaskName = "Sub
Task " + (gantt.Count + root + 1).ToString(), StartDate = new DateTime(1992,

```

```
06, 07), EndDate = new DateTime(1994, 08, 25), Progress = (ran.Next(10,
100)), ParentID = iD, Duration = (ran.Next(1, 50)).ToString()});
}
}
}
}
}
return gantt;
}
// Implementing custom adaptor by extending the DataAdaptor class
public class CustomAdaptor : DataAdaptor
{
//here you can inject your service
[Inject]
public TaskDataAccessLayer context { get; set; } = new
TaskDataAccessLayer();
// Performs data Read operation
public override object Read(DataManagerRequest dm, string key = null)
{
IEnumerable<TaskData> DataSource = GanttData;
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<TaskData>().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;
}
}
protected override void OnInitialized()
{
GanttData = GetGantt().ToList();
}
}
```

### CRUD operation

The CRUD operations for the custom bounded data in the Gantt 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 Gantt, use BatchUpdate/BatchUpdateAsync method to handle the corresponding CRUD operation

The following sample code demonstrates implementing CRUD operations for the custom bounded data,

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt;
@using Syncfusion.Blazor.Data;
@using Syncfusion.Blazor;
<SfGantt TValue="TaskData" Height="450px" Width="1000px"
AllowFiltering="true" AllowSorting="true" Toolbar="@ (new List<string>() {
"Add", "Edit", "Delete", "Update", "Cancel", "Search" })">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor)"
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GanttTaskFields Id="TaskID" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Progress="Progress" Duration="Duration"
ParentID="ParentID">
</GanttTaskFields>
<GanttEditSettings AllowEditing="true" AllowAdding="true"
AllowDeleting="true"></GanttEditSettings>
</SfGantt>
@code{
public static List<TaskData> GanttData { get; set; }
public static List<TaskData> gantt = new List<TaskData>();
public class TaskData
{
public int? TaskID { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public int Progress { get; set; }
public string Duration { get; set; }
public int? ParentID { get; set; }
public TaskData() { }
}
protected override void OnInitialized()
{
GanttData = GetGantt().ToList();
}
public static List<TaskData> GetGantt()
{
if (gantt.Count == 0)
{
int root = -1;
for (var t = 1; t <= 8; t++)
```

```

{
    Random ran = new Random();
    string math = (ran.Next() % 3) == 0 ? "High" : (ran.Next() % 2) == 0 ?
    "Release Breaker" : "Critical";
    root++;
    int rootItem = gantt.Count + root + 1;
    gantt.Add(new TaskData() { TaskID = rootItem, TaskName = "Parent Task " +
    rootItem.ToString(), StartDate = new DateTime(1992, 06, 07), EndDate = new
    DateTime(1994, 08, 25), Progress = (ran.Next(10, 100)), ParentID = null,
    Duration = (ran.Next(1, 50)).ToString() });
    int parent = gantt.Count;
    for (var c = 0; c < 3; c++)
    {
        root++;
        string val = ((parent + c + 1) % 3 == 0) ? "Low" : "Critical";
        int parn = parent + c + 1;
        int iD = gantt.Count + root + 1;
        gantt.Add(new TaskData() { TaskID = iD, TaskName = "Child Task " +
        iD.ToString(), StartDate = new DateTime(1992, 06, 07), EndDate = new
        DateTime(1994, 08, 25), Progress = (ran.Next(10, 100)), ParentID = rootItem,
        Duration = (ran.Next(1, 50)).ToString() });
        if (((parent + c + 1) % 3) == 0)
        {
            int immParent = gantt.Count;
            for (var s = 0; s <= 1; s++)
            {
                root++;
                gantt.Add(new TaskData() { TaskID = gantt.Count + root + 1, TaskName = "Sub
                Task " + (gantt.Count + root + 1).ToString(), StartDate = new DateTime(1992,
                06, 07), EndDate = new DateTime(1994, 08, 25), Progress = (ran.Next(10,
                100)), ParentID = iD, Duration = (ran.Next(1, 50)).ToString()});
            }
        }
    }
    return gantt;
}

// 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<TaskData> DataSource = GanttData;
        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<TaskData>().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;
}
public override object Insert(DataManager dm, object value, string key)
{
GanttData.Insert(0, value as TaskData);
return value;
}
// Performs Remove operation
public override object Remove(DataManager dm, object value, string keyField,
string key)
{
foreach (var record in value as List<TaskData>)
{
GanttData.Remove(GanttData.Where(or => or.TaskID ==
record.TaskID).FirstOrDefault());
}
return value;
}
// Performs Update operation
public override object Update(DataManager dm, object value, string keyField,
string key)
{
foreach(var record in value as List<TaskData>)
{
var data = GanttData.Where(or => or.TaskID ==
record.TaskID).FirstOrDefault();
if (data != null)
{
data.TaskID = record.TaskID;
data.TaskName = record.TaskName;
data.StartDate = record.StartDate;
data.EndDate = record.EndDate;
data.Duration = record.Duration;
data.Progress = record.Progress;
}
}
return value;
}
}
```

You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

## Virtualization in Blazor Gantt Chart Component

Gantt Chart allows you to load a large amount of data without performance degradation.

### Row Virtualization

Row virtualization allows you to render rows only in the content viewport on load time. It is an alternative way of paging in which the rows will be loaded while scrolling vertically. To set up the row virtualization, you need to define `EnableVirtualization` as true and content height by `Height` property.

The number of records displayed in the Gantt Chart is determined implicitly by the height of the content area.

### CSHARP

```
namespace GanttComponent.Data
{
    public class GanttData
    {
    }
    public class GanttDataSource
    {
        public int TaskId { get; set; }
        public string TaskName { get; set; }
        public DateTime StartDate { get; set; }
        public DateTime EndDate { get; set; }
        public string Duration { get; set; }
        public string Predecessor { get; set; }
        public int Progress { get; set; }
        public int? ParentID { get; set; }
    }
    public static List<GanttDataSource> VirtualData()
    {
        List<GanttDataSource> list = new List<GanttDataSource>();
        for (var i = 0; i < 2; i++)
        {
            var x = list.Count + 1;
            list.Add(new GanttDataSource()
            {
                TaskId = x,
                TaskName = "Project " + (i + 1)
            });
            for (var j = 0; j < TempData().ToList().Count; j++)
            {
                list.Add(new GanttDataSource()
                {
                    TaskId = TempData()[j].TaskId + x,
                    TaskName = TempData()[j].TaskName,
                    StartDate = TempData()[j].StartDate,
                    Duration = TempData()[j].Duration,
                    Progress = TempData()[j].Progress,
                    ParentID = TempData()[j].ParentID + x
                });
            }
        }
    }
}
```

```
return list;
}
public static List<GanttDataSource> TempData()
{
    List<GanttDataSource> GanttDataSourceCollection = new
    List<GanttDataSource>();
    GanttDataSource Record1 = new GanttDataSource()
    {
        TaskId = 1,
        TaskName = "Product concept",
        StartDate = new DateTime(2019, 04, 02),
        EndDate = new DateTime(2019, 04, 29),
        ParentID = 0
    };
    GanttDataSource Record2 = new GanttDataSource()
    {
        TaskId = 2,
        TaskName = "Defining the product and its usage",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "3",
        Progress = 30,
        ParentID = 1,
    };
    GanttDataSource Record3 = new GanttDataSource()
    {
        TaskId = 3,
        TaskName = "Defining target audience",
        StartDate = new DateTime(2019, 04, 02),
        ParentID = 1,
        Duration = "3"
    };
    GanttDataSource Record4 = new GanttDataSource()
    {
        TaskId = 4,
        TaskName = "Prepare product sketch and notes",
        StartDate = new DateTime(2019, 04, 05),
        Duration = "2",
        ParentID = 1,
        Predecessor = "2",
        Progress = 30
    };
    GanttDataSource Record5 = new GanttDataSource()
    {
        TaskId = 5,
        TaskName = "Concept approval",
        StartDate = new DateTime(2019, 04, 08),
        ParentID = 0,
        Duration = "0",
        Predecessor = "3,4"
    };
    GanttDataSource Record6 = new GanttDataSource()
    {
        TaskId = 6,
        TaskName = "Market research",
        StartDate = new DateTime(2019, 04, 02),
        ParentID = 0,
        EndDate = new DateTime(2019, 04, 21),
```



```
};
GanttDataSource Record7 = new GanttDataSource()
{
    TaskId = 7,
    TaskName = "Demand analysis",
    StartDate = new DateTime(2019, 04, 04),
    EndDate = new DateTime(2019, 04, 21),
    ParentID = 6,
};
GanttDataSource Record8 = new GanttDataSource()
{
    TaskId = 8,
    TaskName = "Customer strength",
    StartDate = new DateTime(2019, 04, 09),
    Duration = "4",
    Predecessor = "5",
    ParentID = 7,
    Progress = 30
};
GanttDataSource Record9 = new GanttDataSource()
{
    TaskId = 9,
    TaskName = "Market opportunity analysis",
    StartDate = new DateTime(2019, 04, 9),
    Duration = "4",
    ParentID = 7,
    Predecessor = "5"
};
GanttDataSource Record10 = new GanttDataSource()
{
    TaskId = 10,
    TaskName = "Competitor analysis",
    StartDate = new DateTime(2019, 04, 15),
    Duration = "4",
    Predecessor = "7, 8",
    ParentID = 6,
    Progress = 30
};
GanttDataSource Record11 = new GanttDataSource()
{
    TaskId = 11,
    TaskName = "Product strength analysis",
    StartDate = new DateTime(2019, 04, 15),
    Duration = "4",
    ParentID = 6,
    Predecessor = "9"
};
GanttDataSource Record12 = new GanttDataSource()
{
    TaskId = 12,
    TaskName = "Research complete",
    StartDate = new DateTime(2019, 04, 18),
    Duration = "0",
    ParentID = 6,
    Predecessor = "10"
};
GanttDataSource Record13 = new GanttDataSource()
```

```
{
    TaskId = 13,
    TaskName = "Product design and development",
    StartDate = new DateTime(2019, 04, 04),
    ParentID = 0,
    EndDate = new DateTime(2019, 04, 21),
};
GanttDataSource Record14 = new GanttDataSource()
{
    TaskId = 14,
    TaskName = "Functionality design",
    StartDate = new DateTime(2019, 04, 19),
    Duration = "3",
    ParentID = 13,
    Progress = 30,
    Predecessor = "12"
};
GanttDataSource Record15 = new GanttDataSource()
{
    TaskId = 15,
    TaskName = "Quality design",
    StartDate = new DateTime(2019, 04, 19),
    Duration = "3",
    ParentID = 13,
    Predecessor = "12"
};
GanttDataSource Record16 = new GanttDataSource()
{
    TaskId = 16,
    TaskName = "Define reliability",
    StartDate = new DateTime(2019, 04, 24),
    Duration = "2",
    Progress = 30,
    ParentID = 13,
    Predecessor = "15"
};
GanttDataSource Record17 = new GanttDataSource()
{
    TaskId = 17,
    TaskName = "Identifying raw materials",
    StartDate = new DateTime(2019, 04, 24),
    Duration = "2",
    ParentID = 13,
    Predecessor = "15"
};
GanttDataSource Record18 = new GanttDataSource()
{
    TaskId = 18,
    TaskName = "Define cost plan",
    StartDate = new DateTime(2019, 04, 04),
    ParentID = 13,
    EndDate = new DateTime(2019, 04, 21),
};
GanttDataSource Record19 = new GanttDataSource()
{
    TaskId = 19,
    TaskName = "Manufacturing cost",
```

```
StartDate = new DateTime(2019, 04, 26),
Duration = "2",
Progress = 30,
ParentID = 18,
Predecessor = "17"
};
GanttDataSource Record20 = new GanttDataSource()
{
    TaskId = 20,
    TaskName = "Selling cost",
    StartDate = new DateTime(2019, 04, 26),
    Duration = "2",
    ParentID = 18,
    Predecessor = "17"
};
GanttDataSource Record21 = new GanttDataSource()
{
    TaskId = 21,
    TaskName = "Development of the final design",
    StartDate = new DateTime(2019, 04, 30),
    ParentID = 13,
    EndDate = new DateTime(2019, 04, 21),
};
GanttDataSource Record22 = new GanttDataSource()
{
    TaskId = 22,
    TaskName = "Defining dimensions and package volume",
    StartDate = new DateTime(2019, 04, 30),
    Duration = "2",
    ParentID = 21,
    Progress = 30,
    Predecessor = "19,20"
};
GanttDataSource Record23 = new GanttDataSource()
{
    TaskId = 23,
    TaskName = "Develop design to meet industry standards",
    StartDate = new DateTime(2019, 05, 02),
    Duration = "2",
    ParentID = 21,
    Predecessor = "22"
};
GanttDataSource Record24 = new GanttDataSource()
{
    TaskId = 24,
    TaskName = "Include all the details",
    StartDate = new DateTime(2019, 05, 06),
    Duration = "3",
    ParentID = 21,
    Predecessor = "23"
};
GanttDataSource Record25 = new GanttDataSource()
{
    TaskId = 25,
    TaskName = "CAD computer-aided design",
    StartDate = new DateTime(2019, 05, 09),
    Duration = "3",
```

```
ParentID = 13,
Progress = 30,
Predecessor = "24"
};
GanttDataSource Record26 = new GanttDataSource()
{
    TaskId = 26,
    TaskName = "CAM computer-aided manufacturing",
    StartDate = new DateTime(2019, 05, 14),
    Duration = "3",
    ParentID = 13,
    Predecessor = "25"
};
GanttDataSource Record27 = new GanttDataSource()
{
    TaskId = 27,
    TaskName = "Design complete",
    StartDate = new DateTime(2019, 05, 16),
    Duration = "0",
    ParentID = 13,
    Predecessor = "26"
};
GanttDataSource Record28 = new GanttDataSource()
{
    TaskId = 28,
    TaskName = "Prototype testing",
    StartDate = new DateTime(2019, 05, 17),
    Duration = "4",
    Progress = 30,
    ParentID = 0,
    Predecessor = "27"
};
GanttDataSource Record29 = new GanttDataSource()
{
    TaskId = 29,
    TaskName = "Include feedback",
    StartDate = new DateTime(2019, 05, 17),
    Duration = "4",
    ParentID = 0,
    Predecessor = "28ss"
};
GanttDataSource Record30 = new GanttDataSource()
{
    TaskId = 30,
    TaskName = "Manufacturing",
    StartDate = new DateTime(2019, 05, 23),
    Duration = "5",
    Progress = 30,
    ParentID = 0,
    Predecessor = "28,29"
};
GanttDataSource Record31 = new GanttDataSource()
{
    TaskId = 31,
    TaskName = "Assembling materials to finsihed goods",
    StartDate = new DateTime(2019, 05, 30),
    Duration = "5",
```

```
ParentID = 0,
Predecessor = "30"
};
GanttDataSource Record32 = new GanttDataSource()
{
    TaskId = 32,
    TaskName = "Feedback and testing",
    StartDate = new DateTime(2019, 05, 04),
    ParentID = 0,
    EndDate = new DateTime(2019, 05, 21),
};
GanttDataSource Record33 = new GanttDataSource()
{
    TaskId = 33,
    TaskName = "Internal testing and feedback",
    StartDate = new DateTime(2019, 06, 06),
    Duration = "3",
    ParentID = 32,
    Progress = 45,
    Predecessor = "31"
};
GanttDataSource Record34 = new GanttDataSource()
{
    TaskId = 34,
    TaskName = "Customer testing and feedback",
    StartDate = new DateTime(2019, 06, 11),
    Duration = "3",
    ParentID = 32,
    Progress = 50,
    Predecessor = "33"
};
GanttDataSource Record35 = new GanttDataSource()
{
    TaskId = 35,
    TaskName = "Final product development",
    StartDate = new DateTime(2019, 06, 06),
    ParentID = 0,
    EndDate = new DateTime(2019, 06, 06),
};
GanttDataSource Record36 = new GanttDataSource()
{
    TaskId = 36,
    TaskName = "Important improvements",
    StartDate = new DateTime(2019, 06, 14),
    Duration = "4",
    Progress = 30,
    ParentID = 35,
    Predecessor = "34"
};
GanttDataSource Record37 = new GanttDataSource()
{
    TaskId = 37,
    TaskName = "Address any unforeseen issues",
    StartDate = new DateTime(2019, 06, 14),
    Duration = "4",
    Progress = 30,
    ParentID = 35,
```

```
Predecessor = "36ss"
};
GanttDataSource Record38 = new GanttDataSource()
{
    TaskId = 38,
    TaskName = "Final product",
    StartDate = new DateTime(2019, 06, 06),
    ParentID = 0,
    EndDate = new DateTime(2019, 06, 06),
};
GanttDataSource Record39 = new GanttDataSource()
{
    TaskId = 39,
    TaskName = "Branding product",
    StartDate = new DateTime(2019, 06, 20),
    Duration = "4",
    ParentID = 38,
    Predecessor = "37"
};
GanttDataSource Record40 = new GanttDataSource()
{
    TaskId = 40,
    TaskName = "Marketing and presales",
    StartDate = new DateTime(2019, 06, 26),
    Duration = "4",
    Progress = 30,
    ParentID = 38,
    Predecessor = "39"
};
GanttDataSourceCollection.Add(Record1);
GanttDataSourceCollection.Add(Record2);
GanttDataSourceCollection.Add(Record3);
GanttDataSourceCollection.Add(Record4);
GanttDataSourceCollection.Add(Record5);
GanttDataSourceCollection.Add(Record6);
GanttDataSourceCollection.Add(Record7);
GanttDataSourceCollection.Add(Record8);
GanttDataSourceCollection.Add(Record9);
GanttDataSourceCollection.Add(Record10);
GanttDataSourceCollection.Add(Record11);
GanttDataSourceCollection.Add(Record12);
GanttDataSourceCollection.Add(Record13);
GanttDataSourceCollection.Add(Record14);
GanttDataSourceCollection.Add(Record15);
GanttDataSourceCollection.Add(Record16);
GanttDataSourceCollection.Add(Record17);
GanttDataSourceCollection.Add(Record18);
GanttDataSourceCollection.Add(Record19);
GanttDataSourceCollection.Add(Record20);
GanttDataSourceCollection.Add(Record21);
GanttDataSourceCollection.Add(Record22);
GanttDataSourceCollection.Add(Record23);
GanttDataSourceCollection.Add(Record24);
GanttDataSourceCollection.Add(Record25);
GanttDataSourceCollection.Add(Record26);
GanttDataSourceCollection.Add(Record27);
GanttDataSourceCollection.Add(Record28);
```

```
GanttDataSourceCollection.Add(Record29);
GanttDataSourceCollection.Add(Record30);
GanttDataSourceCollection.Add(Record31);
GanttDataSourceCollection.Add(Record32);
GanttDataSourceCollection.Add(Record33);
GanttDataSourceCollection.Add(Record34);
GanttDataSourceCollection.Add(Record35);
GanttDataSourceCollection.Add(Record36);
GanttDataSourceCollection.Add(Record37);
GanttDataSourceCollection.Add(Record38);
GanttDataSourceCollection.Add(Record39);
GanttDataSourceCollection.Add(Record40);
return GanttDataSourceCollection;
}
}
}
```

### Limitations for Virtualization

- Due to the element height limitation in browsers, the maximum number of records loaded by the Gantt chart is limited by the browser capability.
- It is necessary to mention the height of the Gantt in pixels when enabling Virtual Scrolling.
- Cell selection will not be persisted in a row.
- Programmatic selection using the **SelectRows** method is not supported in virtual scrolling.
- Currently Editing, Row Drag and Drop are not supported with Virtual scrolling.

### Columns in Blazor Gantt Chart Component

The column displays information from a bound data source, and you can edit the values of column to update the task details through Tree Grid. The operations such as sorting, filtering, and searching can be performed based on column definitions. To display a Gantt Chart column, the **Field** property should be mapped from the data source to the column.

---

If the column **Field** is not specified in the data source, the column values will be empty.

---

The **TreeColumnIndex** property is used to define the expander column in the Gantt Chart component to expand and collapse the child rows.

### Defining Columns

Using the **GanttColumns** property, you can define the columns in Gantt Chart. If the columns are not defined, then the default columns will be rendered based on the mapped data source fields in the **GanttTaskFields** property. Refer to the following code example for defining the columns in Gantt Chart along with their widths.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttColumns>
    <GanttColumn Field="TaskId" Width="150"></GanttColumn>
```

```

<GanttColumn Field="TaskName" HeaderText="Job Name"
Width="250"></GanttColumn>
</GanttColumns>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {

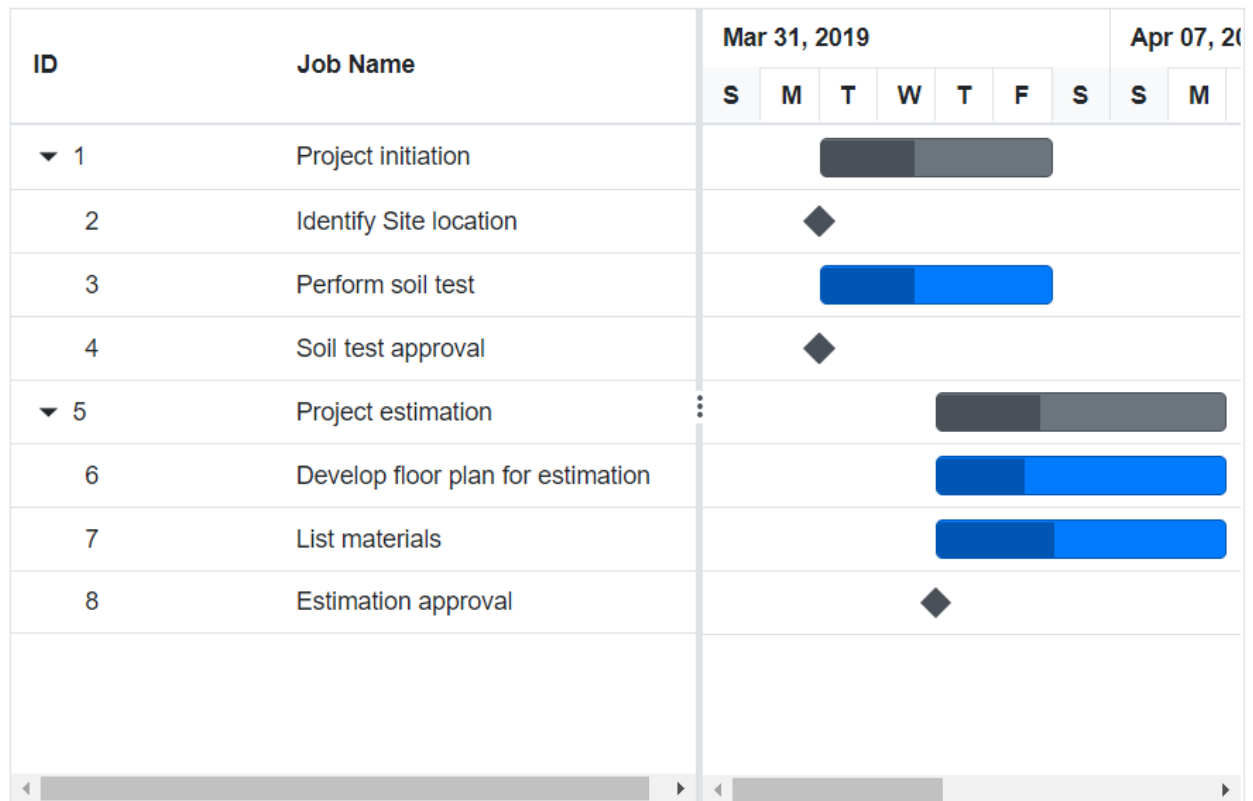
```



```

new TaskData() {
    TaskId = 6,
    TaskName = "Develop floor plan for estimation",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "3",
    Progress = 30,
},
new TaskData() {
    TaskId = 7,
    TaskName = "List materials",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "3",
    Progress = 40
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
}
})
}
};
return Tasks;
}
}

```



## Header template

Before adding the header template to the Gantt Chart, it is strongly recommended to go through the [template](#) section topic to configure the template.

The Header Template has options to display custom element values or content in the header. You can use the [HeaderTemplate](#) of the [GanttColumn](#) component to specify the custom content.

### ASPX-CS

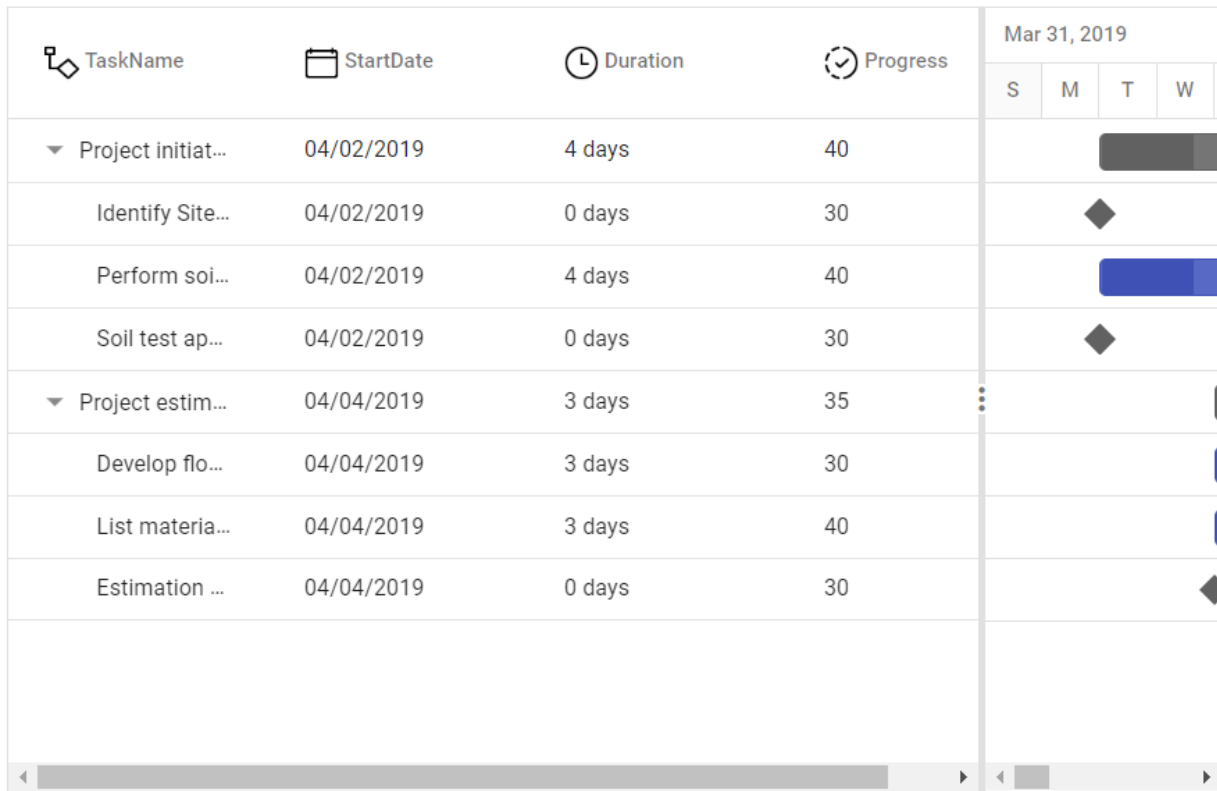
```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Width="700px" Height="400px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress"
  ParentID="Parent_Id"></GanttTaskFields>
  <GanttColumns>
    <GanttColumn Field="TaskName" HeaderText="Job Name" Width="250">
      <HeaderTemplate>
        <div>
          
            @((context as GridColumn).HeaderText)
          </div>
        </HeaderTemplate>
      </GanttColumn>
      <GanttColumn Field="StartDate" HeaderText="Start Date" Width="250">
        <HeaderTemplate>
          <div>
            
            @((context as GridColumn).HeaderText)
          </div>
        </HeaderTemplate>
      </GanttColumn>
      <GanttColumn Field="EndDate" HeaderText="End Date" Width="250">
        <HeaderTemplate>
          <div>
            
            @((context as GridColumn).HeaderText)
          </div>
        </HeaderTemplate>
      </GanttColumn>
      <GanttColumn Field="Duration" HeaderText="Duration" Width="250">
        <HeaderTemplate>
          <div>
            
            @((context as GridColumn).HeaderText)
          </div>
        </HeaderTemplate>
      </GanttColumn>
      <GanttColumn Field="Progress" HeaderText="Progress" Width="250">
        <HeaderTemplate>
          <div>
```

```


@((context as GridColumn).HeaderText)
</div>
</HeaderTemplate>
</GanttColumn>
</GanttColumns>
</SfGantt>
<style>
@@font-face {
font-family: 'ej2-headertemplate';
src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAkAIAAAwAgTlMvMj1vTFIAAAEoAAAAVmNtYXDS2c5qAAABjAAAAEBnbHlmQmN
FrQAAAdQAAANoaGVhZBRdbkIAAADQAAANmhoZWEIuQQEAAAArAAAACRobXR4DAAAAAAAAAYAAAA
MbG9jYQCOAbQAAAHMAAACG1heHABHgENAAABCAAAACBuYw1lohGZJQAABTWAAAKpcG9zdA2o3w0
AAAFoAAAAQAABAAAEAAAAAFwEAAAAAAD9AABAAAAAAAAAAAAAAAAAAAAAwABAAAAQAATBXy9l8
PPPUACwQAAAAANillKkAAAAA2KWUqQAAAAAD9APzAAAACAACAAAAAAAAAAAAEAAAADAQEAEQAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQAAZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wLpYAQAAAAAXAQAAAAAAAAABAAAAAAAAABAAAAAQAAAA
EAAAAAAAAAgAAAAMAAAUAAMAAQAAABQABAAsAAAABgAEAAEAucC6WD//wAA5wLpYP//AAAAAA
BAAYABgAAAAIAAQAAAAAAjgG0ABEAAAAAA8kD8wADAACACwAPABMAFwAbAB8AIwAnACsALwAzADc
AOwBPAGsAACUVizUjFSM1IxUjNSMVIzUjFSM1JRujNSMVIzUjFSM1IxUjNSMVIzU1FSM1IxUjNSM
VIzUjFSM1IxUjNQMVHwYhPwCRITcjDwghNS8HIzUjFSE1IwN2U1NTU1RTU1NTAuxTU1NTVFNTU1M
C7FNTU1NUU1NTU1QCAwUGBgGIA0QICAcHBQQBavxsp30ICAcHAgUDAQEDlAECBAUHBwgIfVP+YFO
zU1NTU1NTU1NTU6dUVFRUVFRUVFRUplNTU1NTU1NTU1P+NgQIBwcGBAMCAQIEBQcHAWgCdPoBAGQ
FAwCHCIF8CQgHBgUEAgFTU1MABAAAAAAD9APeADQABQCuAQAAAAEfCDc1Lwc1PwIPBy8HHw3HwQ
PATMVPwclLx0jDwMfAgUdAR8GBTUzLxQjDx0BFxUFEDsBPxA1Nyc1LxIPEhUCCg8OGxcVExANCgM
BAQMDCQQDAgECAxESEhMTExUUFRTFBISEhEHCwYHCAkKCw0NDw8SuQQGAWIBAgRxlgsKCQCGBAM
BAGMDAwUFBQcGBWgICQkKCgsKDAsMDQwNDQ4NDg45BQUDAQEEA/1eAgUGBwkKCwHjeggKDhEUFXs
1FRMSEA4NCwoJBwcJBjwODg0ODQ0MDQwLDAoLCgoJCQgIBwYHBQUFAwMDAgEBAQECAgYICg0ODxI
SFBUXFwwMDA0MDQwMFxcVFBISDw4MCwgGAgIBAQICAwcJCw0PERITFRUXDAwMDA0NDawMDAsXFRQ
TEQ8ODQoIBgICAVQEAWgJCgsMCwwbEBAREREZE8VDAwKCgoIBwYFAwIBAQIDBQYHCAoUFQwLCws
LCgoJCACGMwsWFhUVHB3QAQIEBggICgueDg4ODg0NDA0MCwsLCwoKCQgJBWgGBwUFBAQDAwECCw8
NDxETCrIlawSKCAGBAIB0AoLCwoLCQgNCAkLDA0NDg4ODg4ZFAlBAwMEBAUGBgYIBwkICQoKCws
LDAwMDA0ODQ4ODgH7DQwMDBgWFRQTERAODaoIBgICAQECAgYICgwoEBETFBWGAwMDA0MDQwMCxc
WFRMSEA8NDakHawIBAQEBAQECAwMICwwoEBETFBWfwwMDQAAAAASAN4AAQAAAAAAAAABAAAAQA
AAAAAQASAAEAQAAAAAAAAgAHABMAAQAAAAAAwASABoAAQAAAAAABAASACwAAQAAAAAABQALAD4
AAQAAAAAABgASAEkAAQAAAAAACgAsAFsAAQAAAAAACwASAIcAAwABBakAAAACAjKAawABBakAAQA
kAJsAAwABBakAAgAOAL8AAwABBakAAwAkAM0AAwABBakABAakAPEAAwABBakABQAWARUAawABBak
ABgAkASsAAwABBakACgBYAU8AAwABBakACwAkAacgZWoyLWhlYWRLcnRlbXBsYXRlUmVndWxhcmV
qMi1oZWfkZXJ0ZW1wbGF0ZWVqMi1oZWfkZXJ0ZW1wbGF0ZVZlcnNpb24gMS4wZWoyLWhlYWRLcnR
lbXBsYXRlRm9udCBnZW5lcmF0ZWQgdXNpbmcgU3luY2Zlc2lvbiBNZXRYbyBTdHVKaW93d3cuc3l
uY2Zlc2lvbi5jb20AIAblAGoAMgAtAGgAZQBhAGQAZQBByAHQAZQBtAHAAbABhAHQAZQBSAGUAZwB
lAGwAYQByAGUAagAyAC0AaABlAGEAZABlAHlAdABlAG0AcABsAGEAdABlAGUAagAyAC0AaABlAGE
AZABlAHlAdABlAG0AcABsAGEAdABlAFYAZQBByAHMAaQBvAG4AIAAxAAC4AMABlAGoAMgAtAGgAZQB
hAGQAZQBByAHQAZQBtAHAAbABhAHQAZQBGAG8AbgB0ACAAZwBlAG4AZQBByAGEAdABlAGQAIABlAHM
AaQBuaGcAIABTAHkAbgBjAGYAdQBzAGkAbwBuACAATQBlAHQAcgBvACAAUwB0AHUAZABpAG8AdwB
3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYwBvAG0AAAAAAGAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAADAQIBAwEEAAATCVF9DYWxlbmRhcghlbXBsb3l1ZQAQ)
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<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? Parent_Id { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>()
{
new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
new DateTime(2022, 04, 02), EndDate = new DateTime(2022, 04, 21)},
new TaskData() { TaskId = 2, TaskName = "Identify Site location", StartDate
= new DateTime(2022, 04, 02), Duration = "0", Progress = 30, Parent_Id = 1},
new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
DateTime(2022, 04, 02), Duration = "4", Progress = 40, Parent_Id = 1},
new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
new DateTime(2022, 04, 02), Duration = "0", Progress = 30, Parent_Id =1},
new TaskData() { TaskId = 5, TaskName = "Project estimation", StartDate =
new DateTime(2022, 04, 02), EndDate = new DateTime(2022, 04, 21)},
new TaskData() { TaskId = 6, TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2022, 04, 04), Duration = "3", Progress = 30,
Parent_Id =5},
new TaskData() { TaskId = 7, TaskName = "List materials", StartDate = new
DateTime(2022, 04, 04), Duration = "3", Progress = 40, Parent_Id =5},
new TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate =
new DateTime(2022, 04, 04), Duration = "0", Progress = 30, Parent_Id =5}
};
return Tasks;
} }
```



### Format

To format the cell values based on a specific culture, use the `GanttColumn.Format` property. The [Blazor Gantt Chart](#) component uses the `Internationalization` library to format number and date values.

### ASPX-CS

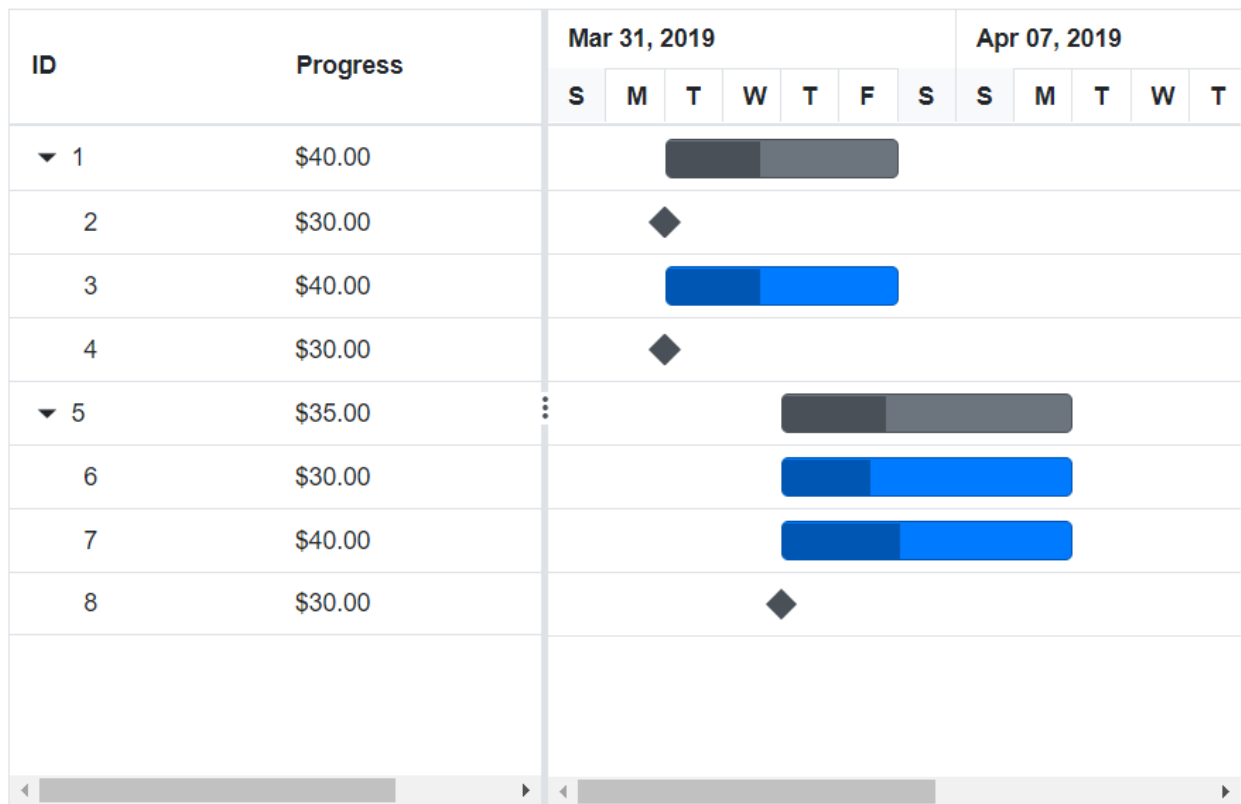
```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttColumns>
    <GanttColumn Field="TaskId" Width="150"></GanttColumn>
    <GanttColumn Field="Progress" Format="@NumberFormat"
    Width="250"></GanttColumn>
  </GanttColumns>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
public string NumberFormat = "C";
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
}
```

```
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
```

```

Progress = 40
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
}
})
}
};
return Tasks;
}
}

```



By default, the number and date values are formatted in en-US culture.

#### [Number formatting](#)

The number or integer values can be formatted using the following format strings.

Format | Description | Remarks

{ type:'date', format:'dd/MM/yyyy' } | 04/07/2019

{ type:'date', format:'dd.MM/yyyy' } | 04.07.2019

{ type:'date', skeleton:'short' } | 7/4/19

{ type:'dateTime', format:'dd/MM/yyyy hh:mm tt' } | 04/07/2019 12:00 AM

```
{ type: 'dateTime', format: 'MM/dd/yyyy hh:mm:ss tt' } | 07/04/2019 12:00:00 AM
```

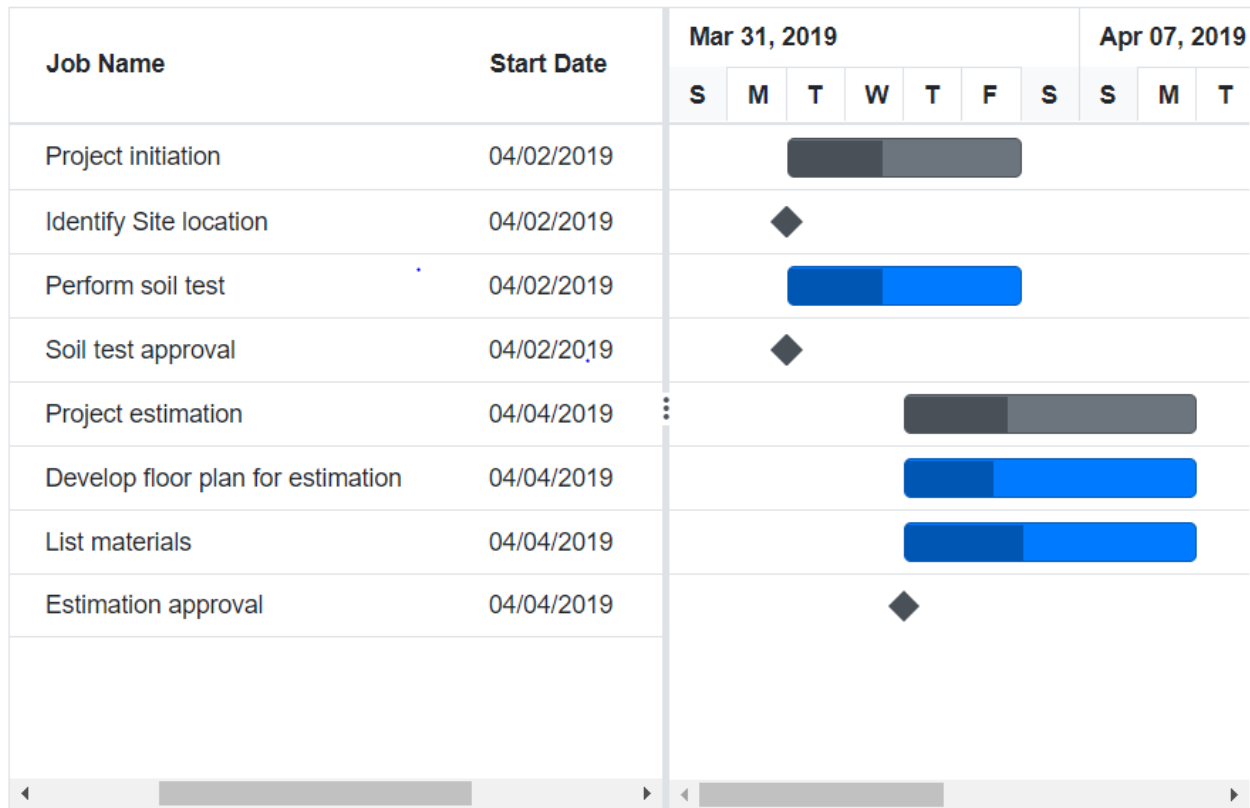
**ASPX-CS**

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" Width="150"></GanttColumn>
<GanttColumn Field="TaskName" HeaderText="Job Name"
Width="250"></GanttColumn>
<GanttColumn Field="StartDate" Format="@DateFormat"></GanttColumn>
<GanttColumn Field="Duration"></GanttColumn>
</GanttColumns>
</SfGantt>

@code{
public List<TaskData> TaskCollection { get; set; }
public string DateFormat = "MM/dd/yyyy";
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
```



```
    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "0",
        Progress = 30
    },
    })
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
},
};
return Tasks;
}
```



### Reordering

The column reordering can be done by dragging a column header from one index to another index within the Tree Grid. To enable reordering, set the `AllowReordering` property to true.

### ASPX-CS

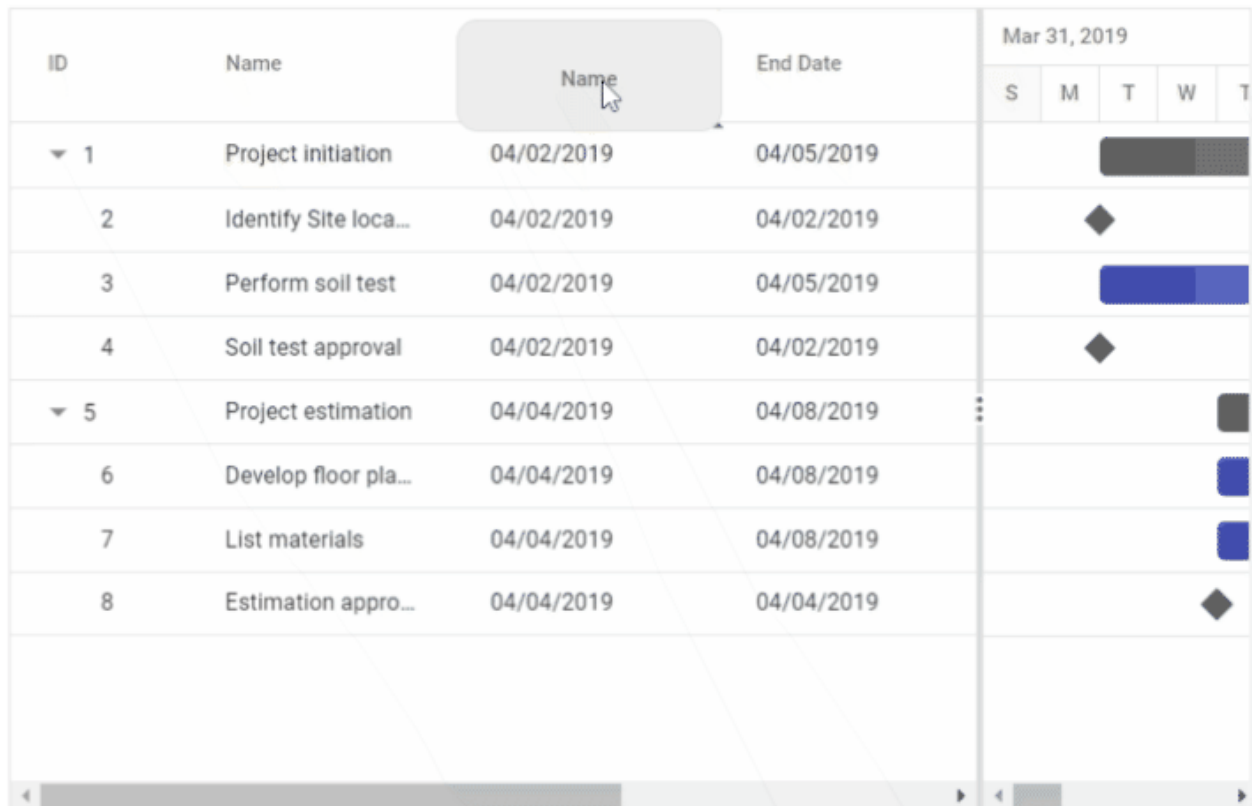
```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowReordering="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
```

```
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 40
                },
                new TaskData() {
                    TaskId = 8,
                    TaskName = "Estimation approval",
                    StartDate = new DateTime(2019, 04, 04),
```

```

Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```



You can disable the reordering of a particular column by setting the `GanttColumn.AllowReordering` property to `false`.

#### Reorder Multiple Columns

Multiple columns can be reordered at a time by using the `ReorderColumnsAsync` method.

#### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<button onclick="ReorderColumn">Reorder TaskName and StartDate to
last</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px" AllowReordering="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" HeaderText="Task Id" Width="150"></GanttColumn>

```

```

<GanttColumn Field="TaskName" HeaderText="Task Name"></GanttColumn>
<GanttColumn Field="StartDate" HeaderText="Start Date"></GanttColumn>
<GanttColumn Field="Duration" HeaderText="Duration"></GanttColumn>
<GanttColumn Field="Progress" HeaderText="Progress"></GanttColumn>
</GanttColumns>
<GanttSplitterSettings Position="100%"></GanttSplitterSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void ReorderColumn()
{
this.Gantt.ReorderColumnsAsync(new List<string>() {"TaskName",
"StartDate"}, "Progress");
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
}
}
}
}
}
}

```

```
Progress = 30
},
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
};
return Tasks;
}
```

Reorder TaskName and StartDate to last

Task Id	Duration	Progress	Task Name	Start Date
▼ 1	4 days	40	Project initiation	04/02/2019
2	0 days	30	Identify Site loca...	04/02/2019
3	4 days	40	Perform soil test	04/02/2019
4	0 days	30	Soil test approval	04/02/2019
▼ 5	3 days	35	Project estimation	04/04/2019
6	3 days	30	Develop floor pla...	04/04/2019
7	3 days	40	List materials	04/04/2019
8	0 days	30	Estimation appro...	04/04/2019

### Resizing

The column width can be resized by clicking and dragging the right edge of the column header. While dragging, the width of the column will be resized immediately. Each column can be auto resized by double-clicking the right edge of the column header to fit the width of that column based on the widest cell content. To resize the column, set the **AllowResizing** property to true. The following code example shows how to enable the column resize feature in the Gantt Chart component.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowResizing="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
```

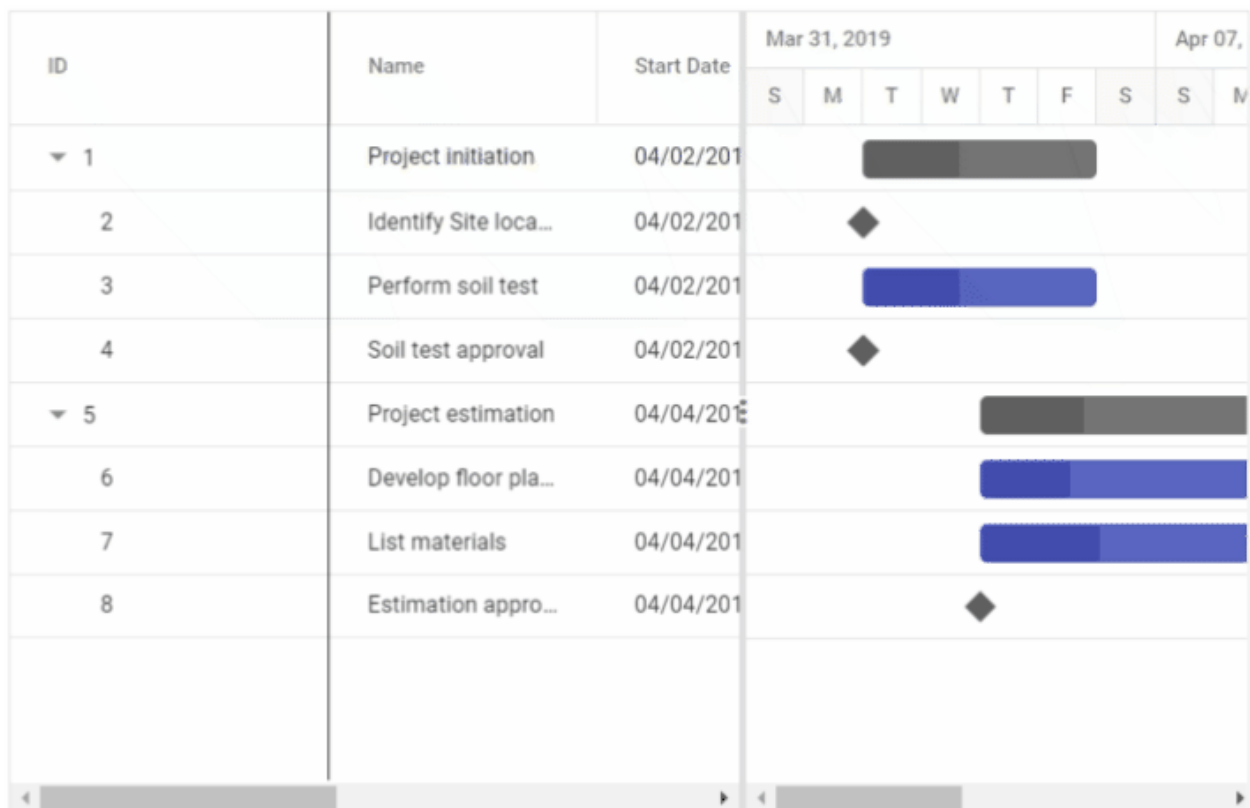
```
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 40
                }
            })
        }
    }
}
```



```

},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
}
})
}
};
return Tasks;
}
}

```



You can disable resizing for a particular column by setting the `GanttColumn.AllowResizing` to false.

#### Defining Minimum and Maximum Column Width

The column resizing can be restricted between minimum and maximum widths by defining the `GanttColumn.MinWidth` and `GanttColumn.MaxWidth` properties.

In the following example, the minimum and maximum widths are defined for the `Duration`, and `Task Name` columns.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
```

```

<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowResizing="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" Width="50"></GanttColumn>
<GanttColumn Field="TaskName" Width="200" MinWidth="150"
MaxWidth="250"></GanttColumn>
<GanttColumn Field="StartDate"></GanttColumn>
<GanttColumn Field="Duration" Width="100" MinWidth="50"
MaxWidth="200"></GanttColumn>
<GanttColumn Field="Progress"></GanttColumn>
</GanttColumns>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,

```

```

TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### Column Template

A column template is used to customize the column's look. The following code example explains how to define the custom template in Gantt Chart using the `Template` property.

#### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Buttons
<SfGantt DataSource="@TaskCollection" Height="450px" Width="100%"
HighlightWeekends="true" ProjectStartDate="@ProjectStart"
ProjectEndDate="@ProjectEnd">
<GanttColumns>
<GanttColumn Field="TaskId" HeaderText="Task ID" MinWidth="150"
MaxWidth="250" AllowReordering="false"></GanttColumn>
<GanttColumn Field="TaskName" HeaderText="Task Name">

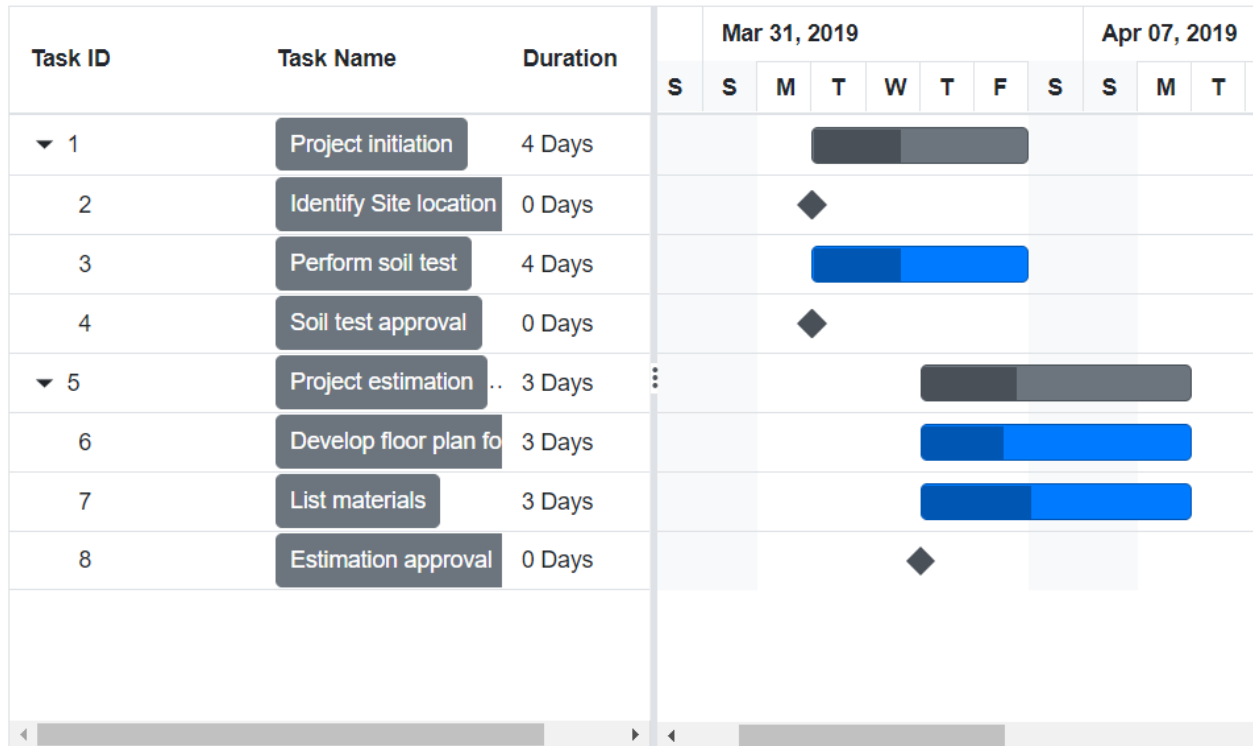
```

```

<Template>
@{
@if (context != null)
{
<SfButton CssClass="e-bigger" Content="@((context as
TaskData).TaskName)"></SfButton>
}
}
</Template>
</GanttColumn>
<GanttColumn Field="Duration" HeaderText="Duration"></GanttColumn>
</GanttColumns>
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId">
</GanttTaskFields>
</SfGantt>
@code{
public DateTime ProjectStart = new DateTime(2019, 3, 25);
public DateTime ProjectEnd = new DateTime(2019, 7, 28);
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime? StartDate { get; set; }
public DateTime? EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",

```

```
Progress = 40,
ParentId = 1
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
ParentId = 5
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
ParentId = 5
}
};
return Tasks;
}
}
```



### Column Menu

The column menu has options to integrate features like sorting, filtering, and autofit. It will show a menu with the integrated feature when users click the Multiple icon of the column. To enable the column menu, you should set the `ShowColumnMenu` property to true.

The default items are displayed in the following table:

Item	Description
SortAscending	Sort the current column in ascending order.
SortDescending	Sort the current column in descending order.
AutoFit	Auto fit the current column.
AutoFitAll	Auto fit all columns.
ColumnChooser	Choose the column visibility.
Filter	Shows the filter menu based on column type.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowResizing="true" ShowColumnMenu="true" AllowFiltering="true"
AllowSorting="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
```

```
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 6,
```

```

TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

ID		Name	Start Date	End Date	Mar 31, 2019				
					S	M	T	W	T
▼ 1		Autofit all columns	4/2/2019	4/5/2019					
2		Autofit this column	4/2/2019	4/2/2019			◆		
3	↑ Sort Ascending		4/2/2019	4/5/2019					
4	↓ Sort Descending		4/2/2019	4/5/2019					
4	Columns		4/2/2019	4/2/2019			◆		
▼ 5	Filter		4/4/2019	4/8/2019					
6		Develop floor pla...	4/4/2019	4/8/2019					
7		List materials	4/4/2019	4/8/2019					
8		Estimation approval	4/4/2019	4/4/2019					◆

You can disable the column menu for a particular column by setting the `GanttColumn.ShowColumnMenu` to false.



### Responsive Columns

You can toggle the column visibility based on media queries, which are defined in the `HideAtMedia`. The `HideAtMedia` accepts valid [Media Queries](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks"></GanttTaskFields>
  <GanttColumns>
    <GanttColumn Field="TaskId" Width="150" HideAtMedia="(min-width:
    700px)"></GanttColumn>
    <GanttColumn Field="TaskName" HeaderText="Job Name"
    Width="250"></GanttColumn>
    <GanttColumn Field="StartDate"></GanttColumn>
    <GanttColumn Field="Duration" HideAtMedia="(min-width:
    500px)"></GanttColumn>
  </GanttColumns>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
  List<TaskData> Tasks = new List<TaskData>() {
    new TaskData() {
      TaskId = 1,
      TaskName = "Project initiation",
      StartDate = new DateTime(2019, 04, 02),
      EndDate = new DateTime(2019, 04, 21),
      SubTasks = (new List<TaskData> () {
        new TaskData() {
          TaskId = 2,
          TaskName = "Identify Site location",
          StartDate = new DateTime(2019, 04, 02),
          Duration = "0",
          Progress = 30,
        },
        new TaskData() {
          TaskId = 3,
          TaskName = "Perform soil test",
```

```

StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### Change Tree / Expander column

The tree/expander column is a column in the Gantt Chart component, that has icons to expand or collapse the parent records. You can define the tree column index in the Gantt Chart component by using the `TreeColumnIndex` property and the default value of this property is 0. The following code example shows how to use this property.

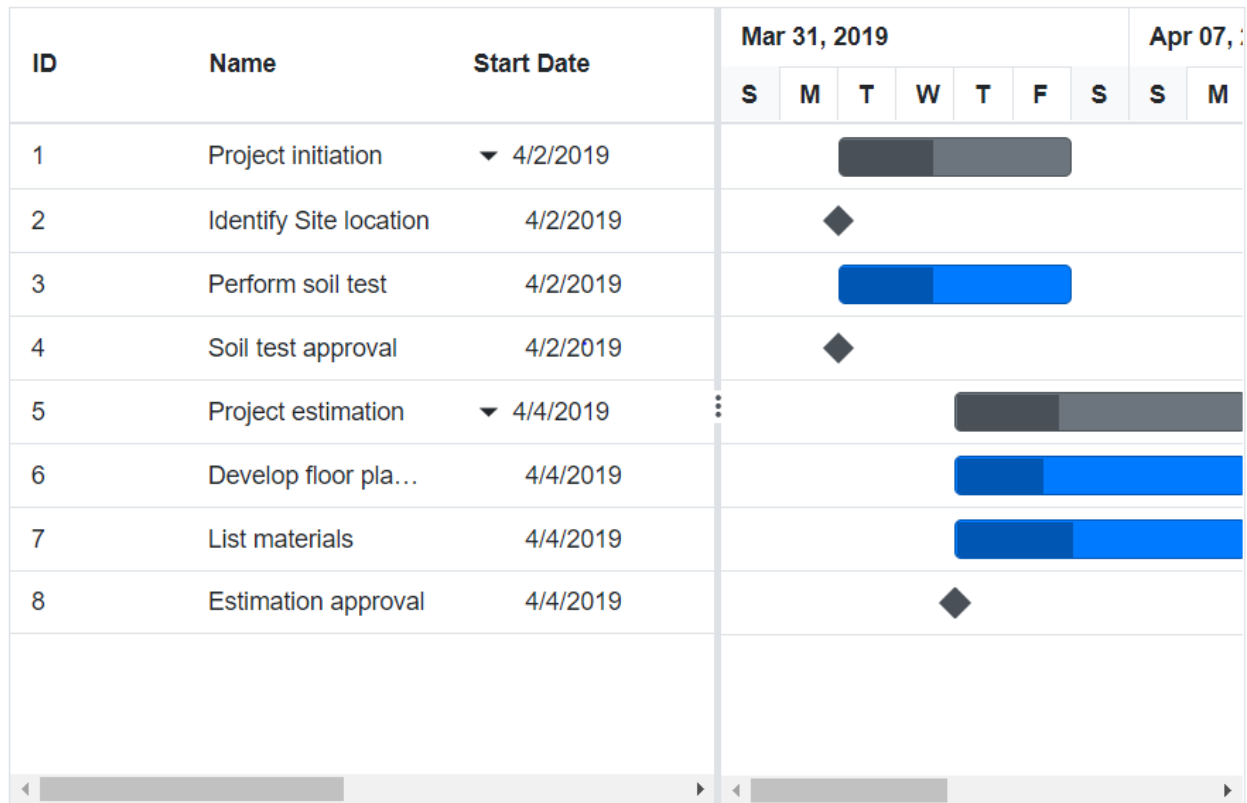
### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" TreeColumnIndex="2" Height="450px"
Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks"></GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,

```

```
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 6,
        TaskName = "Develop floor plan for estimation",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 7,
        TaskName = "List materials",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 40
    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "0",
        Progress = 30,
    }
})
};
return Tasks;
}
```



### Show or Hide Columns dynamically

You can show or hide gantt component columns dynamically using external buttons by invoking the `ShowColumnsAsync` or `HideColumnsAsync` method.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<button @onclick="show">Show columns</button>
<button @onclick="hide">Hide columns</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttColumns>
    <GanttColumn Field="TaskId" Width="100"></GanttColumn>
    <GanttColumn Field="TaskName" HeaderText="Job Name"
Width="250"></GanttColumn>
    <GanttColumn Field="StartDate"></GanttColumn>
    <GanttColumn Field="EndDate"></GanttColumn>
    <GanttColumn Field="Duration"></GanttColumn>
    <GanttColumn Field="Progress"></GanttColumn>
  </GanttColumns>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
public string[] ColumnList = {"TaskName", "StartDate"};
public void show() {
```

```
this.Gantt.ShowColumnsAsync(ColumnList, "Field");
}
public void hide() {
this.Gantt.HideColumnsAsync(ColumnList, "Field");
}
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
```

```

TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
},
}))
}
};
return Tasks;
}
}

```

### Controlling Gantt Column actions

You can enable or disable gantt component action for a particular column by setting the `AllowFiltering`, `AllowSorting`, `AllowReordering`, and `AllowEditing` properties.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="900px" AllowSorting="true" AllowFiltering="true"
AllowReordering="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" Width="100"></GanttColumn>
<GanttColumn Field="TaskName" AllowReordering="false"
Width="250"></GanttColumn>
<GanttColumn Field="StartDate" AllowEditing="false"></GanttColumn>
<GanttColumn Field="Duration" AllowSorting="false"></GanttColumn>
<GanttColumn Field="Progress" AllowFiltering="false"></GanttColumn>
</GanttColumns>
<GanttEditSettings AllowEditing="true"></GanttEditSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
}

```

```
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public bool Verified { get; set; }
    public List<TaskData> SubTasks { get; set; }
}

public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            Verified = true,
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Verified = true,
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Verified = false,
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Verified = true,
                    Progress = 30,
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            Verified = false,
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
```



```
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Verified = true,
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Verified = false,
Progress = 40,
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Verified = true,
Progress = 30,
},
})
}
};
return Tasks;
}
}
```

### Column Type

Column type can be specified using the `GanttColumn.Type` property. It specifies the type of data the column binds. If the `GanttColumn.Format` is defined for a column, the column uses `GanttColumn.Type` to select the appropriate format option number or date.

Gantt column supports the following types:

- String
- Number
- Boolean
- Date
- DateTime

If the `GanttColumn.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 `GanttColumn.Type` for that column.

### Custom Columns

Using the `GanttColumns` property, you can define the Custom Columns in Gantt Chart. If custom columns are required, then you can generate columns that was not defined in the `GanttTaskFields` property. Refer to the following code example for defining the custom columns in Gantt Chart.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
```

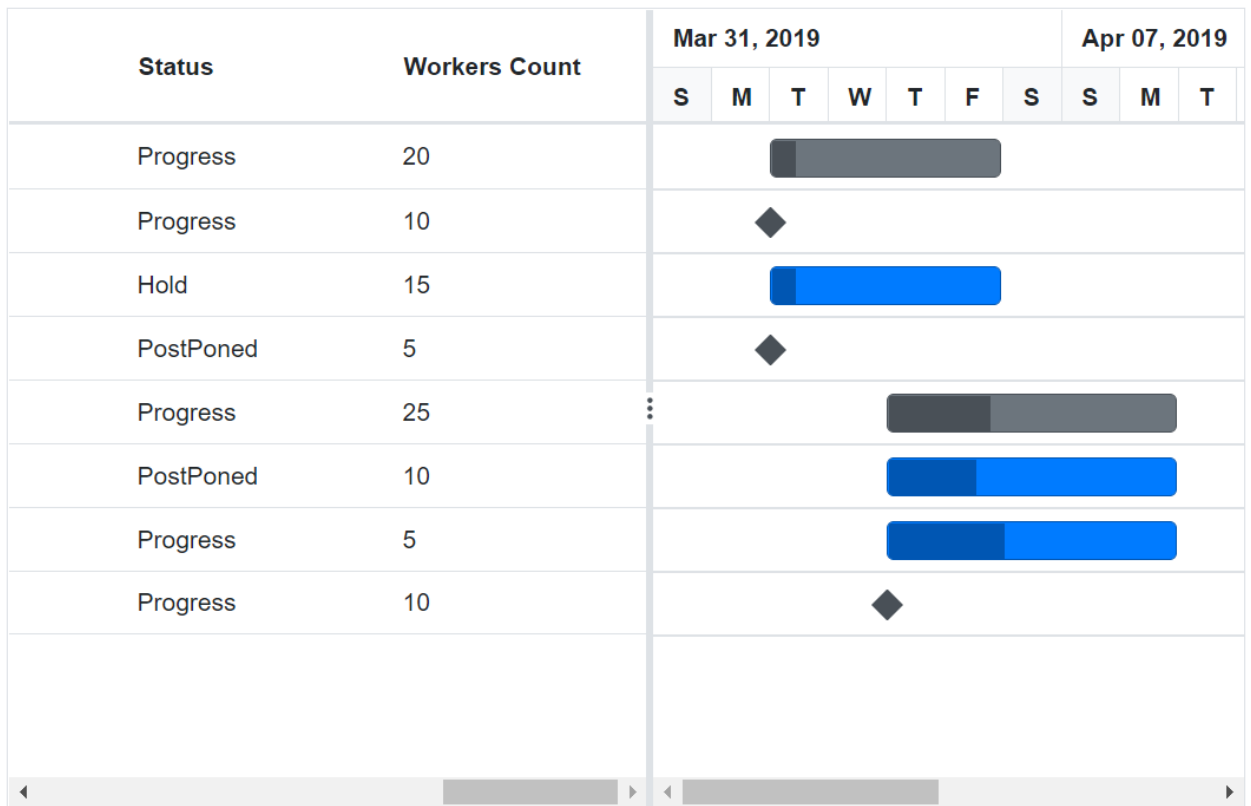
```

<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttColumns>
    <GanttColumn Field="TaskId" Width="150"></GanttColumn>
    <GanttColumn Field="TaskName" HeaderText="Job Name"
    Width="150"></GanttColumn>
    <GanttColumn Field="StartDate" HeaderText="Start Date"
    Width="150"></GanttColumn>
    <GanttColumn Field="EndDate" HeaderText="End Date"
    Width="150"></GanttColumn>
    <GanttColumn Field="Duration" HeaderText="Duration"
    Width="150"></GanttColumn>
    <GanttColumn Field="Progress" HeaderText="Progress"
    Width="150"></GanttColumn>
    <GanttColumn Field="Status" HeaderText="Status" Width="150"
    EditType=Syncfusion.Blazor.Grids.EditType.DefaultEdit></GanttColumn>
    <GanttColumn Field="WorkersCount" HeaderText="Workers Count" Width="150"
    EditType=Syncfusion.Blazor.Grids.EditType.NumericEdit></GanttColumn>
  </GanttColumns>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public string Status { get; set; }
  public int WorkersCount { get; set; }
  public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
  List<TaskData> Tasks = new List<TaskData>() {
    new TaskData() {
      TaskId = 1,
      TaskName = "Project initiation",
      StartDate = new DateTime(2019, 04, 02),
      EndDate = new DateTime(2019, 04, 21),
      Status="Progress",
      WorkersCount=20,
      SubTasks = (new List<TaskData> () {
        new TaskData() {
          TaskId = 2,
          TaskName = "Identify Site location",
          StartDate = new DateTime(2019, 04, 02),
          Duration = "0",

```

```
Progress = 5,
Status="Progress",
WorkersCount=10,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 10,
Status="Hold",
WorkersCount=15,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
Status="PostPoned",
WorkersCount=5,
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
Status="Progress",
WorkersCount=25,
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
Status="PostPoned",
WorkersCount=10,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
Status="Progress",
WorkersCount=5,
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
Status="Progress",
```

```
WorkersCount=10,
}
})
}
};
return Tasks;
}
}
```



Column Chooser

The column chooser has options to show or hide columns dynamically. It can be enabled by defining the [ShowColumnChooser](#) as true.

ASPX-CS

CODESNIPPET

ASPX-CS

,

ASPX-CS

、  
-

### ASPX-CS

、  
-

### ASPX-CS

-->

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttSelectionSettings Mode="SelectionMode.Both"></GanttSelectionSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
```

```
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}
```

## Row

The row selection in the Gantt Chart component can be enabled or disabled using the `AllowSelection` property. You can get the selected row object using the `GetSelectedRecordsAsync` method. The following code example shows how to disable the row selection in Gantt Chart.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowSelection="false">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
```

```

Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

**Row** selection is the default type of Gantt Chart selection mode.

#### *Selecting a Row on Initial Load*

You can select a row at the time of loading by setting the index of the row to the **SelectedRowIndex** property. Find the following code example for details.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowSelection="true" SelectedRowIndex="3">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
}

```

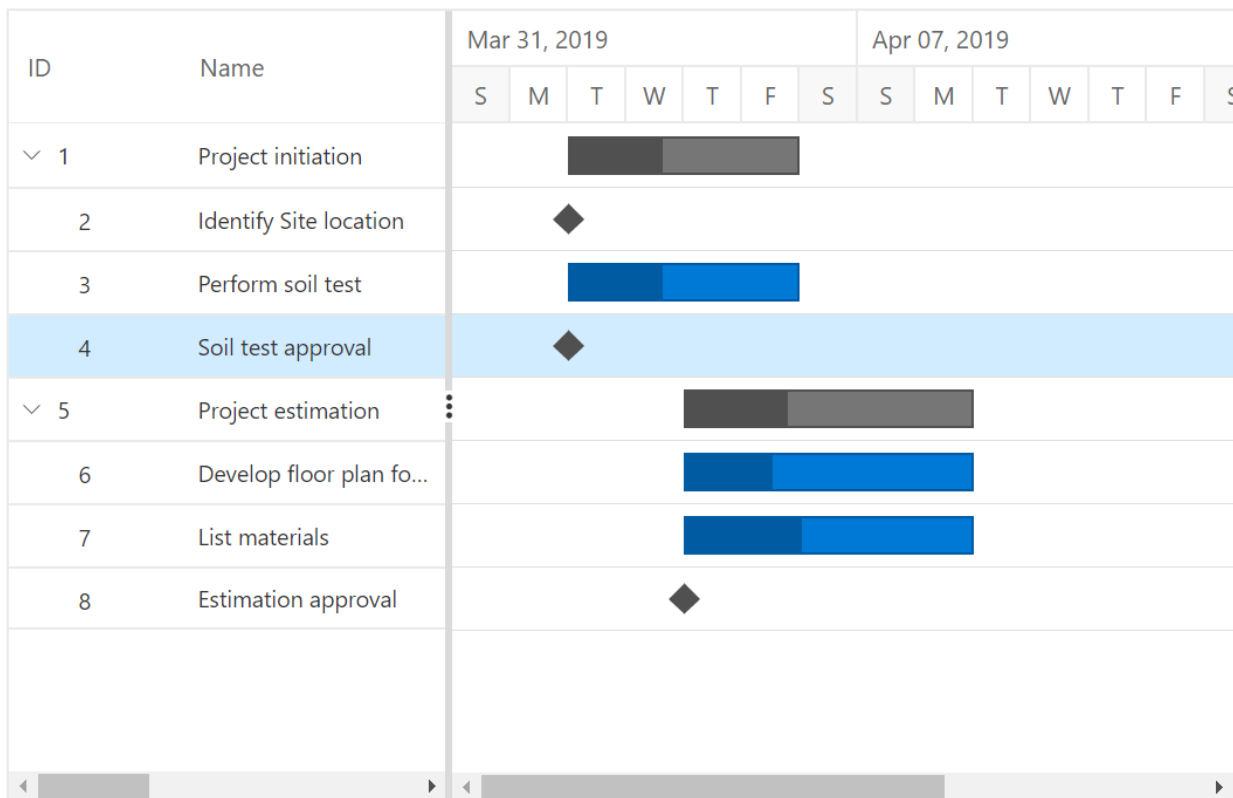


```
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
```

```

},
new TaskData() {
    TaskId = 7,
    TaskName = "List materials",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "3",
    Progress = 40
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
}
})
}
};
return Tasks;
}
}

```



#### Selecting a Row dynamically

You can also select a row dynamically using the `SelectRowAsync` method. The following code demonstrates how to select a row dynamically by clicking the custom button.

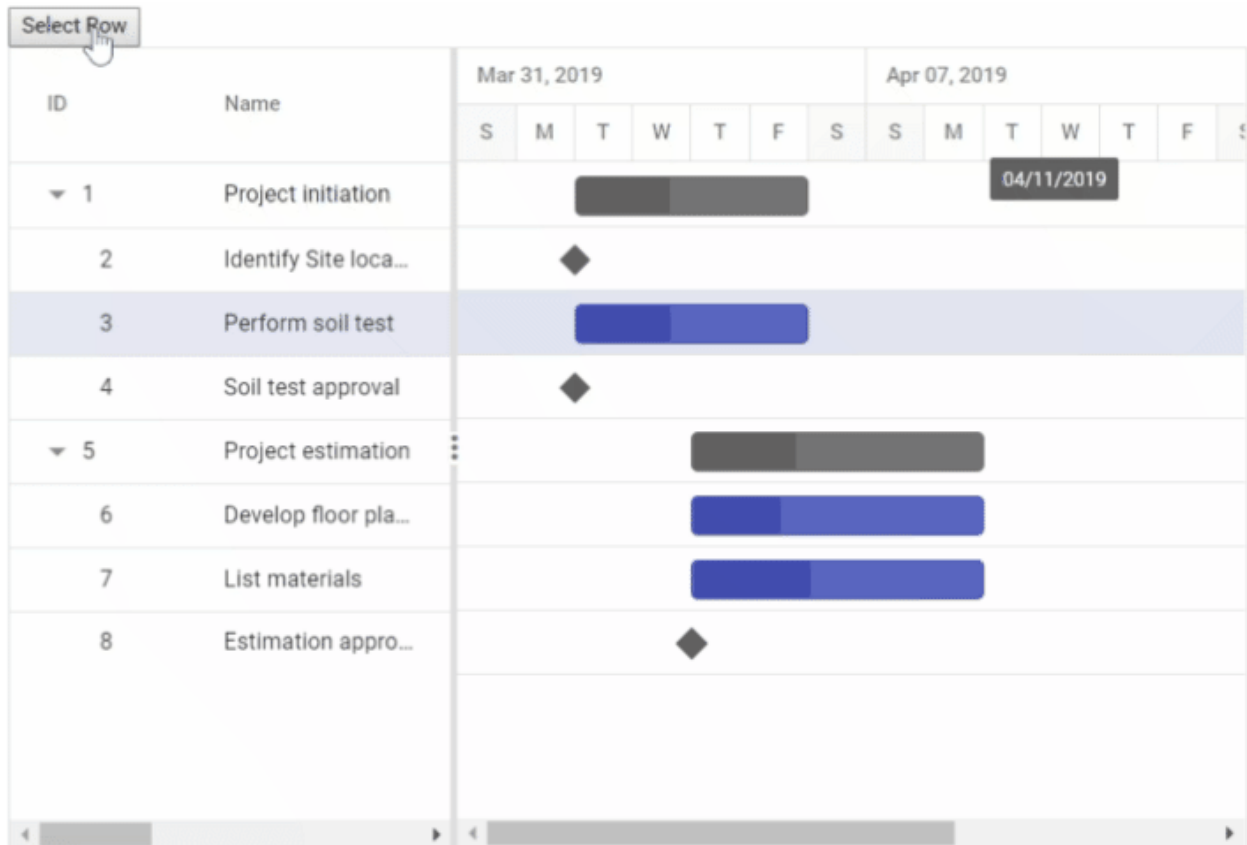
#### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<button @onclick="SelectRow">Select Row</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void SelectRow()
{
this.Gantt.SelectRowAsync(2);
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30

```

```
},
}))
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
});
return Tasks;
}
```



### Multiple Row Selection

You can select multiple rows by setting the `SelectionSettings.Type` property to `Multiple`. You can select more than one row by holding down the CTRL key while selecting multiple rows. The following code example explains how to enable multiple selection in Gantt Chart.

### ASPX-CS

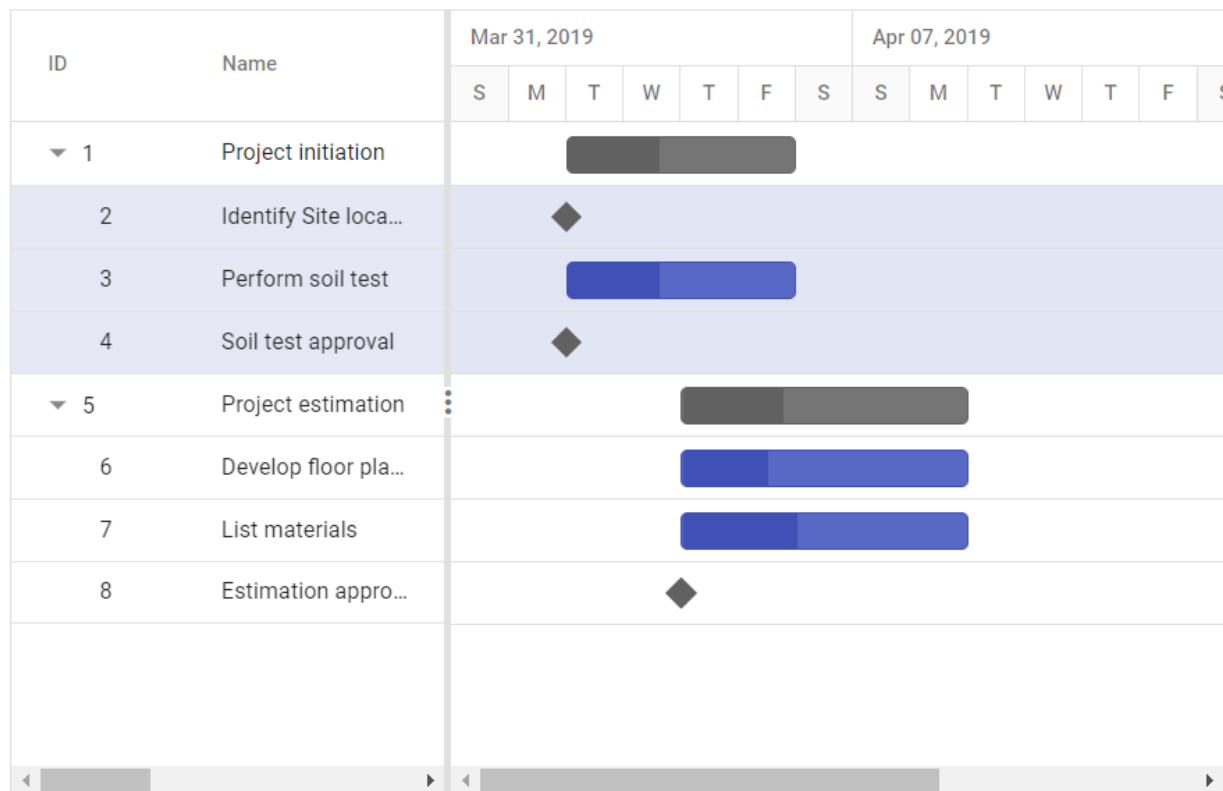
```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttSelectionSettings Mode="SelectionMode.Row"
  Type="Syncfusion.Blazor.Grids.SelectionType.Multiple"></GanttSelectionSettin
  gs>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
}
```

```
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
List<TaskData> Tasks = new List<TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
}))
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
```

```

TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```



### Selecting Multiple Rows dynamically

You can also select rows dynamically using the `SelectRowsAsync` method. The following code demonstrates how to select rows dynamically by clicking the custom button.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<button @onclick="SelectRows">Select Rows</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>

```

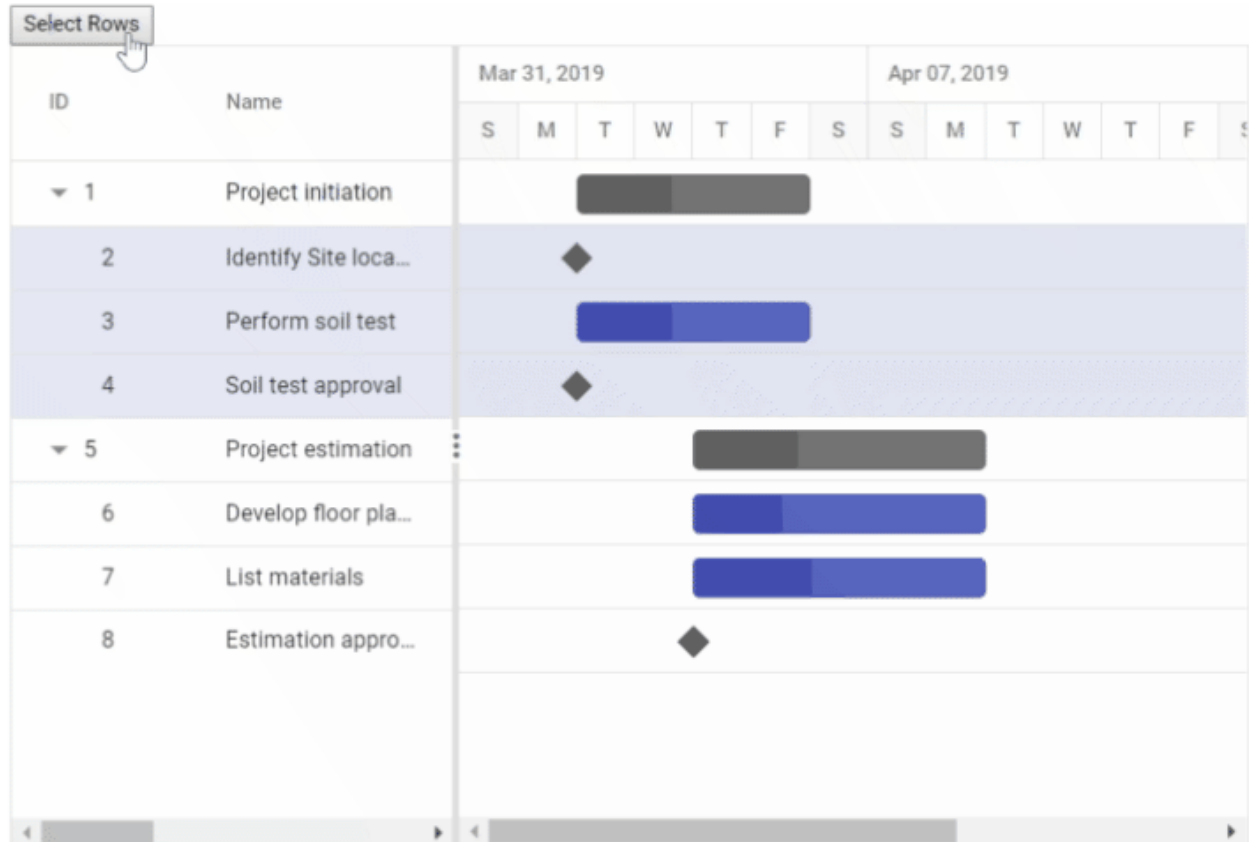
```

<GanttSelectionSettings Mode="SelectionMode.Row"
Type="Syncfusion.Blazor.Grids.SelectionType.Multiple"></GanttSelectionSettin
gs>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void SelectRows()
{
this.Gantt.SelectRowsAsync(new double[] {1,2,3});
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {

```



```
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 6,
        TaskName = "Develop floor plan for estimation",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 7,
        TaskName = "List materials",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 40
    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "0",
        Progress = 30,
    }
})
};
return Tasks;
}
```



### Cell

You can select a cell in the Gantt Chart component by setting the `SelectionSettings.Mode` property to cell. You can get the selected cell information using the `GetSelectedRowCellIndexes` method. This method returns the result as an object collection, which has `CellIndexes` and `RowIndex` information of the selected cells.

Find the code example below to enable the cell selection in Gantt Chart.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSelectionSettings Mode="SelectionMode.Cell"></GanttSelectionSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
}
```

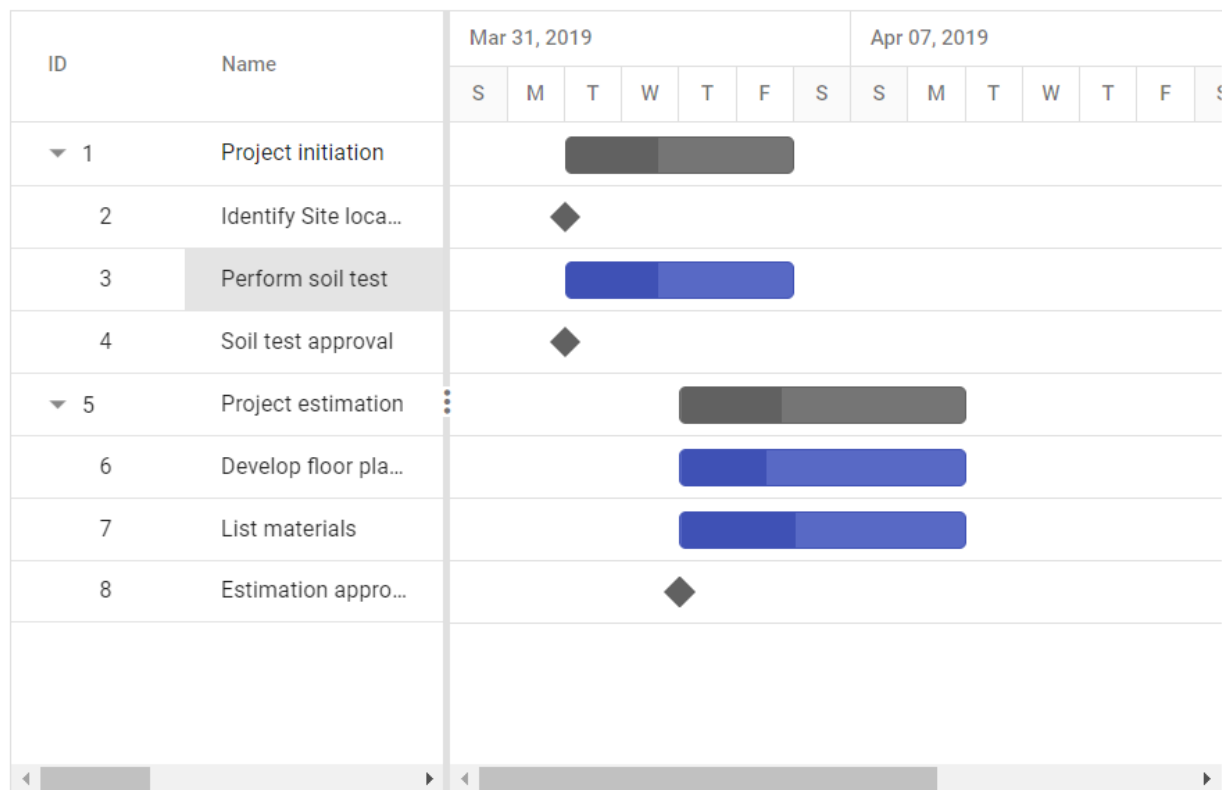
```
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}

public static List<TaskData> GetTaskCollection() {
    List<TaskData> Tasks = new List<TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 40
                },
            })
        },
    }
}
```

```

new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
}
})
}
};
return Tasks;
}
}

```



### Selecting Multiple Cells

You can select multiple cells by setting the `SelectionSettings.Type` property to `multiple` and the `SelectionSettings.Mode` property to `Cell`. Multiple cells can be selected by holding the CTRL key and selecting the cells. The following code example demonstrates how to select multiple cells.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
    <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
    EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
    </GanttTaskFields>

```

```

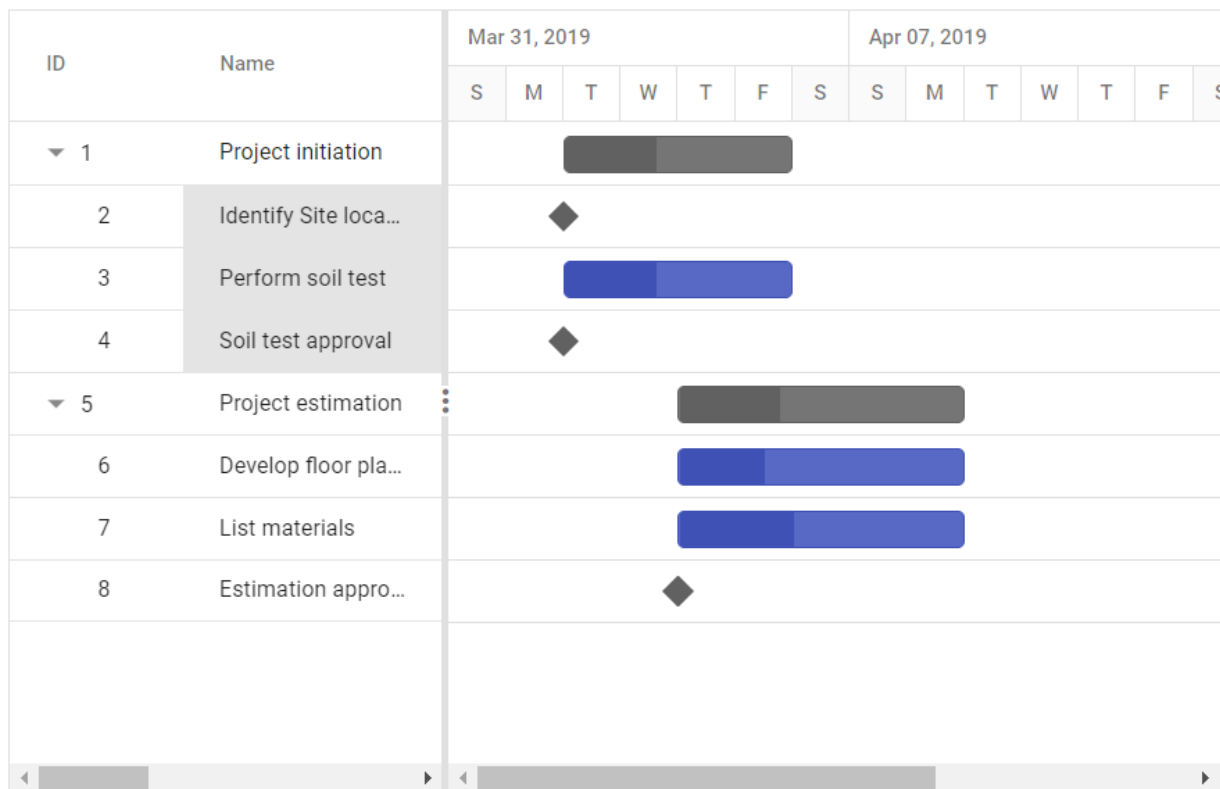
<GanttSelectionSettings Mode="SelectionMode.Cell"
Type="Syncfusion.Blazor.Grids.SelectionType.Multiple"></GanttSelectionSettin
gs>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {

```

```

new TaskData() {
    TaskId = 6,
    TaskName = "Develop floor plan for estimation",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "3",
    Progress = 30,
},
new TaskData() {
    TaskId = 7,
    TaskName = "List materials",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "3",
    Progress = 40
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
}
})
}
};
return Tasks;
}
}

```



*Selecting a Cell dynamically*

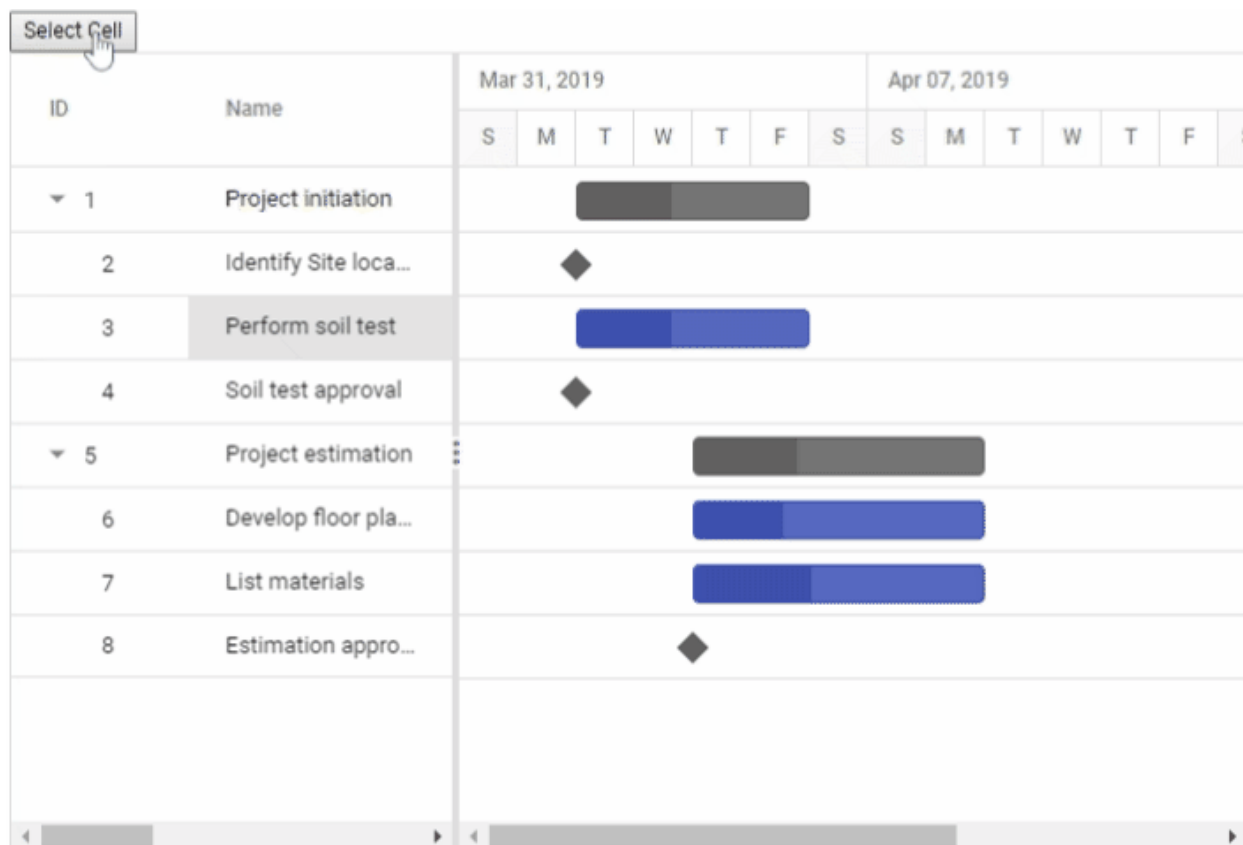
You can select a cell dynamically using the `SelectCellAsync` method. Refer to the following code example for details.

**ASPX-CS**

```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<button @onclick="SelectCell">Select Cell</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSelectionSettings Mode="SelectionMode.Cell"></GanttSelectionSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void SelectCell()
{
this.Gantt.SelectCellAsync( new ValueTuple<int, int> (1, 2) );
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
```

```
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}
```





#### Customize Cell Selection action

While selecting a cell in Gantt Chart, the `CellSelecting` and `CellSelected` event will be triggered. The `CellSelecting` event will be triggered on initialization of cell selection action, and you can get the current selecting cell information to prevent the selection of a particular cell in a particular row. The `CellSelected` event will be triggered on completion of cell selection action, and you can get the current selected cell's information. The following code example demonstrates how to prevent the selection of the cell using the `CellSelecting` event.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttSelectionSettings
  Mode="Syncfusion.Blazor.Grids.SelectionMode.Cell"></GanttSelectionSettings>
  <GanttEvents CellSelecting="CellSelecting" TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public void
CellSelecting(Syncfusion.Blazor.Grids.CellSelectingEventArgs<TaskData> args)
{
if (args.Data.TaskId == 2)
{
args.Cancel = true;
}
}
```

```

}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),

```

```

Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### Toggle Selection

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

By default, the `EnableToggle` property is set to `false`.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<button @onclick="DisableToggle">Disable Toggle</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSelectionSettings Mode="SelectionMode.Row"
Type="Syncfusion.Blazor.Grids.SelectionType.Multiple"
EnableToggle="@toggle"></GanttSelectionSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public bool toggle = true;
public void DisableToggle()
{
this.toggle = false;
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
}

```

```
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}

public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
```

```

TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### Clear Selection

You can clear the selected cells and selected rows by using a method called `ClearSelectionAsync`. The following code example demonstrates how to clear the selected rows in Gantt Chart.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<button @onclick="select">Select Rows</button>
<button @onclick="clear">Clear Selection</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSelectionSettings Mode="SelectionMode.Row"
Type="Syncfusion.Blazor.Grids.SelectionType.Multiple"></GanttSelectionSettin
gs>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void select()
{
this.Gantt.SelectRowsAsync(new double[] {1,2,3});
}
public void clear() {
this.Gantt.ClearSelectionAsync();
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
}
}

```

```
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}

public static List<TaskData> GetTaskCollection() {
    List<TaskData> Tasks = new List<TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 40
                },
            })
        },
    }
}
```

```

new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### Get Selected Row Indexes and Records

You can get the selected row indexes by using the `GetSelectedRowIndexesAsync` method. And by using `GetSelectedRecordsAsync` method, you can get the selected record details.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px">
    <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
    </GanttTaskFields>
    <GanttSelectionSettings Mode="SelectionMode.Row"
Type="Syncfusion.Blazor.Grids.SelectionType.Multiple"></GanttSelectionSettin
gs>
    <GanttEvents TValue="TaskData" RowSelected="rowSelect"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void rowSelect(RowSelectEventArgs<TaskData> args)
{
    Console.WriteLine(this.Gantt.GetSelectedRowIndexesAsync());
    Console.WriteLine(this.Gantt.GetSelectedRecordsAsync());
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {

```

```
new TaskData() {
    TaskId = 1,
    TaskName = "Project initiation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 3,
            TaskName = "Perform soil test",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "4",
            Progress = 40,
        },
        new TaskData() {
            TaskId = 4,
            TaskName = "Soil test approval",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30
        },
    })
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
})
```



```

    }
    };
    return Tasks;
  }
}

```

## Filtering in Blazor Gantt Chart Component

Filtering allows you to view specific or related records based on filter criteria. This can be done in the Gantt Chart component by using the filter menu and toolbar search. To enable filtering in the Gantt Chart component, set the `AllowFiltering` to `true`. Menu filtering support can be configured using the `GanttFilterSettings` property and toolbar searching can be configured using the `GanttSearchSettings` property.

### Menu Filtering

The [Blazor Gantt Chart](#) component provides menu-filtering support for each column. You can enable the filter menu by setting the `AllowFiltering` to `true`. 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.

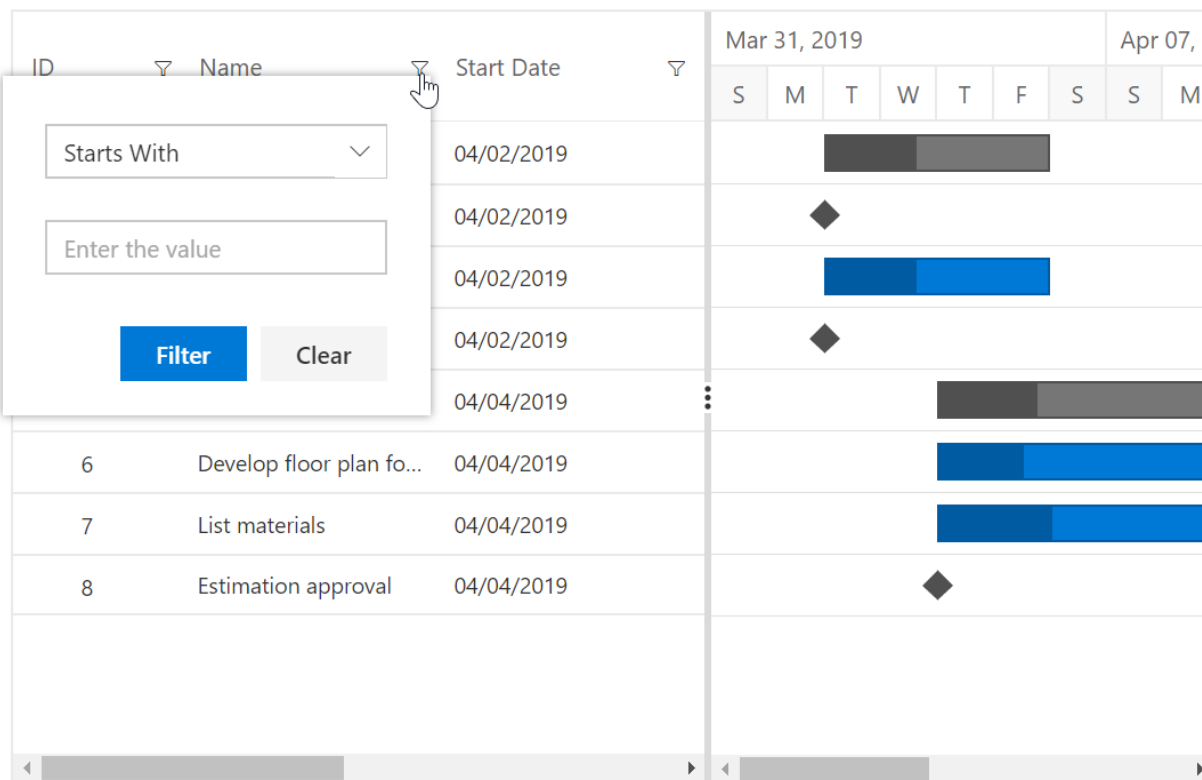
### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowFiltering="true">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
    List<TaskData> Tasks = new List<TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),

```

```
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
},
})
}
};
return Tasks;
}
}
```



The `AllowFiltering` property should be set to `true` to enable the filter menu. Setting the `GanttColumn.AllowFiltering` property to `false` prevents rendering the filter menu for a particular column.

#### Filter Hierarchy Modes

The Gantt Chart supports a set of filtering modes with the `GanttFilterSettings.HierarchyMode` property. The following are the types of filter hierarchy modes available in the Gantt Chart component:

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

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowFiltering="true">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttFilterSettings HierarchyMode
="FilterHierarchyMode.None"></GanttFilterSettings>
```

```
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
```

```

StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
},
})
}
};
return Tasks;
}
}

```

### Initial Filter

To apply the filter at initial rendering, set the filter **Predicate** collections in the **GanttFilterSettings.Columns** property.

### ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowFiltering="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttFilterSettings Columns="@FilterColumns"></GanttFilterSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
public List<PredicateModel> FilterColumns { get; set; } = new
List<PredicateModel>
{
new PredicateModel() { Field = "TaskName", MatchCase=false, Operator =
Operator.StartsWith, Predicate = "and", Value = "Identify" },
new PredicateModel() { Field = "TaskId", MatchCase=false, Operator =
Operator.Equal, Predicate = "and", Value = 2 }
};
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData

```

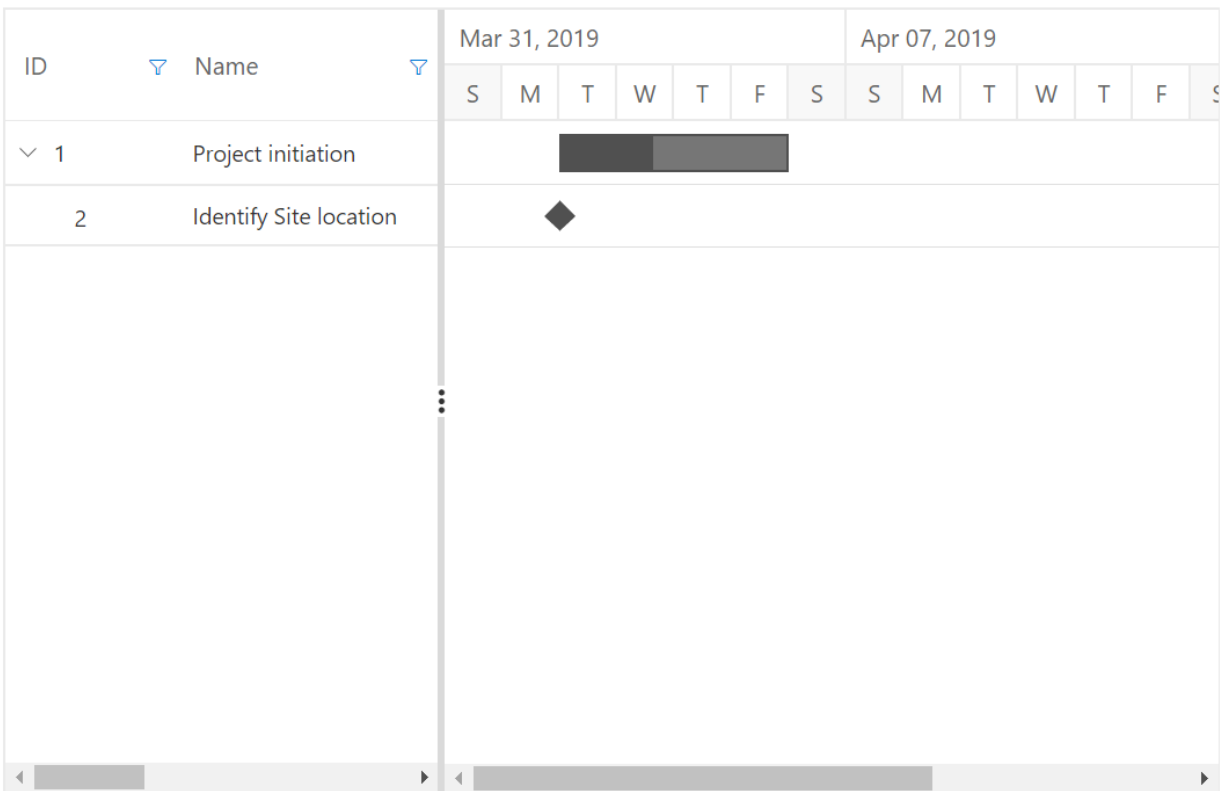
```
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}

public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
```

```

Duration = "3",
Progress = 40,
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
},
})
}
};
return Tasks;
}
}

```



### Filter Operators

The filter operator for a column can be defined in the `GanttFilterSettings.Columns.Operator` property.

The available operators and their supported data types are:

Operator	Description	Supported Types
----------	-------------	-----------------

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 the values that are not equal to the specified value.

By default, the `GanttSearchSettings.Operator` value is *contains*.

[Search by external button](#)

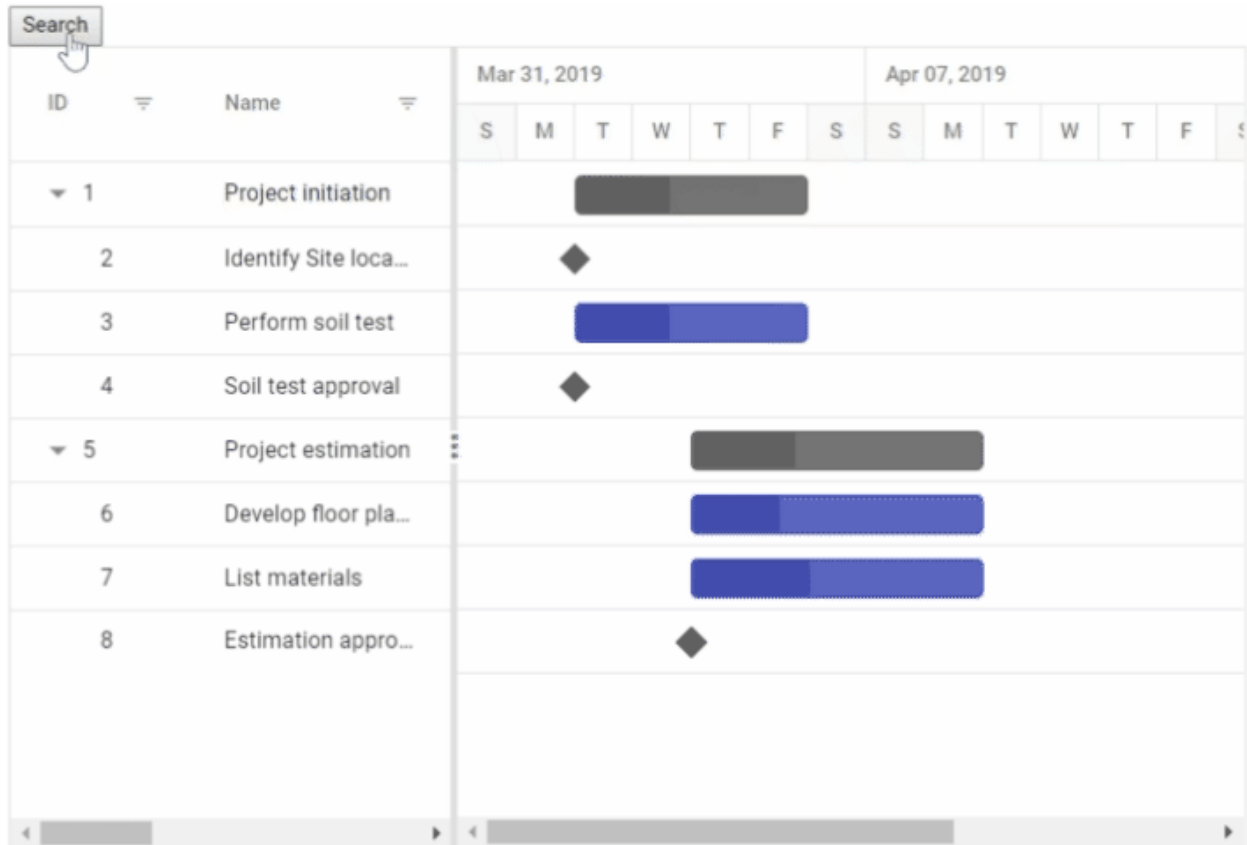
To search the Gantt Chart records from an external button, invoke the `SearchAsync` method.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<button @onclick="Search">Search</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px" AllowFiltering="true">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void Search()
{
  this.Gantt.SearchAsync("Perform");
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
  List<TaskData> Tasks = new List<TaskData>() {
    new TaskData() {
      TaskId = 1,
      TaskName = "Project initiation",
      StartDate = new DateTime(2019, 04, 02),
      EndDate = new DateTime(2019, 04, 21),
      SubTasks = (new List<TaskData> () {
        new TaskData() {
          TaskId = 2,
          TaskName = "Identify Site location",
          StartDate = new DateTime(2019, 04, 02),
          Duration = "0",
          Progress = 30,
        },
        new TaskData() {
          TaskId = 3,
```



```
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
},
})
}
};
return Tasks;
}
}
```



You should set the `AllowFiltering` property to `true` for searching the content externally.

#### Search Specific Columns

By default, the Gantt Chart component searches all the columns. You can search specific columns by defining the specific columns' field names in the `GanttSearchSettings.Fields` property.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
Toolbar="@ (new List<string>() { "Search" })">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSearchSettings Fields="@ (new string[] { "TaskName", "Duration" })">
</GanttSearchSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
}
```

```
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 40,
                },
            })
        },
    }
}
```

```

new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
},
})
}
};
return Tasks;
}
}

```

In above sample, you can search only **TaskName** and **Duration** column values.

#### *Clear Search by External Button*

You can pass the **empty** string to **SearchAsync** method, to clear the searched Gantt records from external button.

#### **ASPX-CS**

```

@using Syncfusion.Blazor.Gantt
<button @onclick="Clear">Clear Search</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px" Toolbar="@ (new List<string>() { "Search" })">
    <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
    EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
    </GanttTaskFields>
    <GanttSearchSettings Fields="@ (new string[] { "TaskName" })"
    Operator="Operator.Contains"
    Key="List" IgnoreCase="true"></GanttSearchSettings>
</SfGantt>
@code{
    public SfGantt<TaskData> Gantt;
    public List<TaskData> TaskCollection { get; set; }
    protected override void OnInitialized()
    {
        this.TaskCollection = GetTaskCollection();
    }
    public void Clear()
    {
        this.Gantt.SearchAsync("");
    }
    public class TaskData
    {
        public int TaskId { get; set; }
        public string TaskName { get; set; }
        public DateTime StartDate { get; set; }
        public DateTime EndDate { get; set; }
        public string Duration { get; set; }
        public int Progress { get; set; }
        public List<TaskData> SubTasks { get; set; }
    }
    public static List<TaskData> GetTaskCollection() {
        List<TaskData> Tasks = new List<TaskData> () {

```

```
new TaskData() {
    TaskId = 1,
    TaskName = "Project initiation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 3,
            TaskName = "Perform soil test",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "4",
            Progress = 40,
        },
        new TaskData() {
            TaskId = 4,
            TaskName = "Soil test approval",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30,
        },
    })
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40,
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        },
    })
}
```

```

    }
    };
    return Tasks;
  }
}

```

## Managing Tasks in Blazor Gantt Chart Component

The [Blazor Gantt Chart](#) component has options to dynamically insert, delete, and update tasks in a project. The primary key column is necessary to manage the tasks and perform CRUD operations in Gantt Chart. To define the primary key, set the `GanttColumn.IsPrimaryKey` property to `true` in the column.

### Adding New Tasks

Tasks can be dynamically added to the Gantt Chart project by enabling the `GanttEditSettings.AllowAdding` property.

#### Toolbar

A row can be added to the Gantt Chart component from the toolbar while the `GanttEditSettings.AllowAdding` property is set to `true`. After clicking the toolbar add icon, you should provide the task information in the add dialog.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Toolbar="@ (new List<string>() { "Add"
})" Height="450px" Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttEditSettings AllowAdding="true"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
  List<TaskData> Tasks = new List<TaskData> () {
    new TaskData() {
      TaskId = 1,
      TaskName = "Project initiation",
      StartDate = new DateTime(2019, 04, 02),
      EndDate = new DateTime(2019, 04, 21),
      SubTasks = (new List<TaskData> () {

```

```
new TaskData() {
    TaskId = 2,
    TaskName = "Identify Site location",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "0",
    Progress = 30,
},
new TaskData() {
    TaskId = 3,
    TaskName = "Perform soil test",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "4",
    Progress = 40,
},
new TaskData() {
    TaskId = 4,
    TaskName = "Soil test approval",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "0",
    Progress = 30
},
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
},
};
return Tasks;
}
```

ID	Name
1	Project in
2	Identify S
3	Perform s
4	Soil test a
5	Project es
6	Develop f
7	List mater
8	Estimation

By default, a new row will be added to the top most row in the Gantt Chart component.

#### Context Menu

A row can also be added above, below or child of the selected row by using context menu support. For this, we need to enable the property `EnableContextMenu`.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px"
EnableContextMenu="true" Width="900px" HighlightWeekends="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
Dependency="Predecessor" ParentID="ParentId"></GanttTaskFields>
<GanttEditSettings AllowAdding="true"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public double Progress { get; set; }
public string Predecessor { get; set; }
public int? ParentId { get; set; }
}
public static List<TaskData> GetTaskCollection()
```



```
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
ParentId = 1
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "3",
Progress = 30,
Predecessor = "2",
ParentId = 1
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
ParentId = 5
},
new TaskData() {
TaskId = 8,
```

```

TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
Predecessor = "6",
ParentId = 5
}
};
return Tasks;
}
}

```

### Using method

You can add rows to the Gantt Chart component dynamically using the `AddRecordAsync` method and you can define the add position of the default new record by using the `RowPosition` property. You can also pass the `RowIndex` as an additional parameter.

- Top of all the rows.
- Bottom to all the existing rows.
- Above the selected row.
- Below the selected row.
- As child to the selected row.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<button onclick="AddRow">Add Row</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId">
  </GanttTaskFields>
  <GanttEditSettings AllowAdding="true"></GanttEditSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public TaskData record = new TaskData() { TaskId = 9, TaskName = "New Added
Record", StartDate = new DateTime(2019, 04, 02), Duration = "3", Progress =
50};
public void AddRow()
{
  this.Gantt.AddRecordAsync(record, 2, RowPosition.Below);
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
}

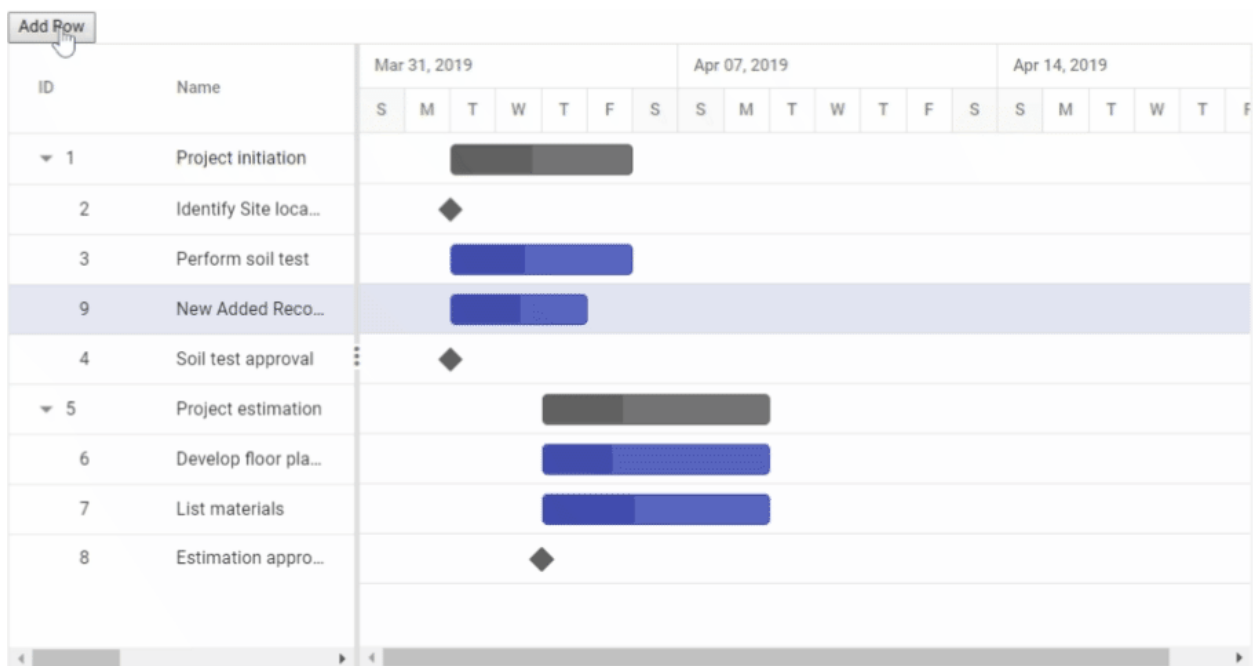
```

```
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
ParentId = 1
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
```

```

ParentId = 5
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
    ParentId = 5
}
};
return Tasks;
}
}

```



### Editing Tasks

The editing feature can be enabled in the Gantt Chart component by enabling the `GanttEditSettings.AllowEditing` and `GanttEditSettings.AllowTaskbarEditing` properties.

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

- Cell
- Dialog
- Taskbar

### Cell Editing

By setting the edit mode to auto using the `GanttEditSettings.Mode` property, the tasks can be edited by double-clicking the Tree Grid cells.

The following code example shows you how to enable the cell editing in Gantt Chart component.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowEditing="true"
Mode="Syncfusion.Blazor.Gantt.EditMode.Auto"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,

```

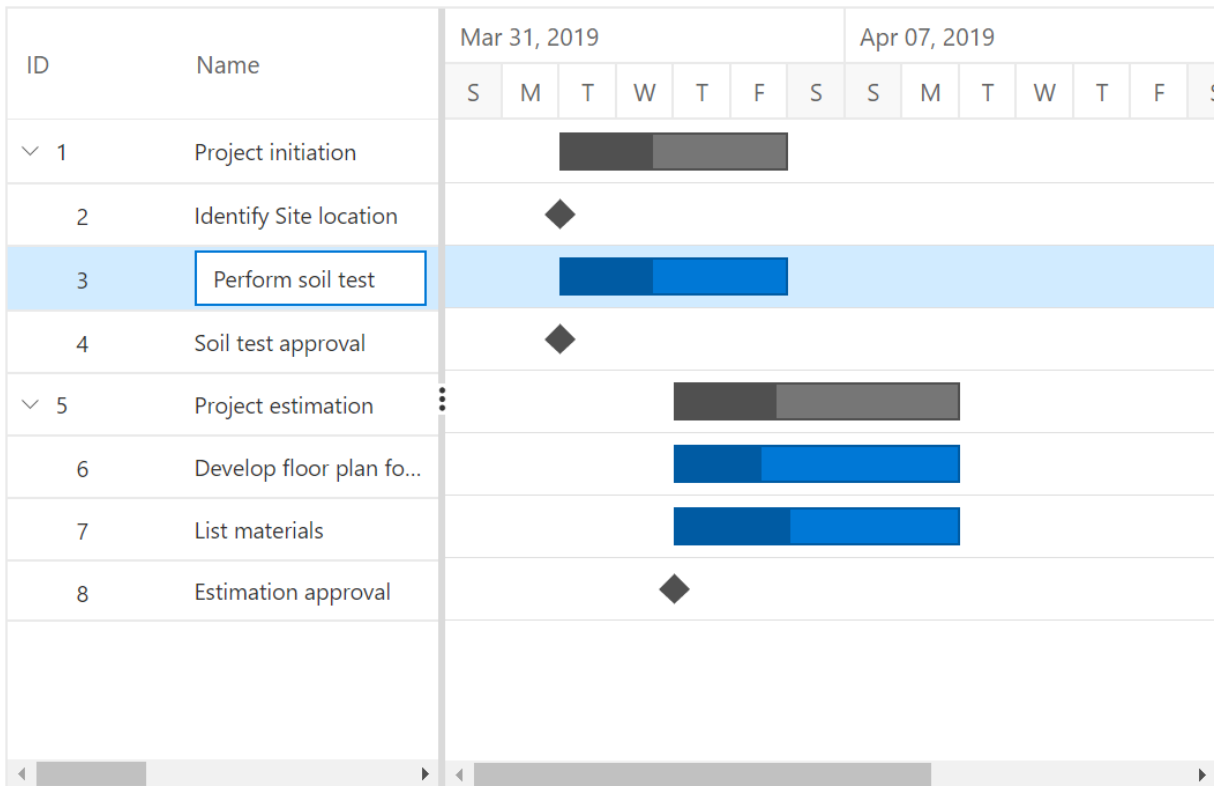
```
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 6,
        TaskName = "Develop floor plan for estimation",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 7,
        TaskName = "List materials",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 40
    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "0",
        Progress = 30,
    }
})
};
return Tasks;
}
```

---

When the edit mode is set to **Auto**, double-clicking on the Tree Grid side changes, the cells to editable mode. Double-clicking on the chart side opens, the edit dialog for editing the task details.

---

double click action on Tree Grid side



double click action on chart side

Task Information

General

ID

3

Name

Perform soil test

Start Date

04/02/2019

End Date

04/05/2019

Duration

4 days

Progress

40

Save

Cancel

*Dialog Editing*

Modify the task details through the edit dialog by setting the `GanttEditSettings.Mode` as `Dialog`.

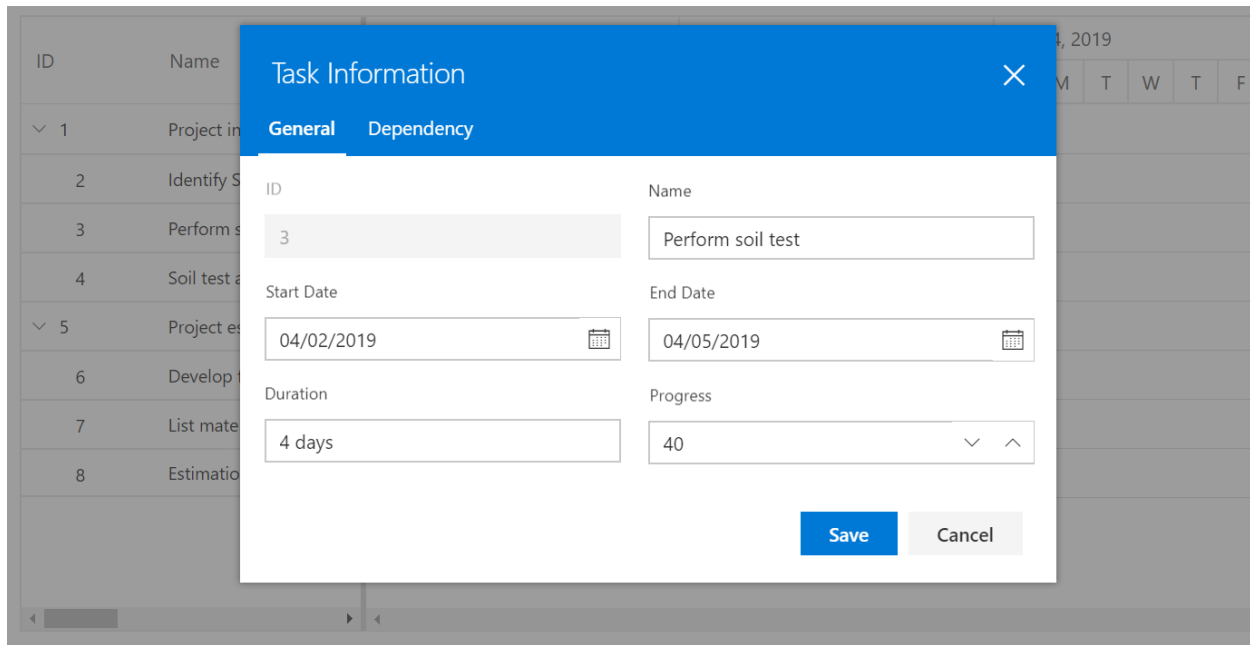
**ASPX-CS**

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowEditing="true"
Mode="Syncfusion.Blazor.Gantt.EditMode.Dialog"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
}
```



```
    },  
    })  
  },  
  new TaskData() {  
    TaskId = 5,  
    TaskName = "Project estimation",  
    StartDate = new DateTime(2019, 04, 02),  
    EndDate = new DateTime(2019, 04, 21),  
    SubTasks = (new List <TaskData> () {  
      new TaskData() {  
        TaskId = 6,  
        TaskName = "Develop floor plan for estimation",  
        StartDate = new DateTime(2019, 04, 04),  
        Duration = "3",  
        Progress = 30,  
      },  
      new TaskData() {  
        TaskId = 7,  
        TaskName = "List materials",  
        StartDate = new DateTime(2019, 04, 04),  
        Duration = "3",  
        Progress = 40  
      },  
      new TaskData() {  
        TaskId = 8,  
        TaskName = "Estimation approval",  
        StartDate = new DateTime(2019, 04, 04),  
        Duration = "0",  
        Progress = 30,  
      }  
    })  
  },  
  };  
  return Tasks;  
}
```

**Note:** In dialog editing mode, the edit dialog appears when the Tree Grid or Gantt chart sides are double-clicked.



### Sections or Tabs in Dialog

In the Gantt Chart dialog, you can define the required tabs or editing sections using the `GanttAddDialogFields` and `GanttEditDialogFields` properties. Every tab is defined using the `GanttAddDialogField.Type` or `GanttEditDialogField.Type` property.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Toolbar="@ (new List<string>() { "Add",
"Edit" })" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks"
  ResourceInfo="ResourceId" Notes="Notes"
  Dependency="Predecessor">
  </GanttTaskFields>
  <GanttResourceFields TResources="TaskResources" Name="ResourceName"
  Id="ResourceId" Resources="@ResourceCollection"></GanttResourceFields>
  <GanttEditSettings AllowAdding="true" AllowEditing="true"
  Mode="Syncfusion.Blazor.Gantt.EditMode.Dialog">
  </GanttEditSettings>
  <GanttEditDialogFields>
  <GanttEditDialogField Type="GanttDialogFieldType.General"
  HeaderText="General">
  </GanttEditDialogField>
  <GanttEditDialogField
  Type="GanttDialogFieldType.Dependency"></GanttEditDialogField>
  <GanttEditDialogField
  Type="GanttDialogFieldType.Resources"></GanttEditDialogField>
  <GanttEditDialogField
  Type="GanttDialogFieldType.Notes"></GanttEditDialogField>
  </GanttEditDialogFields>
  <GanttAddDialogFields>
  <GanttAddDialogField Type="GanttDialogFieldType.General" HeaderText="General
  Tab"></GanttAddDialogField>
```

```

<GanttAddDialogField
Type="GanttDialogFieldType.Dependency"></GanttAddDialogField>
</GanttAddDialogFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
public List<TaskResources> ResourceCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
this.ResourceCollection = GetResourceCollections();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
public List<TaskResources> ResourceId { get; set; }
public string Notes { get; set; }
public string Predecessor { get; set; }
}
public class TaskResources
{
public int ResourceId { get; set; }
public string ResourceName { get; set; }
}
public static List <TaskResources> GetResourceCollections() {
List <TaskResources> Resources = new List <TaskResources> () {
new TaskResources() {
ResourceId = 1,
ResourceName = "Martin Tamer"
},
new TaskResources() {
ResourceId = 2,
ResourceName = "Rose Fuller"
},
new TaskResources() {
ResourceId = 3,
ResourceName = "Margaret Buchanan"
},
new TaskResources() {
ResourceId = 4,
ResourceName = "Fuller King"
},
new TaskResources() {
ResourceId= 5,
ResourceName= "Davolio Fuller"
},
};
return Resources;
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {

```

```
new TaskData() {
    TaskId = 1,
    TaskName = "Project initiation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30,
            ResourceId = new List<TaskResources>(){ new TaskResources() { ResourceId=1}
        },
            Notes = "Measure the total property area allotted for construction"
        },
        new TaskData() {
            TaskId = 3,
            TaskName = "Perform soil test",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "4",
            Predecessor = "2",
            ResourceId = new List<TaskResources>(){ new TaskResources() { ResourceId=2},
            new TaskResources() { ResourceId=3} },
            Notes = "Obtain an engineered soil test of lot where construction is
            planned.From an engineer or company specializing in soil testing"
        },
        new TaskData() {
            TaskId = 4,
            TaskName = "Soil test approval",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30,
            Predecessor = "3"
        },
    })
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
            Predecessor = "4",
            ResourceId = new List<TaskResources>(){ new TaskResources() {
            ResourceId=4}},
            Notes = "Develop floor plans and obtain a materials list for estimations"
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
```

```
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Predecessor = "6",
Notes = ""
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Predecessor = "7",
ResourceId = new List<TaskResources>() { new TaskResources() { ResourceId=1},
new TaskResources() { ResourceId=5} },
Notes = ""
}
})
}
};
return Tasks;
}
}
```

Tabs in Edit Dialog

The screenshot shows a 'Task Information' dialog box with a blue header and a close button (X) in the top right corner. Below the header are four tabs: 'General' (selected), 'Dependency', 'Resources', and 'Notes'. The 'General' tab contains the following fields:

- ID:** A numeric input field with the value '3' and up/down arrow buttons.
- Name:** A text input field with the value 'Perform soil test'.
- Start Date:** A date input field with the value '4/2/2019' and a calendar icon.
- End Date:** A date input field with the value '4/5/2019' and a calendar icon.
- Duration:** A text input field with the value '4 Days'.
- Progress:** A numeric input field with the value '0.00' and up/down arrow buttons.

At the bottom right of the dialog are two buttons: 'Save' (blue) and 'Cancel' (grey).

Tabs in Add Dialog

The screenshot shows a 'New Task' dialog box with a blue header and a white body. It has two tabs: 'General Tab' and 'Dependency'. The 'General Tab' is active and contains the following fields:

- ID:** A text input with the value '9' and up/down arrow buttons.
- Name:** A text input with the value 'New Task 9'.
- Start Date:** A date picker showing '4/2/2019' with a calendar icon.
- End Date:** A date picker showing '4/2/2019' with a calendar icon.
- Duration:** A text input with the value '1 Day'.
- Progress:** A text input with the value '0.00' and up/down arrow buttons.

At the bottom right of the dialog are two buttons: 'Save' (blue) and 'Cancel' (gray).

#### Limiting Data Fields in General Tab

In the Gantt Chart dialog, you can make only specific data source fields visible for editing by using the `GanttAddDialogFields` and `GanttEditDialogFields` properties. The data fields are defined with `GanttEditDialogField.Type` and `GanttEditDialogField.Fields` properties.

**Note:** You can also define the custom fields in the add/edit dialog General tab using the `Fields` property.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Toolbar="@ (new List<string>() { "Add",
"Edit" })" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks"
  ResourceInfo="ResourceId" Notes="Notes"
  Dependency="Predecessor">
  </GanttTaskFields>
  <GanttResourceFields TResources="TaskResources" Name="ResourceName"
  Id="ResourceId" Resources="@ResourceCollections"></GanttResourceFields>
  <GanttEditSettings AllowAdding="true" AllowEditing="true"
  Mode="Syncfusion.Blazor.Gantt.EditMode.Dialog">
  </GanttEditSettings>
  <GanttColumns>
  <GanttColumn Field="TaskId" Width="100"></GanttColumn>
  <GanttColumn Field="TaskName"></GanttColumn>
  <GanttColumn Field="StartDate"></GanttColumn>
  <GanttColumn Field="Duration"></GanttColumn>
```

```

<GanttColumn Field="Progress"></GanttColumn>
</GanttColumns>
<GanttEditDialogFields>
<GanttEditDialogField Type="GanttDialogFieldType.General"
HeaderText="General"
Fields="@ (new string[] { "TaskId", "TaskName", "Duration"
}) "></GanttEditDialogField>
<GanttEditDialogField
Type="GanttDialogFieldType.Notes"></GanttEditDialogField>
</GanttEditDialogFields>
<GanttAddDialogFields>
<GanttAddDialogField Type="GanttDialogFieldType.General" HeaderText="General
Tab"
Fields="@ (new string[] { "TaskId", "TaskName", "Duration"
}) "></GanttAddDialogField>
<GanttAddDialogField
Type="GanttDialogFieldType.Dependency"></GanttAddDialogField>
</GanttAddDialogFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
public List<TaskResources> ResourceCollections { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
this.ResourceCollections = GetResourceCollections();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
public List<TaskResources> ResourceId { get; set; }
public string Notes { get; set; }
public string Predecessor { get; set; }
}
public class TaskResources
{
public int ResourceId { get; set; }
public string ResourceName { get; set; }
}
public static List <TaskResources> GetResourceCollections() {
List <TaskResources> Resources = new List <TaskResources> () {
new TaskResources() {
ResourceId = 1,
ResourceName = "Martin Tamer"
},
new TaskResources() {
ResourceId = 2,
ResourceName = "Rose Fuller"
},
new TaskResources() {
ResourceId = 3,

```

```

ResourceName = "Margaret Buchanan"
},
new TaskResources() {
ResourceId = 4,
ResourceName = "Fuller King"
},
new TaskResources() {
ResourceId= 5,
ResourceName= "Davolio Fuller"
},
};
return Resources;
}

public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ResourceId = new List<TaskResources>(){ new TaskResources() { ResourceId=1}
},
Notes = "Measure the total property area allotted for construction"
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Predecessor = "2",
ResourceId = new List<TaskResources>(){ new TaskResources() { ResourceId=2},
new TaskResources() { ResourceId=3} },
Notes = "Obtain an engineered soil test of lot where construction is
planned.From an engineer or company specializing in soil testing"
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
Predecessor = "3"
},
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),

```



```
SubTasks = (new List<TaskData> () {  
    new TaskData() {  
        TaskId = 6,  
        TaskName = "Develop floor plan for estimation",  
        StartDate = new DateTime(2019, 04, 04),  
        Duration = "3",  
        Progress = 30,  
        Predecessor = "4",  
        ResourceId = new List<TaskResources>() { new TaskResources() {  
            ResourceId=4}},  
        Notes = "Develop floor plans and obtain a materials list for estimations"  
    },  
    new TaskData() {  
        TaskId = 7,  
        TaskName = "List materials",  
        StartDate = new DateTime(2019, 04, 04),  
        Duration = "3",  
        Predecessor = "6",  
        Notes = ""  
    },  
    new TaskData() {  
        TaskId = 8,  
        TaskName = "Estimation approval",  
        StartDate = new DateTime(2019, 04, 04),  
        Duration = "0",  
        Predecessor = "7",  
        ResourceId = new List<TaskResources>() { new TaskResources() { ResourceId=1},  
            new TaskResources() { ResourceId=5} },  
        Notes = ""  
    }  
})  
};  
return Tasks;  
}
```

The following screenshot show the output of above code example.

The image shows a 'Task Information' dialog box with a blue header and a close button (X) in the top right corner. Below the header are two tabs: 'General' (selected) and 'Notes'. The 'General' tab contains three input fields: 'ID' with the value '3', 'Name' with the value 'Perform soil test', and 'Duration' with the value '4 Days'. At the bottom right of the dialog are two buttons: 'Save' (blue) and 'Cancel' (grey).

### Taskbar Editing

Modify the task details through user interaction, such as resizing and dragging the taskbar, by enabling the `GanttEditSettings.AllowTaskbarEditing` property.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
```

```
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 40
                },
                new TaskData() {
                    TaskId = 8,
                    TaskName = "Estimation approval",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "0",
```

```

Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### Task Dependencies

In the Gantt Chart component, you can update the dependencies between tasks and link the tasks interactively. The task dependencies can be mapped from the data source using the `GanttTaskFields.Dependency` property.

You can update the task dependencies using the following ways:

- Edit dialog: Create or remove the task dependencies using the `Dependency` tab in the edit dialog.
- Cell editing: Create or remove the task links using cell editing.

The following code example demonstrates how to enable task dependency editing in the Gantt chart using the `EditSettings` property.

### ASPX-CS

```

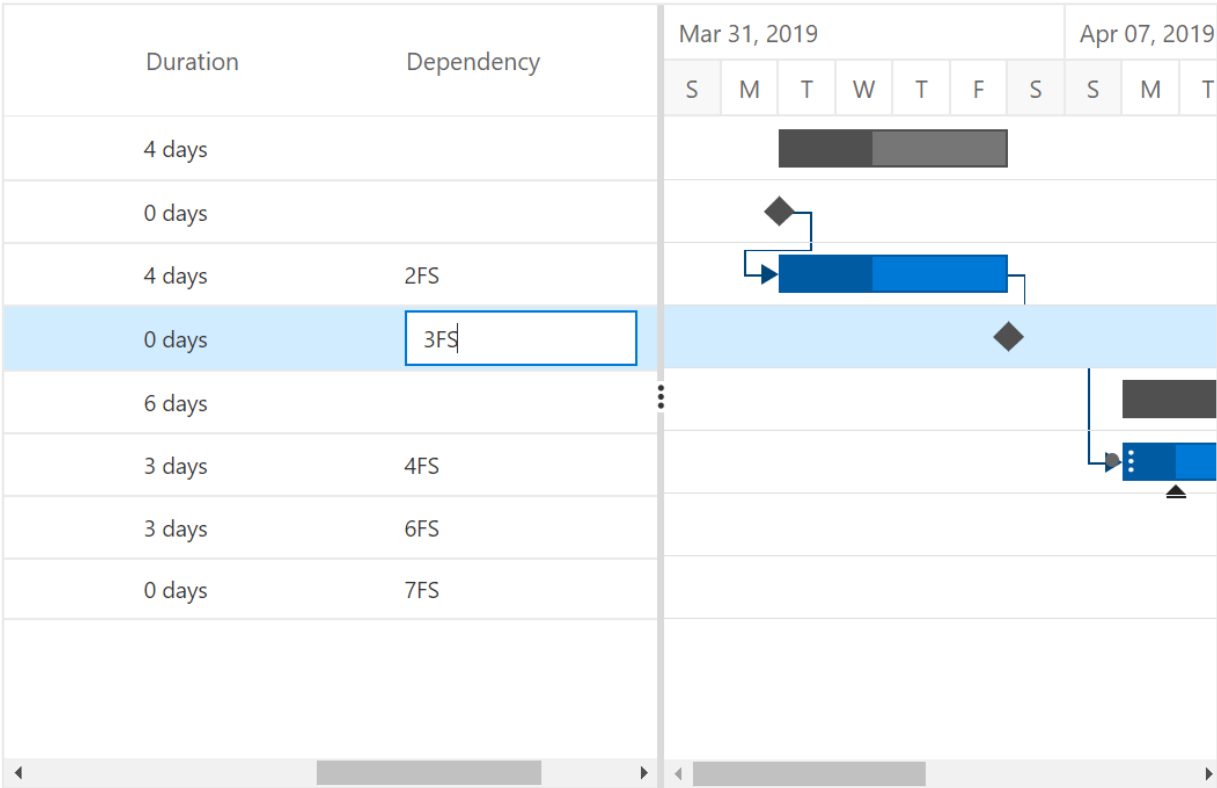
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks"
Dependency="Predecessor">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true" AllowEditing="true"
Mode="Syncfusion.Blazor.Gantt.EditMode.Auto"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public string Predecessor { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
List<TaskData> Tasks = new List<TaskData> ();
Tasks.Add(new TaskData() {

```

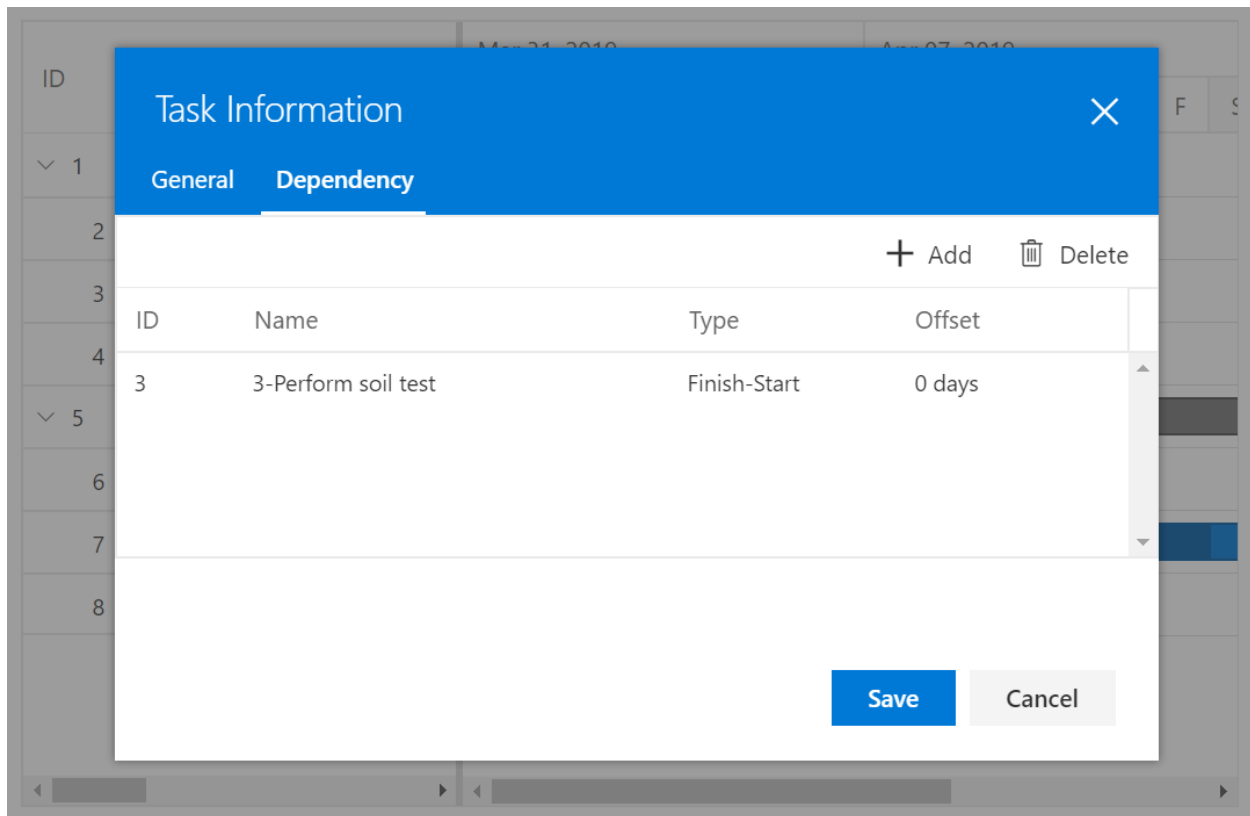
```
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 2,
        TaskName = "Identify Site location",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "0",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 3,
        TaskName = "Perform soil test",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "4",
        Progress = 40,
        Predecessor = "2"
    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "0",
        Progress = 30,
        Predecessor = "3"
    },
})
});
Tasks.Add(new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
            Predecessor = "4"
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40,
            Predecessor = "6"
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
```

```
Progress = 30,  
Predecessor = "7"  
},  
})  
});  
return Tasks;  
}  
}
```

Updating with cell Edit



Updating with Dialog



#### Update Task Values using Method

Tasks' value can be dynamically updated by using the `UpdateRecordByIdAsync` method. You can call this method on any custom action. The following code example shows how to use this method to update a task.

Using the `UpdateRecordByIdAsync` method, you cannot update the task ID value.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<button onclick="UpdateRecord">Update Task 3</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttEditSettings AllowEditing="true"></GanttEditSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void UpdateRecord()
{
  this.Gantt.UpdateRecordByIdAsync(new TaskData() { TaskId = 3, TaskName =
  "Updated by ID value", Progress = 60});
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
```

```
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}

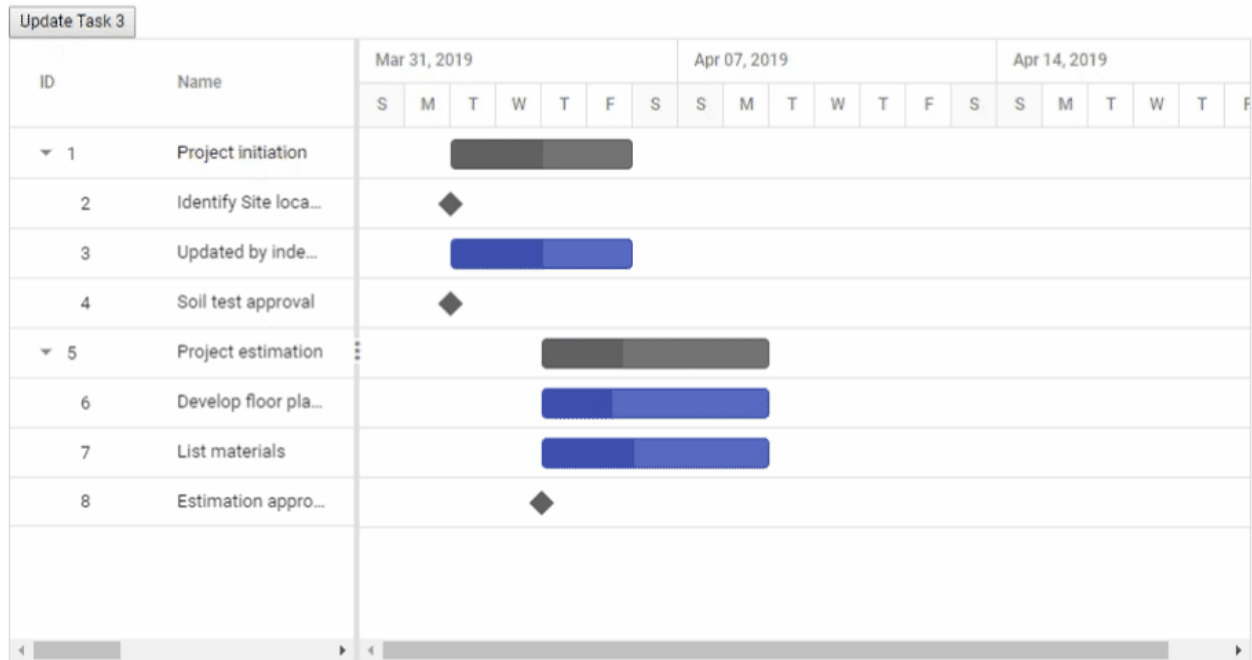
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
```



```

TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```



### Cell Edit Type and its Params

The `GanttColumn.EditType` is used to customize the edit type of the particular column. You can set the `GanttColumn.EditType` based on data type of the column.

- [NumericTextBox](#) component for integers, double, and decimal data types.
- [TextBox](#) component for string data type.
- [DropDownList](#) component for list data type.
- [DatePicker](#) component for date values.
- [DateTimePicker](#) component for datetime type.
- [Checkbox](#) component for boolean type.

Also, you can customize model of the `GanttColumn.EditType` component through the `GanttColumn.EditorSettings`.

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

Component | Example

[NumericTextBox](#) | `@(new { @params = new { format = "n" } })`

[TextBox](#) | -

[DropDownList](#) | `@(new { @params = new { value = "Germany" } })`

[DatePicker](#) | `@(new { @params = new { format = "yyyy-MM-dd" } })`

[DateTimePicker](#) | `@(new { @params = new { strictMode = true } })`

[Checkbox](#) | `@(new { @params = new { checked = true } })`

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="900px" Toolbar="@ (new List<string>() { "Add", "Edit", "Delete",
"Update", "Cancel" }) ">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" Width="100"></GanttColumn>
<GanttColumn Field="TaskName" Width="250"></GanttColumn>
<GanttColumn Field="StartDate" EditorSettings="DateParams"
Type="Syncfusion.Blazor.Grids.ColumnType.DateTime"
EditType="Syncfusion.Blazor.Grids.EditType.DateTimePickerEdit"></GanttColumn
>
<GanttColumn Field="Duration"></GanttColumn>
<GanttColumn Field="Progress" EditorSettings="NumericParams"
EditType="Syncfusion.Blazor.Grids.EditType.NumericEdit"></GanttColumn>
</GanttColumns>
<GanttEditSettings AllowEditing="true"
AllowAdding="true"></GanttEditSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
public Syncfusion.Blazor.Grids.NumericEditCellParams NumericParams = new
Syncfusion.Blazor.Grids.NumericEditCellParams()
{
Params = new Syncfusion.Blazor.Inputs.NumericTextBoxModel<object>() { Format
= "N2" }
};
public Syncfusion.Blazor.Grids.DateEditCellParams DateParams = new
Syncfusion.Blazor.Grids.DateEditCellParams()
{
Params = new Syncfusion.Blazor.Calendars.DatePickerModel() { Format = "d" }
};
protected override void OnInitialized()
{
}
```

```

this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
            new DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 21), SubTasks =
            (new List<TaskData> () {
                new TaskData() { TaskId = 2, TaskName = "Identify Site location", StartDate =
                    new DateTime(2019, 04, 02), Duration = "0", Progress = 30},
                new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
                    DateTime(2019, 04, 02), Duration = "4", Progress = 40},
                new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
                    new DateTime(2019, 04, 02), Duration = "0", Progress = 30} })
            },
        new TaskData() { TaskId = 5, TaskName = "Project estimation", StartDate =
            new DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 21), SubTasks =
            (new List<TaskData> () {
                new TaskData() { TaskId = 6, TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04), Duration = "3", Progress = 30},
                new TaskData() { TaskId = 7, TaskName = "List materials", StartDate = new
                    DateTime(2019, 04, 04), Duration = "3", Progress = 40},
                new TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate =
                    new DateTime(2019, 04, 04), Duration = "0", Progress = 30} })
            }
    };
    return Tasks;
}
}

```

If edit type is not defined in the column, then it will be considered as the **StringEdit** type (Textbox component).

#### Cell Edit Template

The cell edit template is used to add a custom component for a particular column when the column is edited.

The following code example describes, how to define the Edit template for a particular column.

#### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.DropDowns;
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
    Toolbar="@ (new List<string>() { "Add", "Cancel", "Edit", "Update", }) ">

```

```

<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId">
</GanttTaskFields>
<GanttEditSettings AllowAdding="true"
AllowEditing="true"></GanttEditSettings>
<GanttColumns>
<GanttColumn Field="TaskId" IsPrimaryKey=true></GanttColumn>
<GanttColumn Field="TaskName">
<EditTemplate>
@{
var task = (context as TaskData);
}
<SfDropDownList @ref="dropdown" ID="TaskName" @bind-Value="task.TaskName"
TItem="string" TValue="string" DataSource="@DropDownData"></SfDropDownList>
</EditTemplate>
</GanttColumn>
<GanttColumn Field="Duration"></GanttColumn>
<GanttColumn Field="StartDate"></GanttColumn>
<GanttColumn Field="Progress"></GanttColumn>
</GanttColumns>
<GanttEditSettings AllowAdding="true" AllowEditing=true
AllowTaskbarEditing=true></GanttEditSettings>
<GanttEvents OnActionBegin="ActionBeginHandler"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
SfDropDownList<string, string> dropdown;
public void ActionBeginHandler(GanttActionEventArgs<TaskData> args)
{
if (args.RequestType.Equals(Syncfusion.Blazor.Gantt.Action.BeforeSave))
{
var data = args.Data as TaskData;
data.TaskName = dropdown.Value.ToString();
}
}
public List<TaskData> TaskCollection { get; set; }
public List<string> DropDownData { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
this.DropDownData = TaskCollection.Select(s =>
s.TaskName).Distinct().ToList();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {

```

```
new TaskData() {
    TaskId = 1,
    TaskName = "Project initiation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
    TaskId = 2,
    TaskName = "Identify Site location",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "4",
    Progress = 30,
    ParentId = 1
},
new TaskData() {
    TaskId = 3,
    TaskName = "Perform soil test",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "4",
    Progress = 40,
    ParentId = 1
},
new TaskData() {
    TaskId = 4,
    TaskName = "Soil test approval",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "4",
    Progress = 30,
    ParentId = 1
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
    TaskId = 6,
    TaskName = "Develop floor plan for estimation",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "3",
    Progress = 30,
    ParentId = 5
},
new TaskData() {
    TaskId = 7,
    TaskName = "List materials",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "3",
    Progress = 40,
    ParentId = 5
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "3",
```

```

Progress = 30,
ParentId = 5
};
return Tasks;
}
}

```

### Disable Editing for Particular Column

You can disable editing for particular columns, by using the `GanttColumn.AllowEditing` property.

In the following demo, editing is disabled for the `TaskName` column.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="900px" Toolbar="@ (new List<string>() { "Edit" })">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" Width="100"></GanttColumn>
<GanttColumn Field="TaskName" Width="250"
AllowEditing="false"></GanttColumn>
<GanttColumn Field="StartDate"></GanttColumn>
<GanttColumn Field="Duration"></GanttColumn>
<GanttColumn Field="Progress"></GanttColumn>
</GanttColumns>
<GanttEditSettings AllowEditing="true"></GanttEditSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
List<TaskData> Tasks = new List<TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {

```

```
new TaskData() {
    TaskId = 2,
    TaskName = "Identify Site location",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "0",
    Progress = 30,
},
new TaskData() {
    TaskId = 3,
    TaskName = "Perform soil test",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "4",
    Progress = 40,
},
new TaskData() {
    TaskId = 4,
    TaskName = "Soil test approval",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "0",
    Progress = 30,
},
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40,
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        },
    })
},
};
return Tasks;
}
```

## Deleting Tasks

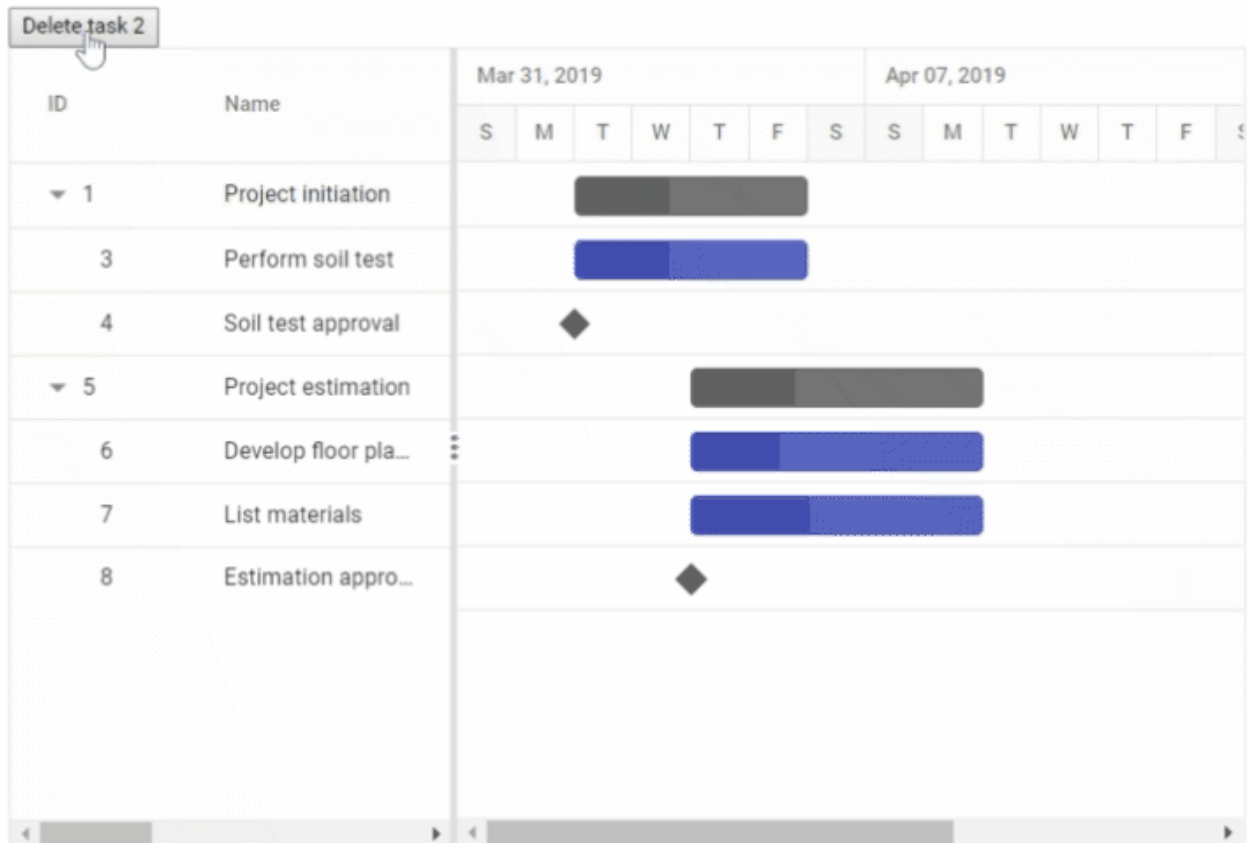
A task delete option in the Gantt Chart component can be enabled by enabling the `GanttEditSettings.AllowDeleting` property. Tasks can be deleted by clicking the delete toolbar item or using the `DeleteRecordAsync` method. You can call this method dynamically on any custom actions like button click. The following code example shows how to enable the delete option in the Gantt Chart component.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<button @onclick="DeleteRow">Delete task 2</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowDeleting="true"></GanttEditSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void DeleteRow()
{
this.Gantt.DeleteRecordAsync(2);
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
List<TaskData> Tasks = new List<TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
},
},
},
}
```



```
new TaskData() {
    TaskId = 3,
    TaskName = "Perform soil test",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "4",
    Progress = 40,
},
new TaskData() {
    TaskId = 4,
    TaskName = "Soil test approval",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "0",
    Progress = 30
},
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
},
};
return Tasks;
}
```



You should set the `AllowDeleting` value to `true` to delete the record dynamically.

#### *Delete confirmation message*

Delete confirmation message is used to get confirmation from users before deleting a task. This confirmation message can be enabled by setting the `GanttEditSettings.ShowDeleteConfirmDialog` property to `true`.

The following code snippet explains how to enable the delete confirmation message in Gantt Chart.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
Toolbar="@ (new List<string>() { "Delete" }) ">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowDeleting="true"
ShowDeleteConfirmDialog="true"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{

```

```
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}

public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
```

```
Progress = 40
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
}
}))
}
};
return Tasks;
}
```



### Entity Framework

You need to follow the below steps to consume data from the **Entity Framework** in the Gantt Chart component.

**Step 1:** The first step is to create a SQL database in your Blazor project. Please refer this [link](#) to create SQL database.

**Step 2:** Install the below packages for Entity Framework Support using Nuget or Package manager console using the below command.

### BASH

```
Install-Package Microsoft.EntityFrameworkCore.Tools -Version 3.0.0
Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 3.0.0
```

**Step 3:** Create a model class `GanttDataDetails.cs` for the existing database file.

#### **CSHARP**

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System;
namespace MyBlazorApp.Data
{
    public class GanttDataDetails
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        [Key]
        public int Id { get; set; }
        public string Name { get; set; }
        public DateTime? Sdate { get; set; }
        public DateTime? Edate { get; set; }
        public string Progress { get; set; }
        public int? ParentId { get; set; }
        public string Duration { get; set; }
        public string ProjectName { get; set; }
        public DateTime? BaselineStartDate { get; set; }
        public DateTime? BaselineEndDate { get; set; }
        public string Predecessor { get; set; }
        public string Notes { get; set; }
        public string TaskType { get; set; }
        public List<TaskResources> ResourceId { get; set; }
        public string ProjectId { get; set; }
        public bool? IsExpand { get; set; }
    }
}
```

**Step 4:** Create a DB Context class to connect to the Microsoft SQL Server database using the below command in Package Manager Console.

#### **BASH**

```
Server=(localdb)\\MSSQLLocalDB;Database=master;Integrated Security=True
```

#### **CSHARP**

```
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;
namespace GanttEF.Models
{
    public partial class masterContext : DbContext
    {
        public masterContext()
        {
        }
        public masterContext(DbContextOptions<masterContext> options)
            : base(options)
        {
        }
        public virtual DbSet<Table> Table { get; set; }
    }
}
```

```
protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer("Server=(localdb)\\MSSQLLocalDB;Database=master;
Integrated Security=True");
    }
}

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Table>(entity =>
    {
        entity.Property(e => e.Id).ValueGeneratedNever();
        entity.Property(e => e.Edate).HasColumnType("datetime");
        entity.Property(e => e.Name)
            .HasMaxLength(10)
            .IsFixedLength();
        entity.Property(e => e.Progress)
            .HasMaxLength(10)
            .IsFixedLength();
        entity.Property(e => e.Sdate).HasColumnType("datetime");
    });
    OnModelCreatingPartial(modelBuilder);
}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}
```

**Step 5:** Update the connection string in the appsettings.json file.

#### **CSHARP**

```
"ConnectionStrings": {
  "EmployeeDatabase": "Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=\\\"D:\\client-side-gantt-
chart-application-with-entity-
framework\\Gantt_wasm_crud_sample1804339720\\Gantt_wasm_crud_sample\\Shared\\
App_DataNORTHWND.MDF\\\";Integrated Security=True"
}
```

The following sections will give details about the steps needed to be followed when working with server-side and client-side applications individually. You can also find samples attached at the end of each section for server-side applications and client-side applications.

#### *Entity Framework in Server-Side Application*

You need to follow the following steps when working with a server-side application.

#### *Custom Adaptor*

In Gantt Chart, we can fetch data from the SQL database using **Entity Framework** Data Model and the update the changes on CRUD action to the server by using **DataManager** support. To communicate with the remote data, we are using **CustomAdaptor** of **DataManager** property to call the server method. You can know more about **CustomAdaptor** from [here](#). We can populate the datasource in

Gantt from the SQL table using Entity Framework using **Read** method. Please Check the below code snippet to assign the data source to Gantt.

#### ASPX-CS

```
@using GanttEF.Models
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor

<SfGantt ID="GanttExport" TValue="GanttData" Height="450px" Width="700px"
Toolbar="@ (new List<string>() { "Add", "Edit", "Update", "Delete", "Cancel",
"ExpandAll", "CollapseAll" }) ">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor) "
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GanttColumns>
<GanttColumn Field=@nameof(GanttData.Id) Width="100"></GanttColumn>
<GanttColumn Field=@nameof(GanttData.Name) Width="250"></GanttColumn>
<GanttColumn Field=@nameof(GanttData.Sdate)></GanttColumn>
<GanttColumn Field=@nameof(GanttData.Edate)></GanttColumn>
<GanttColumn Field=@nameof(GanttData.Progress)></GanttColumn>
</GanttColumns>
<GanttEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" AllowTaskbarEditing="true"
ShowDeleteConfirmDialog="true"
NewRowPosition="RowPosition.Child"></GanttEditSettings>
<GanttTimelineSettings>
<GanttTopTierSettings Unit="TimelineViewMode.Week" Format="MMM dd,
y"></GanttTopTierSettings>
<GanttBottomTierSettings
Unit="TimelineViewMode.Day"></GanttBottomTierSettings>
</GanttTimelineSettings>
<GanttTaskFields Id="Id" Name="Name" StartDate="Sdate" EndDate="Edate"
Progress="Progress"
ParentID="ParentId"></GanttTaskFields>
</SfGantt>

@code {
// Implementing custom adaptor by extending the DataAdaptor class
public class CustomAdaptor : DataAdaptor
{
masterContext db = new masterContext();
// Performs data Read operation
public override object Read(DataManagerRequest dm, string key = null)
{
IEnumerable<GanttData> DataSource = db.GanttData;
int count = DataSource.Cast<GanttData>().Count();
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
.../////
}
}
```

#### Perform CRUD operation in CustomAdaptor

All the CRUD operations in the Gantt Chart are done through DataManager. The DataManager has an option to bind all the CRUD related data in server-side.

We can do CRUD operations over Gantt data and save the changes into the database. By using **BatchUpdate** method of DataManager, we can communicate with the controller method to update the data source by CRUD operation. In Gantt Chart, the CRUD actions on a task are dependent on other tasks. For example, if you edit the child record on the chart side, the corresponding parent item also will get affected and predecessor dependency task as well get affected. So, all the CRUD operations in Gantt Chart are considered to be batch editing, where you will get all the affected records as collection.

This server method will be triggered for all the CRUD operations like adding, editing, and deleting actions. We can handle each operation separately inside this method with corresponding data received in this method argument.

The following sample code explains you about, how to implement CRUD operations for the custom bounded data.

### ASPX-CS

```
@using GanttEF.Models
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor

<SfGantt ID="GanttExport" TValue="GanttData" Height="450px" Width="700px"
Toolbar="@ (new List<string>() { "Add", "Edit", "Update", "Delete", "Cancel",
"ExpandAll", "CollapseAll" }) ">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor) "
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GanttColumns>
<GanttColumn Field=@nameof(GanttData.Id) Width="100"></GanttColumn>
<GanttColumn Field=@nameof(GanttData.Name) Width="250"></GanttColumn>
<GanttColumn Field=@nameof(GanttData.Sdate)></GanttColumn>
<GanttColumn Field=@nameof(GanttData.Edate)></GanttColumn>
<GanttColumn Field=@nameof(GanttData.Progress)></GanttColumn>
</GanttColumns>
<GanttEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" AllowTaskbarEditing="true"
ShowDeleteConfirmDialog="true"
NewRowPosition="RowPosition.Child"></GanttEditSettings>
<GanttTimelineSettings>
<GanttTopTierSettings Unit="TimelineViewMode.Week" Format="MMM dd,
y"></GanttTopTierSettings>
<GanttBottomTierSettings
Unit="TimelineViewMode.Day"></GanttBottomTierSettings>
</GanttTimelineSettings>
<GanttTaskFields Id="Id" Name="Name" StartDate="Sdate" EndDate="Edate"
Progress="Progress"
ParentID="ParentId"></GanttTaskFields>
</SfGantt>
@code {
// Implementing custom adaptor by extending the DataAdaptor class
public class CustomAdaptor : DataAdaptor
{
masterContext db = new masterContext();
// Performs data Read operation
public override object Read(DataManagerRequest dm, string key = null)
{
IEnumerable<GanttData> DataSource = db.GanttData;
int count = DataSource.Cast<GanttData>().Count();
```



```
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count = count } : (object)DataSource;
}
// Performs CRUD operation
public override object BatchUpdate(DataManager dm, object changedRecords, object addedRecords, object deletedRecords, string keyField, string key, int? dropIndex)
{
    List<GanttData> addRecord = addedRecords as List<GanttData>;
    List<GanttData> changed = changedRecords as List<GanttData>;
    List<GanttData> deleteRecord = deletedRecords as List<GanttData>;
    if (changed != null)
    {
        for (var i = 0; i < changed.Count(); i++)
        {
            var value = changed[i];
            GanttData result = db.GanttData.Where(or => or.Id == value.Id).FirstOrDefault();
            result.Id = value.Id;
            result.Name = value.Name;
            result.Sdate = value.Sdate;
            result.Edate = value.Edate;
            result.Progress = value.Progress;
            db.SaveChanges();
        }
    }
    if (deleteRecord != null)
    {
        for (var i = 0; i < deleteRecord.Count(); i++)
        {
            db.GanttData.Remove(db.GanttData.Where(or => or.Id == deleteRecord[i].Id).FirstOrDefault());
            db.SaveChanges();
        }
    }
    if (addRecord != null)
    {
        for (var i = 0; i < addRecord.Count(); i++)
        {
            db.GanttData.Add(addRecord[i] as GanttData);
            db.SaveChanges();
        }
    }
    return (new { addedRecords = addRecord, changedRecords = changed, deletedRecords = deleteRecord });
}
```

---

You can find the sample for server-side application using entity framework [here](#).

---

#### *Entity Framework in Client-Side Application*

You need to follow the following steps when working with a client-side application.

### Custom Adaptor

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

remote data, we are using **CustomAdaptor** of **DataManager** property to call the server method. You can know more about **CustomAdaptor** from [here](#). We can populate the datasource in Gantt from the SQL table using Entity Framework using **ReadAsync** method. Please Check the below code snippet to assign the data source to Gantt.

### ASPX-CS

```
@using MyBlazorApp.Shared.DataAccess
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor
@using MyBlazorApp.Data

<SfGantt ID="GanttExport" TValue="GanttDataDetails" HighlightWeekends="true"
RowHeight="75" DurationUnit="DurationUnit.Day"
Toolbar="@ (new List<string>() { "Add", "Edit", "Update", "Delete", "Cancel",
"ExpandAll", "CollapseAll" }) ">
<SfDataManager Adaptor="Adaptors.CustomAdaptor" DataType="GanttDataDetails">
<CustomDataAdaptor></CustomDataAdaptor>
</SfDataManager>
<GanttColumns>
<GanttColumn Field=@nameof(GanttDataDetails.Id) Width="100"></GanttColumn>
<GanttColumn Field=@nameof(GanttDataDetails.Name) Width="250"></GanttColumn>
<GanttColumn Field=@nameof(GanttDataDetails.Sdate)></GanttColumn>
<GanttColumn Field=@nameof(GanttDataDetails.Edate)></GanttColumn>
<GanttColumn Field=@nameof(GanttDataDetails.Duration)></GanttColumn>
<GanttColumn Field=@nameof(GanttDataDetails.Progress)></GanttColumn>
<GanttColumn Field=@nameof(GanttDataDetails.ParentId)></GanttColumn>
</GanttColumns>
<GanttEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" AllowTaskbarEditing="true"></GanttEditSettings>
<GanttTimelineSettings>
<GanttTopTierSettings Unit="TimelineViewMode.Week" Format="MMM dd,
y"></GanttTopTierSettings>
<GanttBottomTierSettings
Unit="TimelineViewMode.Day"></GanttBottomTierSettings>
</GanttTimelineSettings>
<GanttTaskFields Id="Id" Name="Name" StartDate="Sdate" EndDate="Edate"
ParentID="ParentId" Progress="Progress"
Duration="Duration"></GanttTaskFields>
</SfGantt>
@code {
// Performs data Read operation
public override async Task<object> ReadAsync(DataManagerRequest dm, string
key = null)
{
IEnumerable<object> data = await
http.GetJsonAsync<IEnumerable<GanttDataDetails>>("api/GanttDataDetails") as
IEnumerable<object>;
return dm.RequiresCounts ? new DataResult() { Result = data, Count =
data.Count() } : (object)data;
}
```

```
.../////
}
```

### Perform CRUD operation in CustomAdaptor

All the CRUD operations in the Gantt Chart are done through DataManager. The DataManager has an option to bind all the CRUD related data in server-side.

The following sample code explains you about, how to implement CRUD operations for the custom bounded data.

### CSHARP

```
@using Newtonsoft.Json
@using MyBlazorApp.Data
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor
@using MyBlazorApp.Shared.DataAccess
@inject HttpClient http
@inherits DataAdaptor<GanttDataContext>
@code {
    // Performs data Read operation
    public override async Task<object> ReadAsync(DataManagerRequest dm, string
    key = null)
    {
        IEnumerable<object> data = await
        http.GetJsonAsync<IEnumerable<GanttDataDetails>>("api/GanttDataDetails") as
        IEnumerable<object>;
        return dm.RequiresCounts ? new DataResult() { Result = data, Count =
        data.Count() } : (object)data;
    }
    // Performs CRUD operation
    public override async Task<object> BatchUpdateAsync(DataManager dm, object
    changedRecords, object addedRecords, object deletedRecords, string keyField,
    string key, int? dropIndex)
    {
        List<GanttDataDetails> addRecord = addedRecords as List<GanttDataDetails>;
        List<GanttDataDetails> changed = changedRecords as List<GanttDataDetails>;
        List<GanttDataDetails> deleteRecord = deletedRecords as
        List<GanttDataDetails>;
        if (changed != null)
        {
            for (var i = 0; i < changed.Count(); i++)
            {
                var value = changed[i];
                await http.SendJsonAsync(HttpMethod.Put, "/api/GanttDataDetails/" +
                changed[i].Id, changed[i] as GanttDataDetails);
            }
        }
        if (deleteRecord != null)
        {
            for (var i = 0; i < deleteRecord.Count(); i++)
            {
                await http.DeleteAsync("api/GanttDataDetails/" + deleteRecord[i].Id);
            }
        }
    }
}
```

```

}
if (addRecord != null)
{
    for (var i = 0; i < addRecord.Count(); i++)
    {
        await http.SendJsonAsync(HttpMethod.Post, "/api/GanttDataDetails",
            addRecord[i] as GanttDataDetails);
    }
}
return (new { addedRecords = addRecord, changedRecords = changed,
    deletedRecords = deleteRecord });
}
}

```

You can find the sample for client-side application using entity framework [here](#).

### Indent and Outdent

Indent and Outdent of a task are used to update the level of task in the hierarchical order of the task. It can be performed by enabling the `GanttEditSettings.AllowEditing` property.

**Indent** - Selected task can be indented to the level of task to the hierarchical order. It can be performed by using in-built context menu or toolbar items. It can also be invoked by using the `indent` method dynamically on any action like external button click. The following code example shows how to enable indent option in the Gantt chart.

**Outdent** - Selected task can be outdented to the level of task from the hierarchical order. It can be performed by using in-built context menu or toolbar items. It can also be invoked by using the `outdent` method dynamically on any action like external button click. The following code example shows how to enable outdent option in the Gantt chart.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Toolbar="@ (new List<string>() {
    "Indent", "Outdent" })" Height="450px" Width="900px">
    <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
        EndDate="EndDate"
        Duration="Duration" Progress="Progress" Child="SubTasks"></GanttTaskFields>
    <GanttEditSettings AllowEditing="true" AllowAdding="true"
        AllowDeleting="true">
    </GanttEditSettings>
</SfGantt>
@code{
    public List<TaskData> TaskCollection { get; set; }
    protected override void OnInitialized()
    {
        this.TaskCollection = GetTaskCollection();
    }
    public class TaskData
    {
        public int TaskId { get; set; }
        public string TaskName { get; set; }
        public DateTime StartDate { get; set; }
        public DateTime EndDate { get; set; }
        public string Duration { get; set; }
    }
}

```

```
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
```

```
StartDate = new DateTime(2019, 04, 04),  
Duration = "0",  
Progress = 30,  
}  
})  
}  
};  
return Tasks;  
}  
}
```

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

Editing feature requires a primary key column for CRUD operations. While defining columns in Gantt using the `GanttColumns` property, it is mandatory that any one of the columns, must be a primary column. By default, the `Id` column will be the primary key column. If `Id` column is not defined, we need to enable `IsPrimaryKey` for any one of the columns defined in the `GanttColumns` property.

#### Task Dependencies in Blazor Gantt Chart Component

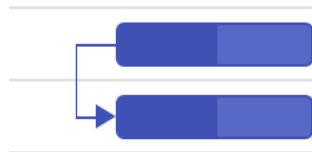
Task dependency or task relationship can be established between two tasks in Gantt Chart. This dependency affects the project schedule. If you change the predecessor of a task, it will affect the successor task, which will affect the next task, and so on.

##### Task relationship types

Task relationships are categorized into four types based on the start and finish dates of the task.

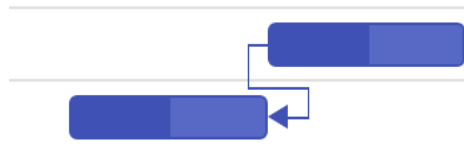
##### *Start to Start (SS)*

You cannot start a task until the dependent task also starts.



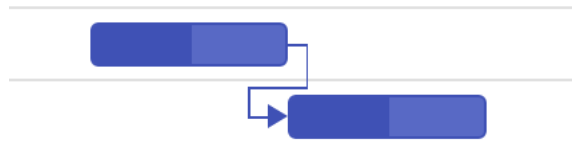
##### *Start to Finish (SF)*

You cannot finish a task until the dependent task is started.



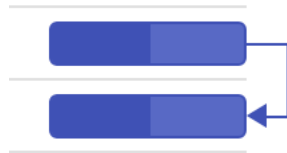
##### *Finish to Start (FS)*

You cannot start a task until the dependent task is completed.



##### *Finish to Finish (FF)*

You cannot finish a task until the dependent task is completed.



### Define task relationship

Task relationship is defined in the data source as a string value, and this value is mapped to the Gantt Chart component by using the `GanttTaskFields.Dependency` property. The following code example demonstrates how to enable the predecessor in the Gantt Chart component.

### ASPX-CS

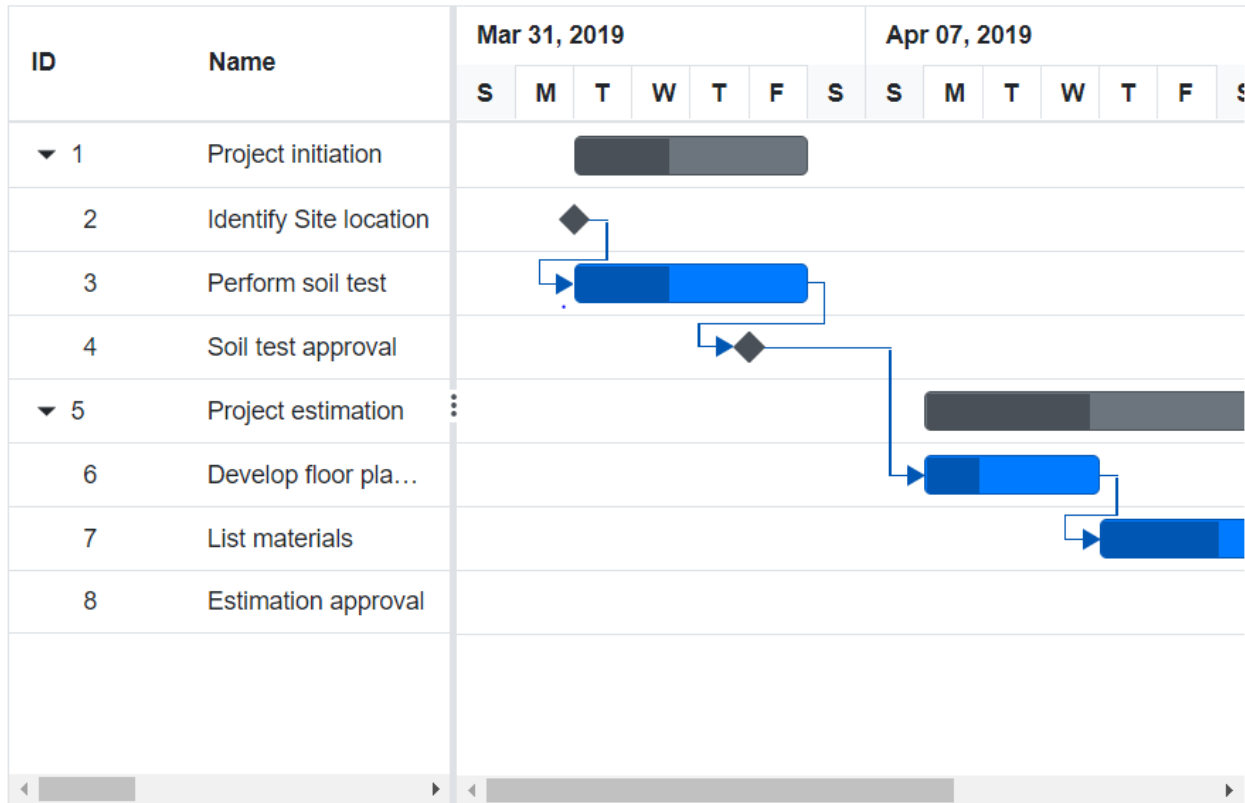
```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks"
  Dependency="Predecessor">
  </GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public string Predecessor { get; set; }
  public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
  List<TaskData> Tasks = new List<TaskData> () {
    new TaskData() {
      TaskId = 1,
      TaskName = "Project initiation",
      StartDate = new DateTime(2019, 04, 02),
      EndDate = new DateTime(2019, 04, 21),
      SubTasks = (new List<TaskData> () {
        new TaskData() {
          TaskId = 2,
          TaskName = "Identify Site location",
          StartDate = new DateTime(2019, 04, 02),
          Duration = "0",
          Progress = 30,
        },
        new TaskData() {
          TaskId = 3,
          TaskName = "Perform soil test",
          StartDate = new DateTime(2019, 04, 02),
          Duration = "4",

```

```
Progress = 40,  
Predecessor = "2"  
},  
new TaskData() {  
    TaskId = 4,  
    TaskName = "Soil test approval",  
    StartDate = new DateTime(2019, 04, 02),  
    Duration = "0",  
    Progress = 30,  
    Predecessor = "3"  
},  
})  
},  
new TaskData() {  
    TaskId = 5,  
    TaskName = "Project estimation",  
    StartDate = new DateTime(2019, 04, 02),  
    EndDate = new DateTime(2019, 04, 21),  
    SubTasks = (new List <TaskData> () {  
        new TaskData() {  
            TaskId = 6,  
            TaskName = "Develop floor plan for estimation",  
            StartDate = new DateTime(2019, 04, 04),  
            Duration = "3",  
            Progress = 30,  
            Predecessor = "4"  
        },  
        new TaskData() {  
            TaskId = 7,  
            TaskName = "List materials",  
            StartDate = new DateTime(2019, 04, 04),  
            Duration = "3",  
            Progress = 40,  
            Predecessor = "6"  
        },  
        new TaskData() {  
            TaskId = 8,  
            TaskName = "Estimation approval",  
            StartDate = new DateTime(2019, 04, 04),  
            Duration = "0",  
            Progress = 30,  
            Predecessor = "7"  
        },  
    })  
}  
};  
return Tasks;  
}  
}
```

The following screenshot displays the output of the above code.





### Predecessor offset with duration units

In the Gantt Chart component, the predecessor offset can be defined with the following duration units:

- Day
- Hour
- Minute

You can define an offset with various offset duration units for predecessors by using the following code example.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
    EndDate="EndDate" Duration="Duration" Progress="Progress"
    Dependency="Predecessor" Child="SubTasks">
  </GanttTaskFields>
</SfGantt>
@code{
  public List<TaskData> TaskCollection { get; set; }
  protected override void OnInitialized()
  {
    this.TaskCollection = GetTaskCollection();
  }
  public class TaskData
  {
    public int TaskId { get; set; }
  }
}
```

```
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public string Predecessor { get; set; }
public List<TaskData> SubTasks { get; set; }
}

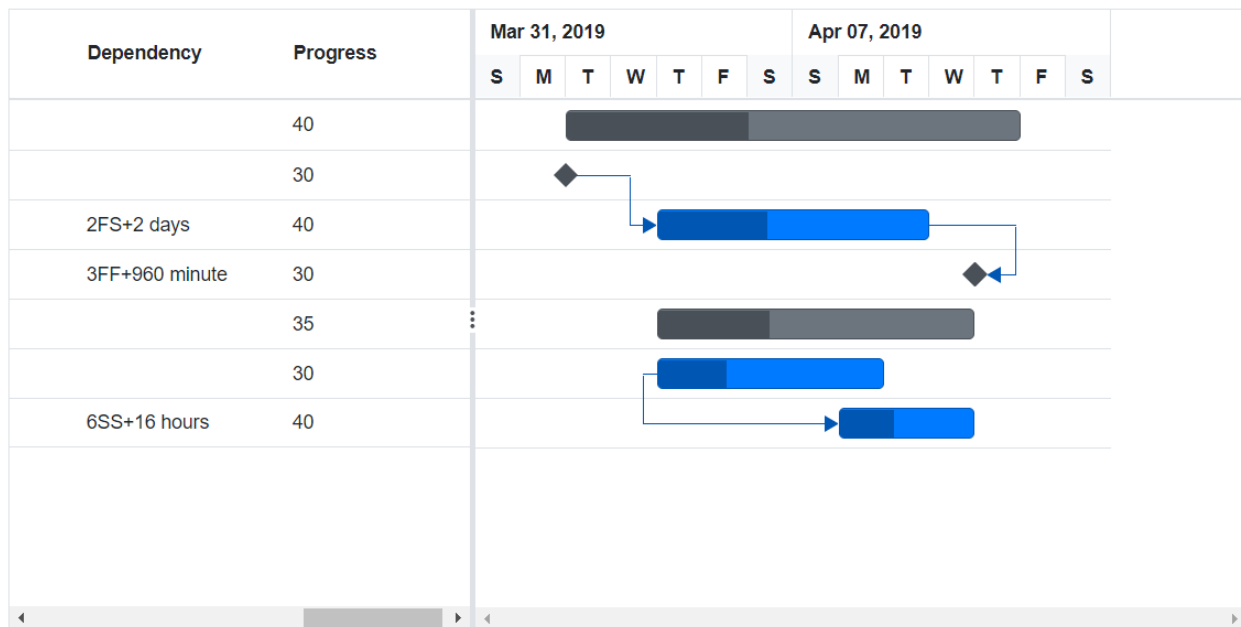
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
Predecessor = "2FS+2d"
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
Predecessor = "3FF+960m"
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
```

```

StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
Predecessor = "6SS+16h"
},
})
}
};
return Tasks;
}
}

```

The following screen shot depicts the duration unit support in the predecessor offset.



### Validate predecessor links on editing

In Gantt, task relationship link can be broken by editing the start date, end date and duration value of task. When the task relationship broken on any edit action. This can be handled in Gantt in two ways.

- Using `actionBegin` event
- Using predecessor validation dialog

### Using `actionBegin` event

#### On taskbar editing

When the task relationship link is broken on any edit action, then the `OnActionBegin` event will be triggered with `RequestType` argument as `ValidateLinkedTask`. You can validate the editing action within the `actionBegin` event using the `ValidateMode` event argument. The `ValidateMode` event argument has the following properties:

Argument | Default value | Description

`args.ValidateMode.RespectLink | false` | In this validation mode, the predecessor links get high priority. With this mode enabled, when the successor task is moved before the predecessor task's end date, the editing will be reverted, and dates will be validated based on the dependency links.

`args.ValidateMode.RemoveLink | false` | In this validation mode, the taskbar editing gets high priority. For inappropriate task dates, the dependency links will be removed and tasks will be moved to the edited date.

`args.ValidateMode.PreserveLinkWithEditing | true` | In this validation mode, the taskbar editing will be considered along with the dependency links. This relationship will be maintained by updating the offset value of predecessors.

By default, the `PreserveLinkWithEditing` validation mode will be enabled, so the predecessors are updated with offset values.

The following sample explains enabling the `RespectLink` validation mode while editing the linked tasks in the `OnActionBegin` event.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress"
  Dependency="Predecessor" ParentID="ParentId"></GanttTaskFields>
  <GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
  <GanttEvents TValue="TaskData" OnActionBegin="ActionBegin"></GanttEvents>
</SfGantt>

@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public void ActionBegin(GanttActionEventArgs<TaskData> args) {
  if(args.RequestType.ToString() == "ValidateLinkedTask") {
    args.ValidateMode.RespectLink = true;
  }
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public string Predecessor { get; set; }
  public int? ParentId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
  List <TaskData> Tasks = new List <TaskData> () {
    new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
    new DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 21) },
    new TaskData() { TaskId = 2, TaskName = "Identify Site location", StartDate
    = new DateTime(2019, 04, 02), Duration = "0", Progress = 30, ParentId = 1 },
```

```

new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
DateTime(2019, 04, 02), Duration = "4", Progress = 40, Predecessor = "2",
ParentId = 1 },
new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
new DateTime(2019, 04, 02), Duration = "0", Progress = 30, ParentId = 1 },
new TaskData() { TaskId = 5, TaskName = "Project estimation", StartDate =
new DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 21) },
new TaskData() { TaskId = 6, TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04), Duration = "3", Progress = 30,
ParentId = 5 },
new TaskData() { TaskId = 7, TaskName = "List materials", StartDate = new
DateTime(2019, 04, 04), Duration = "3", Progress = 40, Predecessor = "6",
ParentId = 5 },
new TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate =
new DateTime(2019, 04, 04), Duration = "0", Progress = 30, ParentId = 5 }
};
return Tasks;
}
}

```

#### On drawing connector lines

When the connector lines are drawn between tasks, the task date gets validated based on predecessor values. You can restrict this validation on predecessor drawing using the [OnActionBegin](#) event which gets triggered with the [Action](#) argument as `DrawConnectorLine`. You can enable/disable the validation using [EnableAutoLinkValidation](#) event argument. By default [EnableAutoLinkValidation](#) is true.

#### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="1000px" Toolbar="@ (new List<string>() { "Add", "Edit", "Update",
"Cancel"}) ">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
Duration="Duration" Dependency="Predecessor"
ParentID="ParentId"></GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true" AllowEditing="true"
AllowAdding="true"></GanttEditSettings>
<GanttEvents OnActionBegin="ActionBegin" TValue="TaskData"></GanttEvents>
</SfGantt>
@code {
public SfGantt<TaskData> Gantt;
public DateTime ProjectStart = new DateTime(2019, 3, 24);
public DateTime ProjectEnd = new DateTime(2019, 7, 6);
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public void ActionBegin(GanttActionEventArgs<TaskData> args)
{
if (args.Action != null && args.Action.Equals("DrawConnectorLine",
StringComparison.Ordinal))
{
args.EnableAutoLinkValidation = false;
}
}
}

```

```

public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public string Duration { get; set; }
    public string Predecessor { get; set; }
    public int Progress { get; set; }
    public int? ParentId { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
            new DateTime(2019, 04, 02)},
        new TaskData() { TaskId = 2, TaskName = "Identify Site location", StartDate =
            new DateTime(2019, 04, 02), Progress = 30, ParentId = 1},
        new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
            DateTime(2019, 04, 02), Duration = "3", Progress = 40, Predecessor = "2fs",
            ParentId = 1 },
        new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
            new DateTime(2019, 04, 02), Duration = "1", Progress = 30, ParentId = 1 },
        new TaskData() { TaskId = 5, TaskName = "Project estimation", StartDate =
            new DateTime(2019, 04, 02) },
        new TaskData() { TaskId = 6, TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04), Duration = "3", Progress = 30,
            ParentId = 5 },
        new TaskData() { TaskId = 7, TaskName = "List materials", StartDate = new
            DateTime(2019, 04, 04), Duration = "3", Progress = 40, ParentId = 5 },
        new TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate =
            new DateTime(2019, 04, 04), Duration = "2", Progress = 30, ParentId = 5 }
    };
    return Tasks;
}

```

EnablePredecessorValidation is used to enable/disable validation based on predecessor values both on load time and on edit actions like cell editing, dialog editing, and on predecessor drawing. Whereas, [EnableAutoLinkValidation](#) event argument is used to enable/disable validation only on predecessor drawing.

You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

### Sorting in Blazor Gantt Chart Component

Sorting enables you to sort data in the ascending or descending order. To sort a column, click the column header. To sort multiple columns, press and hold the CTRL key and click the column header. You can clear sorting of any one of the multi-sorted columns by pressing and holding the SHIFT key and clicking the specific column header.

To enable sorting in the Gantt Chart component, set the `AllowSorting` property to true. Sorting options can be configured through the `GanttSortSettings` property.

**ASPX-CS**

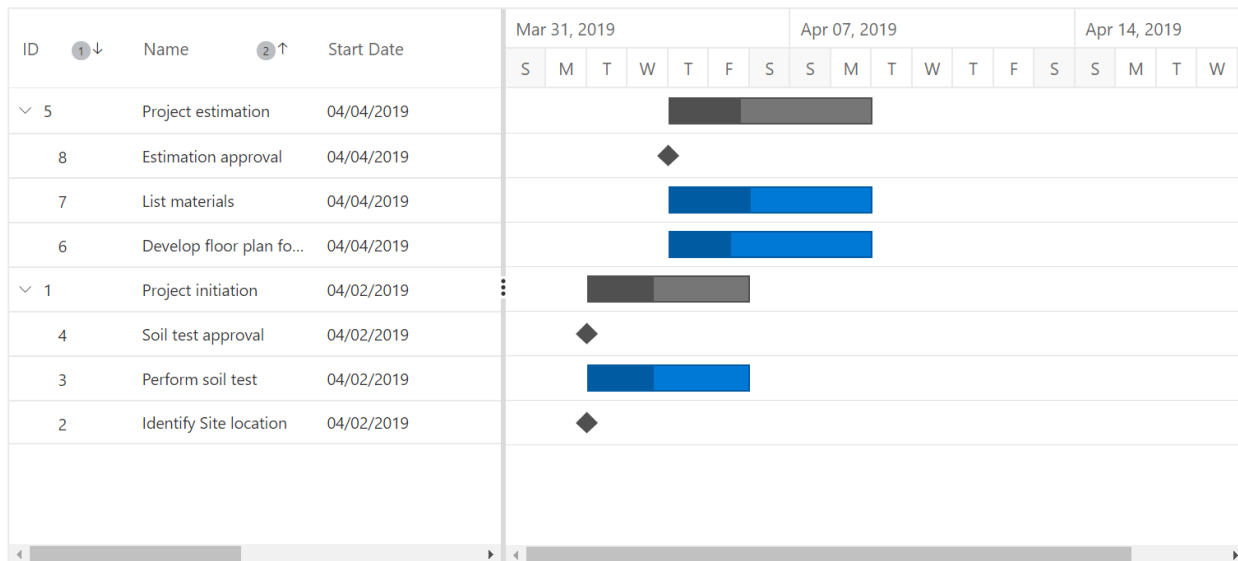
```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowSorting="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Obtain necessary approvals",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
}
```

```

TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 6,
        TaskName = "Develop floor plan for estimation",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 7,
        TaskName = "List materials",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 40
    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "0",
        Progress = 30,
    }
})
};
return Tasks;
}
}

```

The following screenshot shows the output of multicolumn sorting in Gantt Chart component.





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

\* To disable sorting for a particular column, set the `GanttColumn.AllowSorting` property to false.

### Sorting Column on Gantt Chart Initialization

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

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowSorting="true">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttSortSettings>
  <GanttSortDescriptors>
  <GanttSortDescriptor Field="TaskId"
  Direction="SortDirection.Descending"></GanttSortDescriptor>
  <GanttSortDescriptor Field="TaskName"
  Direction="SortDirection.Ascending"></GanttSortDescriptor>
  </GanttSortDescriptors>
  </GanttSortSettings>
</SfGantt>

@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
  List<TaskData> Tasks = new List<TaskData> () {
  new TaskData() {
    TaskId = 1,
    TaskName = "Project initiation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List<TaskData> () {
    new TaskData() {
      TaskId = 2,
      TaskName = "Identify Site location",
      StartDate = new DateTime(2019, 04, 02),
```

```
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}
```

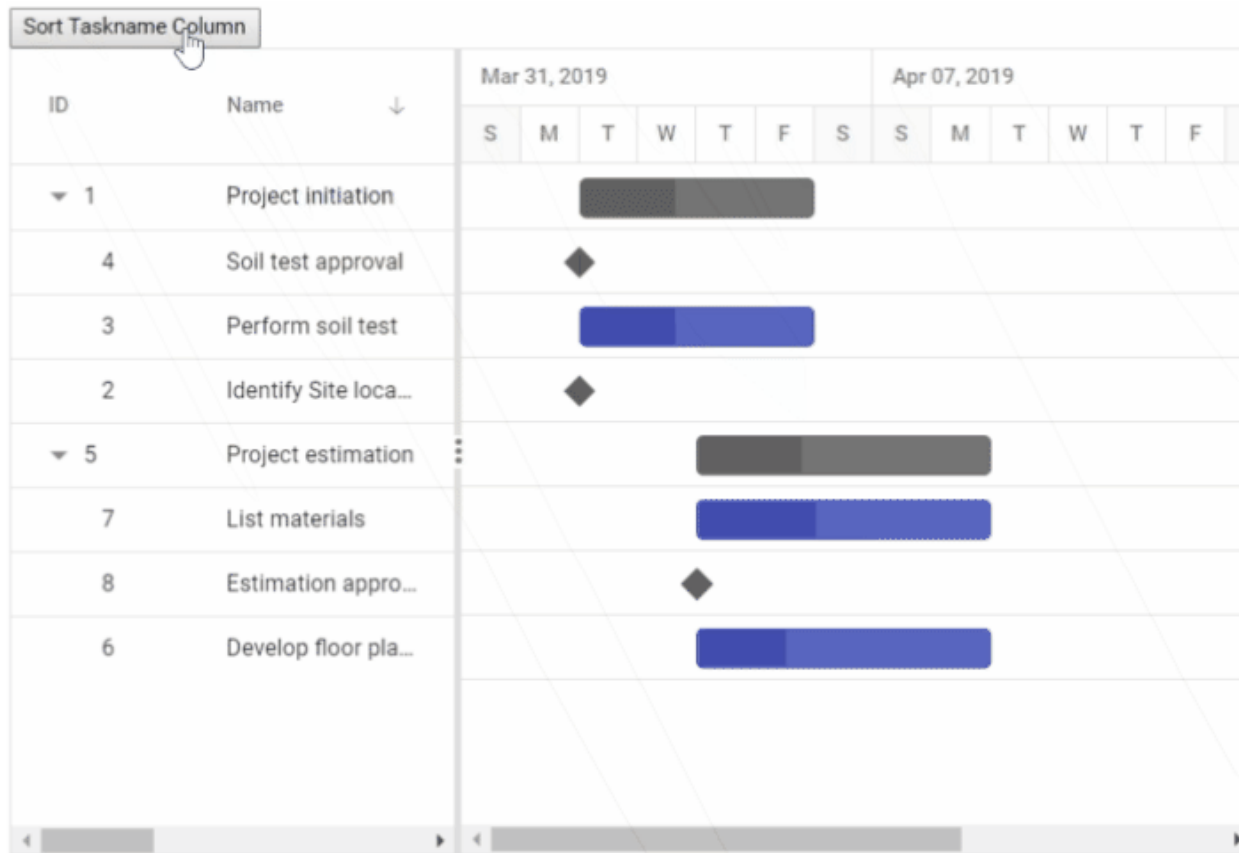
### Sorting Column dynamically

Columns in the Gantt Chart component can be sorted dynamically using the `SortByColumnAsync` method. The following code example demonstrates how to invoke the `SortByColumnAsync` method by clicking the custom button.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<button @onclick="Sorting">Sort Taskname Column</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px" AllowSorting="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void Sorting()
{
this.Gantt.SortByColumnAsync("TaskName",
Syncfusion.Blazor.Grids.SortDirection.Descending, false);
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
List<TaskData> Tasks = new List<TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
```

```
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}
```



Clear all the Sorted Columns dynamically

In the Gantt Chart component, you can clear all the sorted columns and return to previous position using the `ClearSortingAsync` public method. The following code snippet shows how to clear all the sorted columns by clicking the custom button.

#### ASPX-CS

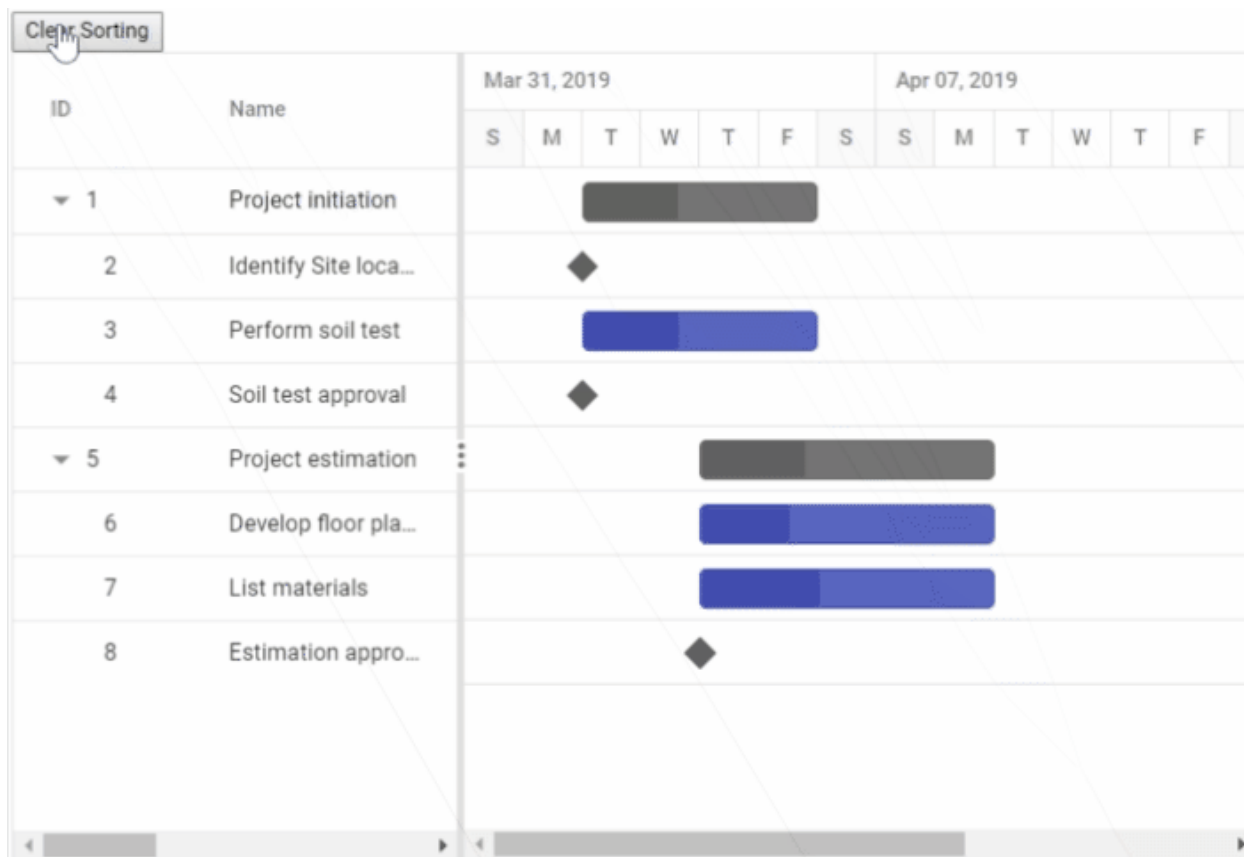
```
@using Syncfusion.Blazor.Gantt
<button @onclick="ClearSorting">Clear Sorting</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px" AllowSorting="true">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttSortSettings>
    <GanttSortDescriptors>
      <GanttSortDescriptor Field="TaskId"
Direction="Syncfusion.Blazor.Grids.SortDirection.Descending"></GanttSortDesc
ripor>
      <GanttSortDescriptor Field="TaskName"
Direction="Syncfusion.Blazor.Grids.SortDirection.Ascending"></GanttSortDescr
iptor>
    </GanttSortDescriptors>
  </GanttSortSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
```

```
public void ClearSorting()
{
    this.Gantt.ClearSortingAsync();
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
```

```

TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```



### Sorting Events

During the sort action, the Gantt Chart component triggers two events. The `OnActionBegin` event triggers before the sort action starts, and the `OnActionComplete` event triggers after the sort action is completed.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowSorting="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEvents OnActionBegin="OnActionBegin"
OnActionComplete="OnActionComplete" TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public void OnActionBegin(GanttActionEventArgs<TaskData> args)
{
Console.WriteLine(args.RequestType + " " + args.Type);
}
public void OnActionComplete(GanttActionEventArgs<TaskData> args)
{
Console.WriteLine(args.RequestType + " " + args.Type);
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
List<TaskData> Tasks = new List<TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
```



```
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}
```

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

### Sorting Custom Columns

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

The following code snippets explains how to achieve this.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<button @onclick="Sorting">Sort Custom Column</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px" AllowSorting="true">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
Child="SubTasks"></GanttTaskFields>
  <GanttColumns>
    <GanttColumn Field="TaskId" Width="100"></GanttColumn>
    <GanttColumn Field="TaskName" Width="250"></GanttColumn>
    <GanttColumn Field="StartDate"></GanttColumn>
    <GanttColumn Field="Duration"></GanttColumn>
    <GanttColumn Field="Progress"></GanttColumn>
    <GanttColumn Field="CustomColumn"></GanttColumn>
  </GanttColumns>
</SfGantt>

@code{
public SfGantt<TaskData> Gantt;
public void Sorting()
{
  this.Gantt.SortByColumnAsync("CustomColumn",
Syncfusion.Blazor.Grids.SortDirection.Descending, false);
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public List<TaskData> SubTasks { get; set; }
  public int CustomColumn { get; set; }
}
public static List<TaskData> GetTaskCollection() {
  List<TaskData> Tasks = new List<TaskData> () {
    new TaskData() {
      TaskId = 1,
      TaskName = "Project initiation",
      StartDate = new DateTime(2019, 04, 02),
      EndDate = new DateTime(2019, 04, 21),
      CustomColumn = 2,
      SubTasks = (new List<TaskData> () {
```

```
new TaskData() {
    TaskId = 2,
    TaskName = "Identify Site location",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "0",
    Progress = 30,
    CustomColumn = 3
},
new TaskData() {
    TaskId = 3,
    TaskName = "Perform soil test",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "4",
    Progress = 40,
    CustomColumn = 4
},
new TaskData() {
    TaskId = 4,
    TaskName = "Soil test approval",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "0",
    Progress = 30,
    CustomColumn = 1
},
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    CustomColumn = 7,
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
            CustomColumn = 8
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40,
            CustomColumn = 5
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
            CustomColumn = 6
        }
    })
}
```

```

    })
  }
};
return Tasks;
}
}

```

### Scrolling in Blazor Gantt Component

The scrollbar will be displayed in the Gantt when the content exceeds the element [Width](#) or [Height](#). The vertical and horizontal scrollbars will be displayed based on the following criteria:

The vertical scrollbar appears when the total height of rows present in the Gantt exceeds its element height. The horizontal scrollbar appears when the sum of the columns' width exceeds the Gantt element width. The [Height](#) and [Width](#) are used to set the Gantt height and width, respectively.

---

The default value for [Height](#) and [Width](#) is **auto**.

---

#### Set width and height

To specify the [Height](#) and [Width](#) of the scroller in the pixel, set the pixel value to a number.

#### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="900px" Toolbar="@ (new List<string>() { "Add","Edit" }) ">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" Width="100"></GanttColumn>
<GanttColumn Field="TaskName" Width="250"
AllowEditing="false"></GanttColumn>
<GanttColumn Field="StartDate"></GanttColumn>
<GanttColumn Field="Duration"></GanttColumn>
<GanttColumn Field="Progress"></GanttColumn>
</GanttColumns>
<GanttEditSettings AllowEditing="true"
AllowAdding="true"></GanttEditSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
}

```

```

public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {TaskId = 1,TaskName = "Project initiation",StartDate = new
        DateTime(2019, 04, 02),EndDate = new DateTime(2019, 04, 21),SubTasks = (new
        List <TaskData> () {new TaskData() {TaskId = 2,TaskName = "Identify Site
        location",StartDate = new DateTime(2019, 04, 02),Duration = "0",Progress =
        30,},new TaskData() {TaskId = 3,TaskName = "Perform soil test",StartDate =
        new DateTime(2019, 04, 02),Duration = "4",Progress = 40,},new TaskData()
        {TaskId = 4, TaskName = "Soil test approval",StartDate = new DateTime(2019,
        04, 02),Duration = "0",Progress = 30,}}}},
        new TaskData() {TaskId = 5,TaskName = "Project estimation",StartDate = new
        DateTime(2019, 04, 02),EndDate = new DateTime(2019, 04, 21),SubTasks = (new
        List <TaskData> () {new TaskData() {TaskId = 6,TaskName = "Develop floor
        plan for estimation",StartDate = new DateTime(2019, 04, 04),Duration =
        "3",Progress = 30,},new TaskData() {TaskId = 7,TaskName = "List
        materials",StartDate = new DateTime(2019, 04, 04),Duration = "3",Progress =
        40,},new TaskData() {TaskId = 8,TaskName = "Estimation approval",StartDate =
        new DateTime(2019, 04, 04),Duration = "0",Progress = 30,}}}}
    };
    return Tasks;
}

```

### Responsive with the parent container

Specify the [Height](#) and [Width](#) as **100%** to make the Gantt element fill its parent container. Setting the [Height](#) to **100%** requires the Gantt parent element to have explicit height or you can use viewport height to set explicit height based on the browser layout.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<div class="gantt">
    <SfGantt DataSource="@TaskCollection" Toolbar="@ (new List<string>() { "Add"
    })" Height="100%" Width="100%">
        <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
        EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
        </GanttTaskFields>
        <GanttEditSettings AllowAdding="true"></GanttEditSettings>
    </SfGantt>
</div>
<style>
.gantt {
    height: 100vh;
    width: 100vw;
    border: 2px solid;
    padding: 20px;
    resize: both;
    overflow: auto;
}
</style>
@code{
    public List<TaskData> TaskCollection { get; set; }
    protected override void OnInitialized()
    {
        this.TaskCollection = GetTaskCollection();
    }
}

```

```

}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>()
    {
        new TaskData() {TaskId = 1,TaskName = "Project initiation",StartDate = new
        DateTime(2019, 04, 02),EndDate = new DateTime(2019, 04, 21),SubTasks = (new
        List<TaskData> () {new TaskData() {TaskId = 2,TaskName = "Identify Site
        location",StartDate = new DateTime(2019, 04, 02),Duration = "0",Progress =
        30,},new TaskData() {TaskId = 3,TaskName = "Perform soil test",StartDate =
        new DateTime(2019, 04, 02),Duration = "4",Progress = 40,},new TaskData()
        {TaskId = 4,TaskName = "Soil test approval",StartDate = new DateTime(2019,
        04, 02),Duration = "0",Progress = 30},)},),
        new TaskData() {TaskId = 5,TaskName = "Project estimation",StartDate = new
        DateTime(2019, 04, 02),EndDate = new DateTime(2019, 04, 21),SubTasks = (new
        List<TaskData> () {new TaskData() {TaskId = 6,TaskName = "Develop floor
        plan for estimation",StartDate = new DateTime(2019, 04, 04),Duration =
        "3",Progress = 30,},new TaskData() {TaskId = 7,TaskName = "List
        materials",StartDate = new DateTime(2019, 04, 04),Duration = "3",Progress =
        40},new TaskData() {TaskId = 8,TaskName = "Estimation approval",StartDate =
        new DateTime(2019, 04, 04),Duration = "0",Progress = 30,}})},
    };
    return Tasks;
}
}

```

### Context Menu in Blazor Gantt Chart Component

The [Blazor Gantt Chart](#) component allows you to perform quick actions by using the context menu. When right-clicking the context menu, the context menu options will be shown. To enable this feature, set the `EnableContextMenu` to true. The default context menu options are enabled using the `GanttEditSettings` property. The context menu options can be customized using the `ContextMenuItems` property.

The default items are listed in the following table.

Items	Description
<code>AutoFit</code>	Auto-fits the current column.
<code>AutoFitAll</code>	Auto-fits all columns.
<code>SortAscending</code>	Sorts the current column in ascending order.
<code>SortDescending</code>	Sorts the current column in descending order.
<code>TaskInformation</code>	Edits the current task.

**Add** | Adds a new row to the Gantt.

**Indent** | Indent the selected record to one level.

**Outdent** | Outdent the selected record to one level.

**DeleteTask** | Deletes the current task.

**Save** | Saves the edited task.

**Cancel** | Cancels the edited task.

**DeleteDependency** | Deletes the current dependency task link.

**Convert** | Converts current task to milestone or vice-versa.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px"
EnableContextMenu="true" AllowSorting="true" AllowResizing="true"
Width="900px" HighlightWeekends="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
Dependency="Predecessor" ParentID="ParentId"></GanttTaskFields>
<GanttEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public double Progress { get; set; }
public string Predecessor { get; set; }
public int? ParentId { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
```

```
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
ParentId = 1
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "3",
Progress = 30,
Predecessor = "2",
ParentId = 1
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
ParentId = 5
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
Predecessor = "6",
ParentId = 5
}
};
return Tasks;
}
```



## Baseline in Blazor Gantt Chart Component

The baseline feature enables users to view the deviation between the planned dates and actual dates of the tasks in a project. Baseline dates or planned dates of a task may or may not be same as the actual task dates. The baseline can be enabled by setting the `RenderBaseline` property to `true` and the baseline color can be changed using the `BaselineColor` property. To render the baseline, you should map the baseline start and end date values from the data source. This can be done using the `GanttTaskFields.BaselineStartDate` and `GanttTaskFields.BaselineEndDate` properties. The following code example shows how to enable a baseline in the Gantt Chart component.

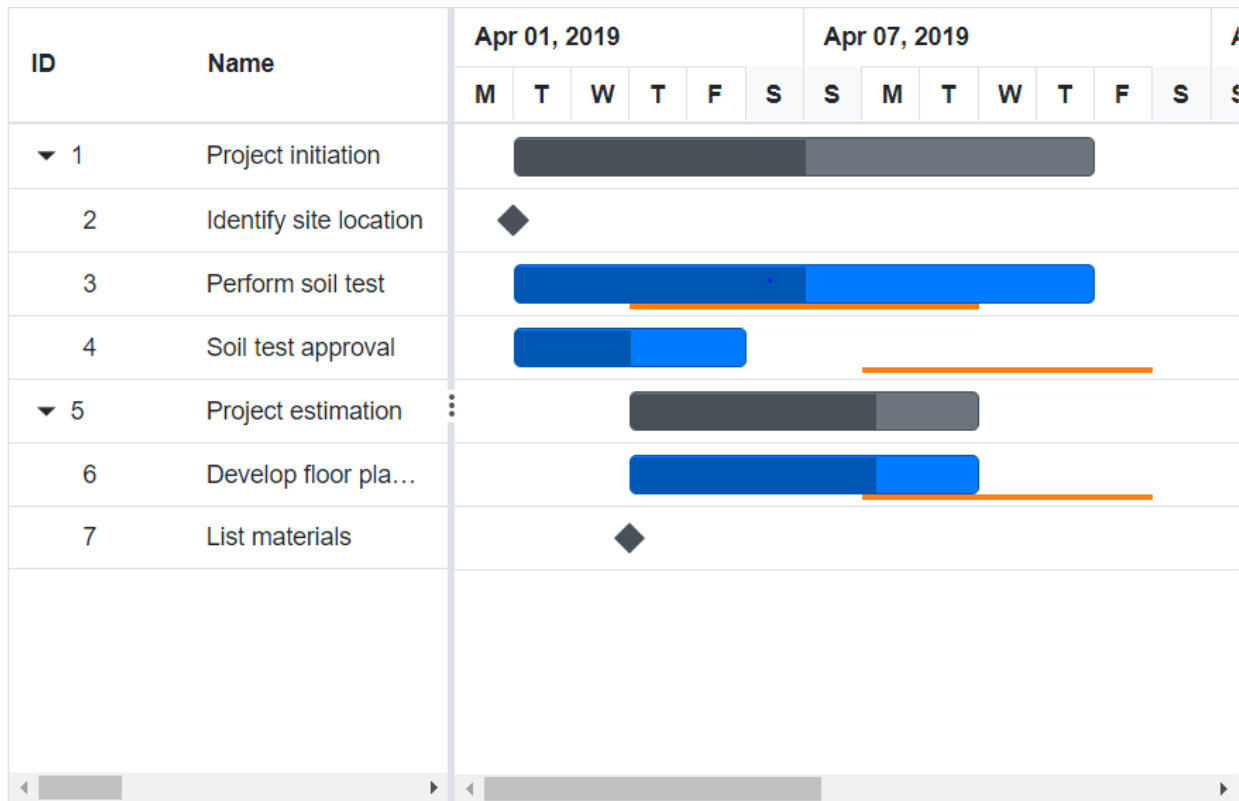
### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" RenderBaseline="true"
ProjectStartDate="@ProjectStart" ProjectEndDate="@ProjectEnd" Height="450px"
Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks"
  BaselineStartDate="BaselineStartDate"
  BaselineEndDate="BaselineEndDate">
  </GanttTaskFields>
</SfGantt>

@code{
public DateTime ProjectStart = new DateTime(2019, 04, 01);
public DateTime ProjectEnd = new DateTime(2019, 04, 30);
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public DateTime BaselineStartDate { get; set; }
public DateTime BaselineEndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify site location",
StartDate = new DateTime(2019, 04, 02),
```

```
BaselineStartDate = new DateTime(2019, 04, 02),
BaselineEndDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 70
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
BaselineStartDate = new DateTime(2019, 04, 04),
BaselineEndDate = new DateTime(2019, 04, 09),
Duration = "8",
Progress = 50
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
BaselineStartDate = new DateTime(2019, 04, 08),
BaselineEndDate = new DateTime(2019, 04, 12),
Duration = "4",
Progress = 50
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
BaselineStartDate = new DateTime(2019, 04, 08),
BaselineEndDate = new DateTime(2019, 04, 12),
Duration = "4",
Progress = 70
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
BaselineStartDate = new DateTime(2019, 04, 02),
BaselineEndDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 50
},
})
}
};
return Tasks;
}
}
```

The following screenshot shows the baselines in Gantt Chart component.



You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

### Time Line in Blazor Gantt Chart Component

In the [Blazor Gantt Chart](#) component, timeline is used to represent the project duration as individual cells with defined unit and formats.

#### Timeline View Modes

Gantt Chart contains the following in-built timeline view modes:

- Hour
- Week
- Month
- Year

Timescale mode in Gantt Chart can be defined by using `GanttTimelineSettings.TimelineViewMode` property and also we can define timescale mode of top tier and bottom tier by using `GanttTopTierSettings.Unit` and `GanttBottomTierSettings.Unit` properties.

Following table explains how top tier and bottom tier unit changes as per `GanttTimelineSettings.TimelineViewMode` property.

Timeline ViewMode | Top tier Unit | Bottom tier Unit

Year | Year | Month

Month | Month | Week

Week | Week | Day

Day | Day | Hour

Hour | Hour | Minute

#### *Week Timeline Mode*

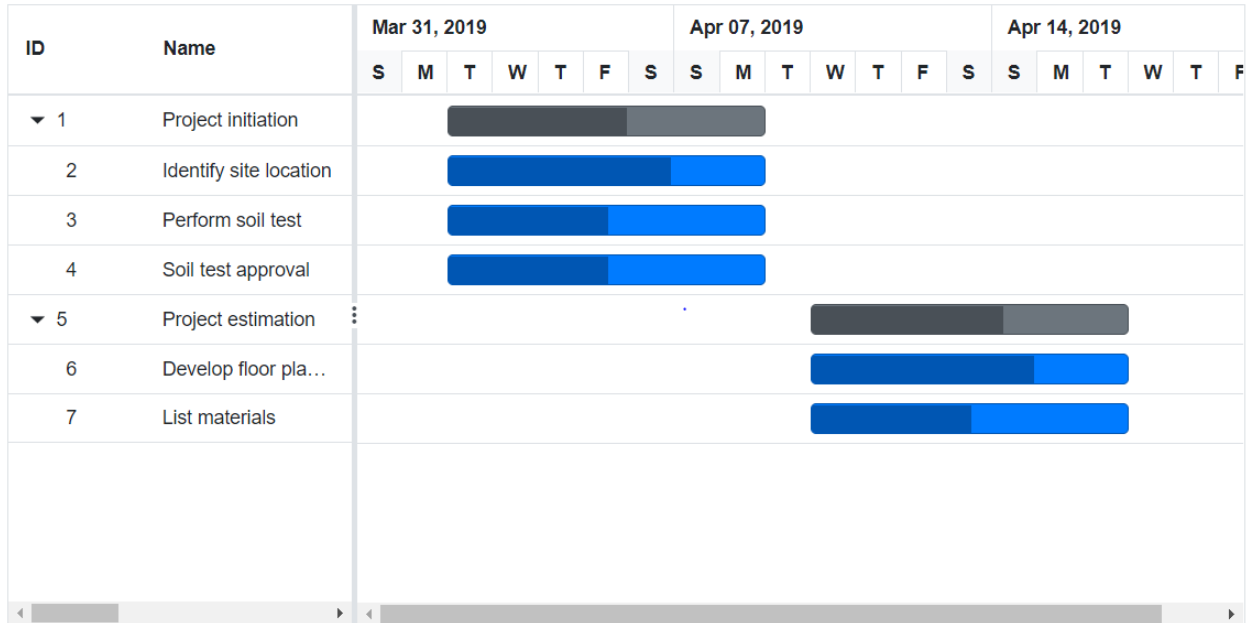
In the **Week** timeline mode, top tier of the schedule header displays the weeks, where as bottom tier of the header displays the days. Refer the following code example.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttTimelineSettings
  TimelineViewMode="TimelineViewMode.Week"></GanttTimelineSettings>
</SfGantt>

@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
  List <TaskData> Tasks = new List <TaskData> () {
    new TaskData() {
      TaskId = 1,
      TaskName = "Project initiation",
      StartDate = new DateTime(2019, 04, 02),
      EndDate = new DateTime(2019, 04, 21),
      SubTasks = (new List <TaskData> () {
        new TaskData() {
          TaskId = 2,
          TaskName = "Identify site location",
          StartDate = new DateTime(2019, 04, 02),
          Duration = "5",
          Progress = 70,
        },
        new TaskData() {
          TaskId = 3,
          TaskName = "Perform soil test",
          StartDate = new DateTime(2019, 04, 02),
```

```
Duration = "5",
Progress = 50,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "5",
Progress = 50,
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 10),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 10),
Duration = "5",
Progress = 70,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 10),
Duration = "5",
Progress = 50
},
})
}
};
return Tasks;
}
}
```



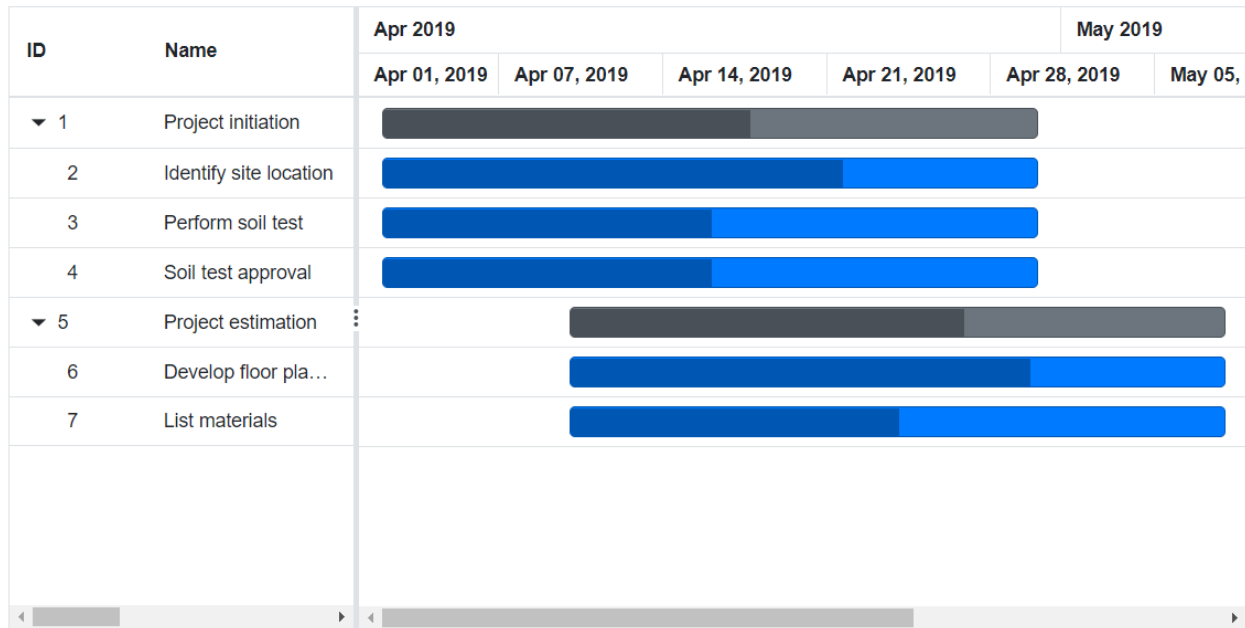
### Month Timeline Mode

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

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttTimelineSettings TimelineUnitSize=120
  TimelineViewMode="TimelineViewMode.Month"></GanttTimelineSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
```

```
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 2,
        TaskName = "Identify site location",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "20",
        Progress = 70,
    },
    new TaskData() {
        TaskId = 3,
        TaskName = "Perform soil test",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "20",
        Progress = 50,
    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "20",
        Progress = 50,
    },
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 10),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 10),
            Duration = "20",
            Progress = 70,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 10),
            Duration = "20",
            Progress = 50
        },
    })
}
};
return Tasks;
}
}
```



### Year Timeline Mode

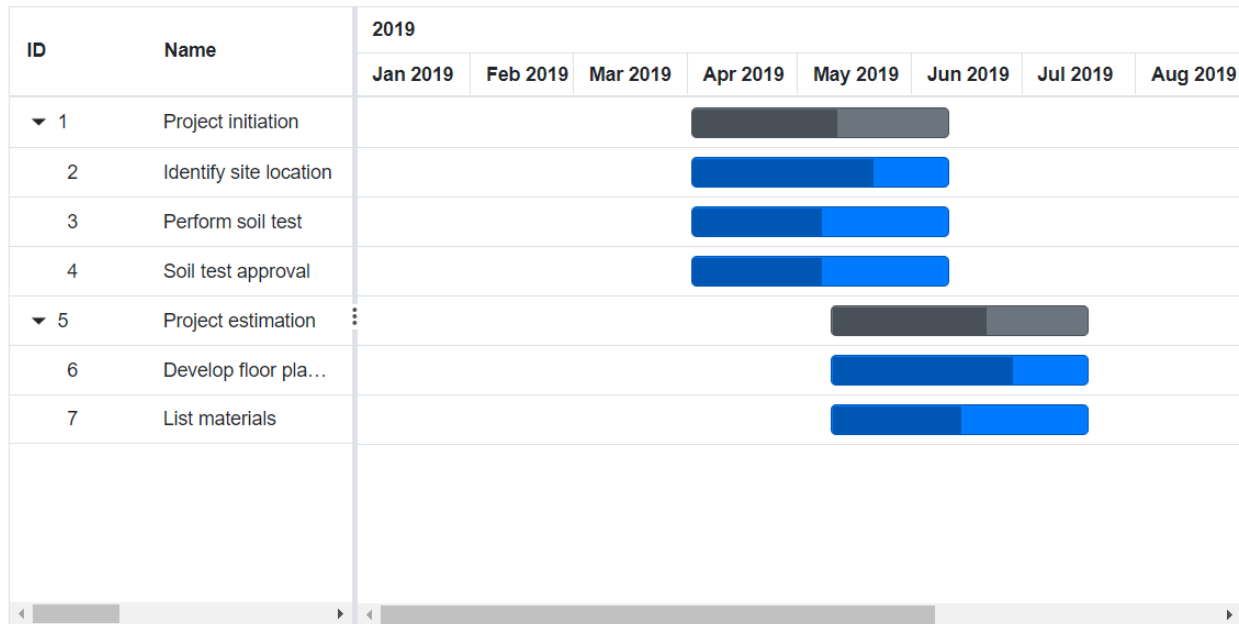
In the **Year** timeline mode, top tier of the schedule header displays the years, where as bottom tier of the schedule header displays its corresponding months. Refer the following code example.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttTimelineSettings TimelineUnitSize=75
TimelineViewMode="TimelineViewMode.Year">
</GanttTimelineSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
```



```
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 2,
        TaskName = "Identify site location",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "50",
        Progress = 70,
    },
    new TaskData() {
        TaskId = 3,
        TaskName = "Perform soil test",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "50",
        Progress = 50,
    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "50",
        Progress = 50,
    },
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 05, 02),
    EndDate = new DateTime(2019, 09, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 05, 10),
            Duration = "50",
            Progress = 70,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 05, 10),
            Duration = "50",
            Progress = 50
        },
    })
}
};
return Tasks;
}
}
```



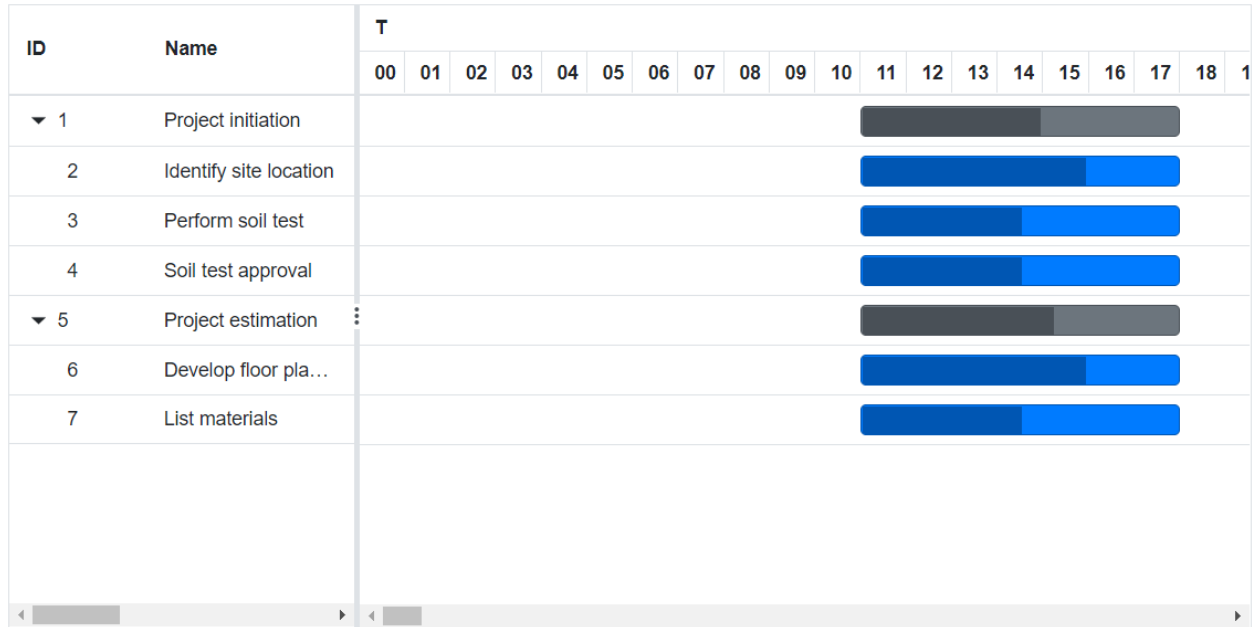
### Day Timeline Mode

In the **Day** timeline mode, top tier of the schedule header displays the days, where as the bottom tier of the schedule header displays its corresponding hours. Refer the following code example.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" DurationUnit="DurationUnit.Hour"
Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttTimelineSettings
TimelineViewMode="TimelineViewMode.Day"></GanttTimelineSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
List<TaskData> Tasks = new List<TaskData> () {
new TaskData() {
```

```
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 2,
        TaskName = "Identify site location",
        StartDate = new DateTime(2019, 04, 02, 09, 0, 0),
        Duration = "5",
        Progress = 70,
    },
    new TaskData() {
        TaskId = 3,
        TaskName = "Perform soil test",
        StartDate = new DateTime(2019, 04, 02, 09, 0, 0),
        Duration = "5",
        Progress = 50,
    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02, 09, 0, 0),
        Duration = "5",
        Progress = 50,
    },
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 02, 11, 0, 0),
            Duration = "5",
            Progress = 70,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 02, 11, 0, 0),
            Duration = "5",
            Progress = 50
        },
    })
}
};
return Tasks;
}
}
```



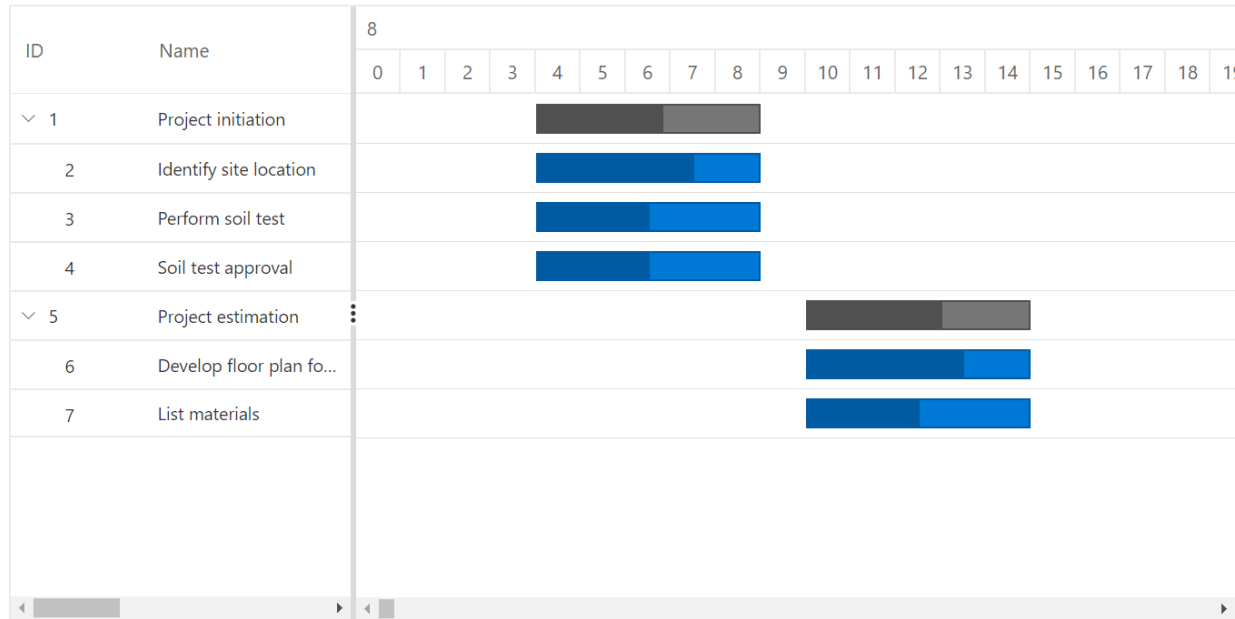
### Hour Timeline Mode

An **Hour** timeline mode tracks the tasks in minutes scale. In this mode, the top tier of the schedule header displays hour scale and the bottom tier of the header displays its corresponding minutes.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" DurationUnit="DurationUnit.Minute"
Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttTimelineSettings
TimelineViewMode="TimelineViewMode.Hour"></GanttTimelineSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
```

```
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 2,
        TaskName = "Identify site location",
        StartDate = new DateTime(2019, 04, 02, 8, 4, 0),
        Duration = "5",
        Progress = 70,
    },
    new TaskData() {
        TaskId = 3,
        TaskName = "Perform soil test",
        StartDate = new DateTime(2019, 04, 02, 8, 4, 0),
        Duration = "5",
        Progress = 50,
    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02, 8, 4, 0),
        Duration = "5",
        Progress = 50,
    },
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 02, 8, 10, 0),
            Duration = "5",
            Progress = 70,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 02, 8, 10, 0),
            Duration = "5",
            Progress = 50,
        },
    })
}
};
return Tasks;
}
```



### Top Tier and Bottom Tier

Gantt Chart component contains two tier layout in timeline, we can customize the top tier and bottom tier using `GanttTopTierSettings` and `GanttBottomTierSettings` properties. Timeline tier's unit can be defined by using `Unit` property and `Format` property was used to define date format of timeline cell and `Count` property was used to define how many unit will be combined as single cell.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
    EndDate="EndDate"
    Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttTimelineSettings>
    <GanttTopTierSettings Unit="TimelineViewMode.Month"
      Format="MMM"></GanttTopTierSettings>
    <GanttBottomTierSettings
      Unit="TimelineViewMode.Day"></GanttBottomTierSettings>
  </GanttTimelineSettings>
</SfGantt>

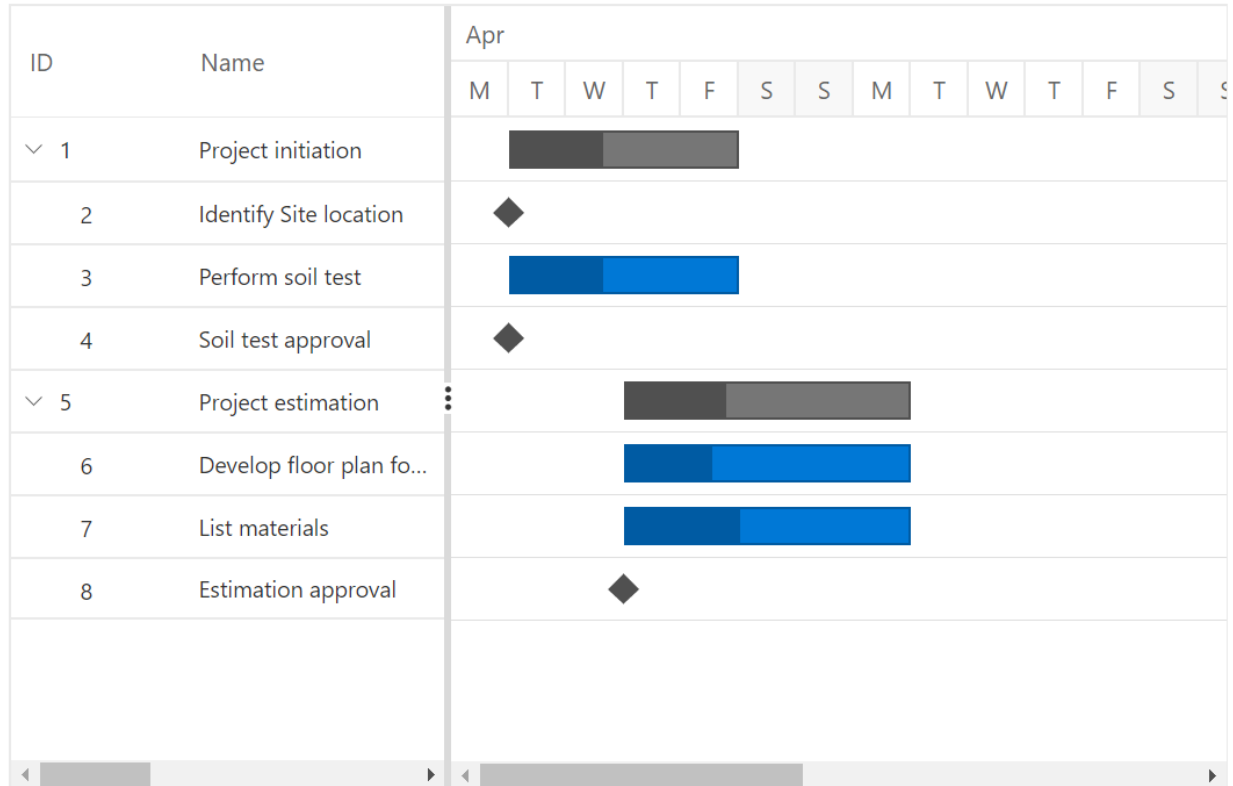
@code{
  public List<TaskData> TaskCollection { get; set; }
  protected override void OnInitialized()
  {
    this.TaskCollection = GetTaskCollection();
  }
  public class TaskData
  {
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
  }
}
```

```
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
```

```

StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
},
})
}
};
return Tasks;
}
}

```



### Combining Timeline Cells

In Gantt Chart, timeline cells in top tier and bottom tier can be combined with number of timeline units, this can be achieved by using `GanttTopTierSettings.Count` and `GanttBottomTierSettings.Count` properties. Please refer the below sample.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttTimelineSettings TimelineUnitSize="200">
    <GanttTopTierSettings Unit="TimelineViewMode.Year"></GanttTopTierSettings>
    <GanttBottomTierSettings Unit="TimelineViewMode.Month" Format="MMM"
    Count="6"></GanttBottomTierSettings>
  </GanttTimelineSettings>
</SfGantt>

```



```
</GanttTimelineSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 08, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "150",
Progress = 70,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "150",
Progress = 50,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "150",
Progress = 50,
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 05, 02),
EndDate = new DateTime(2019, 09, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
```

```

TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 05, 02),
Duration = "150",
Progress = 70,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 05, 02),
Duration = "150",
Progress = 50,
},
})
}
};
return Tasks;
}
}

```



#### Customize Header Timeline Cells

In the Gantt Chart component, you can format the value of top and bottom timeline cells using the standard date format string or the custom formatter method. This can be done using the `GanttTopTierSettings.Format`, `GanttTopTierSettings.FormatterTemplate`, `GanttBottomTierSettings.Format` and `GanttBottomTierSettings.FormatterTemplate` properties. The following example shows how to use the formatter method for timeline cells.

#### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
ProjectStartDate="@ProjectStart" ProjectEndDate="@ProjectEnd">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId">
</GanttTaskFields>
<GanttTimelineSettings TimelineUnitSize=50>
<GanttTopTierSettings Unit="TimelineViewMode.Month" Count="3">
<FormatterTemplate >
@{
@if(context.Tier=="top"){
@this.Formatter((context.Date))
}
}
</FormatterTemplate>
</GanttTopTierSettings>
<GanttBottomTierSettings Unit="TimelineViewMode.Month"
Format="MMM"></GanttBottomTierSettings>
</GanttTimelineSettings>
</SfGantt>
@code{
public DateTime ProjectStart = new DateTime(2019, 01, 10);
public DateTime ProjectEnd = new DateTime(2019, 12, 10);
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public string Formatter(DateTime? date) {
DateTime dateTime=(DateTime)(date);
var month = dateTime.Month;
if (month >= 0 && month <= 2) {
return "Q1";
} else if (month >= 3 && month <= 5) {
return "Q2";
} else if (month >= 6 && month <= 8) {
return "Q3";
} else {
return "Q4";
}
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",

```

```
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 06, 21)
},
new TaskData() {
    TaskId = 2,
    TaskName = "Identify Site location",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "20",
    Progress = 30,
    ParentId = 1
},
new TaskData() {
    TaskId = 3,
    TaskName = "Perform soil test",
    StartDate = new DateTime(2019, 04, 20),
    Duration = "24",
    Progress = 40,
    ParentId = 1
},
new TaskData() {
    TaskId = 4,
    TaskName = "Soil test approval",
    StartDate = new DateTime(2019, 05, 02),
    Duration = "25",
    Progress = 30,
    ParentId = 1
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 06, 02),
    EndDate = new DateTime(2019, 09, 21)
},
new TaskData() {
    TaskId = 6,
    TaskName = "Develop floor plan for estimation",
    StartDate = new DateTime(2019, 06, 04),
    Duration = "33",
    Progress = 30,
    ParentId = 5
},
new TaskData() {
    TaskId = 7,
    TaskName = "List materials",
    StartDate = new DateTime(2019, 07, 04),
    Duration = "23",
    Progress = 40,
    ParentId = 5
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 08, 04),
    Duration = "20",
    Progress = 30,
    ParentId = 5
}
```

```
};
return Tasks;
}
}
```

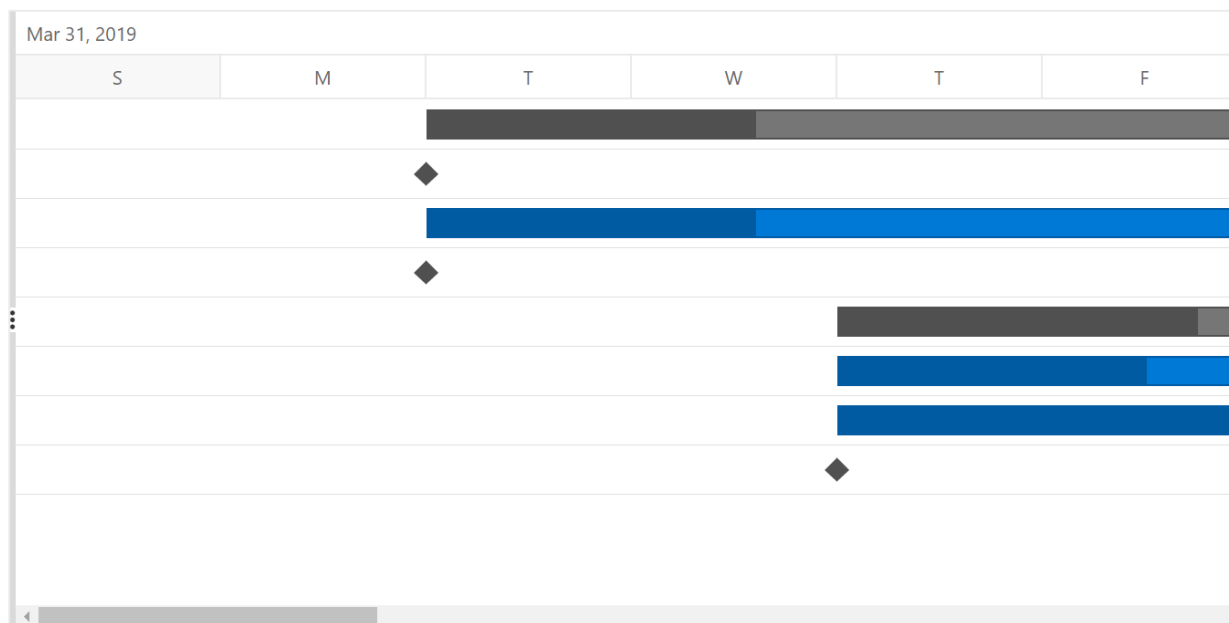
### Timeline Cell Width

In the Gantt Chart component, you can define the width value of timeline cell using the `GanttTimelineSettings.TimelineUnitSize` property. This value will be set to the bottom timeline cell, and the width value of top timeline cell will be calculated automatically based on bottom tier cell width using the `GanttTopTierSettings.Unit` and `GanttTimelineSettings.TimelineUnitSize` properties. Refer the following example.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="1000px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttTimelineSettings TimelineUnitSize=150></GanttTimelineSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
```

```
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30
}
})
}
};
return Tasks;
}
}
```



### Week Start Day Customization

In the Gantt Chart component, you can customize the week start day using the `GanttTimelineSettings.WeekStartDay` property. By default, the `GanttTimelineSettings.WeekStartDay` is set to 0, which specifies the Sunday as a start day of the week. But, you can customize the week start day by using the following code example.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="1000px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttTimelineSettings WeekStartDay=3</GanttTimelineSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
List<TaskData> Tasks = new List<TaskData> () {
```

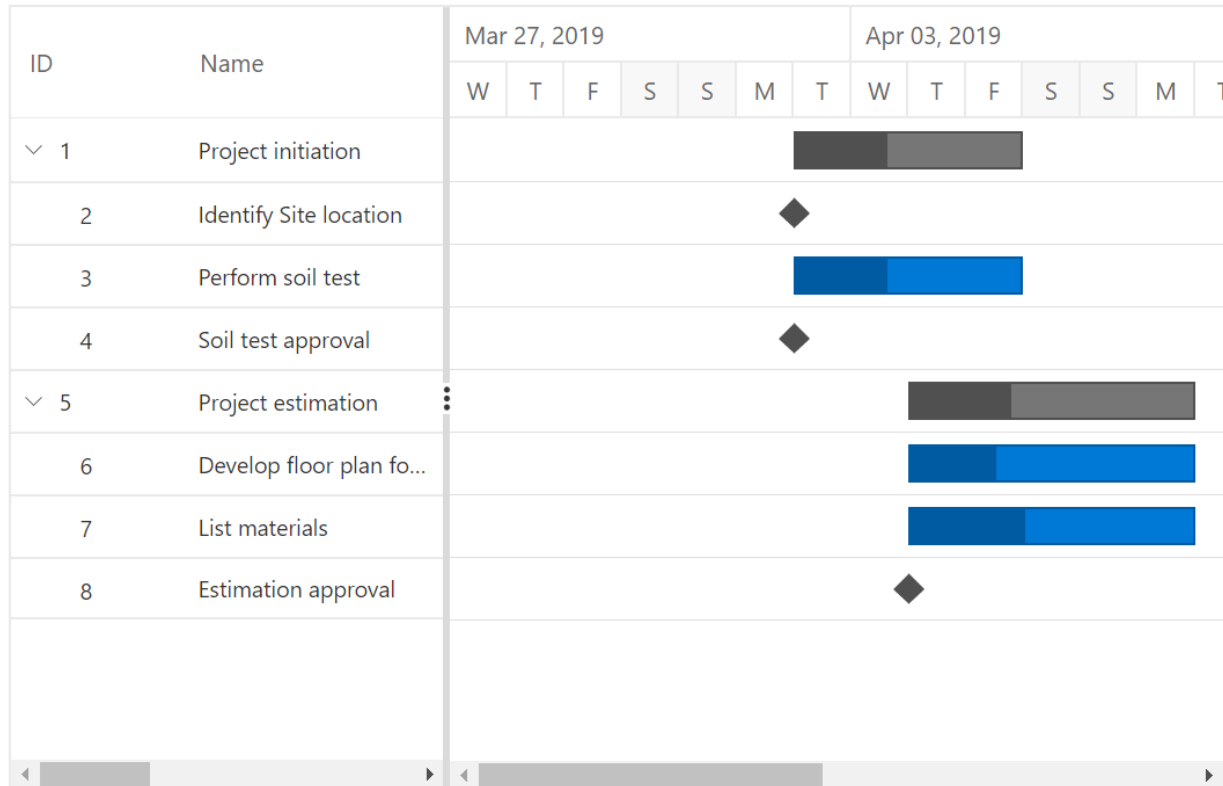
```
new TaskData() {
    TaskId = 1,
    TaskName = "Project initiation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 3,
            TaskName = "Perform soil test",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "4",
            Progress = 40,
        },
        new TaskData() {
            TaskId = 4,
            TaskName = "Soil test approval",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30,
        },
    })
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40,
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30
        }
    })
})
```



```

}
};
return Tasks;
}
}

```



### Customize Automatic Timescale Update action

In the Gantt Chart component, the schedule timeline will be automatically updated when the tasks date values are updated beyond the project start date and end date ranges. This can be enabled or disabled using the `GanttTimelineSettings.UpdateTimescaleView` property.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttEditSettings AllowEditing="true"
  AllowTaskbarEditing="true"></GanttEditSettings>
  <GanttTimelineSettings UpdateTimescaleView="false"></GanttTimelineSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
}

```

```
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
```

```

TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30
}
})
}
};
return Tasks;
}
}

```

### Zooming

Gantt Chart have 25 predefined zooming timespan levels from year timespan to hour timespan. This support enables you to view the tasks in a project clearly from minute to decade timespan. To enable the zooming features, define the **ZoomIn**, **ZoomOut**, and **ZoomToFit** items to toolbar items collections. The following zooming options are available to view the project:

#### Zoom in

This support is used to increase the timeline width and timeline unit from years to minutes timespan. When the **ZoomIn** icon was clicked, the timeline cell width is increased when the cell size exceeds the specified range and the timeline unit is changed based on the current zoom levels.

#### Zoom out

This support is used to increase the timeline width and timeline unit from minutes to years timespan. When the **ZoomOut** icon was clicked, the timeline cell width is decreased when the cell size falls behind the specified range and the timeline view mode is changed based on the current zooming levels.

#### Zoom to fit

This support is used to view all the tasks available in a project to specific timespan which is compatible with available area on the chart part of Gantt Chart. When users click the **ZoomToFit** icon, then all the tasks are rendered within the available chart container width.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Toolbar="@ (new List<string>() {
"ZoomIn", "ZoomOut", "ZoomToFit" })" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks"
Dependency="Predecessor">
</GanttTaskFields>
<GanttLabelSettings LeftLabel="TaskName"
TValue="TaskData"></GanttLabelSettings>
</SfGantt>
@code{

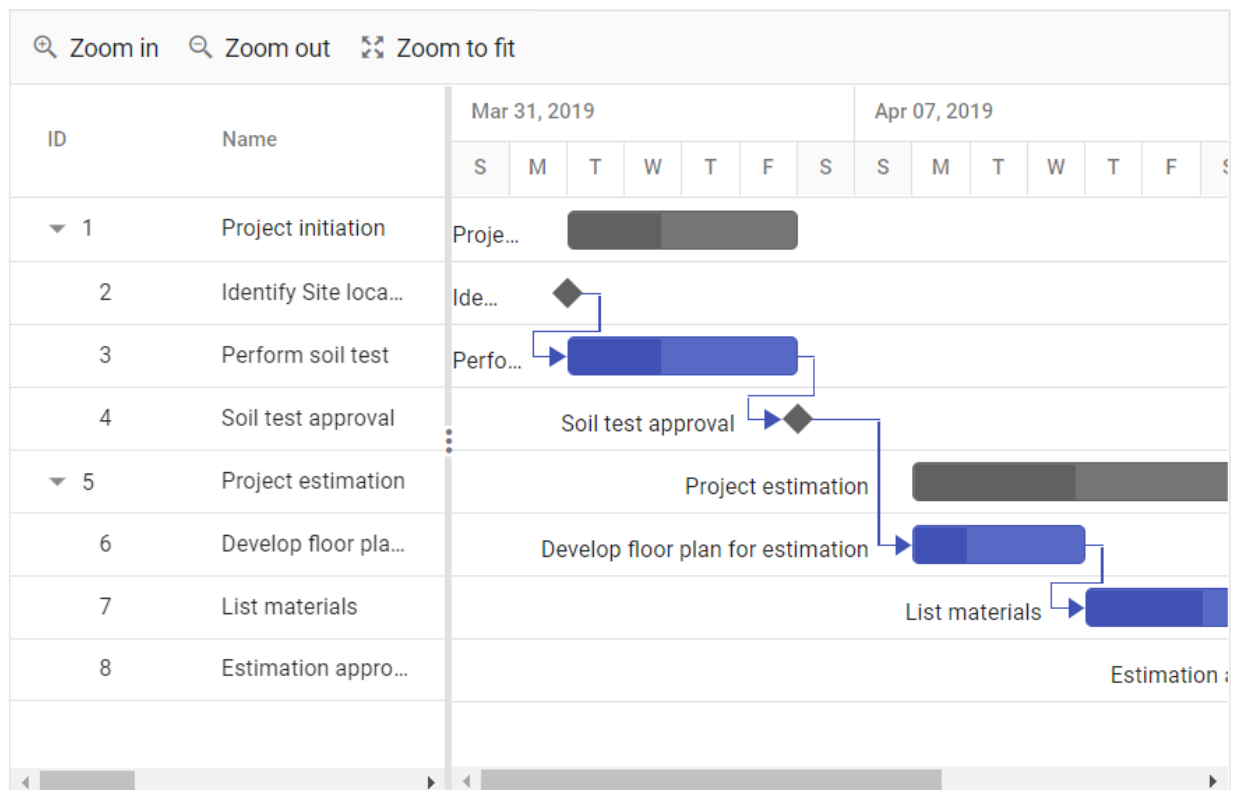
```

```
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public string Predecessor { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                    Predecessor = "2"
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                    Predecessor = "3"
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
```

```

TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
Predecessor = "4"
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
Predecessor = "6"
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
Predecessor = "7"
},
})
}
};
return Tasks;
}
}

```



### Zoom action by methods

Zooming action also can be performed on external actions such as button click using the `ZoomIn()`, `ZoomOut()`, and `ZoomToFitAsync()` built-in methods.

```
<!-- `cshtml`  
@using Syncfusion.Blazor.Gantt  
  
<button @onclick="ZoomIn">Zoom In</button>  
<button @onclick="ZoomOut">Zoom Out</button>  
<button @onclick="ZoomToFit">ZoomToFit</button>  
  
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px" Width="700px">  
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate" EndDate="EndDate"  
    Duration="Duration" Progress="Progress" Child="SubTasks" Dependency="Predecessor">  
  </GanttTaskFields>  
  
  <GanttLabelSettings LeftLabel="TaskName" TValue="TaskData"></GanttLabelSettings>  
</SfGantt>  
  
@code{  
  public SfGantt<TaskData> Gantt;  
  
  public void ZoomIn()  
  {  
    this.Gantt.ZoomIn();  
  }  
  
  public void ZoomOut()  
  {  
    this.Gantt.ZoomOut();  
  }  
  
  public void ZoomToFit()  
  {  
    this.Gantt.ZoomToFit();  
  }  
  
  public List<TaskData> TaskCollection { get; set; }  
  protected override void OnInitialized()  
  {  
    this.TaskCollection = GetTaskCollection();  
  }  
  
  public class TaskData
```

```
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public string Predecessor { get; set; }
public List<TaskData> SubTasks { get; set; }
}

public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
Predecessor = "2"
},

```

```
new TaskData() {
    TaskId = 4,
    TaskName = "Soil test approval",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "0",
    Progress = 30,
    Predecessor = "3"
},
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
            Predecessor = "4"
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40,
            Predecessor = "6"
        },
        new TaskData() {
```



```

TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
Predecessor = "7"
},
})
}
};
return Tasks;
}
}
-->

```

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttEventMarkers>
    <GanttEventMarker Day="@Event" Label="Project approval and kick-off"
    CssClass="e-custom-event-marker"></GanttEventMarker>
  </GanttEventMarkers>
</SfGantt>
@code{
public DateTime Event = new DateTime(2019, 04, 11);
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {

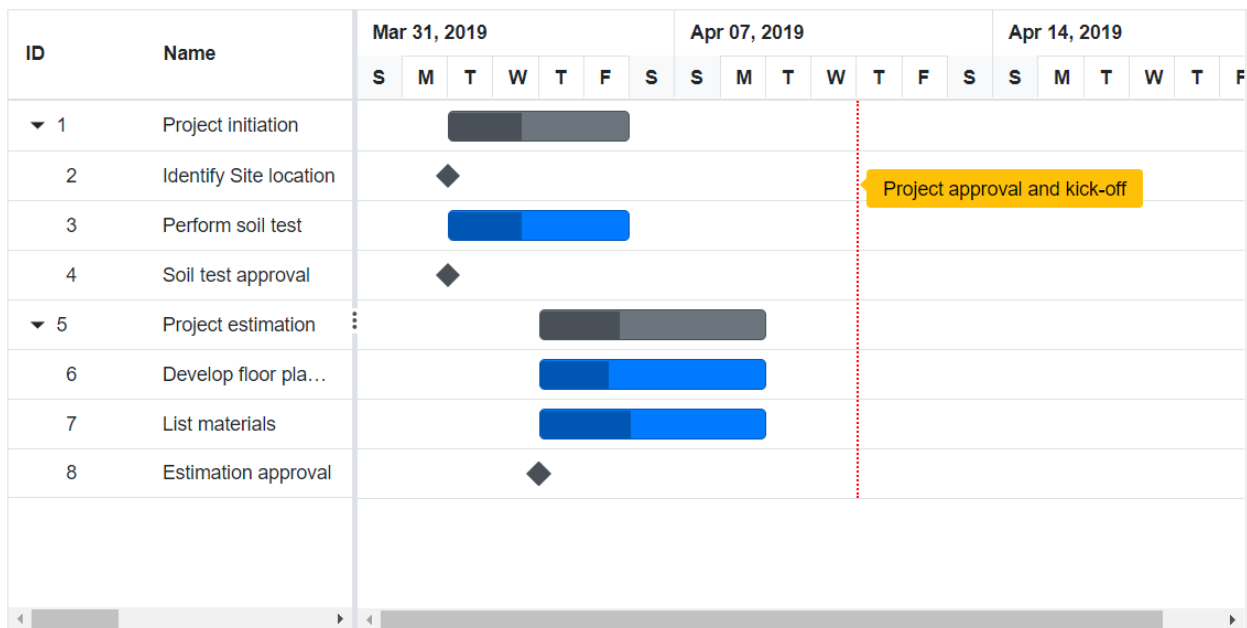
```

```
List <TaskData> Tasks = new List <TaskData> () {
    new TaskData() {
        TaskId = 1,
        TaskName = "Project initiation",
        StartDate = new DateTime(2019, 04, 02),
        EndDate = new DateTime(2019, 04, 21),
        SubTasks = (new List <TaskData> () {
            new TaskData() {
                TaskId = 2,
                TaskName = "Identify Site location",
                StartDate = new DateTime(2019, 04, 02),
                Duration = "0",
                Progress = 30,
            },
            new TaskData() {
                TaskId = 3,
                TaskName = "Perform soil test",
                StartDate = new DateTime(2019, 04, 02),
                Duration = "4",
                Progress = 40,
            },
            new TaskData() {
                TaskId = 4,
                TaskName = "Soil test approval",
                StartDate = new DateTime(2019, 04, 02),
                Duration = "0",
                Progress = 30,
            },
        })
    },
    new TaskData() {
        TaskId = 5,
        TaskName = "Project estimation",
        StartDate = new DateTime(2019, 04, 02),
        EndDate = new DateTime(2019, 04, 21),
        SubTasks = (new List <TaskData> () {
            new TaskData() {
                TaskId = 6,
                TaskName = "Develop floor plan for estimation",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "3",
                Progress = 30,
            },
            new TaskData() {
                TaskId = 7,
                TaskName = "List materials",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "3",
                Progress = 40,
            },
            new TaskData() {
                TaskId = 8,
                TaskName = "Estimation approval",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "0",
                Progress = 30
            }
        })
    }
}
```

```

    })
    }
    };
    return Tasks;
    }
    }
<style>
.e-gantt .e-gantt-chart .e-custom-event-marker {
width: 1px;
border-left: 2px red dotted;
}
</style>

```



You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

## Holidays in Blazor Gantt Chart Component

Non-working days in a project can be displayed in the Gantt Chart component using the `GanttHolidays` property. Each holiday can be defined with the following properties:

- **From:** Defines start date of the holiday(s).
- **To:** Defines end date of the holiday(s).
- **Label:** Defines the description or label for the holiday.
- **CssClass:** Formats the holidays label in the Gantt chart.

The following code example shows how to display the non-working days in the Gantt Chart component.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">

```

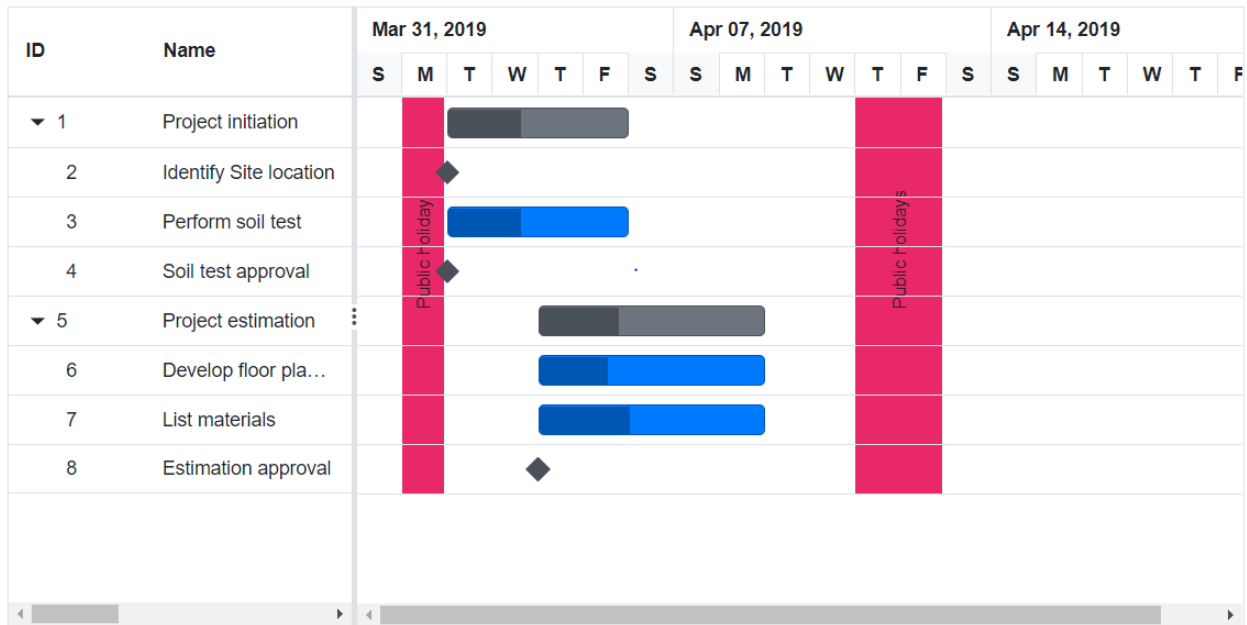
```

<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttHolidays>
<GanttHoliday From="@Holiday1" To="@Holiday2" Label="Public holidays"
CssClass="e-custom-holiday"></GanttHoliday>
<GanttHoliday From="@Holiday3" To="@Holiday4" Label="Public holiday"
CssClass="e-custom-holiday"></GanttHoliday>
</GanttHolidays>
</SfGantt>
@code{
public DateTime Holiday1 = new DateTime(2019, 04, 11);
public DateTime Holiday2 = new DateTime(2019, 04, 12);
public DateTime Holiday3 = new DateTime(2019, 04, 01);
public DateTime Holiday4 = new DateTime(2019, 04, 01);
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
List<TaskData> Tasks = new List<TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),

```

```
Duration = "0",
Progress = 30,
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30
}
})
}
};
return Tasks;
}
}
<style>
.e-gantt .e-gantt-chart .e-custom-holiday {
background-color: #e82869;
}
</style>
```

The following screenshot shows the output of Holidays in Gantt Chart component.



You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

## Resources in Blazor Gantt Chart Component

In [Blazor Gantt Chart](#), the resources are represented by staff, equipment and materials etc. In Gantt Chart component you can show or allocate the resources (human resources) for each task.

### Resource Collection

The resource collection contains details about resources that are used in the project. Resources are List of `TResources` object that contains id, name and unit of the resources and this collection is mapped to the Gantt Chart component using the `GanttResourceFields.Resources` property. Id, name and unit field of the resources are mapped by using the `GanttResourceFields.Id`, `GanttResourceFields.Name` and `GanttResourceFields.Unit` properties. The following code snippets shows resource collection and how it assigned to Gantt Chart component.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="100%"
ProjectStartDate="@ProjectStart" ProjectEndDate="@ProjectEnd"
WorkUnit="WorkUnit.Hour">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress"
  ParentID="ParentId" ResourceInfo="Resources">
  </GanttTaskFields>
  <GanttColumns>
    <GanttColumn Field="TaskId" HeaderText="ID"></GanttColumn>
    <GanttColumn Field="TaskName" HeaderText="Event Name"></GanttColumn>
    <GanttColumn Field="Resources" HeaderText="Event Resources"
    Width="300px"></GanttColumn>
    <GanttColumn Field="Duration" HeaderText="Duration"></GanttColumn>
    <GanttColumn Field="StartDate" HeaderText="Start Date"></GanttColumn>
    <GanttColumn Field="EndDate" HeaderText="End Date"></GanttColumn>
```

```

</GanttColumns>
<GanttResourceFields Resources="@ResourceCollection" Id="ResourceId"
Name="ResourceName" Unit="Unit"
TResources="ResourceAlloacteData"></GanttResourceFields>
<GanttLabelSettings RightLabel="Resources"
TValue="TaskData"></GanttLabelSettings>
</SfGantt>
@code{
public DateTime ProjectStart = new DateTime(2019, 03, 25);
public DateTime ProjectEnd = new DateTime(2019, 05, 10);
public List<TaskData> TaskCollection { get; set; }
public List<ResourceAlloacteData> ResourceCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
this.ResourceCollection = GetResources;
}
public class ResourceAlloacteData
{
public int ResourceId { get; set; }
public string ResourceName { get; set; }
public double Unit { get; set; }
}
public static List<ResourceAlloacteData> GetResources = new
List<ResourceAlloacteData>()
{
new ResourceAlloacteData() { ResourceId= 1, ResourceName= "Martin Tamer"},
new ResourceAlloacteData() { ResourceId= 2, ResourceName= "Rose Fuller" },
new ResourceAlloacteData() { ResourceId= 3, ResourceName= "Margaret
Buchanan" },
new ResourceAlloacteData() { ResourceId= 4, ResourceName= "Fuller King" },
new ResourceAlloacteData() { ResourceId= 5, ResourceName= "Davolio Fuller"
},
new ResourceAlloacteData() { ResourceId= 7, ResourceName= "Fuller Buchanan"
},
new ResourceAlloacteData() { ResourceId= 8, ResourceName= "Jack Davolio" },
new ResourceAlloacteData() { ResourceId= 9, ResourceName= "Tamer Vinet" },
new ResourceAlloacteData() { ResourceId= 10, ResourceName= "Vinet Fuller" },
new ResourceAlloacteData() { ResourceId= 11, ResourceName= "Bergs Anton" },
new ResourceAlloacteData() { ResourceId= 12, ResourceName= "Construction
Supervisor" }
};
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
public List<ResourceAlloacteData> Resources { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,

```

```
TaskName = "Project initiation",
StartDate = new DateTime(2019, 03, 28),
EndDate = new DateTime(2019, 07, 28),
Duration="4"
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 03, 29),
Progress = 30,
ParentId = 1,
Duration="2",
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=1, Unit=70} ,new ResourceAlloacteData() { ResourceId=6} }
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 03, 29),
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=2} ,new ResourceAlloacteData() { ResourceId=3} },
ParentId = 1,
Duration="4"
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 03, 29),
Duration = "1",
Progress = 30,
ParentId = 1,
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=8} ,new ResourceAlloacteData() { ResourceId=9} }
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 03, 29),
EndDate = new DateTime(2019, 04, 2),
Duration="4"
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 03, 29),
Duration = "3",
Progress = 30,
ParentId = 5,
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=4} }
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 01),
Duration = "3",
Progress = 30,
```



```

ParentId = 5,
Resources = new List<ResourceAlloacteData>() { new ResourceAlloacteData() {
ResourceId=4},new ResourceAlloacteData() { ResourceId=8} },
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 01),
Duration = "2",
ParentId = 5,
Resources = new List<ResourceAlloacteData>() { new ResourceAlloacteData() {
ResourceId= 12},new ResourceAlloacteData() { ResourceId= 5} },
}
};
return Tasks;
}
}

```

### Assign Resource

We can assign resources for a task at initial load by using the resource id value of the resources as a collection. This collection is mapped from the dataSource to the Gantt Chart component using the `GanttTaskFields.ResourceInfo` property.

- Gantt TValue for Resource mapping collection name should be the same as `GanttTaskFields.ResourceInfo`.
- Gantt Resource mapping collection should have the value for Id. Both Name and Unit values are optional.
- If the unit is not specified for a specific resource, the amount of work done will be considered as 100% by default. In such cases, the resource unit will not be displayed in Gantt UI.

### Assign Resource with Unit

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

#### CSHARP

```

new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 03, 29),
Progress = 30,
ParentId = 1,
Duration="2",
Resources = new List<ResourceAlloacteData>() { new ResourceAlloacteData() {
ResourceId=1, Unit=70} }
},

```

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

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

**ASPX-CS**

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="100%"
ProjectStartDate="@ProjectStart" ProjectEndDate="@ProjectEnd"
WorkUnit="WorkUnit.Hour">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId" Work="Work" ResourceInfo="Resources"
TaskType="TaskType">
</GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" HeaderText="ID"></GanttColumn>
<GanttColumn Field="TaskName" HeaderText="Event Name"></GanttColumn>
<GanttColumn Field="Resources" HeaderText="Event Resources"
Width="300px"></GanttColumn>
<GanttColumn Field="Work" HeaderText="Work"></GanttColumn>
<GanttColumn Field="Duration" HeaderText="Duration"></GanttColumn>
<GanttColumn Field="TaskType" HeaderText="Task Type"></GanttColumn>
<GanttColumn Field="StartDate" HeaderText="Start Date"></GanttColumn>
<GanttColumn Field="EndDate" HeaderText="End Date"></GanttColumn>
</GanttColumns>
<GanttResourceFields Resources="@ResourceCollection" Id="ResourceId"
Name="ResourceName" Unit="Unit"
TResources="ResourceAlloacteData"></GanttResourceFields>
<GanttLabelSettings RightLabel="Resources"
TValue="TaskData"></GanttLabelSettings>
</SfGantt>
@code{
public DateTime ProjectStart = new DateTime(2019, 03, 25);
public DateTime ProjectEnd = new DateTime(2019, 05, 10);
public List<TaskData> TaskCollection { get; set; }
public List<ResourceAlloacteData> ResourceCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
this.ResourceCollection = GetResources;
}
public class ResourceAlloacteData
{
public int ResourceId { get; set; }
public string ResourceName { get; set; }
public double Unit { get; set; }
}
public static List<ResourceAlloacteData> GetResources = new
List<ResourceAlloacteData>()
{
new ResourceAlloacteData() { ResourceId= 1, ResourceName= "Martin Tamer"},
new ResourceAlloacteData() { ResourceId= 2, ResourceName= "Rose Fuller" },
new ResourceAlloacteData() { ResourceId= 3, ResourceName= "Margaret
Buchanan" },
new ResourceAlloacteData() { ResourceId= 4, ResourceName= "Fuller King" },
new ResourceAlloacteData() { ResourceId= 5, ResourceName= "Davolio Fuller"
},
new ResourceAlloacteData() { ResourceId= 7, ResourceName= "Fuller Buchanan"
},
new ResourceAlloacteData() { ResourceId= 8, ResourceName= "Jack Davolio" },

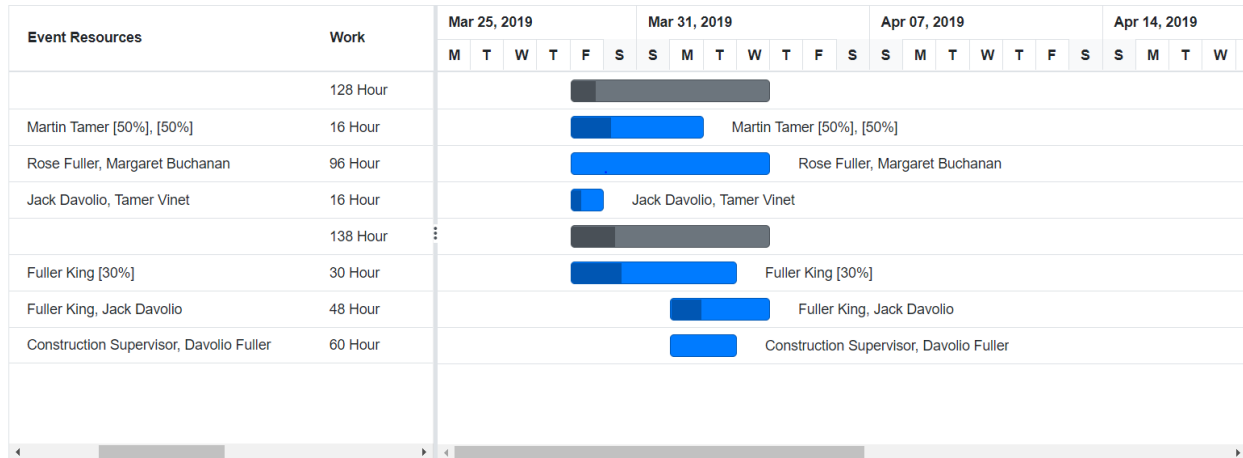
```

```

new ResourceAlloacteData() { ResourceId= 9, ResourceName= "Tamer Vinet" },
new ResourceAlloacteData() { ResourceId= 10, ResourceName= "Vinet Fuller" },
new ResourceAlloacteData() { ResourceId= 11, ResourceName= "Bergs Anton" },
new ResourceAlloacteData() { ResourceId= 12, ResourceName= "Construction
Supervisor" }
};
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public string TaskType { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public int? ParentId { get; set; }
    public double? Work { get; set; }
    public List<ResourceAlloacteData> Resources { get; set; }
}
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 03, 28),
            EndDate = new DateTime(2019, 07, 28),
            TaskType = "FixedDuration",
            Work=128,
            Duration="4"
        },
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 03, 29),
            Progress = 30,
            ParentId = 1,
            Duration="2",
            Work=16,
            Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
                ResourceId=1, Unit=70} ,new ResourceAlloacteData() { ResourceId=6} }
        },
        new TaskData() {
            TaskId = 3,
            TaskName = "Perform soil test",
            StartDate = new DateTime(2019, 03, 29),
            Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
                ResourceId=2} ,new ResourceAlloacteData() { ResourceId=3} },
            ParentId = 1,
            Work=96,
            Duration="4",
            TaskType="Fixed work"
        },
        new TaskData() {
            TaskId = 4,
            TaskName = "Soil test approval",
            StartDate = new DateTime(2019, 03, 29),
            Duration = "1",

```

```
Progress = 30,
ParentId = 1,
Resources = new List<ResourceAlloacteData>() { new ResourceAlloacteData() {
ResourceId=8} ,new ResourceAlloacteData() { ResourceId=9} },
Work=16,
TaskType="Fixed work"
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 03, 29),
EndDate = new DateTime(2019, 04, 2),
TaskType="Fixed Duration",
Duration="4"
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 03, 29),
Duration = "3",
Progress = 30,
ParentId = 5,
Resources = new List<ResourceAlloacteData>() { new ResourceAlloacteData() {
ResourceId=4, Unit=30} },
Work=30,
TaskType="Fixed work"
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 01),
Duration = "3",
Progress = 30,
ParentId = 5,
TaskType="Fixedwork",
Work=48,
Resources = new List<ResourceAlloacteData>() { new ResourceAlloacteData() {
ResourceId=4},new ResourceAlloacteData() { ResourceId=8} },
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 01),
Duration = "2",
ParentId = 5,
Work=60,
TaskType="Fixedwork",
Resources = new List<ResourceAlloacteData>() { new ResourceAlloacteData() {
ResourceId= 12},new ResourceAlloacteData() { ResourceId= 5} },
}
};
return Tasks;
}
}
```



### Add / Edit Resource Collection

By using cell editing or dialog editing, we can add/remove the resource for particular task.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="100%"
ProjectStartDate="@ProjectStart" ProjectEndDate="@ProjectEnd"
WorkUnit="WorkUnit.Hour">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId" Work="Work" ResourceInfo="Resources"
TaskType="TaskType">
</GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" HeaderText="ID"></GanttColumn>
<GanttColumn Field="TaskName" HeaderText="Event Name"></GanttColumn>
<GanttColumn Field="Resources" HeaderText="Event Resources"
Width="300px"></GanttColumn>
<GanttColumn Field="Duration" HeaderText="Duration"></GanttColumn>
<GanttColumn Field="StartDate" HeaderText="Start Date"></GanttColumn>
<GanttColumn Field="EndDate" HeaderText="End Date"></GanttColumn>
</GanttColumns>
<GanttEditSettings AllowEditing="true"></GanttEditSettings>
<GanttResourceFields Resources="@ResourceCollection" Id="ResourceId"
Name="ResourceName" Unit="Unit"
TResources="ResourceAlloactedData"></GanttResourceFields>
<GanttLabelSettings RightLabel="Resources"
TValue="TaskData"></GanttLabelSettings>
</SfGantt>
@code{
public DateTime ProjectStart = new DateTime(2019, 03, 25);
public DateTime ProjectEnd = new DateTime(2019, 05, 10);
public List<TaskData> TaskCollection { get; set; }
public List<ResourceAlloactedData> ResourceCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
this.ResourceCollection = GetResources;
}
public class ResourceAlloactedData
```

```

{
    public int ResourceId { get; set; }
    public string ResourceName { get; set; }
    public double Unit { get; set; }
}
public static List<ResourceAlloacteData> GetResources = new
List<ResourceAlloacteData>()
{
    new ResourceAlloacteData() { ResourceId= 1, ResourceName= "Martin Tamer"},
    new ResourceAlloacteData() { ResourceId= 2, ResourceName= "Rose Fuller" },
    new ResourceAlloacteData() { ResourceId= 3, ResourceName= "Margaret
    Buchanan" },
    new ResourceAlloacteData() { ResourceId= 4, ResourceName= "Fuller King" },
    new ResourceAlloacteData() { ResourceId= 5, ResourceName= "Davolio Fuller"
    },
    new ResourceAlloacteData() { ResourceId= 7, ResourceName= "Fuller Buchanan"
    },
    new ResourceAlloacteData() { ResourceId= 8, ResourceName= "Jack Davolio" },
    new ResourceAlloacteData() { ResourceId= 9, ResourceName= "Tamer Vinet" },
    new ResourceAlloacteData() { ResourceId= 10, ResourceName= "Vinet Fuller" },
    new ResourceAlloacteData() { ResourceId= 11, ResourceName= "Bergs Anton" },
    new ResourceAlloacteData() { ResourceId= 12, ResourceName= "Construction
    Supervisor" }
};
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public string TaskType { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public int? ParentId { get; set; }
    public double? Work { get; set; }
    public List<ResourceAlloacteData> Resources { get; set; }
}
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 03, 28),
            EndDate = new DateTime(2019, 07, 28),
            TaskType ="FixedDuration",
            Work=128,
            Duration="4"
        },
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 03, 29),
            Progress = 30,
            ParentId = 1,
            Duration="2",
            Work=16,

```

```
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=1, Unit=70} ,new ResourceAlloacteData() { ResourceId=6} }
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 03, 29),
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=2} ,new ResourceAlloacteData() { ResourceId=3} },
ParentId = 1,
Work=96,
Duration="4",
TaskType="Fixed work"
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 03, 29),
Duration = "1",
Progress = 30,
ParentId = 1,
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=8} ,new ResourceAlloacteData() { ResourceId=9} },
Work=16,
TaskType="Fixed work"
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 03, 29),
EndDate = new DateTime(2019, 04, 2),
TaskType="Fixed Duration",
Duration="4"
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 03, 29),
Duration = "3",
Progress = 30,
ParentId = 5,
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=4} },
Work=30,
TaskType="Fixed work"
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 01),
Duration = "3",
Progress = 30,
ParentId = 5,
TaskType="Fixedwork",
Work=48,
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=4},new ResourceAlloacteData() { ResourceId=8} },
```

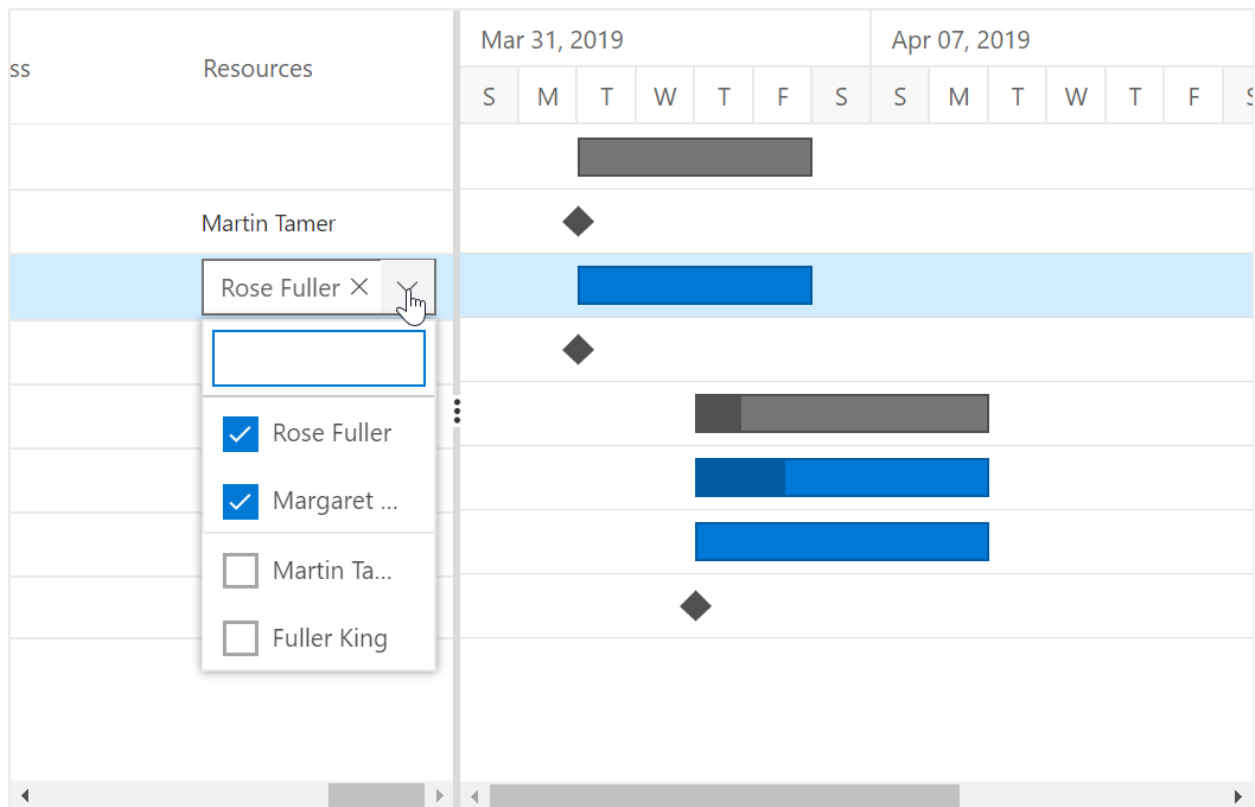
```

},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 01),
    Duration = "2",
    ParentId = 5,
    Work=60,
    TaskType="Fixedwork",
    Resources = new List<ResourceAlloacteData>() { new ResourceAlloacteData() {
        ResourceId= 12},new ResourceAlloacteData() { ResourceId= 5} },
    };
return Tasks;
}
}

```

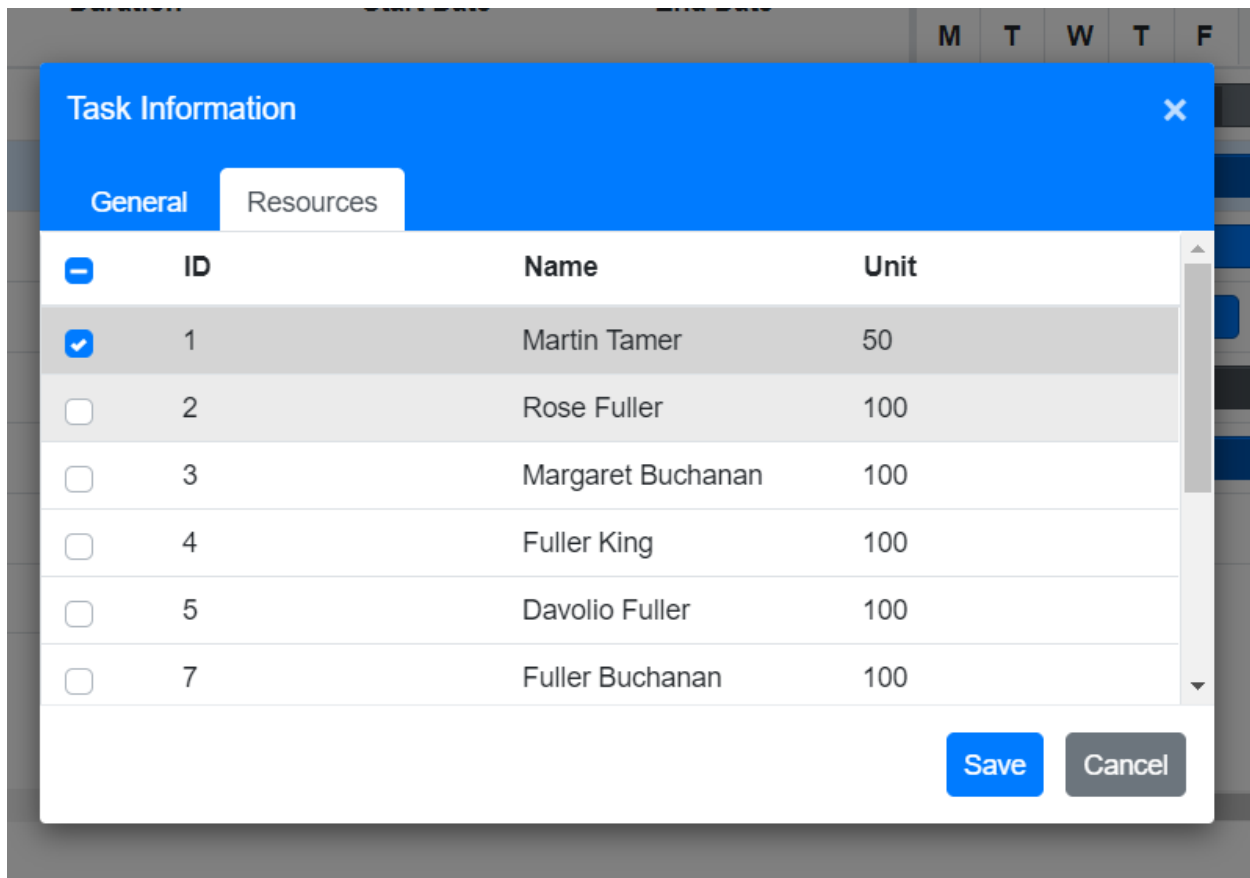
**Note:** When the edit mode is set as **Auto**, on performing double click action on Tree Grid side the cells will be changed to editable mode and on performing double click action on chart side the edit dialog will appear for editing the task details. By using this support we can add/remove the resource for particular task using both cell and edit dialog

Editing resource with cell edit



Editing resource with edit dialog





### Work in Blazor Gantt Chart Component

The work is the total hours required to complete a task. Work can be mapped from the data source field using the property `GanttTaskFields.Work`. Work can be measured in Hour, Day, Minute. By default, work is measured in Hour and it can be changed, by using the property `WorkUnit`.

When the work field is mapped from the data source, the default task type will be `FixedWork`.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="1000px" WorkUnit="WorkUnit.Hour" ProjectStartDate="@ProjectStart"
ProjectEndDate="@ProjectEnd">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress"
  ParentID="ParentId" Work="Work" ResourceInfo="Resources"></GanttTaskFields>
  <GanttEditSettings AllowAdding="true" AllowDeleting="true"
  AllowEditing="true" AllowTaskbarEditing="true"
  ShowDeleteConfirmDialog="true"></GanttEditSettings>
  <GanttResourceFields Resources="GetResources" Id="ResourceId"
  Name="ResourceName" Unit="Unit"
  TResources="ResourceData"></GanttResourceFields>
  <GanttLabelSettings TValue="TaskData"
  RightLabel="Resources"></GanttLabelSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
```

```
public DateTime ProjectStart = new DateTime(2019, 03, 25);
public DateTime ProjectEnd = new DateTime(2019, 05, 10);
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class ResourceData
{
    public int ResourceId { get; set; }
    public string ResourceName { get; set; }
    public double Unit { get; set; }
}
List<ResourceData> GetResources = new List<ResourceData>() {
    new ResourceData() { ResourceId= 1, ResourceName= "Martin Tamer" ,Unit=70},
    new ResourceData() { ResourceId= 2, ResourceName= "Rose Fuller" },
    new ResourceData() { ResourceId= 3, ResourceName= "Margaret Buchanan" },
    new ResourceData() { ResourceId= 4, ResourceName= "Fuller King" },
    new ResourceData() { ResourceId= 5, ResourceName= "Davolio Fuller" },
    new ResourceData() { ResourceId= 6, ResourceName= "Van Jack" },
    new ResourceData() { ResourceId= 7, ResourceName= "Fuller Buchanan" },
    new ResourceData() { ResourceId= 8, ResourceName= "Jack Davolio" },
    new ResourceData() { ResourceId= 9, ResourceName= "Tamer Vinet" },
    new ResourceData() { ResourceId= 10, ResourceName= "Vinet Fuller" },
    new ResourceData() { ResourceId= 11, ResourceName= "Bergs Anton" },
    new ResourceData() { ResourceId= 12, ResourceName= "Construction Supervisor"
    }
};
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public string TaskType { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public int? ParentId { get; set; }
    public double? Work { get; set; }
    public List<ResourceData> Resources { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 03, 29),
            EndDate = new DateTime(2019, 04, 21),
            TaskType = "FixedDuration"
        },
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 03, 29),
            Progress = 30,
            ParentId = 1,
```

```
Work=16,
Resources = new List<ResourceData>(){ new ResourceData() {
ResourceId=1,Unit=70} ,new ResourceData() { ResourceId=6} }
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 03, 29),
Resources = new List<ResourceData>(){ new ResourceData() { ResourceId=2}
,new ResourceData() { ResourceId=3} ,new ResourceData() { ResourceId=5} },
ParentId = 1,
Work=96
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 03, 29),
Duration = "1",
Progress = 30,
ParentId = 1,
Resources = new List<ResourceData>(){ new ResourceData() { ResourceId=8}
,new ResourceData() { ResourceId=9} },
Work=16
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 03, 29),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 03, 29),
Duration = "3",
Progress = 30,
ParentId = 5,
Resources = new List<ResourceData>(){ new ResourceData() { ResourceId=4} },
Work=30
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 01),
Duration = "3",
Progress = 30,
ParentId = 5,
Work=48,
Resources = new List<ResourceData>(){ new ResourceData() { ResourceId=6},new
ResourceData() { ResourceId=8} },
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 01),
Duration = "2",
ParentId = 5,
```

```

Work=60,
Resources = new List<ResourceData>() { new ResourceData() { ResourceId=
12}, new ResourceData() { ResourceId= 5} }
};
return Tasks;
}
}

```

### Task type

The work, duration and resource unit fields of a task depends upon each other and will change automatically on editing any one of these fields. But we can also set these field's values as constant using the **TaskType** property. **FixedUnit** is the default **TaskType**. The following values can be set to the **TaskType** property.

- **FixedDuration** - Duration task field will remain constant while updating resource unit or work field.
- **FixedWork** - Work field will remain constant while updating resource unit or duration fields.
- **FixedUnit** - Resource units will remain constant while updating duration or work field.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="900px" HighlightWeekends="true" ProjectStartDate="@ProjectStart"
ProjectEndDate="@ProjectEnd">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress"
  ParentID="ParentId"
  Work="Work" ResourceInfo="Resources" TaskType="TaskType"></GanttTaskFields>
  <GanttEditSettings AllowAdding="true" AllowDeleting="true"
  AllowEditing="true" AllowTaskbarEditing="true"></GanttEditSettings>
  <GanttResourceFields Resources="@ResourceCollection" Id="ResourceId"
  Name="ResourceName" Unit="Unit"
  TResources="ResourceAlloactedData"></GanttResourceFields>
  <GanttLabelSettings TValue="TaskData"
  RightLabel="Resources"></GanttLabelSettings>
</SfGantt>
@code{
public DateTime ProjectStart = new DateTime(2019, 03, 25);
public DateTime ProjectEnd = new DateTime(2019, 05, 10);
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
public List<ResourceAlloactedData> ResourceCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
  this.ResourceCollection = GetResources;
}
public class ResourceAlloactedData
{
  public int ResourceId { get; set; }
  public string ResourceName { get; set; }
}

```

```

public double Unit { get; set; }
}
public static List<ResourceAlloacteData> GetResources = new
List<ResourceAlloacteData>()
{
    new ResourceAlloacteData() { ResourceId= 1, ResourceName= "Martin Tamer"},
    new ResourceAlloacteData() { ResourceId= 2, ResourceName= "Rose Fuller" },
    new ResourceAlloacteData() { ResourceId= 3, ResourceName= "Margaret
    Buchanan" },
    new ResourceAlloacteData() { ResourceId= 4, ResourceName= "Fuller King" },
    new ResourceAlloacteData() { ResourceId= 5, ResourceName= "Davolio Fuller"
    },
    new ResourceAlloacteData() { ResourceId= 7, ResourceName= "Fuller Buchanan"
    },
    new ResourceAlloacteData() { ResourceId= 8, ResourceName= "Jack Davolio" },
    new ResourceAlloacteData() { ResourceId= 9, ResourceName= "Tamer Vinet" },
    new ResourceAlloacteData() { ResourceId= 10, ResourceName= "Vinet Fuller" },
    new ResourceAlloacteData() { ResourceId= 11, ResourceName= "Bergs Anton" },
    new ResourceAlloacteData() { ResourceId= 12, ResourceName= "Construction
    Supervisor" }
};
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public string TaskType { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public double Progress { get; set; }
    public int? ParentId { get; set; }
    public string Notes { get; set; }
    public double? Work { get; set; }
    public List<ResourceAlloacteData> Resources { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
    new TaskData() {
        TaskId = 1,
        TaskName = "Project initiation",
        StartDate = new DateTime(2019, 03, 28),
        EndDate = new DateTime(2019, 07, 28),
        TaskType = "FixedDuration",
        Work=128,
        Duration="4"
    },
    new TaskData() {
        TaskId = 2,
        TaskName = "Identify Site location",
        StartDate = new DateTime(2019, 03, 29),
        Progress = 30,
        ParentId = 1,
        Duration="2",
        Work=16,
        TaskType="Fixed work",

```

```
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=1, Unit=70} ,new ResourceAlloacteData() { ResourceId=6} }
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 03, 29),
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=2} ,new ResourceAlloacteData() { ResourceId=3} },
ParentId = 1,
Work=96,
Duration="4",
TaskType="Fixed work"
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 03, 29),
Duration = "1",
Progress = 30,
ParentId = 1,
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=8} ,new ResourceAlloacteData() { ResourceId=9} },
Work=16,
TaskType="Fixed work"
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 06),
TaskType="Fixed Duration",
Duration="4"
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 02),
Duration = "3",
Progress = 30,
ParentId = 5,
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=4} },
Work=30,
TaskType="Fixed work"
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 03),
Duration = "3",
Progress = 30,
ParentId = 5,
TaskType="Fixedwork",
Work=48,
Resources = new List<ResourceAlloacteData>(){ new ResourceAlloacteData() {
ResourceId=4},new ResourceAlloacteData() { ResourceId=8} },
```

```

    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 03),
        Duration = "2",
        ParentId = 5,
        Work=60,
        TaskType="Fixedwork",
        Resources = new List<ResourceAlloacteData>() { new ResourceAlloacteData() {
            ResourceId= 12},new ResourceAlloacteData() { ResourceId= 5} },
        };
    return Tasks;
    }
}

```

Following table explains how the work, duration and resource unit fields will gets updated on changing any of the fields

Task Type | Changes in Duration | Changes in work | Changes in Resource Units

Fixed Duration | Will updates work value | Will updates Resource unit | Will updates work value

Fixed Work | Will updates Resource unit. Note: For manually scheduled task work will update. | Will updates Duration value. Note: For manually scheduled task resource unit updates. | Will updates Duration value. Note: For manually scheduled task work field updates.

Fixed Unit | Will updates work value | Will updates Duration value. Note: For manually scheduled task resource unit updates. | Will updates Duration value. Note: For manually scheduled task work field updates.

---

Fixed Unit is the default TaskType in Gantt. The above calculations are not applicable for Milestones.

---

You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to knows how to render and configure the gantt.

### Tooltip in Blazor Gantt Chart Component

The [Blazor Gantt Chart](#) component has a support to display a tooltip for various UI elements like taskbar, timeline cells, and grid cells

#### Enable Tooltip

In the Gantt Chart component, you can enable or disable the mouse hover tooltip for the following UI elements using the `GanttTooltipSettings.ShowTooltip` property:

- Taskbar
- Connector line
- Baseline
- Event marker

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
```

```

<SfGantt DataSource="@TaskCollection" RenderBaseline="true"
BaselineColor="Red" Height="450px" Width="800px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress"
  Dependency="Predecessor"
  BaselineStartDate="BaselineStartDate" BaselineEndDate="BaselineEndDate"
  Child="SubTasks">
  </GanttTaskFields>
  <GanttEventMarkers>
  <GanttEventMarker Day="@Event" Label="Project approval and kick-off"
  CssClass="e-custom-event-marker"></GanttEventMarker>
  </GanttEventMarkers>
</SfGantt>
@code{
public DateTime Event = new DateTime(2019, 04, 10);
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
public DateTime? BaselineStartDate { get; set; }
public DateTime? BaselineEndDate { get; set; }
public string Predecessor { get; set; }
}
public static List<TaskData> GetTaskCollection() {
List<TaskData> Tasks = new List<TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
BaselineStartDate = new DateTime(2019, 04, 02),
BaselineEndDate = new DateTime(2019, 04, 08),
Progress = 70
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Predecessor = "2FS",
Progress = 50
}
}
},
}
}

```



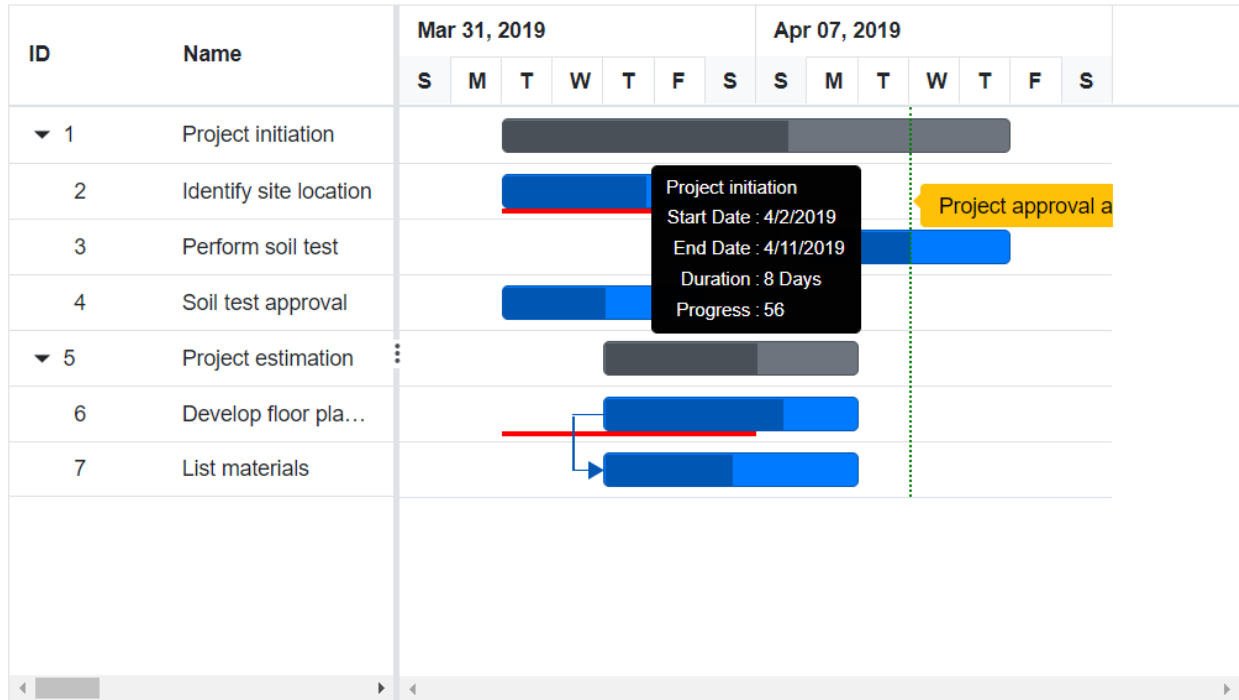
```

    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "4",
        Progress = 50
    },
    })
    },
    new TaskData() {
        TaskId = 5,
        TaskName = "Project estimation",
        StartDate = new DateTime(2019, 04, 02),
        EndDate = new DateTime(2019, 04, 21),
        SubTasks = (new List <TaskData> () {
            new TaskData() {
                TaskId = 6,
                TaskName = "Develop floor plan for estimation",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "3",
                Progress = 70,
                BaselineStartDate = new DateTime(2019, 04, 02),
                BaselineEndDate = new DateTime(2019, 04, 06),
            },
            new TaskData() {
                TaskId = 7,
                TaskName = "List materials",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "3",
                Predecessor = "6SS",
                Progress = 50
            }
        })
    },
    });
    return Tasks;
}
}

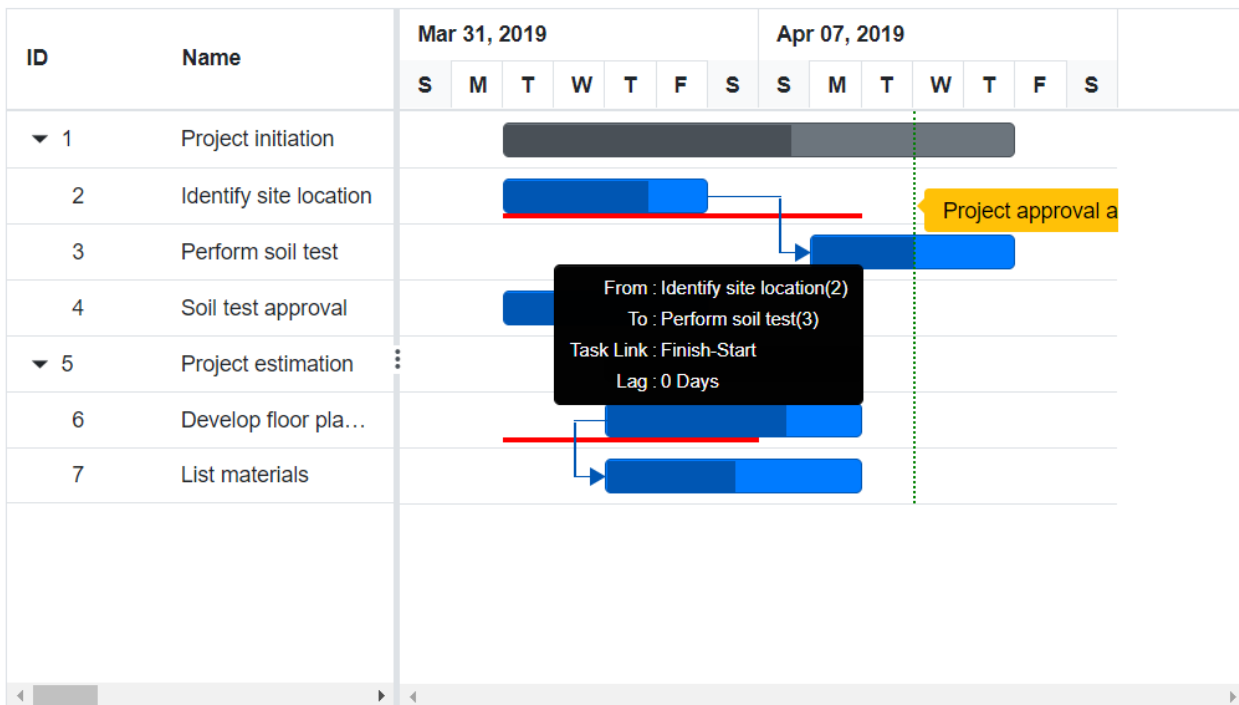
<style>
.e-gantt .e-gantt-chart .e-custom-event-marker {
width: 1px;
border-left: 2px green dotted;
}
</style>

```

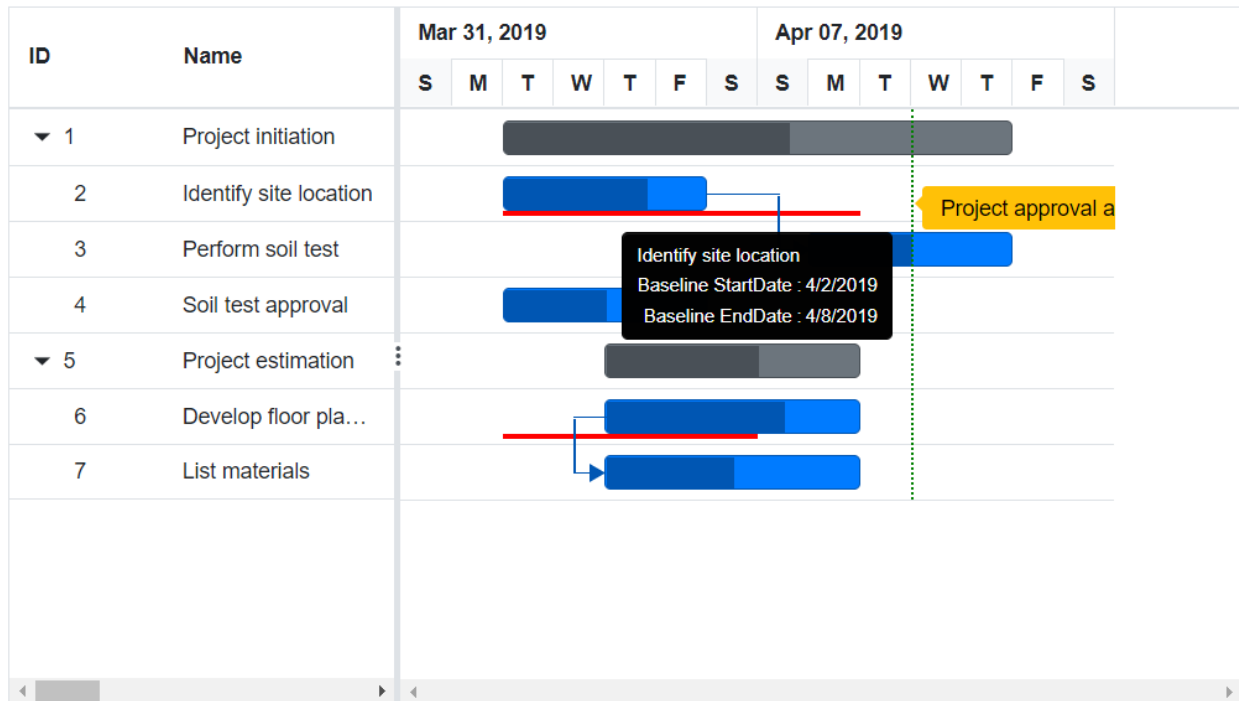
### Taskbar Tooltip



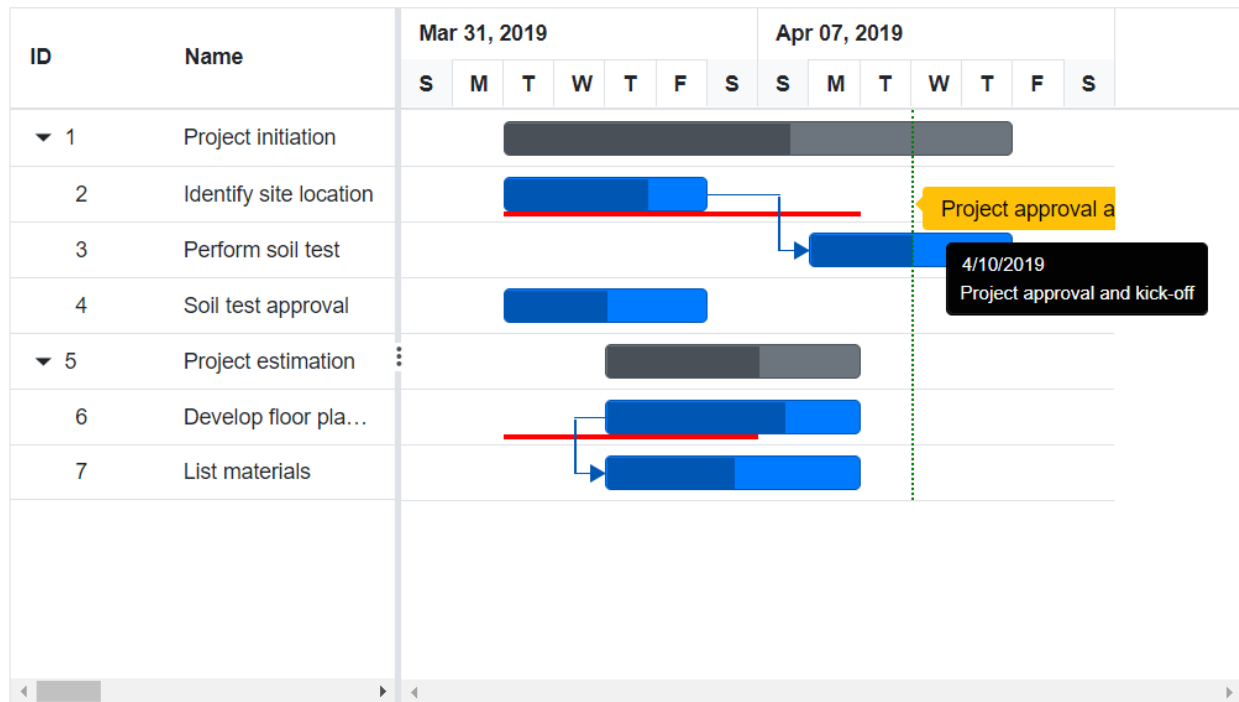
## Dependency Tooltip



## Baseline Tooltip



## Event Marker Tooltip



The default value of the `GanttTooltipSettings.ShowTooltip` property is true.

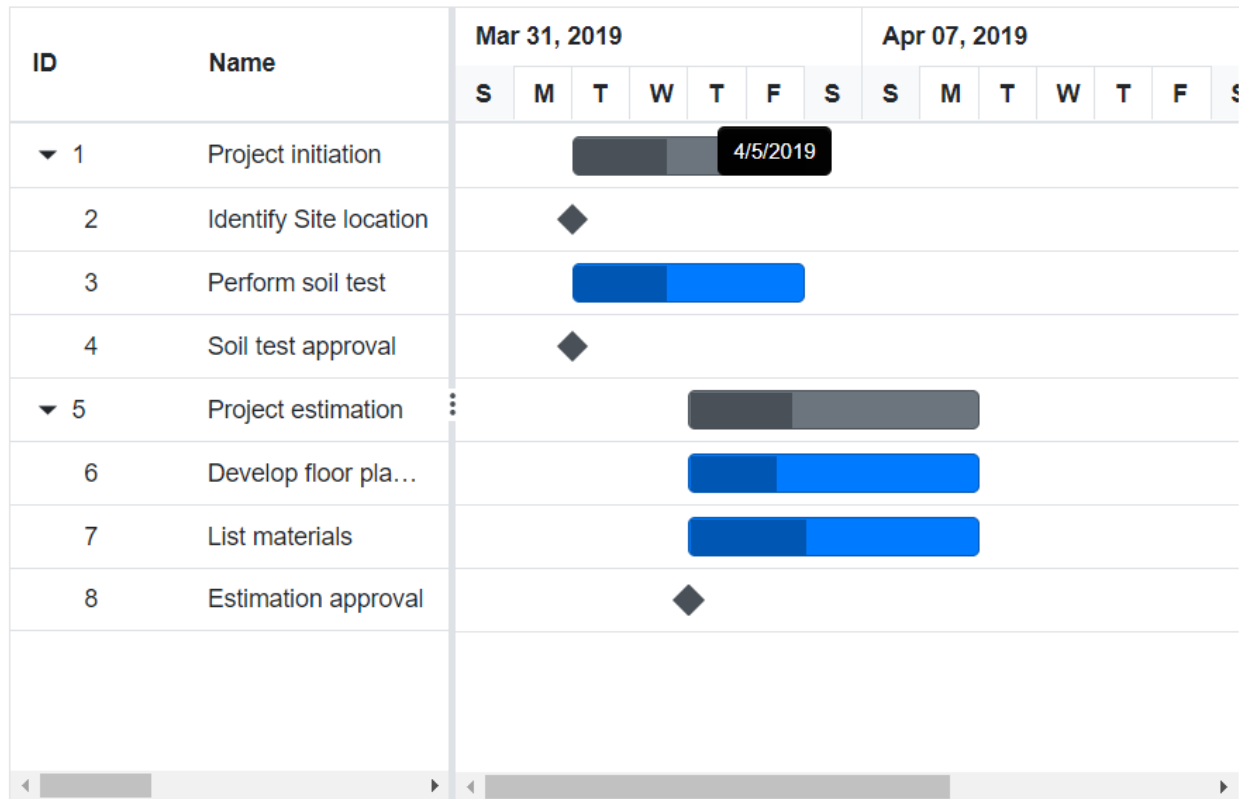
## Timeline Cells Tooltip

In the Gantt Chart component, you can enable or disable the mouse hover tooltip of timeline cells using the `GanttTimelineSettings.ShowTooltip` property. The default value of this property is true. The following code example shows how to enable the timeline cells tooltip in Gantt Chart.

**ASPX-CS**

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttTimelineSettings ShowTooltip="true"></GanttTimelineSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
}
```

```
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
};
return Tasks;
}
```



### Cell Tooltip

You can enable or disable the Grid cell tooltip using the `GanttColumn.ClipMode` property.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttColumns>
    <GanttColumn Field="TaskId"></GanttColumn>
    <GanttColumn Field="TaskName" Width="100"
    ClipMode="ClipMode.EllipsisWithTooltip"></GanttColumn>
    <GanttColumn Field="StartDate"></GanttColumn>
    <GanttColumn Field="Duration" ClipMode="ClipMode.Clip"></GanttColumn>
    <GanttColumn Field="Progress"></GanttColumn>
  </GanttColumns>
</SfGantt>
@code{
  public List<TaskData> TaskCollection { get; set; }
  protected override void OnInitialized()
  {
    this.TaskCollection = GetTaskCollection();
  }
  public class TaskData
  {
    public int TaskId { get; set; }
  }
}
```

```
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}

public static List<TaskData> GetTaskCollection() {
    List<TaskData> Tasks = new List<TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 40
                }
            })
        }
    }
}
```

```

    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "0",
        Progress = 30,
    }
})
};
return Tasks;
}
}

```

### Clip Mode

The clip mode provides options to display its overflow cell content and it can be defined by the `GanttColumn.ClipMode` property.

The following are three types of `ClipMode`:

- `Clip`: Truncates the cell content when it overflows its area.
- `Ellipsis`: Displays ellipsis when content of the cell overflows its area.
- `EllipsisWithTooltip`: Displays ellipsis when content of the cell overflows its area; it displays the tooltip content when hover over ellipsis.

### NOTE

By default, all the column's `ClipMode` property is defined as `EllipsisWithTooltip`.

### Tooltip Template

#### Taskbar Tooltip

The default tooltip in the Gantt Chart component can be customized using the `GanttTooltipSettings.TaskbarTemplate` property. You can map the template script element's ID value or template string directly to this property.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttTooltipSettings ShowTooltip="true" TValue="TaskData">
  <TaskbarTemplate>
    @ {
      <div>TaskID: @context.TaskId</div>
    }
  </TaskbarTemplate>
</GanttTooltipSettings>
</SfGantt>
@code{
    public List<TaskData> TaskCollection { get; set; }
}

```



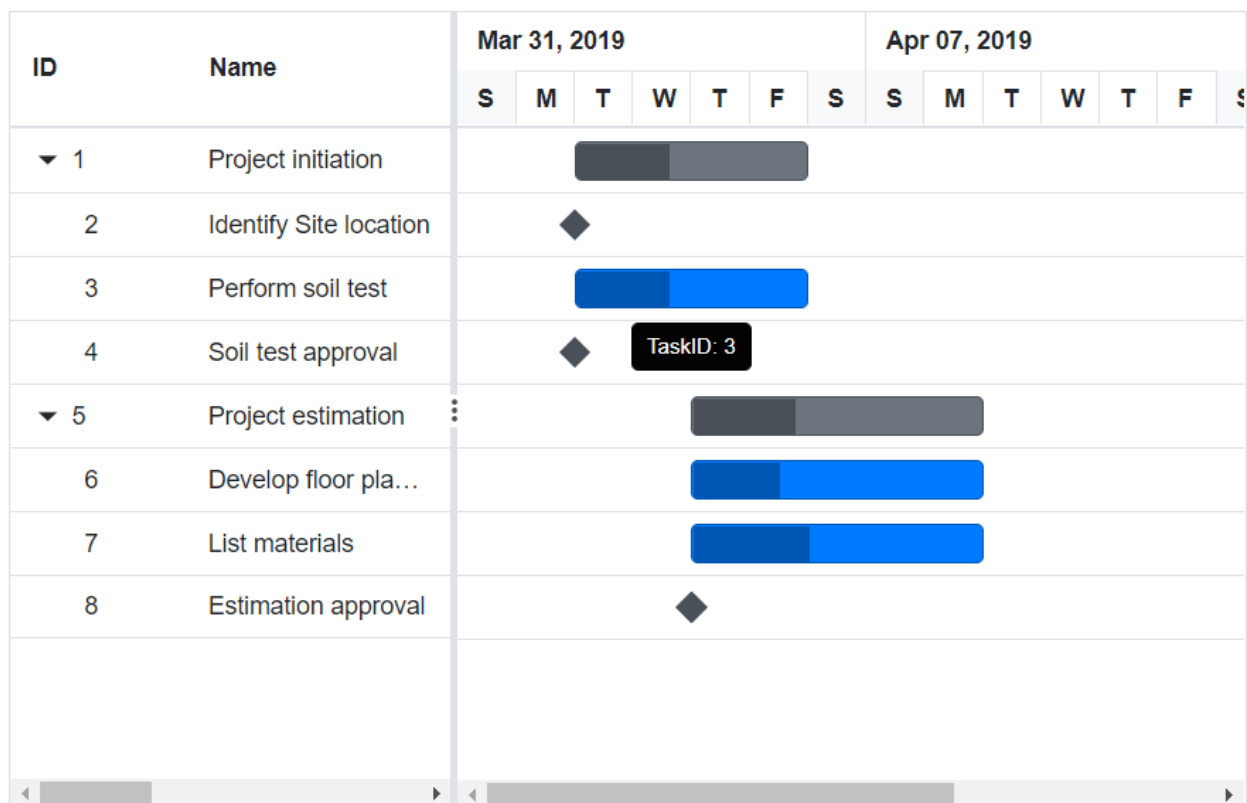
```
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
```

```

},
new TaskData() {
    TaskId = 7,
    TaskName = "List materials",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "3",
    Progress = 40
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
}
})
}
};
return Tasks;
}
}

```

The below screenshot shows the output of above code example.



<!-- Taskbar editing tooltip

The taskbar editing tooltip can be customized using the `GanttTooltipSettings.Editing` property. The following code example shows how to customize the taskbar editing tooltip in Gantt Chart.

**ASPX-CS**

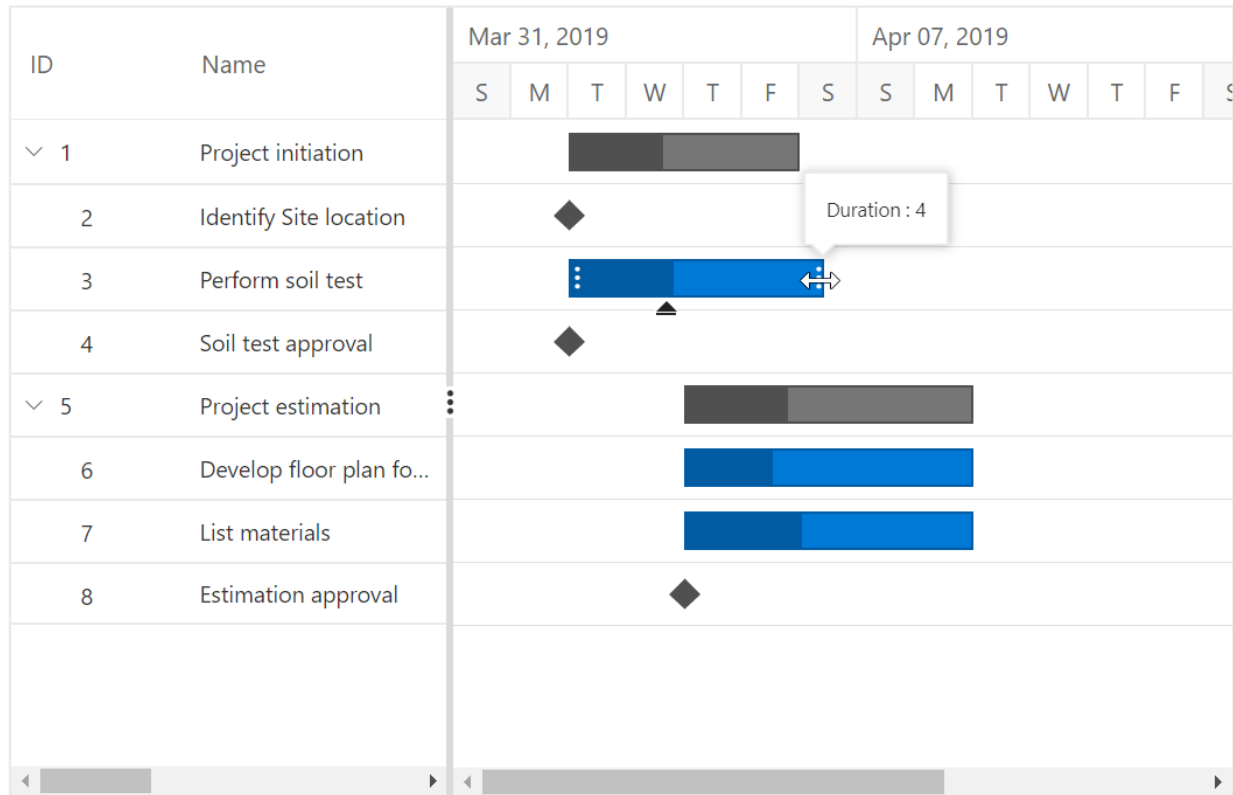
```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttTooltipSettings ShowTooltip="true" TValue="TaskData">
<EditingTemplate>
@{
<div>Duration: @context.Duration</div>
}
</EditingTemplate>
</GanttTooltipSettings>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 70
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",

```

```
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50
},
}))
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 70,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 50
        },
    })
};
return Tasks;
}
```

The below screenshot shows the output of above code example.



--&gt;

### Baseline Tooltip

A baseline tooltip can be customized using the `GanttTooltipSettings.BaselineTemplate` property. The following code example shows how to customize the baseline tooltip in Gantt Chart.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" RenderBaseline="true"
BaselineColor="Red" Height="450px" Width="800px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress"
BaselineStartDate="BaselineStartDate" BaselineEndDate="BaselineEndDate"
Child="SubTasks">
</GanttTaskFields>
<GanttTooltipSettings ShowTooltip="true" TValue="TaskData">
<BaselineTemplate>
@{
<div>Baseline StartDate:
@context.BaselineStartDate.ToShortDateString() </div>
}
</BaselineTemplate>
</GanttTooltipSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{

```

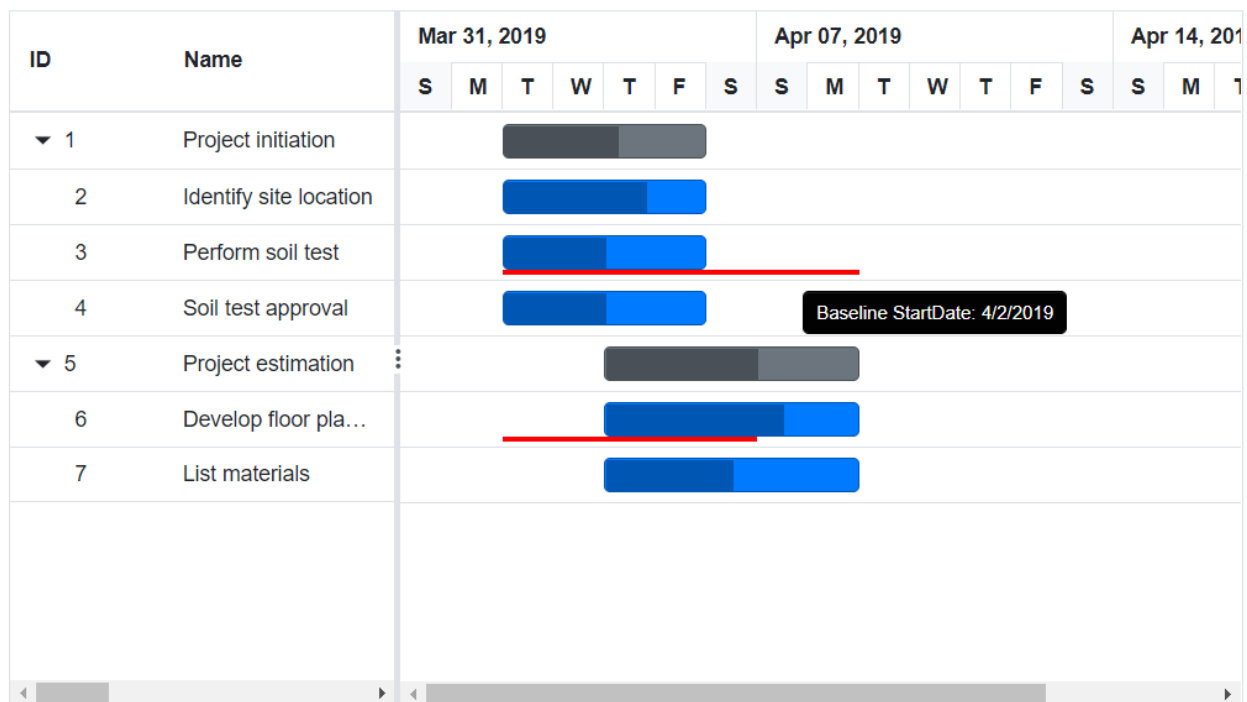
```
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public DateTime BaselineStartDate { get; set; }
    public DateTime BaselineEndDate { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
    List<TaskData> Tasks = new List<TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 70
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 50,
                    BaselineStartDate = new DateTime(2019, 04, 02),
                    BaselineEndDate = new DateTime(2019, 04, 08)
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 50
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
```

```

Duration = "3",
Progress = 70,
BaselineStartDate = new DateTime(2019, 04, 02),
BaselineEndDate = new DateTime(2019, 04, 06),
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 50
},
})
}
};
return Tasks;
}
}

```

The following screenshot shows the template for baseline in Gantt Chart.



#### Manual Taskbar Tooltip

A manual taskbar tooltip can be customized using the `GanttTooltipSettings.ManualTaskbarTemplate` property. The following code example shows how to customize the manual taskbar tooltip in Gantt Chart.

#### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px"
TaskMode="ScheduleMode.Manual" ValidateManualTasksOnLinking="true"

```

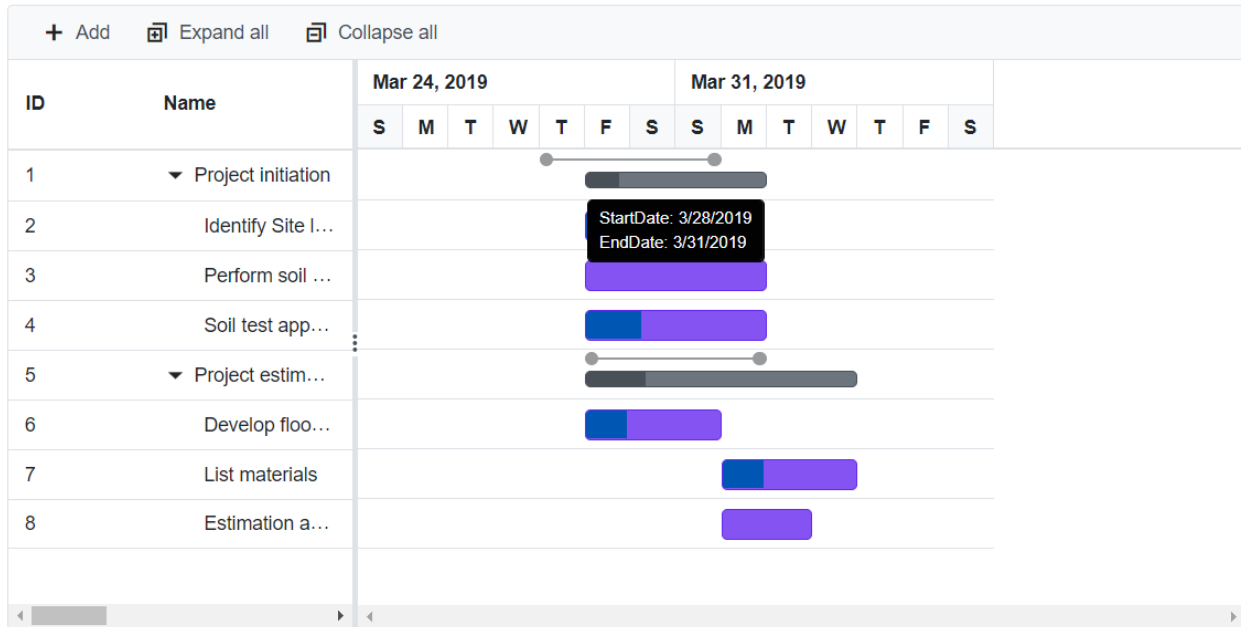
```

Width="900px" TreeColumnIndex="1" Toolbar="@ (new List<string>() { "Add",
"Edit", "Update", "Delete", "Cancel", "ExpandAll", "CollapseAll" })">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId" Dependency="Predecessor">
</GanttTaskFields>
<GanttEditSettings AllowEditing="true" AllowAdding="true"
AllowDeleting="true" AllowTaskbarEditing="true"></GanttEditSettings>
<GanttTooltipSettings ShowTooltip="true" TValue="TaskData">
<ManualTaskbarTemplate>
@{
<div> StartDate: @context.StartDate.ToShortDateString() </div>
<div> EndDate: @context.EndDate.ToShortDateString() </div>
}
</ManualTaskbarTemplate>
</GanttTooltipSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public string Predecessor { get; set; }
public int? ParentId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 03, 28),
EndDate = new DateTime(2019, 07, 28),
Duration="4"
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 03, 29),
Progress = 30,
ParentId = 1,
Duration="2"
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 03, 29),
ParentId = 1,
Duration="4"
}
}
}

```



```
    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 03, 29),
        Duration = "4",
        Progress = 30,
        ParentId = 1,
    },
    new TaskData() {
        TaskId = 5,
        TaskName = "Project estimation",
        StartDate = new DateTime(2019, 03, 29),
        EndDate = new DateTime(2019, 04, 2),
        Duration="4"
    },
    new TaskData() {
        TaskId = 6,
        TaskName = "Develop floor plan for estimation",
        StartDate = new DateTime(2019, 03, 29),
        Duration = "3",
        Progress = 30,
        ParentId = 5
    },
    new TaskData() {
        TaskId = 7,
        TaskName = "List materials",
        StartDate = new DateTime(2019, 04, 01),
        Duration = "3",
        Progress = 30,
        ParentId = 5
    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 01),
        Duration = "2",
        ParentId = 5
    }
};
return Tasks;
}
```



## Templates in Blazor Gantt Chart Component

Blazor has templated components that accept one or more UI segments as input that can be rendered as part of the component during component rendering. Gantt Chart is a templated razor component, that allows customizing various parts of the UI using template parameters. It allows rendering custom components or content based on its logic.

The available template options in Gantt Chart are as follows,

- [Column template](#) - Used to customize cell content.
- [Header template](#) - Used to customize header cell content.

## Template Context

Most of the templates used by the Gantt Chart are of type `RenderFragment<T>` and they will be passed with parameters. The parameters passed can be accessed to the templates using an implicit parameter named `context`. This implicit parameter name can also be changed using the `Context` attribute.

For example, the data of the column template can be accessed using `context` as follows.

## GanttChartTemplates 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 from directly specifying templates such as `TaskbarTemplate` and `MilestoneTemplate` as descendants of the Gantt Chart component. Hence the templates such as `TaskbarTemplate` and `MilestoneTemplate` should be wrapped around a component named `GanttTemplates` as follows.

## Taskbar Template

You can design your taskbars to view the tasks in Gantt Chart by using `GanttTemplates.TaskbarTemplate` property. It is also possible to customize the parent taskbars and

milestones with custom templates by using `GanttTemplates.ParentTaskbarTemplate` and `GanttTemplates.MilestoneTemplate` properties.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt RowHeight="75" TaskbarHeight="50"
ProjectStartDate="@ProjectStart" ProjectEndDate="@ProjectEnd"
DurationUnit="DurationUnit.Minute"
DateFormat="hh:mm tt" DataSource="@TaskCollection" Height="450px"
Width="100%">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Dependency="Predecessor"
ParentID="ParentId"></GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" HeaderText="Event Id"></GanttColumn>
<GanttColumn Field="TaskName" HeaderText="Event Name"
Width="150"></GanttColumn>
<GanttColumn Field="StartDate" HeaderText="Start Time"></GanttColumn>
<GanttColumn Field="EndDate" HeaderText="End Time"></GanttColumn>
<GanttColumn Field="Winner" HeaderText="Winner"></GanttColumn>
<GanttColumn Field="Movie" HeaderText="Movie"></GanttColumn>
<GanttColumn Field="Performance" HeaderText="Performance"
Width="200"></GanttColumn>
</GanttColumns>
<GanttLabelSettings LeftLabel="TaskName"
TValue="TaskbarTemplateData.TaskbarData">
</GanttLabelSettings>
<GanttSplitterSettings Position="30%"> </GanttSplitterSettings>
<GanttTemplates TValue="TaskbarTemplateData.TaskbarData">
<TaskbarTemplate>
@if ((context as TaskbarTemplateData.TaskbarData).TaskName == "Oscar
moments")
{
<div class="e-gantt-child-taskbar e-custom-moments"
style="height:50px;border-radius:5px;">
@if (Convert.ToInt64((context as TaskbarTemplateData.TaskbarData).Duration)
< 4)
{
<img class="moments" height="32" width="44" />
}
</div>
}
else if ((context as TaskbarTemplateData.TaskbarData).TaskName == "Oscar
performance")
{
<div class="e-gantt-child-taskbar e-custom-performance"
style="height:50px;border-radius:5px;">
@if (Convert.ToInt64((context as TaskbarTemplateData.TaskbarData).Duration)
<= 5)
{
<img class="face-mask" height="32" width="32" />
}
</div>
}
else
{

```

```

<div class="e-gantt-parent-taskbar e-custom-parent"
style="height:50px;border-radius:5px;text-overflow:ellipsis;">
@if (Convert.ToInt64((context as TaskbarTemplateData.TaskbarData).Duration)
< 4)
{
<img class="oscar" height="32" width="32" />
}
else
{
@if (!string.IsNullOrEmpty(((context as
TaskbarTemplateData.TaskbarData).Winner)) && !string.IsNullOrEmpty(((context
as TaskbarTemplateData.TaskbarData).Movie)))
{
<img class="oscar" height="32" width="32" />
<span class="e-task-label" style="position:absolute; top:13px;font-
size:14px;">@((context as TaskbarTemplateData.TaskbarData).Winner)</span>
<span class="e-task-label" style="position:absolute;top:33px;font-
size:10px;text-overflow:ellipsis;">@((context as
TaskbarTemplateData.TaskbarData).Movie)</span>
}
else if (!string.IsNullOrEmpty(((context as
TaskbarTemplateData.TaskbarData).Movie)))
{
<img class="oscar" height="32" width="32" />
<span class="e-task-label" style="position:absolute; top:18px;font-
size:12px;text-overflow:ellipsis;">@((context as
TaskbarTemplateData.TaskbarData).Movie)</span>
}
else
{
<span class="e-task-label"></span>
}
}
</div>
}
</TaskbarTemplate>
<MilestoneTemplate>
<div style="margin-top:-7px;">
<div class="e-gantt-milestone" style="position:absolute;">
<img class="moments" height="24" width="48" />
<div class="e-milestone-top" style="border-right-width:26px; margin-top: -
24px;border-left-width:26px;border-bottom-width:26px;"></div>
<div class="e-milestone-bottom" style="top:26px;border-right-width:26px;
border-left-width:26px; border-top-width:26px;"></div>
</div>
</div>
</MilestoneTemplate>
</GanttTemplates>
<GanttTimelineSettings TimelineUnitSize="75">
<GanttTopTierSettings Unit="TimelineViewMode.Hour" Format="MMM dd,
yyyy"></GanttTopTierSettings>
<GanttBottomTierSettings Unit="TimelineViewMode.Minutes" Count="15"
Format="h:mm tt"></GanttBottomTierSettings>
</GanttTimelineSettings>
<GanttDayWorkingTimeCollection>
<GanttDayWorkingTime From="0" To="24"></GanttDayWorkingTime>
</GanttDayWorkingTimeCollection>

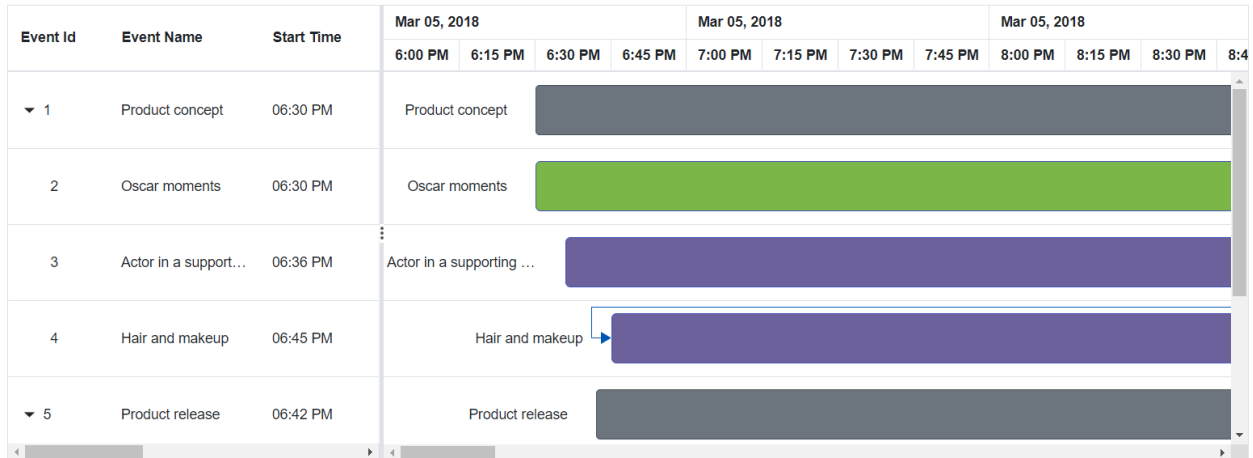
```

```

</SfGantt>
@code{
public DateTime ProjectStart = new DateTime(2018, 3, 5, 18, 0, 0);
public DateTime ProjectEnd = new DateTime(2018, 3, 6, 18, 0, 0);
public List<TaskbarTemplateData.TaskbarData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = TaskbarTemplateData.TaskTemplateData();
}
public class TaskbarTemplateData
{
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public string Predecessor { get; set; }
public int? ParentId { get; set; }
}
public class TaskProperties
{
public string TaskName { get; set; }
public double Duration { get; set; }
}
public class TaskbarData : TaskData
{
public string Performance { get; set; }
public string Winner { get; set; }
public string Movie { get; set; }
public TaskProperties GanttProperties { get; set; }
}
public static List<TaskbarData> TaskTemplateData()
{
List<TaskbarData> TaskDataCollection = new List<TaskbarData> {
new TaskbarData() { TaskId = 1, TaskName = "Product concept", StartDate =
new DateTime(2018, 03, 05, 18, 0, 0), EndDate = new DateTime(2018, 03, 05,
18, 15, 0) },
new TaskbarData() { TaskId = 2, TaskName = "Oscar moments", StartDate = new
DateTime(2018, 03, 05, 18, 30, 0), EndDate = new DateTime(2018, 03, 05, 18,
45, 0), Winner = "", Performance = "90th Academy awards kicks-off and Jimmy
kimmel hosts the show", ParentId=1 },
new TaskbarData() { TaskId = 3, TaskName = "Actor in a supporting role",
StartDate = new DateTime(2018, 03, 05, 18, 36, 0), EndDate = new
DateTime(2018, 03, 05, 18, 42, 0), Predecessor = "1", Winner = "Sam
Rockwell", Movie = "Three Billboards Outside Ebbing, Missouri.", ParentId=1
},
new TaskbarData() { TaskId = 4, TaskName = "Hair and makeup", StartDate =
new DateTime(2018, 03, 05, 18, 33, 0), EndDate = new DateTime(2018, 03, 05,
18, 40, 0), Predecessor = "2", Movie = "Darkest Hour", ParentId=1 },
new TaskbarData() { TaskId = 5, TaskName = "Product release", StartDate =
new DateTime(2018, 03, 05, 18, 41, 0), EndDate = new DateTime(2018, 03, 05,
18, 52, 0) },
new TaskbarData() { TaskId = 6, TaskName = "Costume design", StartDate = new
DateTime(2018, 03, 05, 18, 59, 0), EndDate = new DateTime(2018, 03, 05, 19,

```

```
10, 0), Predecessor = "3", Winner = "Mark Bridges", Movie = "Phantom
Thread", ParentId = 5 },
new TaskbarData() { TaskId = 7, TaskName = "Documentary feature", StartDate
= new DateTime(2018, 03, 05, 19, 11, 0), EndDate = new DateTime(2018, 03,
05, 19, 15, 0), Predecessor = "4", Winner = "Bryan Fogel", Movie = "Icarus",
ParentId = 5 },
new TaskbarData() { TaskId = 8, TaskName = "Best sound editing and sound
mixing", StartDate = new DateTime(2018, 03, 05, 19, 16, 0), EndDate = new
DateTime(2018, 03, 05, 19, 23, 0), Predecessor = "5", Winner = "Richard King
and Alex Gibson", Movie = "Dunkirk", ParentId = 5 }
};
return TaskDataCollection;
}
}
}
<style>
.e-custom-parent {
background-color: #6d619b;
border: 1px solid #3f51b5;
}
.e-custom-performance {
background-color: #ad7a66;
border: 1px solid #3f51b5;
}
.e-custom-moments {
background-color: #7ab748;
border: 1px solid #3f51b5;
}
.moments, .face-mask, .oscar {
position: relative;
top: 2px;
bottom: 2px;
left: 5px;
padding-right: 4px;
}
.e-milestone-top {
border-bottom-color: #7ab748 !important;
border-bottom: 1px solid #3f51b5;
}
.e-milestone-bottom {
border-top-color: #7ab748 !important;
border-top: 1px solid #3f51b5;
}
</style>
```



## Appearance Customization in Blazor Gantt Chart Component

### Taskbar customization

#### Taskbar Height

Height of child taskbars and parent taskbars can be customized by using `TaskbarHeight` property. The following code example shows how to use the property.

#### ASPX-CS

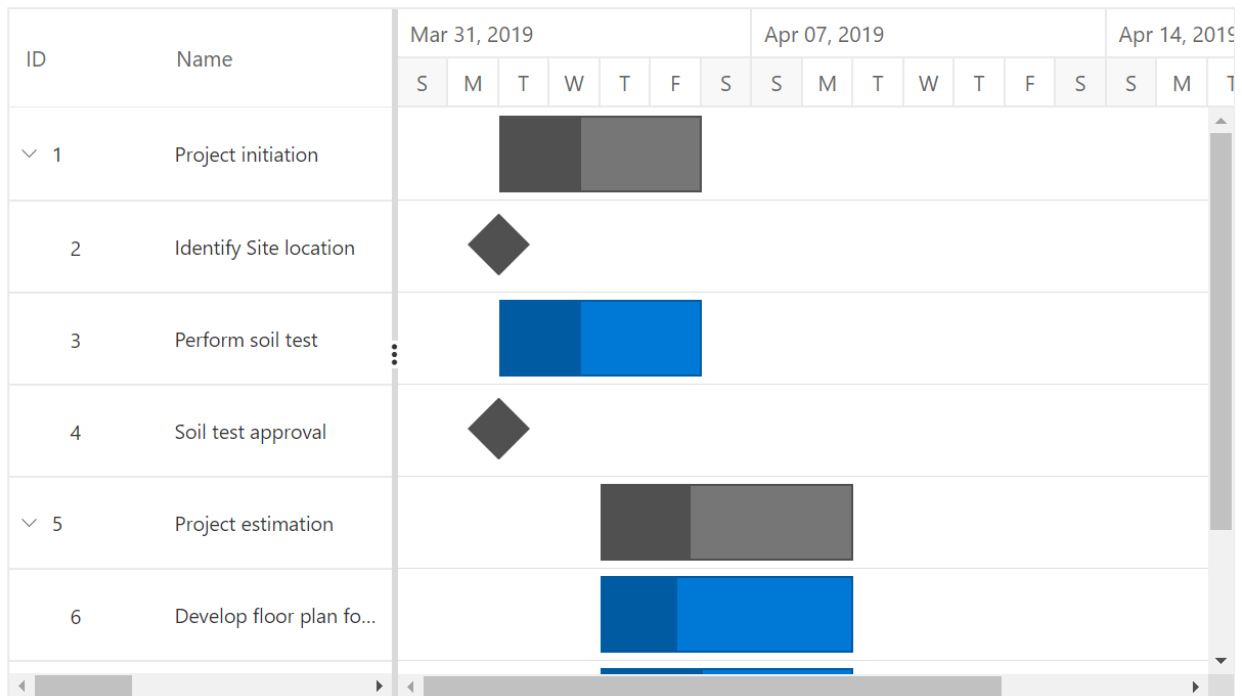
```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
RowHeight=60 TaskbarHeight=50>
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
List<TaskData> Tasks = new List<TaskData> () {
new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() { TaskId = 2, TaskName = "Identify Site location", StartDate =
new DateTime(2019, 04, 02), Duration = "0", Progress = 30 },

```

```

new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
DateTime(2019, 04, 02), Duration = "4", Progress = 40 },
new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
new DateTime(2019, 04, 02), Duration = "0", Progress = 30 },
}),
},
new TaskData() { TaskId = 5, TaskName = "Project estimation", StartDate =
new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() { TaskId = 6, TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04), Duration = "3", Progress = 30 },
new TaskData() { TaskId = 7, TaskName = "List materials", StartDate = new
DateTime(2019, 04, 04), Duration = "3", Progress = 40 },
new TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate =
new DateTime(2019, 04, 04), Duration = "0", Progress = 30 }
})
}
};
return Tasks;
}
}

```



The **TaskbarHeight** property accepts only pixel value.

#### Taskbar Background

In the Gantt Chart component, the appearance can be customized based on the Hierarchy using the **GetHierarchicalData** method with the custom styles. So, the method **GetHierarchicalData** helps the user to do the customization and access the internal properties of Gantt. The following code example



shows how to customize Gantt Chart Rows using Gantt properties as `level` and `hasChildRecords` from `GetHierarchicalData` method in `QueryChartRowInfo` event of Gantt.

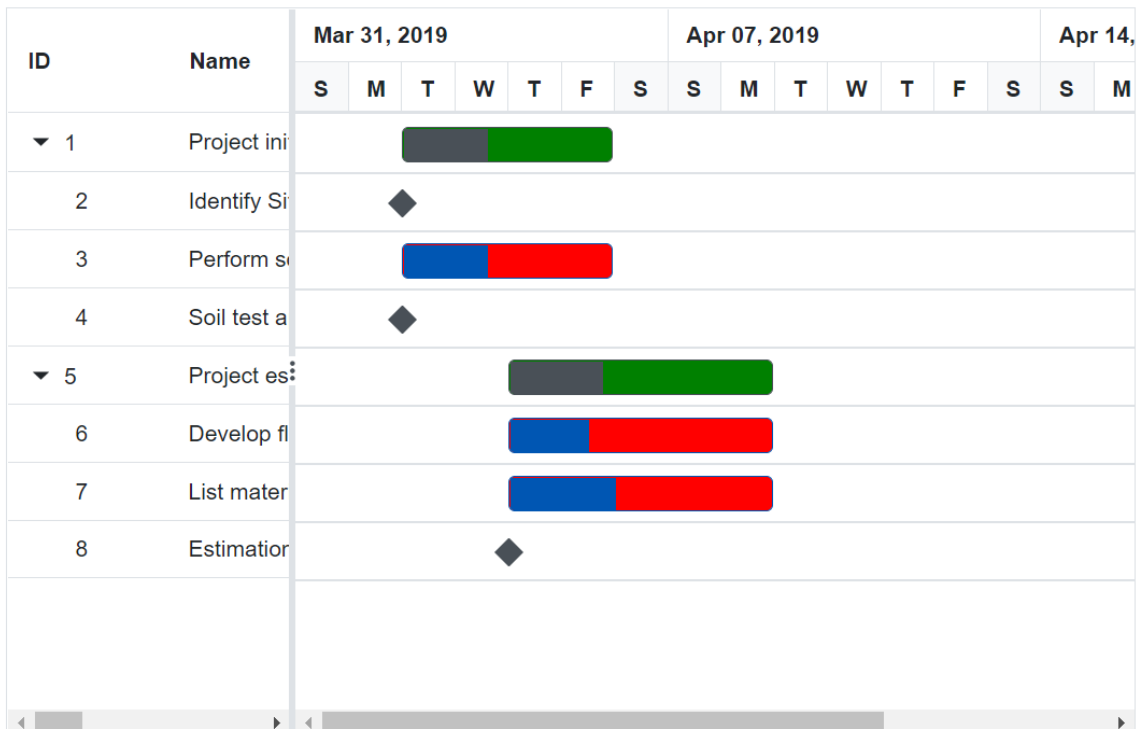
### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress"
  ParentID="ParentId">
  </GanttTaskFields>
  <GanttEvents TValue="TaskData"
  QueryChartRowInfo="GanttChartRowInfo"></GanttEvents>
</SfGantt>
@code{
  SfGantt<TaskData> Gantt;
  public List<TaskData> TaskCollection { get; set; }
  protected override void OnInitialized()
  {
    this.TaskCollection = GetTaskCollection();
  }
  private void GanttChartRowInfo(QueryChartRowInfoEventArgs<TaskData> args)
  {
    dynamic data = Gantt.GetHierarchicalData(args.Data.TaskId);
    if (data.level == 0 && data.hasChildRecords == true)
    {
      args.Row.AddClass(new string[] { "customize-parent" });
    }
    else
    {
      args.Row.AddClass(new string[] { "customize-child" });
    }
  }
  public class TaskData
  {
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public int? ParentId { get; set; }
  }
  public static List<TaskData> GetTaskCollection()
  {
    List<TaskData> Tasks = new List<TaskData>() {
      new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
      new DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 21) },
      new TaskData() { TaskId = 2, TaskName = "Identify Site location", StartDate
      = new DateTime(2019, 04, 02), Duration = "0", Progress = 30, ParentId = 1 },
      new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
      DateTime(2019, 04, 02), Duration = "4", Progress = 40, ParentId = 1 },
      new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
      new DateTime(2019, 04, 02), Duration = "0", Progress = 30, ParentId = 1 },
      new TaskData() { TaskId = 5, TaskName = "Project estimation", StartDate =
      new DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 21) },
    }
```

```

new TaskData() { TaskId = 6, TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04), Duration = "3", Progress = 30,
ParentId = 5 },
new TaskData() { TaskId = 7, TaskName = "List materials", StartDate = new
DateTime(2019, 04, 04), Duration = "3", Progress = 40, ParentId = 5 },
new TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate =
new DateTime(2019, 04, 04), Duration = "0", Progress = 30, ParentId = 5 }
};
return Tasks;
}
}
<style>
.customize-parent .e-gantt-parent-taskbar {
background-color: green !important;
}
.customize-child .e-gantt-child-taskbar {
background-color: red !important;
}
</style>

```



### Task labels

The Gantt Chart component maps any data source fields to task labels using the `GanttLabelSettings.LeftLabel`, `GanttLabelSettings.RightLabel`, and `GanttLabelSettings.TaskLabel` properties. You can customize the task labels with templates using `GanttLabelSettings.LeftLabelTemplate`, `GanttLabelSettings.RightLabelTemplate` and `GanttLabelSettings.TaskLabelTemplate`.

### ASPX-CS

```

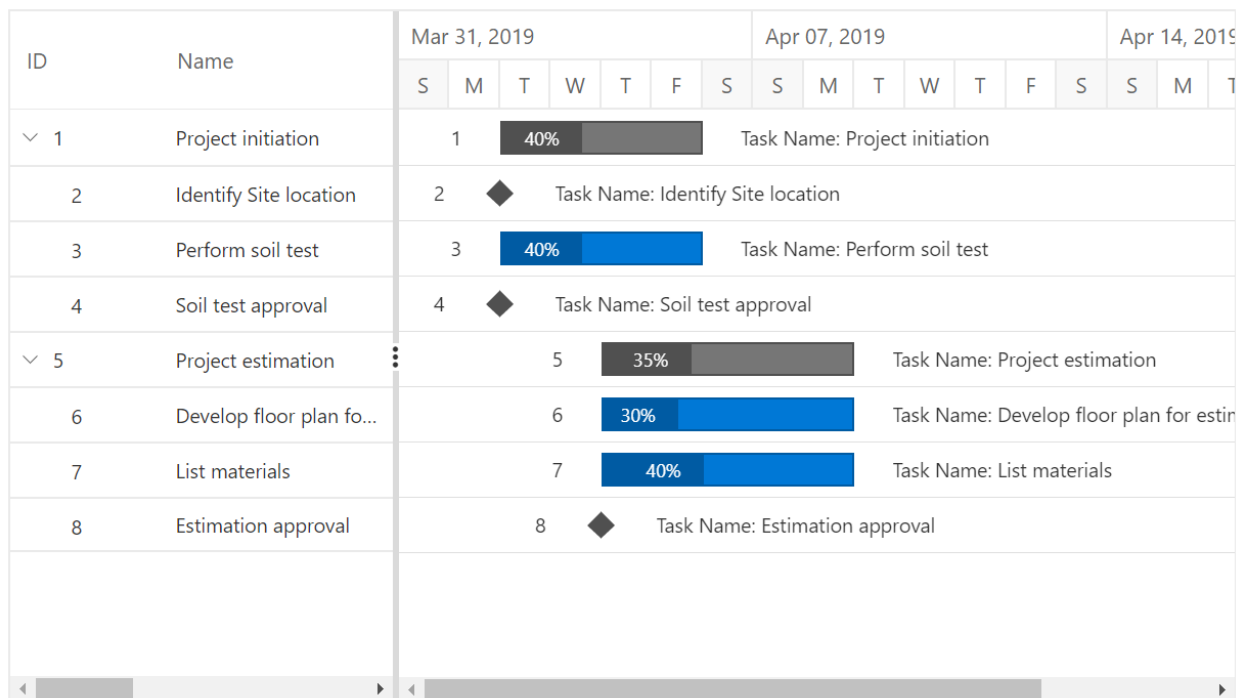
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttLabelSettings LeftLabel="TaskId" TValue="TaskData">
<RightLabelTemplate>
<div class="e-right-label-inner-div" style="height:22px;margin-top:7px;">
<span class="e-label">Task Name: @((context as TaskData).TaskName)</span>
</div>
</RightLabelTemplate>
<TaskLabelTemplate>
<div class="e-task-label-inner-div" style="line-height:21px; text-align:display:width:px; height:22px;">
<span class="e-label">@((context as TaskData).Progress)%</span>
</div>
</TaskLabelTemplate>
</GanttLabelSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() { TaskId = 2, TaskName = "Identify Site location", StartDate
= new DateTime(2019, 04, 02), Duration = "3", Progress = 30 },
new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
DateTime(2019, 04, 02), Duration = "4", Progress = 40 },
new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
new DateTime(2019, 04, 02), Duration = "0", Progress = 30 },
})
},
new TaskData() { TaskId = 5, TaskName = "Project estimation", StartDate =
new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() { TaskId = 6, TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04), Duration = "3", Progress = 30 },

```

```

new TaskData() { TaskId = 7, TaskName = "List materials", StartDate = new
DateTime(2019, 04, 04), Duration = "3", Progress = 40 },
new TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate =
new DateTime(2019, 04, 04), Duration = "0", Progress = 30 }
})
}
};
return Tasks;
}
}
<style>
.e-label {
color: white !important;
}
</style>

```



### Connector lines

The width and background color of connector lines in Gantt Chart can be customized using the `ConnectorLineWidth` and `ConnectorLineBackground` properties. The following code example shows how to use these properties.

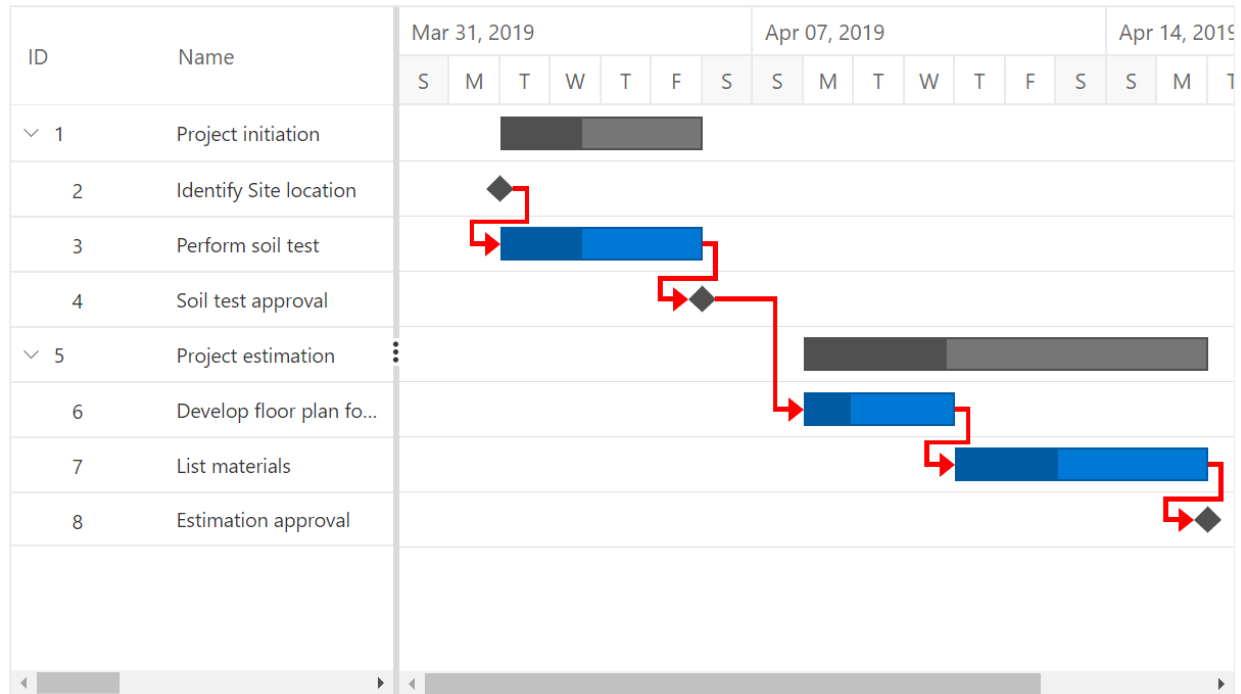
### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="800px"
ConnectorLineWidth="3" ConnectorLineBackground="red">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks"
Dependency="Predecessor">
</GanttTaskFields>
</SfGantt>

```

```
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public string Predecessor { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() { TaskId = 2, TaskName = "Identify Site location", StartDate
= new DateTime(2019, 04, 02), Duration = "0", Progress = 30 },
new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
DateTime(2019, 04, 02), Duration = "4", Progress = 40, Predecessor = "2" },
new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
new DateTime(2019, 04, 02), Duration = "0", Progress = 30, Predecessor = "3"
},
})
},
new TaskData() { TaskId = 5, TaskName = "Project estimation", StartDate =
new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() { TaskId = 6, TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04), Duration = "3", Progress = 30,
Predecessor = "4" },
new TaskData() { TaskId = 7, TaskName = "List materials", StartDate = new
DateTime(2019, 04, 04),Duration = "3", Progress = 40, Predecessor = "6" },
new TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate =
new DateTime(2019, 04, 04), Duration = "0", Progress = 30, Predecessor = "7"
}
})
}
};
return Tasks;
}
}
```



### Customize rows and cells

While rendering the Tree Grid part in Gantt Chart, the `RowDataBound` and `QueryCellInfo` events trigger for every row and cell. Using these events, you can customize the rows and cells. The following code example shows how to customize the cell and row elements using these events.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttColumns>
    <GanttColumn Field="TaskId" Width="150"></GanttColumn>
    <GanttColumn Field="TaskName" HeaderText="Task Name"
    Width="250"></GanttColumn>
    <GanttColumn Field="Progress" Width="150"></GanttColumn>
    <GanttColumn Field="StartDate" HeaderText="Start Date"
    Width="150"></GanttColumn>
    <GanttColumn Field="Duration" HeaderText="Duration"
    Width="150"></GanttColumn>
  </GanttColumns>
  <GanttSplitterSettings ColumnIndex=3></GanttSplitterSettings>
  <GanttEvents QueryCellInfo="QueryCellInfo" RowDataBound="RowDataBound"
  TValue="TaskData"></GanttEvents>
</SfGantt>
<style>
  .custom-row {
    background-color: #90EE90;
  }
  .yellow-cell {
```

```

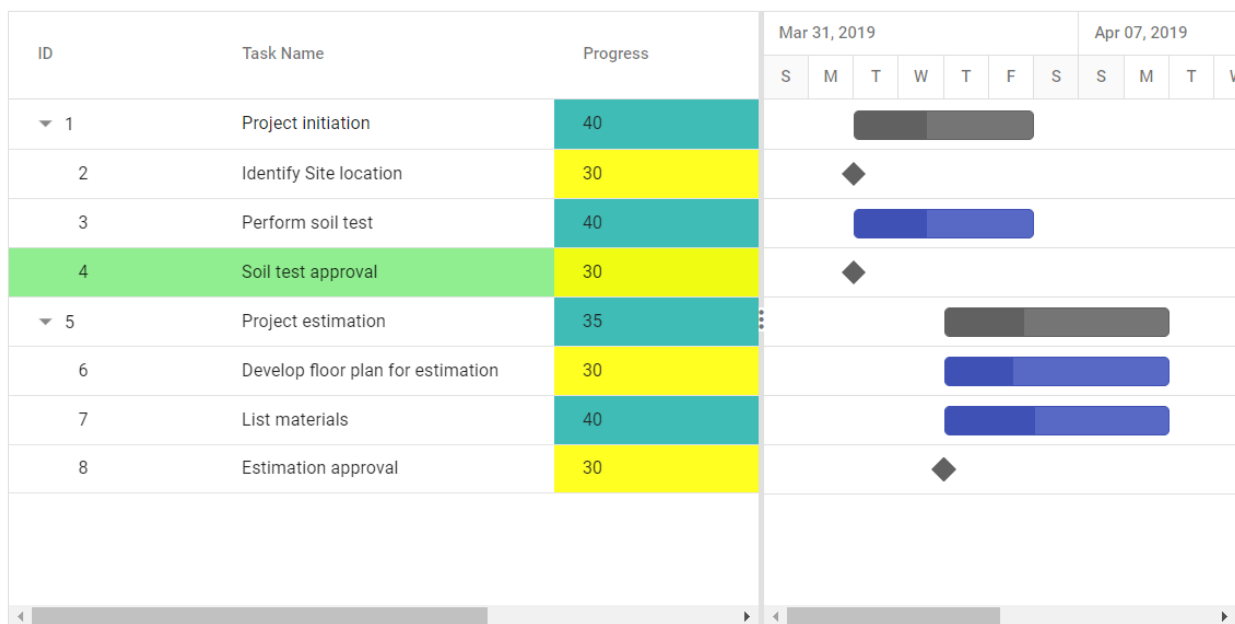
background-color: #FFFF00;
}
.red-cell {
background-color: #20B2AA;
}
</style>
@code{
public void QueryCellInfo(QueryCellInfoEventArgs<TaskData> args)
{
if (args.Column.Field == "Progress")
{
if (args.Data.Progress == 30)
{
args.Cell.AddClass(new string[] { "yellow-cell" });
}
else
{
args.Cell.AddClass(new string[] { "red-cell" });
}
}
}
public void RowDataBound(RowDataBoundEventArgs<TaskData> args)
{
if(args.Data.TaskId == 4) {
args.Row.AddClass(new string[] { "custom-row" });
}
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection() {
List<TaskData> Tasks = new List<TaskData> () {
new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() { TaskId = 2, TaskName = "Identify Site location", StartDate
= new DateTime(2019, 04, 02), Duration = "0", Progress = 30 },
new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
DateTime(2019, 04, 02), Duration = "4", Progress = 40 },
new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
new DateTime(2019, 04, 02), Duration = "0", Progress = 30 },
})
}
},

```

```

new TaskData() { TaskId = 5, TaskName = "Project estimation", StartDate =
new DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() { TaskId = 6, TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04), Duration = "3", Progress = 30 },
new TaskData() { TaskId = 7, TaskName = "List materials", StartDate = new
DateTime(2019, 04, 04), Duration = "3", Progress = 40 },
new TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate =
new DateTime(2019, 04, 04), Duration = "0", Progress = 30 }
})
}
};
return Tasks;
}
}

```



### Grid lines

In the Gantt Chart component, you can show or hide the grid lines in the Tree Grid side and chart side by using the `GridLines` property.

The following options are available in the Gantt Chart component for rendering the grid lines:

- Horizontal: The horizontal grid lines alone will be visible.
- Vertical: The vertical grid lines alone will be visible.
- Both: Both the horizontal and vertical grid lines will be visible on the Tree Grid and chart sides.
- None: Gridlines will not be visible on Tree Grid and chart sides.

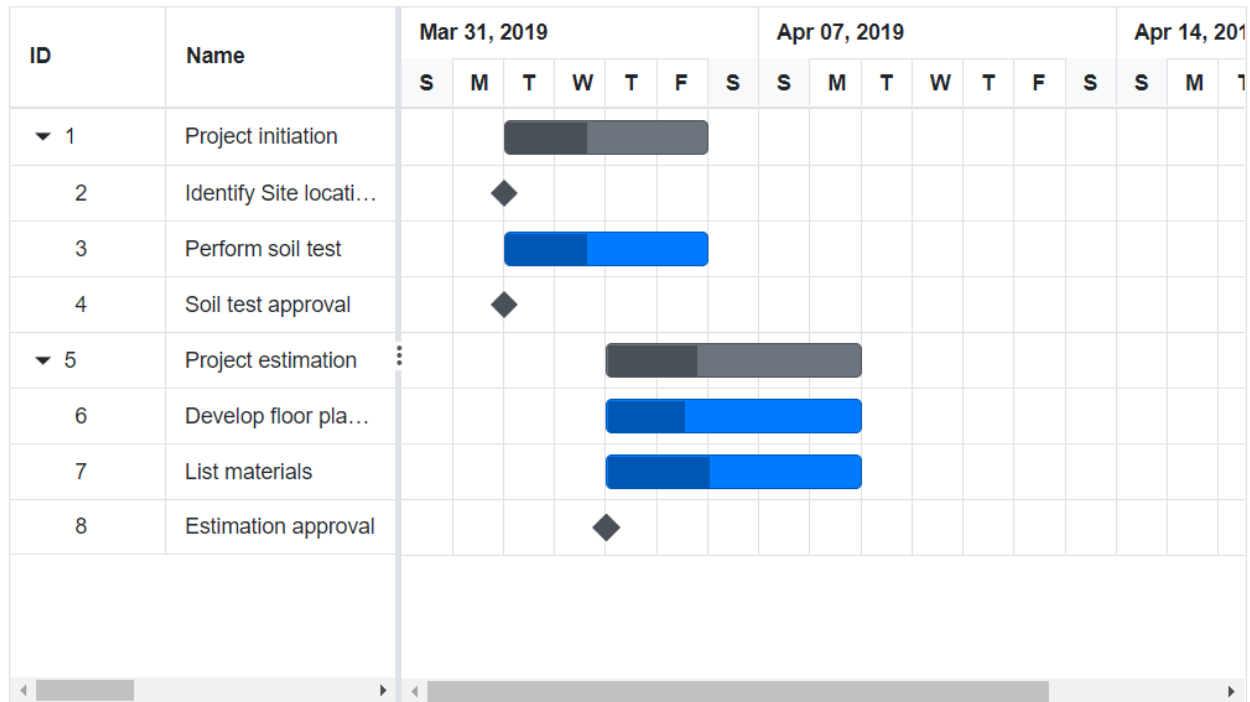
By default, the `GridLines` property is set to `Horizontal` type.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
```



```
<SfGantt DataSource="@TaskCollection" Height="450px" Width="800px"
GridLines="Syncfusion.Blazor.Gantt.GridLine.Both">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
new DateTime(2019, 04, 02),EndDate = new DateTime(2019, 04, 21), SubTasks =
(new List <TaskData> () {
new TaskData() { TaskId = 2, TaskName = "Identify Site location", StartDate
= new DateTime(2019, 04, 02),Duration = "0", Progress = 30 },
new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
DateTime(2019, 04, 02),Duration = "4", Progress = 40 },
new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
new DateTime(2019, 04, 02),Duration = "0", Progress = 30 }}}
},
new TaskData() {TaskId = 5, TaskName = "Project estimation", StartDate = new
DateTime(2019, 04, 02),EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () { new TaskData() { TaskId = 6, TaskName =
"Develop floor plan for estimation", StartDate = new DateTime(2019, 04, 04),
Duration = "3", Progress = 30 },
new TaskData() { TaskId = 7, TaskName = "List materials", StartDate = new
DateTime(2019, 04, 04),Duration = "3", Progress = 40 },
new TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate =
new DateTime(2019, 04, 04),Duration = "0", Progress = 30 }}}
}
};
return Tasks;
}
}
```



### Splitter

Gantt Chart component consists of both Tree Grid part and Chart part. Splitter is used to resize the Tree Grid section from the Chart section. You can change the position of the Splitter when loading the Gantt Chart component using the `SplitterSettings` property. The following list defines possible values for this property:

#### | Splitter Properties | Description |

| --- | --- |

| `GanttSplitterSettings.Position` | This property denotes the percentage of the Tree Grid section's width to be rendered and this property supports both pixels and percentage values |

| `GanttSplitterSettings.ColumnIndex` | This property defines the splitter position as column index value |

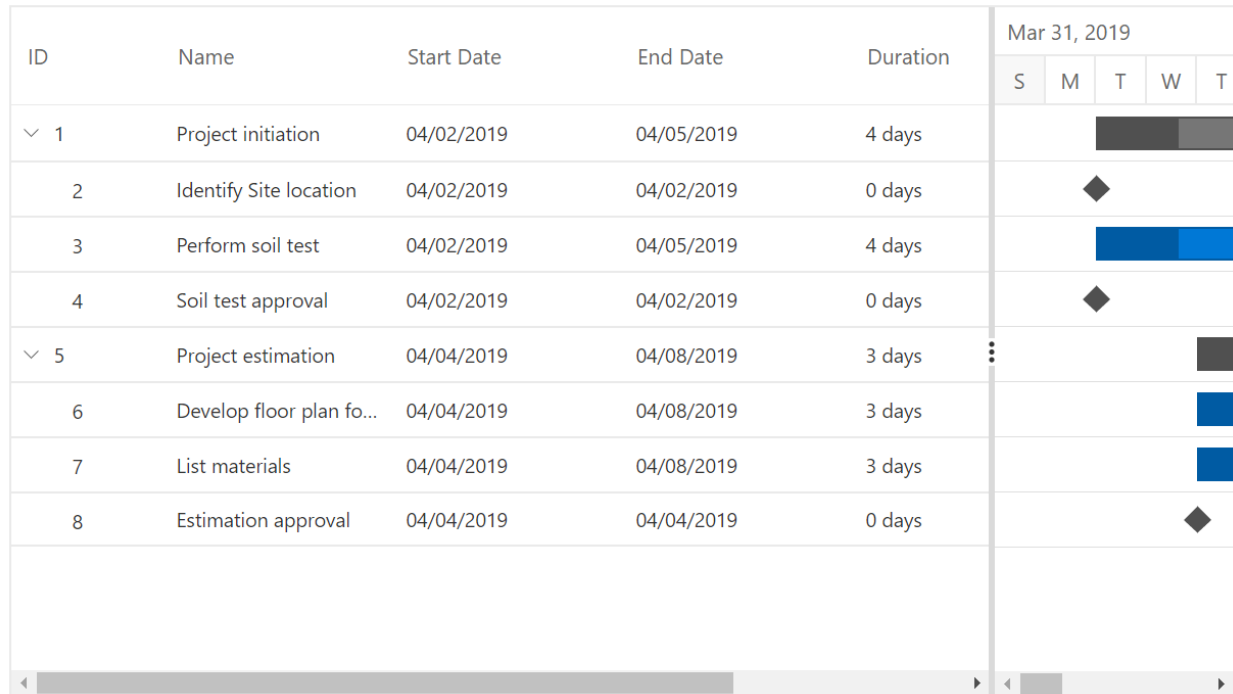
| `GanttSplitterSettings.View` | *Default: Shows Grid side and Gantt Chart side.*   
 `Grid`: Shows Grid side alone in Gantt Chart.   
 `Chart`: Shows chart side alone in Gantt Chart. |

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="800px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttSplitterSettings Position="80%"></GanttSplitterSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
```

```
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}

public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
            new DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () { new TaskData() { TaskId = 2, TaskName =
                "Identify Site location", StartDate = new DateTime(2019, 04, 02), Duration =
                "0", Progress = 30 },
                new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
                    DateTime(2019, 04, 02), Duration = "4", Progress = 40 },
                new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
                    new DateTime(2019, 04, 02), Duration = "0", Progress = 30 }}}),
        new TaskData() { TaskId = 5, TaskName = "Project estimation", StartDate =
            new DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () { new TaskData() { TaskId = 6, TaskName =
                "Develop floor plan for estimation", StartDate = new DateTime(2019, 04, 04),
                Duration = "3", Progress = 30 },
                new TaskData() { TaskId = 7, TaskName = "List materials", StartDate = new
                    DateTime(2019, 04, 04), Duration = "3", Progress = 40 },
                new TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate =
                    new DateTime(2019, 04, 04), Duration = "0", Progress = 30 }}}),
    };
    return Tasks;
}
```



### Change splitter position dynamically

In Gantt Chart, we can change the splitter position dynamically by using `SetSplitterPositionAsync` method. We can change the splitter position by passing value and type parameter to `SetSplitterPositionAsync` method. Type parameter will accept one of the following values 'Position', 'ColumnIndex', 'ViewType'.

The following code example shows how to use this method.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.DropDowns
<button @onclick="UpdateSplitterByPosition">Update splitter by
position</button>
<button @onclick="UpdateSplitterByIndex">Update splitter by index</button>
<SfDropDownList TValue="string" TItem="SplitterView"
DataSource="@SplitterViews" Width="200px">
<DropDownListFieldSettings Value="ID"
Text="Text"></DropDownListFieldSettings>
<DropDownListEvents TValue="string" TItem="SplitterView"
ValueChange="OnChange"></DropDownListEvents>
</SfDropDownList>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="800px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public class SplitterView
{
```

```

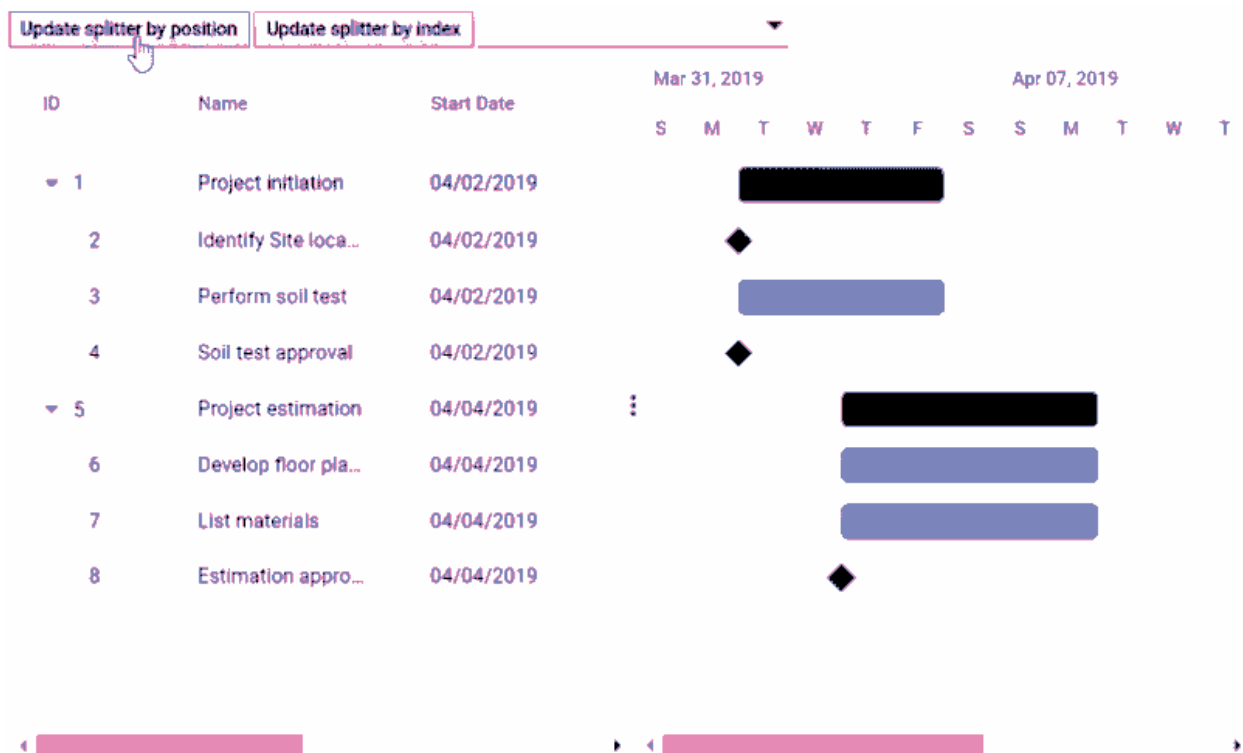
public string ID { get; set; }
public string Text { get; set; }
}
public List<SplitterView>SplitterViews = new List<SplitterView>
{
    new SplitterView() { ID= "Default", Text= "Default" },
    new SplitterView() { ID= "Grid", Text= "Grid" },
    new SplitterView() { ID= "Chart", Text= "Chart" },
};
public void OnChange(Syncfusion.Blazor.DropDowns.ChangeEventArgs<string,
SplitterView> args)
{
    if(args.Value == "Grid") {
        this.Gantt.SetSplitterPositionAsync(Syncfusion.Blazor.Gantt.SplitterView.Grid);
    } else if (args.Value == "Chart") {
        this.Gantt.SetSplitterPositionAsync(Syncfusion.Blazor.Gantt.SplitterView.Chart);
    } else {
        this.Gantt.SetSplitterPositionAsync(Syncfusion.Blazor.Gantt.SplitterView.Default);
    }
}
public void UpdateSplitterByPosition()
{
    this.Gantt.SetSplitterPositionAsync("70%");
}
public void UpdateSplitterByIndex()
{
    this.Gantt.SetSplitterPositionAsync(0);
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
            new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() { TaskId = 2, TaskName = "Identify Site location", StartDate =
                    new DateTime(2019, 04, 02), Duration = "0", Progress = 30 },
                new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
                    DateTime(2019, 04, 02), Duration = "4", Progress = 40 },
            })
    }
}

```

```

new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
new DateTime(2019, 04, 02), Duration = "0", Progress = 30 }})
},
new TaskData() { TaskId = 5, TaskName = "Project estimation", StartDate =
new DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () { new TaskData() { TaskId = 6, TaskName =
"Develop floor plan for estimation", StartDate = new DateTime(2019, 04, 04),
Duration = "3", Progress = 30 },
new TaskData() { TaskId = 7, TaskName = "List materials", StartDate = new
DateTime(2019, 04, 04), Duration = "3", Progress = 40 },
new TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate =
new DateTime(2019, 04, 04), Duration = "0", Progress = 30 }}})
};
return Tasks;
}
}

```



You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

### Duration Unit in Blazor Gantt Chart Component

In Gantt Chart, the tasks' duration value can be measured by the following duration units,

- Day
- Hour
- Minute

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

- Using `GanttTaskFields.DurationUnit` property, to map the duration unit data source field.
- Defining the duration unit value along with the duration field in the data source.

### Mapping the Duration Unit Field

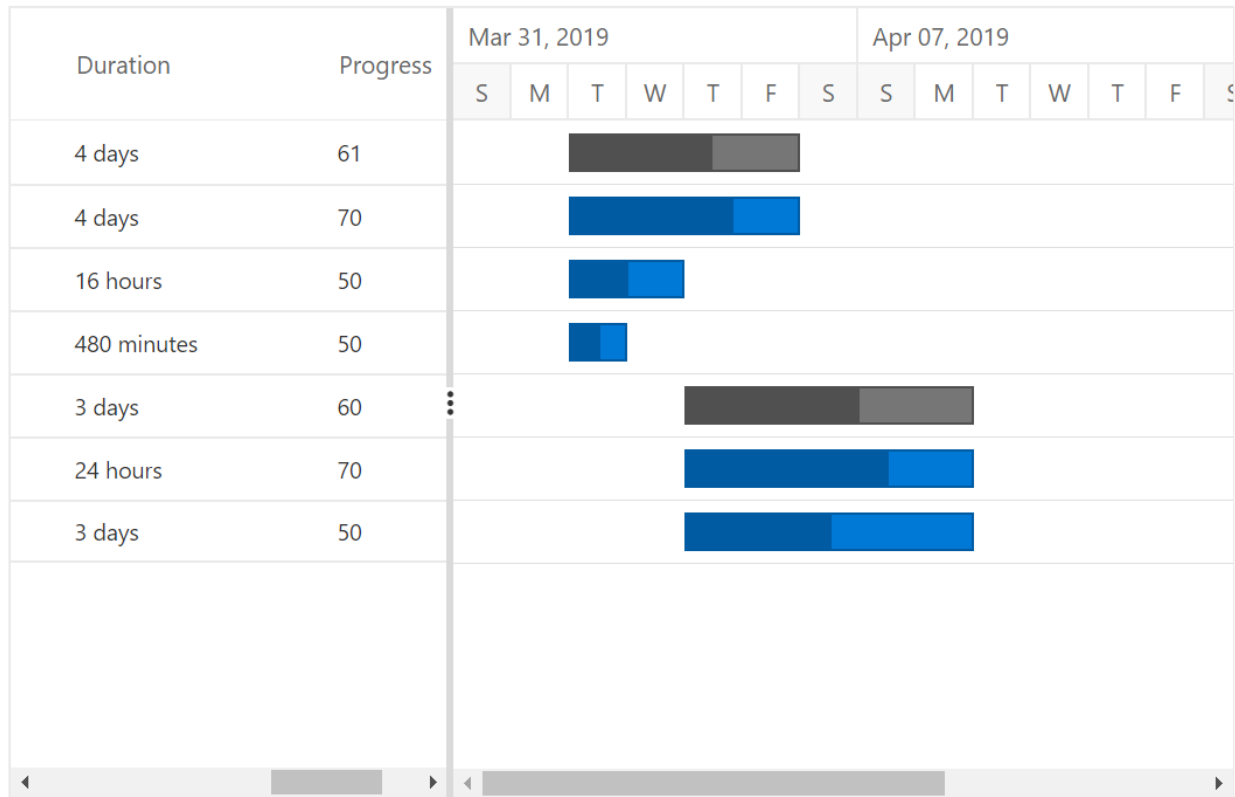
The below code snippet explains the mapping of duration unit data source field to the Gantt Chart component using the `GanttTaskFields.DurationUnit` property.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" DurationUnit="DurationUnit" Progress="Progress"
Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public string DurationUnit { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 70,
DurationUnit = "day"
},
},
},
}
```

```
new TaskData() {
    TaskId = 3,
    TaskName = "Perform soil test",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "16",
    Progress = 50,
    DurationUnit = "hour"
},
new TaskData() {
    TaskId = 4,
    TaskName = "Soil test approval",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "480",
    Progress = 50,
    DurationUnit = "minute"
},
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "24",
            Progress = 70,
            DurationUnit = "hour"
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 50,
            DurationUnit = "day"
        },
    })
};
return Tasks;
}
```





The default value of the `DurationUnit` property is `day`.

#### Defining Duration Unit along With Duration Field

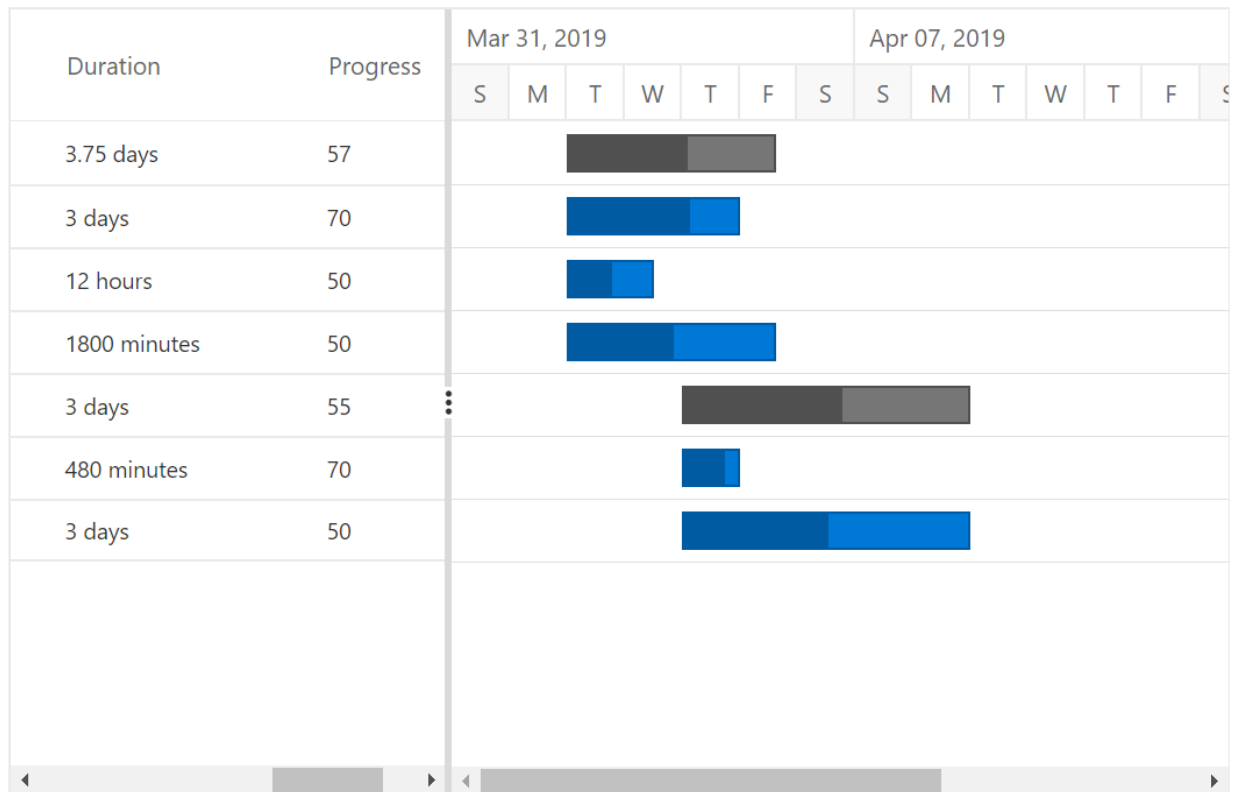
Duration units for the tasks can also be defined along with the duration values, the below code snippet explains the duration unit for a task along with duration value,

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
}
```

```
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "3days",
Progress = 70,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "12hours",
Progress = 50
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "1800minutes",
Progress = 50
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "480minutes",
Progress = 70
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3days",
Progress = 50
},
})
}
};
return Tasks;
```

```
}
}
```



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

You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

### Scheduling Tasks in Blazor Gantt Chart Component

By default, Gantt tasks are validated based on the child tasks with some factors like working time, holidays, weekends, and predecessors. The Gantt provides support for automatic and manual task scheduling modes. It is used to indicate whether the start date and end date of all the tasks will be automatically validated or not. `TaskMode` is the property used to change the schedule mode of a task.

The Gantt Chart component supports three types of modes. They are:

- **Auto**: All the tasks will be automatically validated.
- **Manual**: All the tasks will be manually validated by the user.
- **Custom**: Tasks will be validated as Auto or Manual based on the value mapped in the data source.

The default value of `TaskMode` is `Auto`.

### Automatically Scheduled Tasks

When the `TaskMode` property is set as `Auto`, the start date and end date of all the tasks in the project will be automatically validated. That is, dates will be validated based on various factors such as working time, holidays, weekends, and predecessors.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px"
TaskMode="ScheduleMode.Auto" Width="900px" TreeColumnIndex="1"
Toolbar="@ (new List<string>() { "Add", "Edit", "Update", "Delete", "Cancel",
"ExpandAll", "CollapseAll" }) ">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId">
</GanttTaskFields>
<GanttEditSettings AllowEditing="true" AllowAdding="true"
AllowDeleting="true"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 03, 28),
EndDate = new DateTime(2019, 07, 28),
Duration="4"
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 03, 29),
Progress = 30,
ParentId = 1,
Duration="2"
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 03, 29),
```

```
ParentId = 1,
Duration="4"
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 03, 29),
Duration = "1",
Progress = 30,
ParentId = 1,
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 03, 29),
EndDate = new DateTime(2019, 04, 2),
Duration="4"
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 03, 29),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 01),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 01),
Duration = "2",
ParentId = 5
}
};
return Tasks;
}
```

### Manually Scheduled Tasks

When the **TaskMode** property is set as **Manual**, the start date and end date of all the tasks in the project will be the same as given in the data source. That is, dates will not be validated based on factors such as dependencies between tasks, holidays, weekends, working time. We can restrict this mode in predecessor validation alone. That is, we can automatically validate the dates based on predecessor values by enabling the **ValidateManualTasksOnLinking** property.

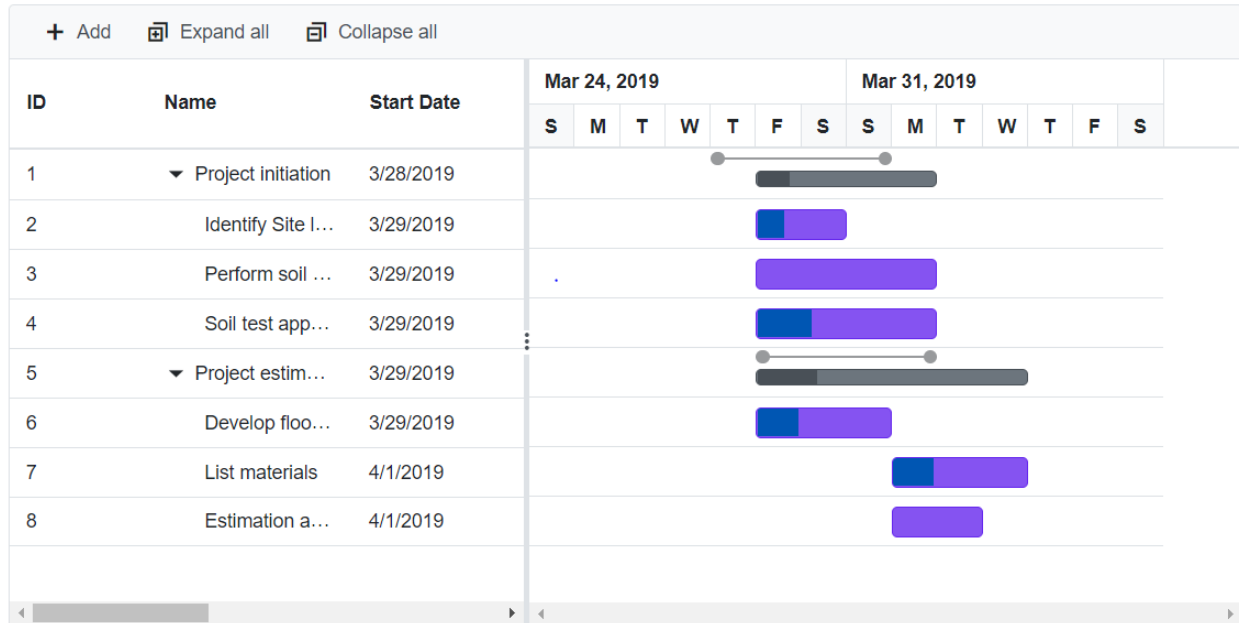
### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px"
TaskMode="ScheduleMode.Manual" ValidateManualTasksOnLinking="true"
Width="900px" TreeColumnIndex="1" Toolbar="@ (new List<string>() { "Add",
"Edit", "Update", "Delete", "Cancel", "ExpandAll", "CollapseAll" })">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentId="ParentId">
</GanttTaskFields>
<GanttEditSettings AllowEditing="true" AllowAdding="true"
AllowDeleting="true" AllowTaskbarEditing="true"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public string Predecessor { get; set; }
public int? ParentId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 03, 28),
EndDate = new DateTime(2019, 07, 28),
Duration="4"
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 03, 29),
Progress = 30,
ParentId = 1,
Duration="2"
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 03, 29),
ParentId = 1,
Duration="4"
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 03, 29),

```

```
Duration = "4",
Progress = 30,
ParentId = 1,
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 03, 29),
EndDate = new DateTime(2019, 04, 2),
Duration="4"
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 03, 29),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 01),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 01),
Duration = "2",
ParentId = 5
}
};
return Tasks;
}
}
```



### Custom

If we want to use some specific task mode for specific tasks, then we can set the `TaskMode` property as `Custom`. So, the scheduling mode for each task will be mapped from the data source field. The Boolean property `GanttTaskFields.Manual` is used to map the manual scheduling mode field from the data source.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" TaskMode="ScheduleMode.Custom"
Toolbar="@ (new List<string>() { "Add", "Cancel", "CollapseAll", "Delete",
"Edit", "ExpandAll", "Update" })" Height="450px" Width="1000px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId" Manual="IsManual">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true" AllowEditing="true"
AllowAdding="true" AllowDeleting="true"
Mode="Syncfusion.Blazor.Gantt.EditMode.Auto"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
```

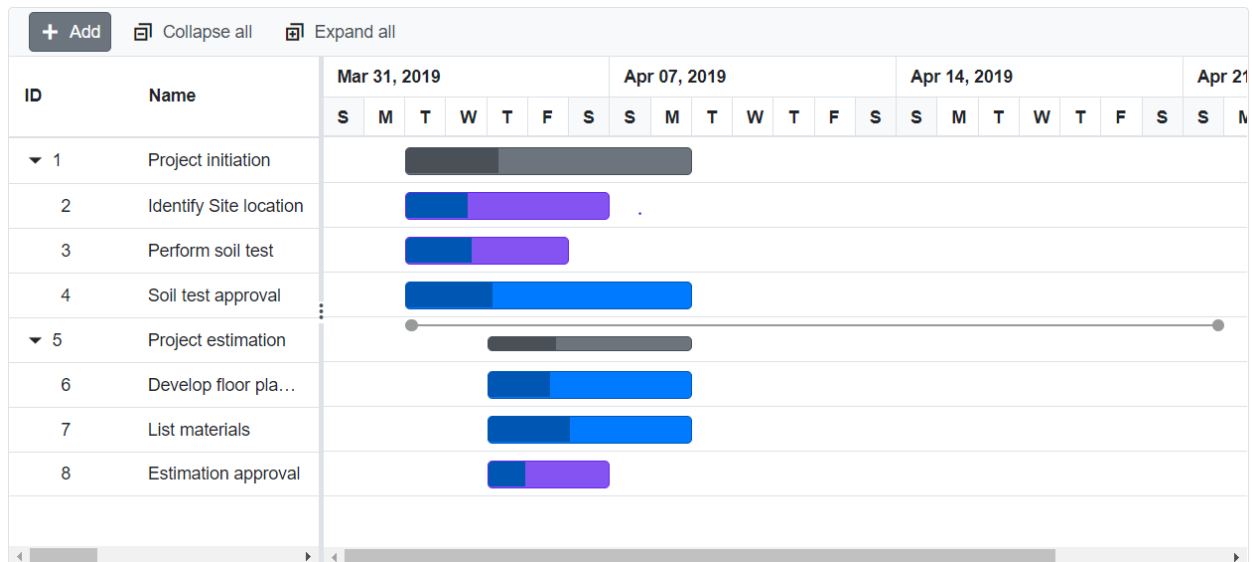


```
public string IsManual { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21)
        },
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "5",
            Progress = 30,
            ParentId = 1,
            IsManual="true",
        },
        new TaskData() {
            TaskId = 3,
            TaskName = "Perform soil test",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "4",
            Progress = 40,
            IsManual="true",
            ParentId = 1
        },
        new TaskData() {
            TaskId = 4,
            TaskName = "Soil test approval",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "5",
            Progress = 30,
            ParentId = 1
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            IsManual="true",
        },
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
            ParentId = 5
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
```

```

Progress = 40,
ParentId = 5
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
IsManual="true",
ParentId = 5
}
};
return Tasks;
}
}

```



### Unscheduled Tasks

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

#### Start Date Only



*End Date Only**Duration Only**Milestone*

A milestone is a task that has no start and end dates, but it has a duration value of zero. It is represented as follows.



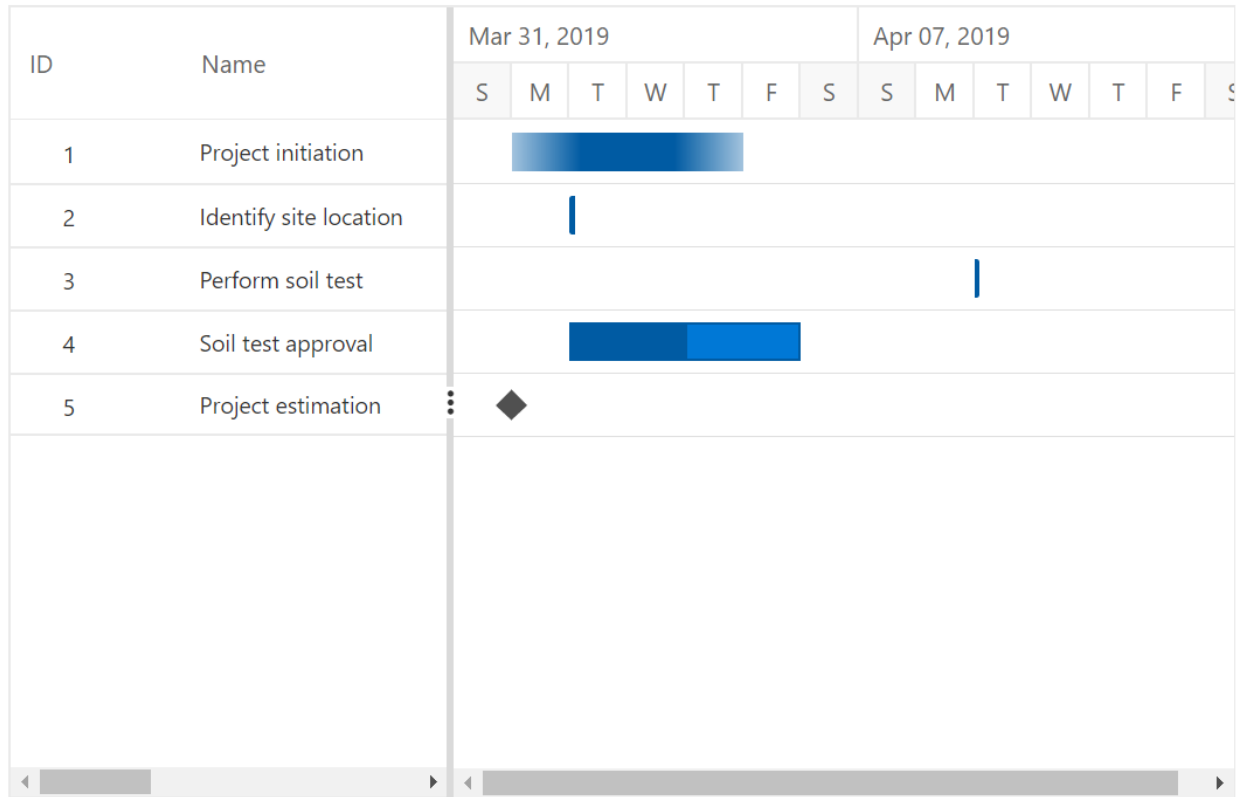
## Define Unscheduled Tasks in Data Source

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

**ASPX-CS**

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowUnscheduledTasks="true">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime? StartDate { get; set; }
  public DateTime? EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
}
public static List <TaskData> GetTaskCollection() {
```

```
List<TaskData> Tasks = new List<TaskData> () {  
    new TaskData() {  
        TaskId = 1,  
        TaskName = "Project initiation",  
        Duration = "4"  
    },  
    new TaskData() {  
        TaskId = 2,  
        TaskName = "Identify site location",  
        StartDate = new DateTime(2019, 04, 02)  
    },  
    new TaskData() {  
        TaskId = 3,  
        TaskName = "Perform soil test",  
        EndDate = new DateTime(2019, 04, 08)  
    },  
    new TaskData() {  
        TaskId = 4,  
        TaskName = "Soil test approval",  
        StartDate = new DateTime(2019, 04, 02),  
        EndDate = new DateTime(2019, 04, 06),  
        Progress = 50  
    },  
    new TaskData() {  
        TaskId = 5,  
        TaskName = "Project estimation",  
        Duration = "0"  
    }  
};  
return Tasks;  
}
```



If the `AllowUnscheduledTasks` property is set to false, then the Gantt Chart component automatically calculates the scheduled date values with a default value of duration 1 and the project start date is considered as the start date for the task.

### Working Time Range

In the Gantt Chart component, working hours in a day for a project can be defined by using the `GanttDayWorkingTime` property. Based on the working hours, automatic date scheduling and duration validations for a task are performed.

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

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
    EndDate="EndDate"
    Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttDayWorkingTimeCollection>
    <GanttDayWorkingTime From="9" To="18"></GanttDayWorkingTime>
  </GanttDayWorkingTimeCollection>
  <GanttTimelineSettings TimelineUnitSize="75">
    <GanttTopTierSettings Unit="TimelineViewMode.Day" Format="MMM
dd, yyyy"></GanttTopTierSettings>
    <GanttBottomTierSettings Unit="TimelineViewMode.Hour" Format="h.mm
tt"></GanttBottomTierSettings>
  </GanttTimelineSettings>
</SfGantt>
```

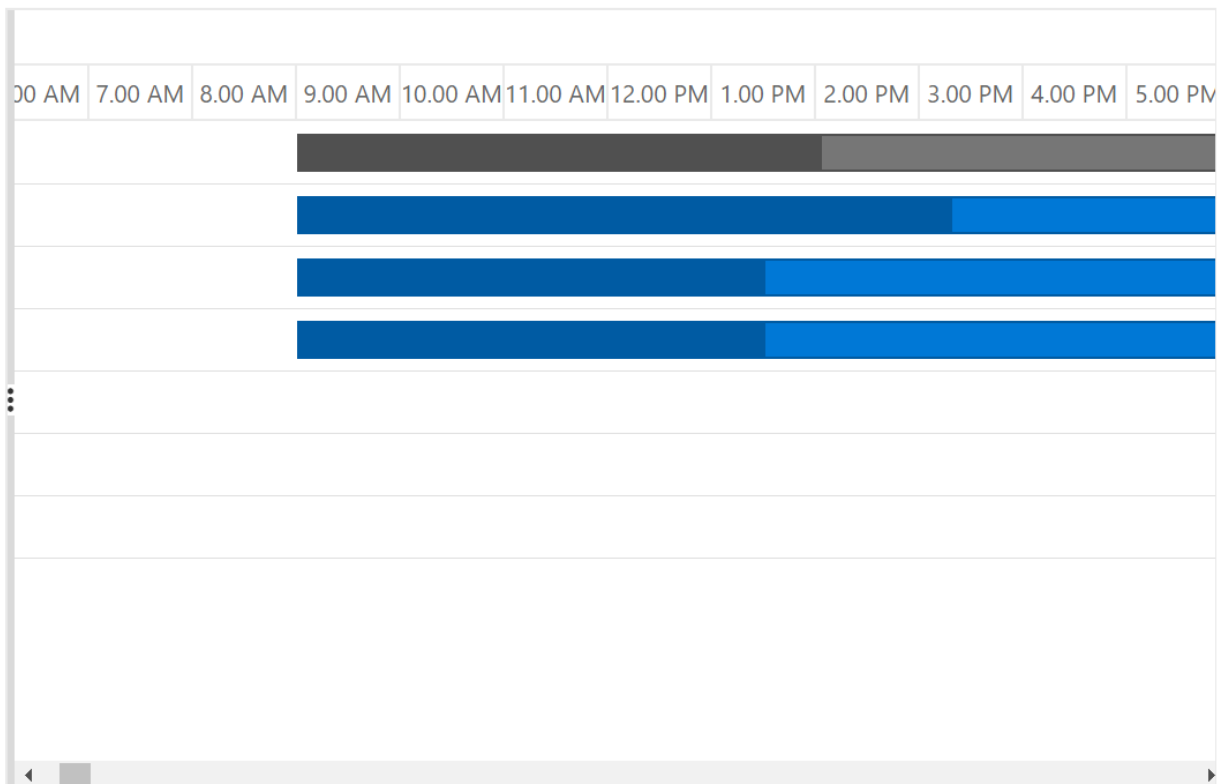
```
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "1",
Progress = 70,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "1",
Progress = 50
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "1",
Progress = 50
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
```

```

Duration = "1",
Progress = 70
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "1",
Progress = 50
},
})
}
};
return Tasks;
}
}

```

The following screen shot shows working time range in Gantt Chart component.



\* Individual tasks can lie between any time within the defined working time range of the project.

\* The `GanttDayWorkingTime` property is used to define the working time for the whole project.

#### Weekend or Non-working Days

Non-working days/weekends are used to represent the non-productive days in a project. You can exclude the non-working days in a work week using the `WorkWeek` property in Gantt Chart.

#### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" WorkWeek="@((new string[] { "Sunday",
"Monday", "Tuesday", "Wednesday", "Thursday" }))"
HighlightWeekends="true" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {

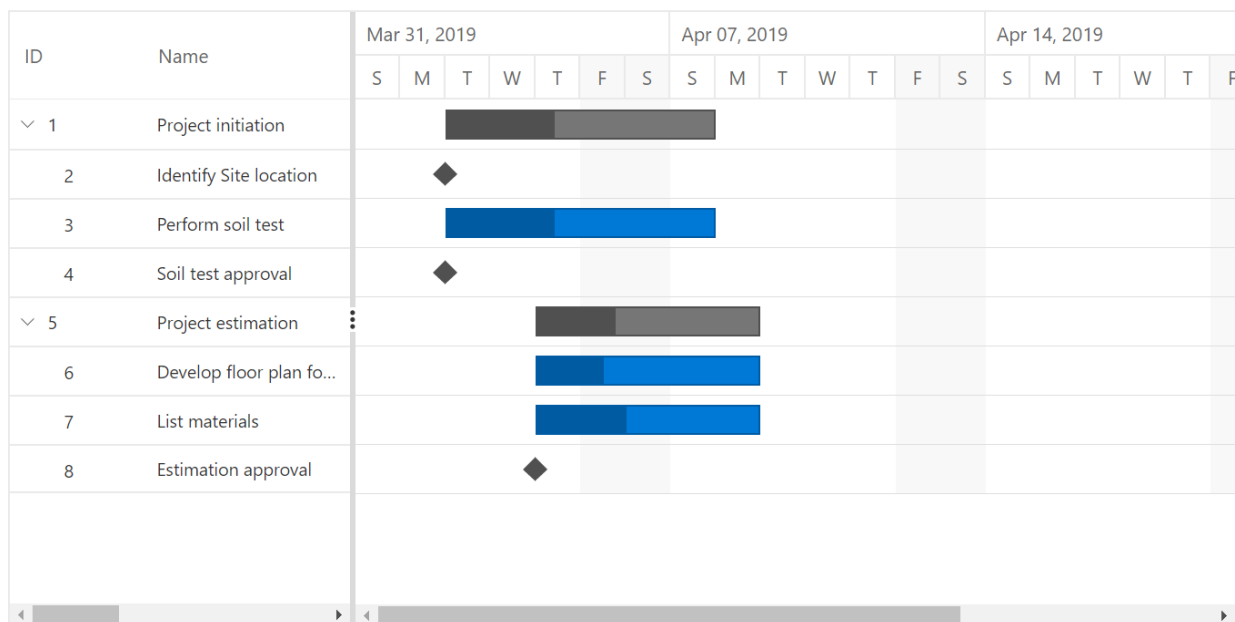
```



```

TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 6,
        TaskName = "Develop floor plan for estimation",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 7,
        TaskName = "List materials",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 40
    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "0",
        Progress = 30,
    }
})
};
return Tasks;
}
}

```



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

In the Gantt Chart component, you can make weekend as working day by setting the `IncludeWeekend` property to `true`.

You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

### Rows in Blazor Gantt Chart Component

Row represents a task information from the data source, and it is possible to perform the following actions in Gantt Chart rows.

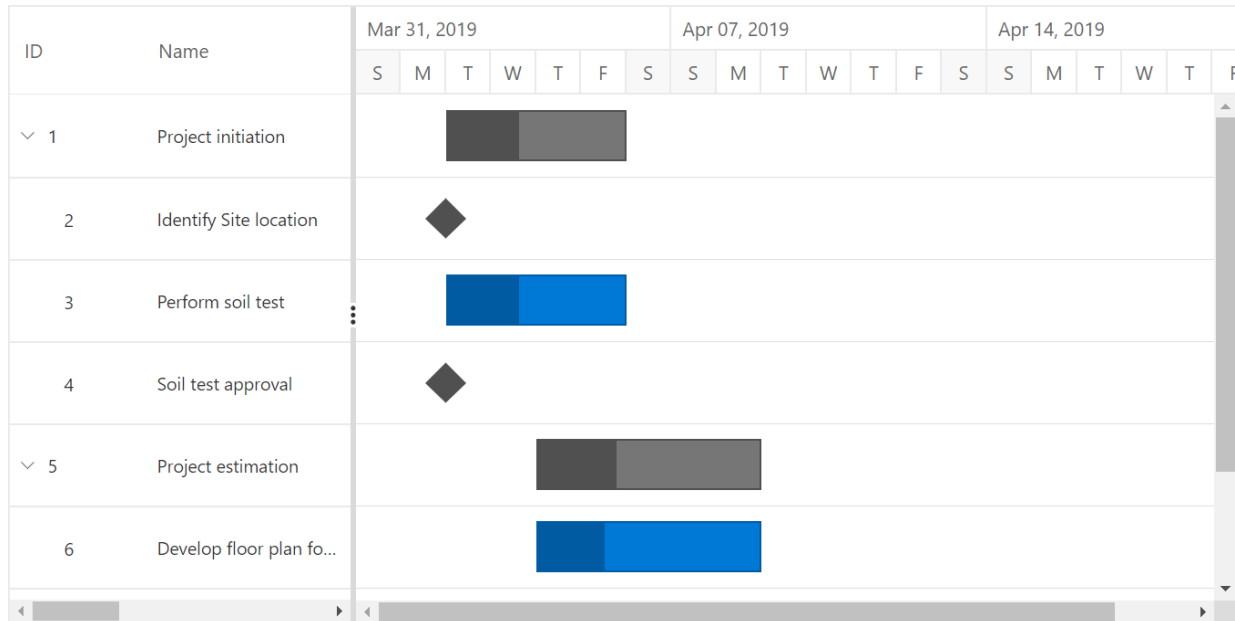
#### Row Height

It is possible to change the height of the row in Gantt Chart by setting row height in pixels to the `RowHeight` property. The following code example explains how to change the row height in Gantt Chart at load time.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" RowHeight=60 Height="450px"
Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
```

```
    },
    new TaskData() {
        TaskId = 3,
        TaskName = "Perform soil test",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "4",
        Progress = 40,
    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "0",
        Progress = 30
    },
    })
    },
    new TaskData() {
        TaskId = 5,
        TaskName = "Project estimation",
        StartDate = new DateTime(2019, 04, 02),
        EndDate = new DateTime(2019, 04, 21),
        SubTasks = (new List <TaskData> () {
            new TaskData() {
                TaskId = 6,
                TaskName = "Develop floor plan for estimation",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "3",
                Progress = 30,
            },
            new TaskData() {
                TaskId = 7,
                TaskName = "List materials",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "3",
                Progress = 40
            },
            new TaskData() {
                TaskId = 8,
                TaskName = "Estimation approval",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "0",
                Progress = 30,
            }
        })
    },
    });
    return Tasks;
}
```



### Expand or Collapse Row

In Gantt Chart parent tasks are expanded/collapsed by using expand/collapse icons, expand all/collapse all toolbar items and by using public methods. By default all tasks in Gantt Chart was rendered in expanded state but we can change this status through properties below.

### Collapse All Tasks at Gantt Chart Load

All tasks available in Gantt Chart was rendered in collapsed state by setting **CollapseAllParentTasks** property as **true**. The following code example shows how to use this property.

### ASPX-CS

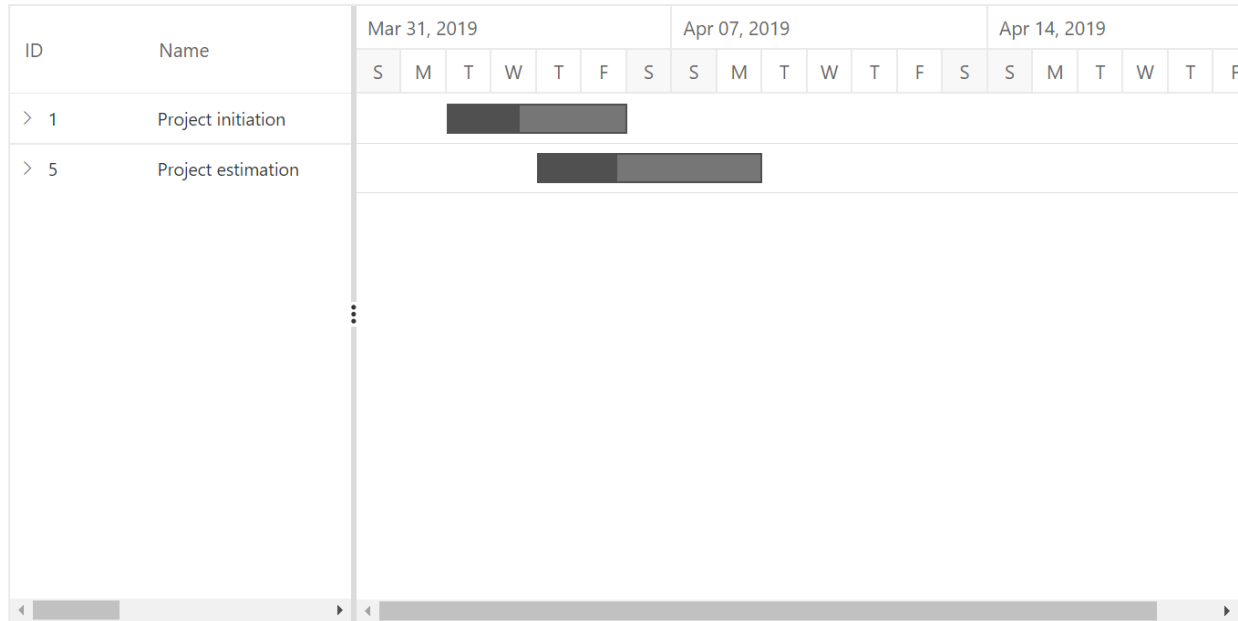
```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" CollapseAllParentTasks="true"
Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" ParentID="ParentId">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
```

```
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
ParentId = 1
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
ParentId = 5
},
new TaskData() {
TaskId = 8,
```

```

TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
ParentId = 5
}
};
return Tasks;
}
}

```



<!-- Define Expand / Collapse Status of Tasks

In Gantt Chart, we can render some tasks in collapsed state and some tasks in expanded state, this can be done by defining expand status of the task in data source. This value was mapped to Gantt Chart component by using `GanttTaskFields.ExpandState` property. The following code example shows how to use this property.

#### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks"
  ExpandState="isExpand">
  </GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
}

```

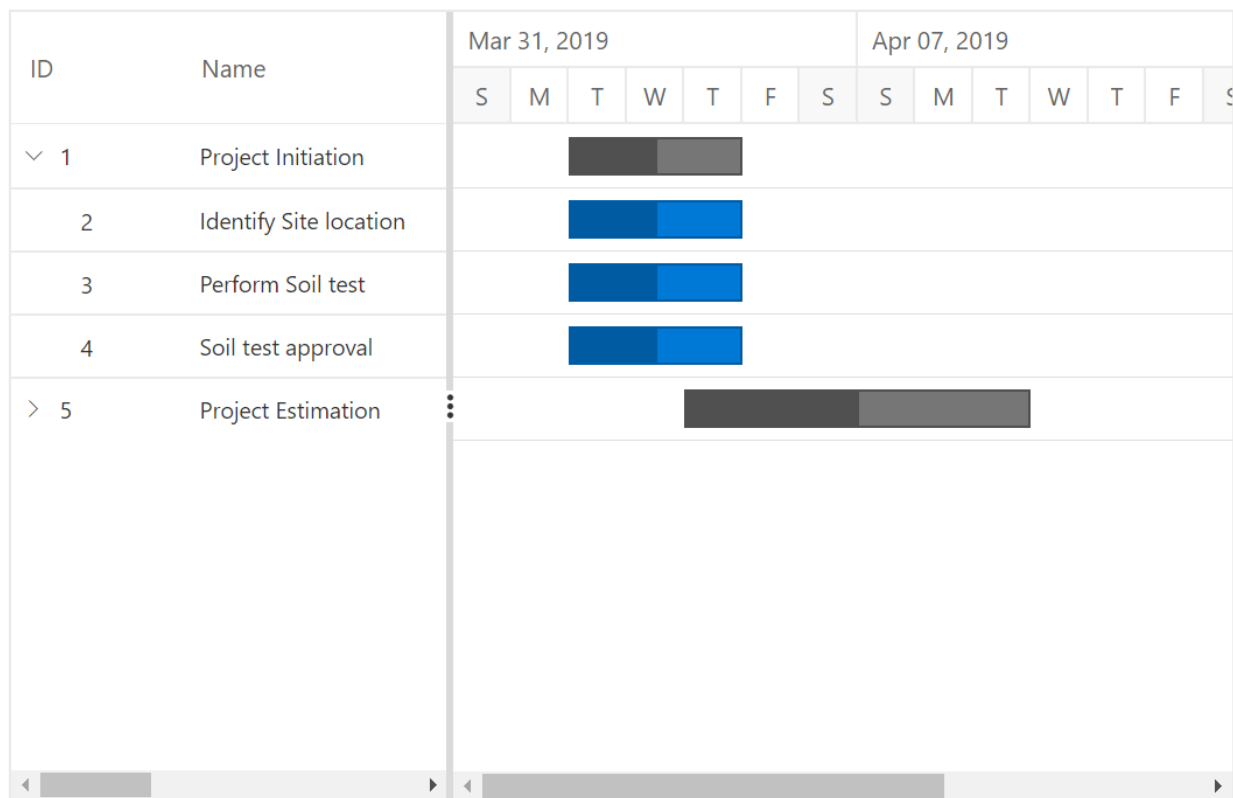
```
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public bool isExpand { get; set; }
    public List<TaskData> SubTasks { get; set; }
}

public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            isExpand = true,
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "3",
                    Progress = 50,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "3",
                    Progress = 50,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "3",
                    Progress = 50
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            isExpand = false,
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
            })
        },
    }
}
```

```

new TaskData() {
    TaskId = 7,
    TaskName = "List materials",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "3",
    Progress = 40
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
}
})
}
};
return Tasks;
}
}

```



-->

#### Customize Expand and Collapse action

On expand action **Expanding** and **Expanded** event will be triggered with current expanding row's information. Similarly on collapse action **Collapsing** and **Collapsed** event will be triggered. Using this events and it's arguments we can customize the expand/collapse action. The following code example



shows how to prevent the particular row from expand/collapse action using **Expanding** and **Collapsing** event.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttEvents Collapsing="Collapsing" Expanding="Expanding"
  TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public void
Collapsing(Syncfusion.Blazor.TreeGrid.RowCollapsingEventArgs<TaskData> args)
{
  if(args.Data.TaskId == 1){
    args.Cancel = true;
  }
}
public void
Expanding(Syncfusion.Blazor.TreeGrid.RowExpandingEventArgs<TaskData> args)
{
  if(args.Data.TaskId == 5){
    args.Cancel = true;
  }
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
  List<TaskData> Tasks = new List<TaskData>() {
    new TaskData() {
      TaskId = 1,
      TaskName = "Project initiation",
      StartDate = new DateTime(2019, 04, 02),
      EndDate = new DateTime(2019, 04, 21),
      SubTasks = (new List<TaskData> () {
        new TaskData() {
          TaskId = 2,
          TaskName = "Identify Site location",
          StartDate = new DateTime(2019, 04, 02),
```

```
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}
```

### Drag and Drop

You can dynamically rearrange the rows in the Gantt Chart component by using the **AllowRowDragAndDrop** property. Using this property, row drag and drop can be enabled or disabled in Gantt. Using this feature, rows can be dropped at above and below as a sibling or child to the existing rows

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
TreeColumnIndex="1" AllowRowDragAndDrop="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentId="ParentId">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 03, 28),
EndDate = new DateTime(2019, 07, 28),
Duration="4"
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 03, 29),
Progress = 30,
ParentId = 1,
Duration="2",
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 03, 29),
ParentId = 1,
Duration="4"
},
},
}
```

```

new TaskData() {
    TaskId = 4,
    TaskName = "Soil test approval",
    StartDate = new DateTime(2019, 03, 29),
    Duration = "4",
    Progress = 30,
    ParentId = 1
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 03, 29),
    EndDate = new DateTime(2019, 04, 2),
    Duration="4",
},
new TaskData() {
    TaskId = 6,
    TaskName = "Develop floor plan for estimation",
    StartDate = new DateTime(2019, 03, 29),
    Duration = "3",
    Progress = 30,
    ParentId = 5
},
new TaskData() {
    TaskId = 7,
    TaskName = "List materials",
    StartDate = new DateTime(2019, 04, 01),
    Duration = "3",
    Progress = 30,
    ParentId = 5
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 01),
    Duration = "2",
    ParentId = 5
}
};
return Tasks;
}
}

```

### Multiple Row Drag and Drop

Gantt also supports dragging multiple rows at a time and drop them on any rows above, below, or at child positions. In Gantt, you can enable the multiple drag and drop by setting the `GanttSelectionSettings.Type` to `Multiple` and you should enable the `AllowRowDragAndDrop` property.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
TreeColumnIndex="1" AllowRowDragAndDrop="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"

```

```

ParentID="ParentId">
</GanttTaskFields>
<GanttSelectionSettings
Type="Syncfusion.Blazor.Grids.SelectionType.Multiple"></GanttSelectionSettin
gs>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 03, 28),
EndDate = new DateTime(2019, 07, 28),
Duration="4"
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 03, 29),
Progress = 30,
ParentId = 1,
Duration="2",
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 03, 29),
ParentId = 1,
Duration="4"
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 03, 29),
Duration = "4",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",

```

```

StartDate = new DateTime(2019, 03, 29),
EndDate = new DateTime(2019, 04, 2),
Duration="4",
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 03, 29),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 01),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 01),
Duration = "2",
ParentId = 5
}
};
return Tasks;
}
}

```

### Drag and Drop Events

We provide various events to customize the row drag and drop action, the following table explains about the available events and its details.

Event Name | Description

**OnRowDragStart** | Triggers when drag action starts in Gantt.

**RowDropped** | Triggers when a drag row was dropped on the target row.

<!-- Customize Row Drag and Drop action

In Gantt, the **OnRowDragStart** and **RowDropped** events are triggered on row drag and drop action. Using this event, you can prevent dragging of particular record, validate the drop position, and cancel the drop action based on the target record and dragged record. The following topics explain about this.

#### Prevent Dragging of Particular Record

You can prevent drag action of the particular record by setting the **Cancel** property to **true**, which is available in the **OnRowDragStart** event argument based on our requirement. In the following sample, drag action was restricted for first parent record and its child records.

### ASPX-CS

-->**ASPX-CS**

```

@using Syncfusion.Blazor.Gantt
<button onclick="drag">Dynamic drag and drop</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="900px" TreeColumnIndex="1" AllowRowDragAndDrop="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId">
</GanttTaskFields>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public void drag() {
this.Gantt.ReorderRowAsync(2, 6, "Below");
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 03, 28),
EndDate = new DateTime(2019, 07, 28),
Duration="4"
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 03, 29),
Progress = 30,
ParentId = 1,
Duration="2",
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 03, 29),

```

```
ParentId = 1,
Duration="4"
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 03, 29),
Duration = "4",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 03, 29),
EndDate = new DateTime(2019, 04, 2),
Duration="4",
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 03, 29),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 01),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 01),
Duration = "2",
ParentId = 5
}
};
return Tasks;
}
```

### Customize Rows

You can customize the appearance of a row in grid side, by using the `RowDataBound` event and in chart side by using `QueryChartRowInfo` event

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
TreeColumnIndex="1">
```

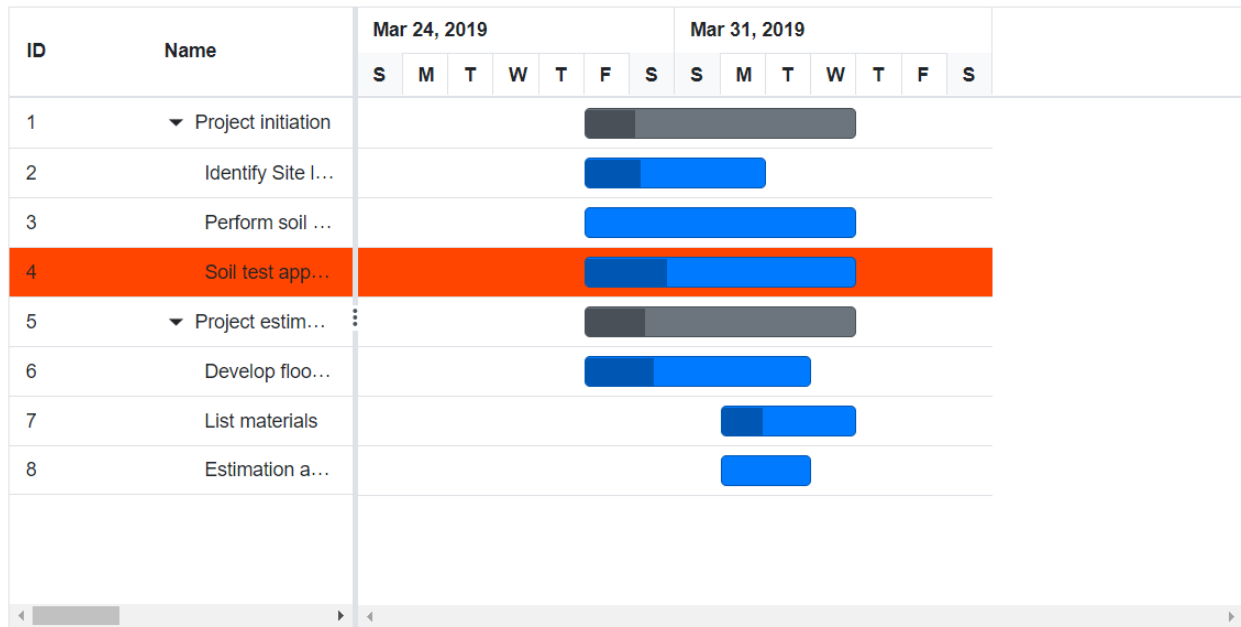


```

<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId">
</GanttTaskFields>
<GanttEvents RowDataBound="rowBound" QueryChartRowInfo="queryChart"
TValue="TaskData"></GanttEvents>
</SfGantt>
<style>
.rowcustom {
background-color: orangered;
}
</style>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public void rowBound(RowDataBoundEventArgs<TaskData> args) {
if (args.Data.TaskId == 4) {
args.Row.AddClass(new string[] { "rowcustom" });
}
}
public void queryChart(QueryChartRowInfoEventArgs<TaskData> args) {
if (args.Data.TaskId == 4) {
args.Row.AddClass(new string[] { "rowcustom" });
}
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 03, 28),
EndDate = new DateTime(2019, 07, 28),
Duration="4"
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 03, 29),
Progress = 30,
ParentId = 1,
Duration="2",
},
new TaskData() {
TaskId = 3,

```

```
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 03, 29),
ParentId = 1,
Duration="4"
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 03, 29),
Duration = "4",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 03, 29),
EndDate = new DateTime(2019, 04, 2),
Duration="4",
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 03, 29),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 01),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 01),
Duration = "2",
ParentId = 5
}
};
return Tasks;
}
```



### Styling Alternate Rows

You can change the background colour of alternative rows in Gantt chart, by overriding the class as shown below.

#### CSS

```
.e-altrow, tr.e-chart-row:nth-child(odd) {
background-color: #f2f2f2;
}
```

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
TreeColumnIndex="1">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId">
</GanttTaskFields>
</SfGantt>
<style>
.e-altrow, tr.e-chart-row:nth-child(odd) {
background-color: #f2f2f2;
}
</style>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
}
```

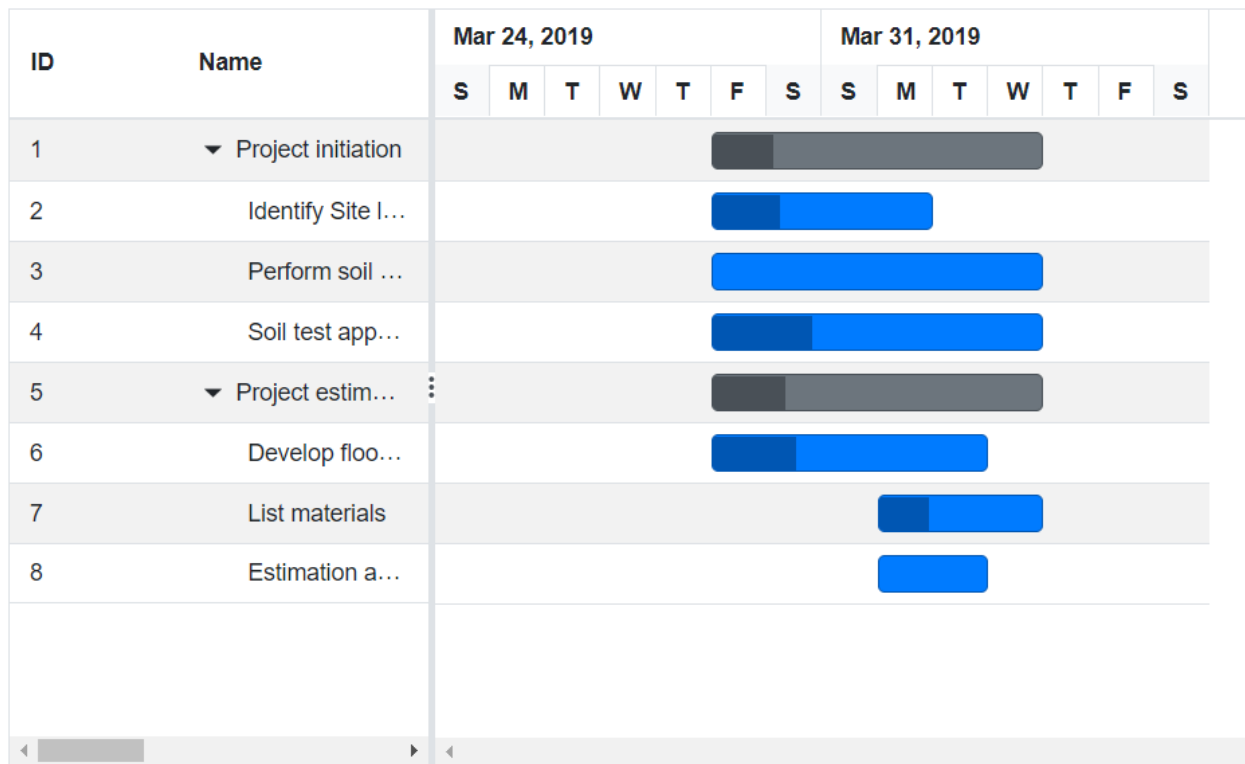
```
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}

public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 03, 28),
            EndDate = new DateTime(2019, 07, 28),
            Duration="4"
        },
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 03, 29),
            Progress = 30,
            ParentId = 1,
            Duration="2",
        },
        new TaskData() {
            TaskId = 3,
            TaskName = "Perform soil test",
            StartDate = new DateTime(2019, 03, 29),
            ParentId = 1,
            Duration="4"
        },
        new TaskData() {
            TaskId = 4,
            TaskName = "Soil test approval",
            StartDate = new DateTime(2019, 03, 29),
            Duration = "4",
            Progress = 30,
            ParentId = 1
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 03, 29),
            EndDate = new DateTime(2019, 04, 2),
            Duration="4",
        },
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 03, 29),
            Duration = "3",
            Progress = 30,
            ParentId = 5
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 01),
```

```

Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 01),
Duration = "2",
ParentId = 5
}
};
return Tasks;
}
}

```



You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

### Toolbar in Blazor Gantt Chart Component

The Gantt Chart component provides the toolbar support to handle Gantt Chart actions. The **Toolbar** property accepts the collection of built-in toolbar items and **ItemModel** objects for custom toolbar items.

### Built-in Toolbar Items

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

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

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

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Toolbar="@ (new List<string>() { "Add",
"Cancel", "CollapseAll", "Delete", "Edit", "ExpandAll", "NextTimeSpan",
"PrevTimeSpan", "Search", "Update", "Indent", "Outdent" })" Height="450px"
Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSearchSettings Fields="@Searchfields"></GanttSearchSettings>
<GanttEditSettings AllowEditing="true" AllowAdding="true"
AllowDeleting="true">
</GanttEditSettings>
</SfGantt>
@code{
public string[] Searchfields = new string[] { "TaskId", "TaskName",
"StartDate", "EndDate", "Duration", "Progress" };
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
```

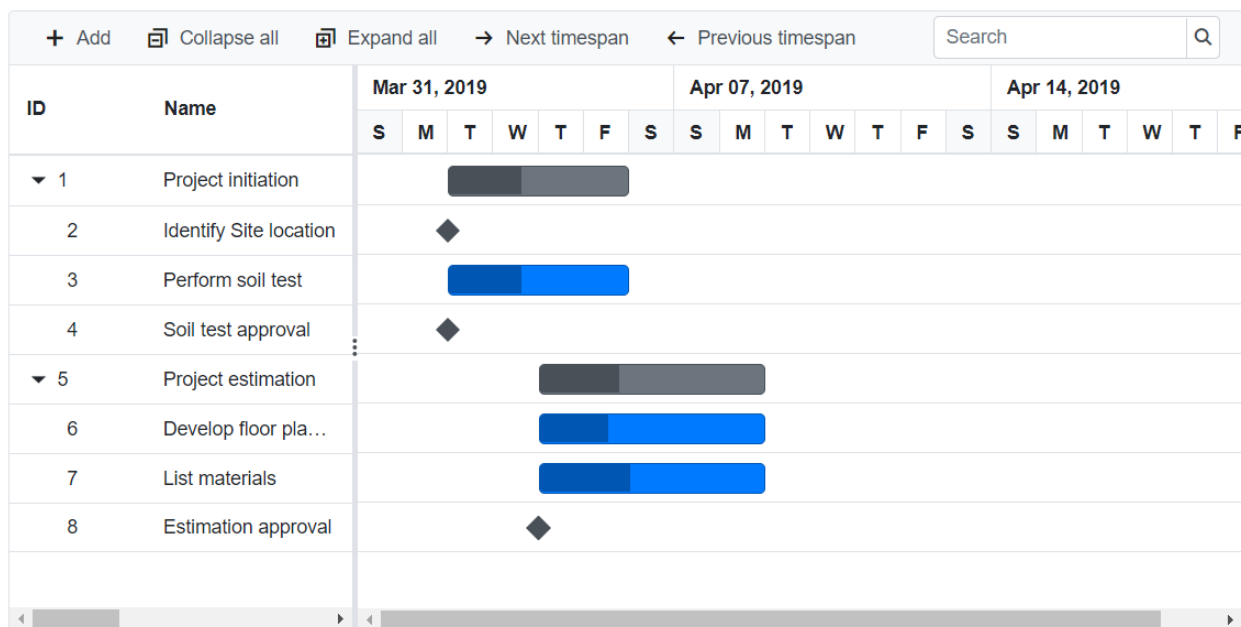
```
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}

public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
```

```

Progress = 40
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
}
})
}
};
return Tasks;
}
}

```



The **Toolbar** has options to define both built-in and custom toolbar items.

### Custom Toolbar Items

Custom toolbar items can be added to the toolbar by defining the **Toolbar** property as a collection of **ItemModels**. Actions for this customized toolbar items are defined in the **OnToolbarClick** event.

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

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Navigations
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Toolbar="Toolbaritems"
Height="450px" Width="900px" AllowFiltering = "true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">

```



```

</GanttTaskFields>
<GanttEvents OnToolbarClick="ToolbarClickHandler"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<ItemModel> ToolbarItems = new List<ItemModel>() { new
ItemModel() { Text = "Quick Filter",
TooltipText = "Quick Filter", Id = "toolbarfilter", Align = ItemAlign.Right
} };
public void ToolbarClickHandler(ClickEventArgs args)
{
if (args.Item.Id == "toolbarfilter")
{
this.Gantt.FilterByColumnAsync("TaskName", "startswith", "Identify");
}
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",

```

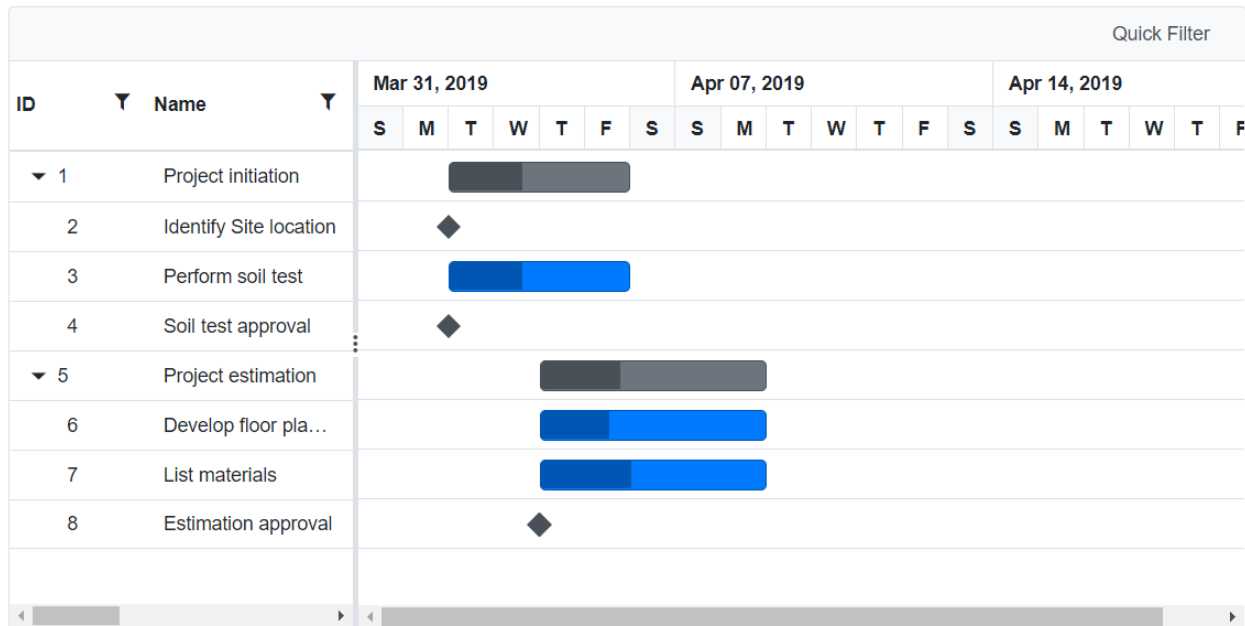
```
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
}))
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
},
});
return Tasks;
}
```

---

\* The **Toolbar** has options to define both built-in and custom toolbar items.

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

---



### Built-in and Custom Items in Toolbar

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

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

### ASPX-CS

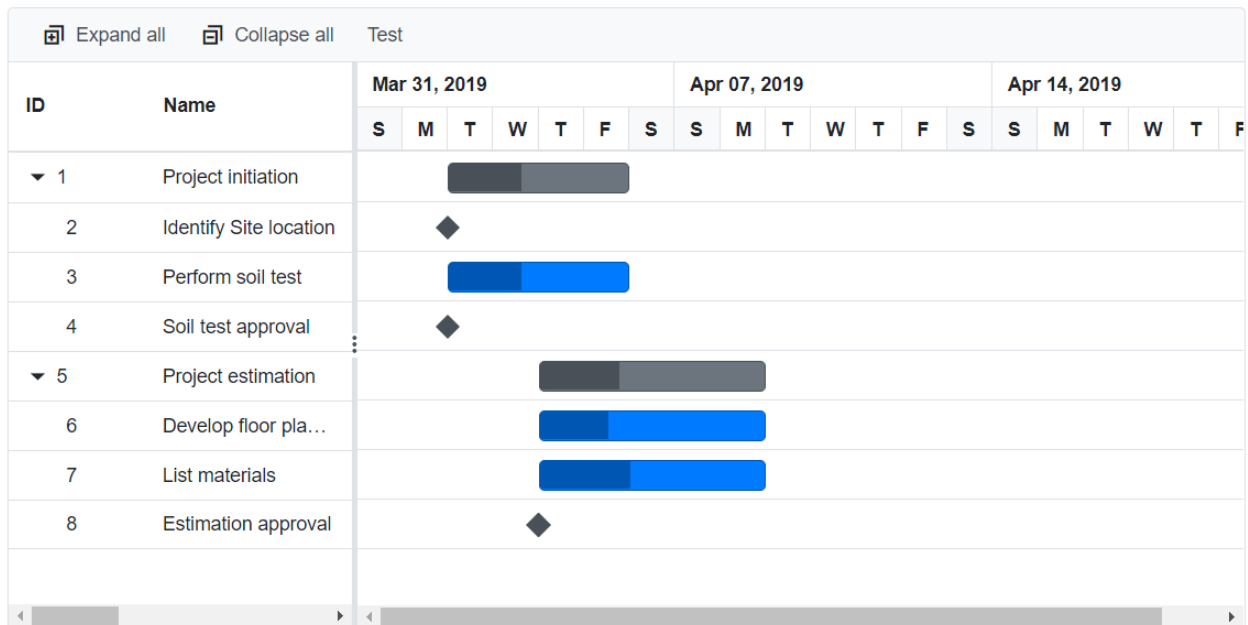
```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Navigations
<SfGantt DataSource="@TaskCollection" Toolbar="Toolbaritems" Height="450px"
Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEvents OnToolbarClick="ToolbarClickHandler"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<Object> Toolbaritems = new List<Object>() { "ExpandAll",
"CollapseAll", new ItemModel() { Text = "Test", TooltipText = "Test", Id =
"Test" } };
public void ToolbarClickHandler(ClickEventArgs args)
{
if (args.Item.Id == "Test")
{
Console.WriteLine("Custom toolbar click...");
}
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
}
```

```
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
```

```

TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```



### Enable or Disable Toolbar Items

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

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Navigations
<button @onclick="EnableToolbar">Enable</button>
<button @onclick="DisableToolbar">Disable</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Toolbar="ToolbarItems"
Height="450px" Width="900px" AllowFiltering = "true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>

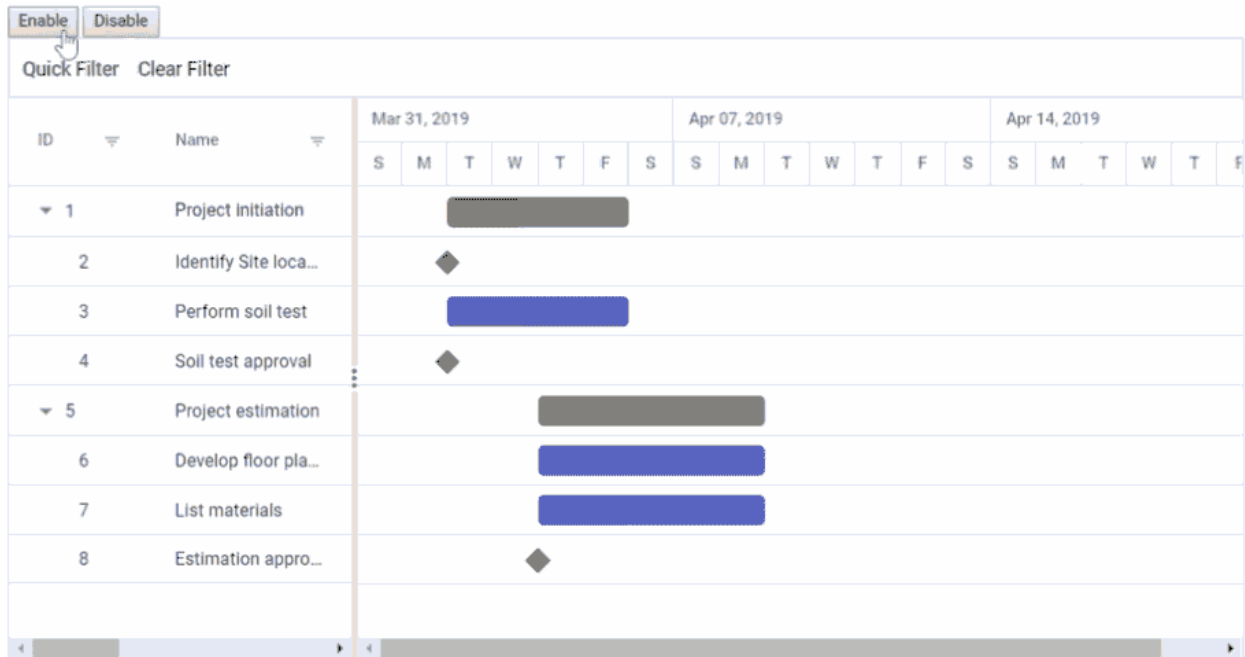
```

```

<GanttEvents OnToolbarClick="ToolbarClickHandler"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<ItemModel> ToolbarItems = new List<ItemModel>() { new
ItemModel() { Text = "Quick Filter", TooltipText = "Quick Filter", Id =
"quickfilter" },
new ItemModel() { Text = "Clear Filter", TooltipText = "Clear Filter", Id =
"clearfilter" } };
public void ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs
args)
{
if (args.Item.Id == "quickfilter")
{
this.Gantt.FilterByColumnAsync("TaskName", "startswith", "Identify");
}
if (args.Item.Id == "clearfilter")
{
this.Gantt.ClearFilteringAsync();
}
}
public void EnableToolbar()
{
this.Gantt.EnableItems(new List<int>() { 0,1 }, true);
}
public void DisableToolbar()
{
this.Gantt.EnableItems(new List<int>() { 0,1 }, false);
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),

```

```
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}
```



You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

### Excel Export in Blazor Gantt Chart Component

The excel export allows exporting GanttChart data to Excel and CSV formats. You need to use the **ExcelExportAsync** and **CsvExportAsync** method for exporting. To enable Excel export in the Gantt chart, set the **AllowExcelExport** property as true.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt ID="GanttContainer" @ref="Gantt" AllowExcelExport="true"
Toolbar="@ (new List<string>() { "ExcelExport", "CsvExport" })"
DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
Dependency="Predecessor" Child="SubTasks"></GanttTaskFields>
<GanttEvents OnToolbarClick="ToolbarClickHandler"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
public void ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs
args)
{
if (args.Item.Id == "GanttContainer_exceleexport")
{
this.Gantt.ExportToExcelAsync();
}
else if (args.Item.Id == "GanttContainer_csvexport")
{
this.Gantt.ExportToCsvAsync();
}
```



```
}
}
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public string Predecessor { get; set; }
    public List<TaskData> SubTasks { get; set; }
    public int[] ResourceId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Predecessor = "2"
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                    Predecessor = "3"
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
```

```

TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
Predecessor = "4"
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Predecessor = "6"
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Predecessor = "7"
}
})
}
};
return Tasks;
}
}

```

### Customize the Excel Export

The excel export provides an option to customize mapping of the gantt chart to excel document.

#### Export Hidden Columns

The excel export provides an option to export hidden columns of gantt chart by defining **IncludeHiddenColumn** as **true**.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt ID="GanttContainer" @ref="Gantt" AllowExcelExport="true"
Toolbar="@ (new List<string>() { "ExcelExport", "CsvExport" })"
DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId"></GanttTaskFields>
<GanttEvents OnToolbarClick="ToolbarClickHandler"
TValue="TaskData"></GanttEvents>
<GanttColumns>
<GanttColumn Field="TaskId" HeaderText="Task Id" Width="150"></GanttColumn>
<GanttColumn Field="TaskName" HeaderText="Task Name"
Width="250"></GanttColumn>
<GanttColumn Field="StartDate" HeaderText="StartDate" Width="250"
Visible="false"></GanttColumn>
<GanttColumn Field="Duration" Width="150" HeaderText="Duration"
Visible="false"></GanttColumn>
<GanttColumn Field="Progress" HeaderText="Progress"
Width="250"></GanttColumn>

```

```
</GanttColumns>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
public void ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs
args)
{
Syncfusion.Blazor.Grids.ExcelExportProperties ExportProperties = new
Syncfusion.Blazor.Grids.ExcelExportProperties();
ExportProperties.IncludeHiddenColumn = true;
if (args.Item.Id == "GanttContainer_excelexport")
{
this.Gantt.ExportToExcelAsync(ExportProperties);
}
else if (args.Item.Id == "GanttContainer_csvexport")
{
this.Gantt.ExportToCsvAsync(ExportProperties);
}
}
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
ParentId = 1
}
```

```

    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "0",
        Progress = 30,
        ParentId = 1
    },
    new TaskData() {
        TaskId = 5,
        TaskName = "Project estimation",
        StartDate = new DateTime(2019, 04, 02),
        EndDate = new DateTime(2019, 04, 21)
    },
    new TaskData() {
        TaskId = 6,
        TaskName = "Develop floor plan for estimation",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 30,
        ParentId = 5
    },
    new TaskData() {
        TaskId = 7,
        TaskName = "List materials",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 40,
        ParentId = 5
    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "0",
        Progress = 30,
        ParentId = 5
    }
};
return Tasks;
}
}

```

### Theme

The Excel export also provides an option to include custom theme for exported Excel document. To apply theme in exported Excel, define the **Theme** in **ExcelExportProperties**.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt ID="GanttContainer" @ref="Gantt" AllowExcelExport="true"
Toolbar="@ (new List<string>() { "ExcelExport" })"
DataSource="@TaskCollection" Height="450px" Width="700px">

```

```

<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
Dependency="Predecessor" Child="SubTasks"></GanttTaskFields>
<GanttEvents OnToolbarClick="ToolbarClickHandler"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
public void ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs
args)
{
Syncfusion.Blazor.Grids.ExcelExportProperties ExportProperties = new
Syncfusion.Blazor.Grids.ExcelExportProperties();
Syncfusion.Blazor.Grids.ExcelTheme Theme = new
Syncfusion.Blazor.Grids.ExcelTheme();
Syncfusion.Blazor.Grids.ExcelStyle ThemeStyle = new
Syncfusion.Blazor.Grids.ExcelStyle()
{
FontName = "Segoe UI",
FontColor = "#666666",
FontSize = 12
};
Theme.Header = ThemeStyle;
Theme.Record = ThemeStyle;
ExportProperties.Theme = Theme;
if (args.Item.Id == "GanttContainer_excelexport")
{
this.Gantt.ExportToExcelAsync(ExportProperties);
}
}
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public string Predecessor { get; set; }
public List<TaskData> SubTasks { get; set; }
public int[] ResourceId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,

```

```
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Predecessor = "2"
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
Predecessor = "3"
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
Predecessor = "4"
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Predecessor = "6"
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Predecessor = "7"
}
})
}
};
return Tasks;
}
}
```

---

By default, material theme is applied to the exported Excel document.

---

#### *File Name for Exported Document*

You can assign the file name for the exported document by defining **FileName** property in excel export properties.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Gantt
<SfGantt ID="GanttContainer" @ref="Gantt" AllowExcelExport="true"
Toolbar="@ (new List<string>() { "ExcelExport", "CsvExport" })"
DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
Dependency="Predecessor" Child="SubTasks"></GanttTaskFields>
<GanttEvents OnToolbarClick="ToolbarClickHandler"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
public void ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs
args)
{
if (args.Item.Id == "GanttContainer_excelexport")
{
Syncfusion.Blazor.Grids.ExcelExportProperties ExportProperties = new
Syncfusion.Blazor.Grids.ExcelExportProperties();
ExportProperties.FileName = "Gantt.xlsx";
this.Gantt.ExportToExcelAsync(ExportProperties);
}
else if (args.Item.Id == "GanttContainer_csvexport")
{
Syncfusion.Blazor.Grids.ExcelExportProperties ExportProperties = new
Syncfusion.Blazor.Grids.ExcelExportProperties();
ExportProperties.FileName = "Gantt.csv";
this.Gantt.ExportToCsvAsync(ExportProperties);
}
}
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public string Predecessor { get; set; }
public List<TaskData> SubTasks { get; set; }
public int[] ResourceId { get; set; }
}
```

```
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Predecessor = "2"
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                    Predecessor = "3"
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                    Predecessor = "4"
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Predecessor = "6"
                },
                new TaskData() {
                    TaskId = 8,
                    TaskName = "Estimation approval",
                    StartDate = new DateTime(2019, 04, 04),
```



```

Duration = "0",
Predecessor = "7"
}
})
}
};
return Tasks;
}
}

```

You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

### Data Markers in Blazor Gantt Chart Component

Data markers are a set of events used to represent the schedule events for a task. Data markers are defined in data source as array of objects, and this value is mapped to the Gantt Chart component using the `GanttTaskFields.Indicators` property. You can represent more than one data marker in a task.

Data markers can be defined using the following properties:

- **Date** : Defines the date of indicator.
- **IconClass** : Defines the icon class of indicator.
- **Name** : Defines the name of indicator.
- **Tooltip** : Defines the tooltip of indicator.

Data Marker **Tooltip** will be rendered only if tooltip property has value.

The following code example demonstrates how to implement data markers in the Gantt chart.

#### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration"
Progress="Progress" ParentID="ParentId" Indicators="Indicators">
</GanttTaskFields>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime? StartDate { get; set; }
public DateTime? EndDate { get; set; }
public string Duration { get; set; }
}
}

```

```

public int Progress { get; set; }
public int? ParentId { get; set; }
public List<GanttIndicator> Indicators { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
            new DateTime(2019, 03, 27), EndDate = new DateTime(2019, 04, 06),
            Indicators=(new List<GanttIndicator>())
            {
                new GanttIndicator() { Name="product", IconClass="e-btn-icon e-notes-info e-
                    icons e-icon-left e-gantt e-notes-info::before", Date=new DateTime(2019, 04,
                    11), Tooltip="sales"}
            }
        },
        new TaskData() { TaskId = 2, TaskName = "Identify Site location", StartDate
            = new DateTime(2019, 04, 02), Duration = "2", Progress = 30, ParentId = 1,
            Indicators=(new List<GanttIndicator>())
            {
                new GanttIndicator(){ Name="customer", IconClass="e-btn-icon e-notes-info e-
                    icons e-icon-left e-gantt e-notes-info::before", Date=new DateTime(2019, 04,
                    14), Tooltip="people" },
                new GanttIndicator(){ Name="product", IconClass="e-btn-icon e-notes-info e-
                    icons e-icon-left e-gantt e-notes-info::before", Date=new DateTime(2019, 04,
                    17), Tooltip="sales" }
            }
        },
        new TaskData() { TaskId = 3, TaskName = "Perform soil test", StartDate = new
            DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 06), Duration =
            "5", Progress = 40, ParentId = 1 },
        new TaskData() { TaskId = 4, TaskName = "Soil test approval", StartDate =
            new DateTime(2019, 04, 02), Duration = "5", EndDate = new DateTime(2019, 04,
            06), Progress = 30, ParentId = 1 },
        new TaskData() { TaskId = 5, TaskName = "Project initiation", StartDate =
            new DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 08) },
        new TaskData() { TaskId = 6, TaskName = "Identify Site location", StartDate
            = new DateTime(2019, 04, 02), Duration = "2", Progress = 30, ParentId = 5 },
        new TaskData() { TaskId = 7, TaskName = "Perform soil test", StartDate = new
            DateTime(2019, 04, 02), Duration = "4", Progress = 40, ParentId = 5 },
        new TaskData() { TaskId = 8, TaskName = "Soil test approval", StartDate =
            new DateTime(2019, 04, 02), Duration = "5", Progress = 30, ParentId = 5 },
    };
    return Tasks;
}
}

```

You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

### WebAssembly Performance in Blazor Gantt Component

This section provides performance guidelines for using the Syncfusion Gantt Chart component efficiently in the Blazor WebAssembly application. The general framework Blazor WebAssembly performance guidelines can be found [here](#).

Refer to the Getting Started with [Blazor Server-Side Gantt](#) and [Blazor WebAssembly Gantt](#) documentation pages for configuration specifications.

### Avoid unnecessary component renders

During Blazor diffing algorithm, every cell of the Gantt Chart component and its child component will be checked for re-rendering. For instance, having [EventCallback](#) on the application or Gantt Chart will check every child component once the event callback is completed.

You can have fine-grained control over Gantt Chart component rendering. The [PreventRender](#) method helps you to avoid unnecessary re-rendering of the Gantt Chart component. This method internally overrides the [ShouldRender](#) method of the Gantt Chart to prevent rendering.

In the following example:

- The [PreventRender](#) method is called in the **IncrementCount** method which is a click callback.
- Now Gantt Chart component will not be a part of the rendering which happens because of the click event and **currentCount** alone will get updated.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<h1>Counter</h1>
<p>Current count: @currentCount</p>
<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="800px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId">
</GanttTaskFields>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
private int currentCount = 0;
private void IncrementCount()
{
  Gantt.PreventRender();
  currentCount++;
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public int? ParentId { get; set; }
}
public static List<TaskData> GetTaskCollection()
```

```
{
  List<TaskData> Tasks = new List<TaskData>() {
    new TaskData() { TaskId = 1, TaskName = "Product concept", StartDate = new
    DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04, 08), Duration =
    "5days"},
    new TaskData() { TaskId = 2, TaskName = "Defining the product usage",
    StartDate = new DateTime(2019, 04, 02), EndDate = new DateTime(2019, 04,
    08), Duration = "3", Progress = 30, ParentId = 1},
    new TaskData() { TaskId = 3, TaskName = "Defining the Target audience",
    EndDate = new DateTime(2019, 04, 04), Progress = 40, ParentId = 1},
    new TaskData() { TaskId = 4, TaskName = "Prepare product sketch and notes",
    StartDate = new DateTime(2019, 04, 05), Duration = "2", Progress = 30,
    ParentId = 1 },
    new TaskData() { TaskId = 5, TaskName = "Concept approval", StartDate = new
    DateTime(2019, 04, 08), EndDate = new DateTime(2019, 04, 08), Duration="0"
    },
    new TaskData() { TaskId = 6, TaskName = "Market Research", StartDate = new
    DateTime(2019, 04, 09), EndDate = new DateTime(2019, 04, 18), Duration =
    "4", Progress = 30 },
    new TaskData() { TaskId = 7, TaskName = "Demand Analysis", Duration = "4",
    Progress = 40, ParentId = 6 },
    new TaskData() { TaskId = 8, TaskName = "Customer Strength", StartDate = new
    DateTime(2019, 04, 09), EndDate = new DateTime(2019, 04, 12), Duration =
    "4", Progress = 30, ParentId = 7 },
    new TaskData() { TaskId = 9, TaskName = "Market Opportunity analysis",
    StartDate = new DateTime(2019, 04, 09), EndDate = new DateTime(2019, 04,
    012), Duration="4", ParentId= 7 }
  };
  return Tasks;
} }
```

---

The [PreventRender](#) method accepts the Boolean argument that accepts true or false to disable or enable rendering respectively.

---

This method can be used only after the Gantt component completed the initial rendering. Calling this method during initial rendering will not have any effect.

### Accessibility in Blazor Gantt Chart Component

Accessibility is achieved in the Gantt component through the WAI-ARIA standard and keyboard navigations. The Gantt features can be effectively accessed through assistive technologies such as screen readers. It is also available with a built-in keyboard navigation support; it makes accessibility easier for the people who use assistive technologies or who completely rely on the Keyboard support.

#### 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. It helps to provide information about elements in a document for assistive technology.

The following ARIA attributes are used in Gantt:

Attributes	Description
------------	-------------

---	---
-----	-----

| grid (role) | This attribute is added to the `e-table` element present in the Gantt, which represents Grid part |

| gridcell (role) | This attribute is added to the `td` elements present within the `e-table`, which represents the work cells of Gantt |

| columnheader (role) | This attribute is added to the `th` elements within the `e-table`, which represents the header cells of Grid table |

| separator (role) | This attribute is added to the `e-split-bar` element, which represents the splitter between the Grid table and Chart |

| dialog (role) | This attribute is added to the `e-dialog` element, which represents the pop-up dialog |

| toolbar (role) | This attribute is added to the `e-gantt-toolbar` element, which represents the toolbars of Gantt |

| aria-label | It indicates the element's information<br> It is assigned to the Gantt UI elements such as timeline cell, taskbar, left label, right label, dependency line, and event markers. |

| aria-selected | This attribute is assigned to the Gantt chart row, and it defaults to `false`. The value is changed to `true` when the user selects a grid cell or task |

| aria-expanded | This attribute is assigned to the Gantt chart parent task row. The value is changed to `true` when the user clicks a parent taskbar to expand. After the user clicked a parent taskbar to collapse, the attribute value is changed to `false` |

| aria-grabbed | This attribute is assigned to the taskbars of Gantt when the user tries to achieve taskbar editing |

### Keyboard Navigation

Gantt functionalities can be interactive with keyboard shortcuts.

The following keyboard shortcuts are supported by Gantt.

Interaction Keys | Description

**Home** | Selects the first row.

**End** | Selects the last row.

**DownArrow** | Moves the cell focus/row or cell selection downward.

**UpArrow** | Moves the cell focus/row or cell selection upward.

**LeftArrow** | Moves the cell focus/row or cell selection left side.

**RightArrow** | Moves the cell focus/row or cell selection right side.

**Ctrl + Up Arrow** | Collapses all tasks.

**Ctrl + Down Arrow** | Expands all tasks.

**Ctrl + Shift + Up Arrow** | Collapses the selected row.

**Ctrl + Shift + Down Arrow** | Expands the selected row.

**Enter** | Saves request.

Esc | Cancels request.

Insert | Adds a new row.

Ctrl + Insert | Opens addRowDialog.

Ctrl + F2 | Opens editRowDialog.

Delete | Deletes the selected row.

Shift + F5 | FocusTask

Ctrl + Shift + F | Focus search

Shift + DownArrow | Extends the row/cell selection downwards.

Shift + UpArrow | Extends the row/cell selection upwards.

Shift + LeftArrow | Extends the cell selection to the left side.

Shift + RightArrow | Extends the cell selection to the right side.

---

You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

---

### Globalization in Blazor Gantt Chart 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 Gantt. The Gantt component has static text on some features (like task information text, context menu options, 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 Gantt.

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)

Gantt\_EmptyRecord | Keine Datensätze zum Anzeigen

Gantt\_Id | ICH WÜRDE

Gantt\_Name | Name

Gantt\_StartDate | Anfangsdatum

Gantt\_EndDate | Endtermin

Gantt\_Duration | Dauer

Gantt\_Progress | Fortschritt

Gantt\_Dependency | Abhängigkeit

Gantt\_Notes | Anmerkungen

Gantt\_BaselineStartDate | Basisstartdatum

Gantt\_BaselineEndDate | Baseline-Enddatum

Gantt\_Type | Art

Gantt\_Offset | Offset

Gantt\_ResourceName | Ressourcen

Gantt\_ResourceID | Ressourcen-ID

Gantt\_Day | Tag

Gantt\_Hour | Stunde

Gantt\_Minute | Minute

Gantt\_Days | Tage

Gantt\_Hours | Std

Gantt\_Minutes | Protokoll

Gantt\_GeneralTab | Allgemeines

Gantt\_CustomTab | Benutzerdefinierte Spalten

Gantt\_WriteNotes | Notizen schreiben

Gantt\_AddDialogTitle | Neue Aufgabe

Gantt\_EditDialogTitle | Aufgabeninformationen

Gantt\_SaveButton | speichern

Gantt\_Add | Hinzufügen

Gantt\_Edit | Bearbeiten

Gantt\_Update | Aktualisieren

Gantt\_Delete | Löschen

Gantt\_Cancel | Stornieren

Gantt\_Search | Suche

Gantt\_Task | Aufgabe

Gantt\_Tasks | Aufgaben

Gantt\_ZoomIn | Hineinzoomen

Gantt\_ZoomOut | Rauszoomen

Gantt\_ZoomToFit | Zoom passend

Gantt\_ExcelExport | Excel-Export

Gantt\_CsvExport | CSV-Export

Gantt\_ExpandAll | Alle erweitern

Gantt\_CollapseAll | Alles einklappen

Gantt\_NextTimeSpan | Nächste Zeitspanne

Gantt\_PrevTimeSpan | Vorherige Zeitspanne

Gantt\_OkText | In Ordnung

Gantt\_ConfirmDelete | Möchten Sie den Datensatz wirklich löschen?

Gantt\_From | Von

Gantt\_To | Zu

Gantt\_TaskLink | Task-Link

Gantt\_Lag | Verzögerung

Gantt\_Start | Start

Gantt\_Finish | Fertig

Gantt\_EnterValue | Geben Sie den Wert ein

GanttTaskBeforePredecessorFS | Sie haben '{0}' verschoben, um vor dem Ende von '{1}' zu beginnen, und die beiden Aufgaben sind miteinander verknüpft. Infolgedessen können die Links nicht beachtet werden. Wählen Sie unten eine Aktion aus, die ausgeführt werden soll

GanttTaskAfterPredecessorFS | Sie haben '{0}' von '{1}' entfernt und die beiden Aufgaben sind miteinander verknüpft. Infolgedessen können die Links nicht beachtet werden. Wählen Sie unten eine Aktion aus, die ausgeführt werden soll

GanttTaskBeforePredecessorSS | Sie haben '{0}' verschoben, um vor dem Start von '{1}' zu beginnen, und die beiden Aufgaben sind miteinander verknüpft. Infolgedessen können die Links nicht beachtet werden. Wählen Sie unten eine Aktion aus, die ausgeführt werden soll



`GanttTaskAfterPredecessorSS` | Sie haben '{0}' verschoben, um nach dem Start von '{1}' zu beginnen, und die beiden Aufgaben sind miteinander verbunden. Infolgedessen können die Links nicht beachtet werden. Wählen Sie unten eine Aktion aus, die ausgeführt werden soll

`GanttTaskBeforePredecessorFF` | Sie haben '{0}' verschoben, um den Vorgang zu beenden, bevor '{1}' abgeschlossen ist, und die beiden Aufgaben sind miteinander verknüpft. Infolgedessen können die Links nicht beachtet werden. Wählen Sie unten eine Aktion aus, die ausgeführt werden soll

`GanttTaskBeforePredecessorSF` | Sie haben '{0}' von '{1}' zum Start verschoben und die beiden Aufgaben sind miteinander verbunden. Infolgedessen können die Links nicht beachtet werden. Wählen Sie unten eine Aktion aus, die ausgeführt werden soll

`GanttTaskAfterPredecessorSF` | Sie haben '{0}' nach dem Start von '{1}' verschoben und die beiden Aufgaben sind miteinander verbunden. Infolgedessen können die Links nicht beachtet werden. Wählen Sie unten eine Aktion aus, die ausgeführt werden soll

`Gantt_TaskInformation` | Aufgabeninformationen

`Gantt_DeleteTask` | Aufgabe löschen

`Gantt_DeleteDependency` | Abhängigkeit löschen

`Gantt_Convert` | Konvertieren

`Gantt_Save` | speichern

`Gantt_Above` | Über

`Gantt_Below` | Unten

`Gantt_Child` | Kind

`Gantt_Milestone` | Meilenstein

`Gantt_ToTask` | Zur Aufgabe

`Gantt_ToMilestone` | Zum Meilenstein

`Gantt_EventMarkers` | Ereignismarker

`Gantt_LeftTaskLabel` | Linke Aufgabenbezeichnung

`Gantt_RightTaskLabel` | Richtige Aufgabenbezeichnung

`Gantt_TimelineCell` | Timeline-Zelle

`Gantt_ConfirmPredecessorDelete` | Möchten Sie den Abhängigkeitslink wirklich entfernen?

`Gantt_Indent` | Einzug

`Gantt_Outdent` | Outdent

#### *Blazor Server Side*

In the following examples, demonstrate how to enable **Localization** for Gantt in server side Blazor samples.

- Open the **Startup.cs** file and add the below configuration in the **ConfigureServices** function as follows.

#### **C#**

```
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 Gantt using **Locale** property.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt ID="GanttContainer" @ref="Gantt" Locale="de"
AllowExcelExport="true" Toolbar="@ (new List<string>() { "Add",
"Cancel","CollapseAll", "Delete", "Edit", "ExpandAll","NextTimeSpan",
"PrevTimeSpan", "Search", "Update", "Indent", "Outdent","ExcelExport",
"CsvExport" })" DataSource="@TaskCollection" Height="450px" Width="700px">
    <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
    EndDate="EndDate" Duration="Duration" Progress="Progress"
    ParentID="ParentId"></GanttTaskFields>
    <GanttColumns>
        <GanttColumn Field="TaskId" HeaderText="Task Id" Width="150"></GanttColumn>
        <GanttColumn Field="TaskName" HeaderText="Task Name"
        Width="250"></GanttColumn>
        <GanttColumn Field="StartDate" HeaderText="StartDate"
        Width="250"></GanttColumn>
        <GanttColumn Field="Duration" Width="150"
        HeaderText="Duration"></GanttColumn>
        <GanttColumn Field="Progress" HeaderText="Progress"
        Width="250"></GanttColumn>
    </GanttColumns>
    <GanttEditSettings AllowEditing="true" AllowAdding="true"
    AllowTaskbarEditing="true"></GanttEditSettings>
</SfGantt>

@code{
    public SfGantt<TaskData> Gantt;
    public List<TaskData> TaskCollection { get; set; }
    protected override void OnInitialized()
    {
        this.TaskCollection = GetTaskCollection();
    }
    public class TaskData
    {
        public int TaskId { get; set; }
        public string TaskName { get; set; }
        public DateTime StartDate { get; set; }
        public DateTime EndDate { get; set; }
        public string Duration { get; set; }
        public int Progress { get; set; }
        public int? ParentId { get; set; }
    }
}
```

```
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21)
        },
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30,
            ParentId = 1
        },
        new TaskData() {
            TaskId = 3,
            TaskName = "Perform soil test",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "4",
            Progress = 40,
            ParentId = 1
        },
        new TaskData() {
            TaskId = 4,
            TaskName = "Soil test approval",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30,
            ParentId = 1
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21)
        },
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
            ParentId = 5
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40,
            ParentId = 5
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
```

```
StartDate = new DateTime(2019, 04, 04),  
Duration = "0",  
Progress = 30,  
ParentId = 5  
}  
};  
return Tasks;  
}  
}
```

### Blazor WebAssembly

In the following examples, demonstrate how to enable **Localization** for Gantt 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 Gantt using **Locale** property.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt ID="GanttContainer" @ref="Gantt" Locale="de"
AllowExcelExport="true" Toolbar="@ (new List<string>() { "Add", "Cancel",
"CollapseAll", "Delete", "Edit", "ExpandAll", "NextTimeSpan",
"PrevTimeSpan", "Search", "Update", "Indent", "Outdent", "ExcelExport",
"CsvExport" })" DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId"></GanttTaskFields>
<GanttColumns>
<GanttColumn Field="TaskId" HeaderText="Task Id" Width="150"></GanttColumn>
<GanttColumn Field="TaskName" HeaderText="Task Name"
Width="250"></GanttColumn>
<GanttColumn Field="StartDate" HeaderText="StartDate"
Width="250"></GanttColumn>
<GanttColumn Field="Duration" Width="150"
HeaderText="Duration"></GanttColumn>
<GanttColumn Field="Progress" HeaderText="Progress"
Width="250"></GanttColumn>
</GanttColumns>
<GanttEditSettings AllowEditing="true" AllowAdding="true"
AllowTaskbarEditing="true"></GanttEditSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
}

```

```
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}

public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21)
        },
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30,
            ParentId = 1
        },
        new TaskData() {
            TaskId = 3,
            TaskName = "Perform soil test",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "4",
            Progress = 40,
            ParentId = 1
        },
        new TaskData() {
            TaskId = 4,
            TaskName = "Soil test approval",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30,
            ParentId = 1
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21)
        },
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
            ParentId = 5
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
```

```

Progress = 40,
ParentId = 5
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
ParentId = 5
}
};
return Tasks;
}
}

```

## 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 Gantt 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 Gantt.

#### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt ID="GanttContainer" @ref="Gantt" EnableRtl="true"
DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
ParentID="ParentId"></GanttTaskFields>
<GanttEditSettings AllowEditing="true" AllowAdding="true"
AllowTaskbarEditing="true"></GanttEditSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
}
}

```



```
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public int? ParentId { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
ParentId = 1
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
ParentId = 1
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21)
},
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
ParentId = 5
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40,
```

```
ParentId = 5
},
new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
    ParentId = 5
}
};
return Tasks;
}
```

---

Gantt chart doesn't have RTL support when predecessors enabled.

You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

---

### Touch Interaction in Blazor Gantt Chart Component

The Gantt Chart component supports to perform user interactions in mobile and tablet devices. This section explains how to interact with the Gantt features in touch-enabled devices.

#### Tooltip

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

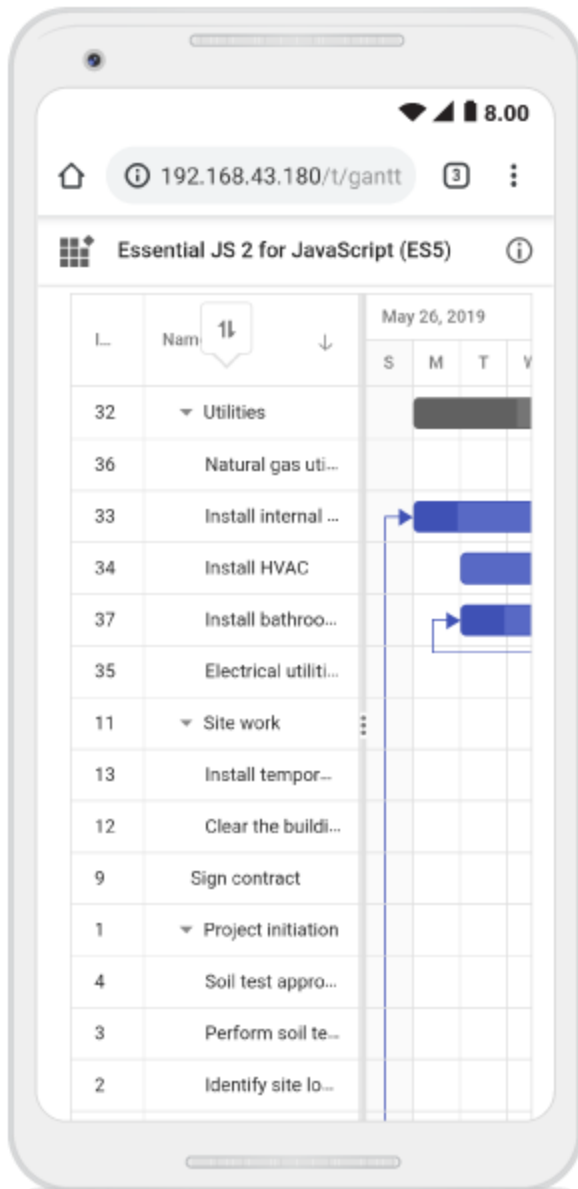
#### Context Menu

To perform **long press** action on a row, [Context Menu](#) is opened, and then tap a menu item to trigger its action.

#### Sorting

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

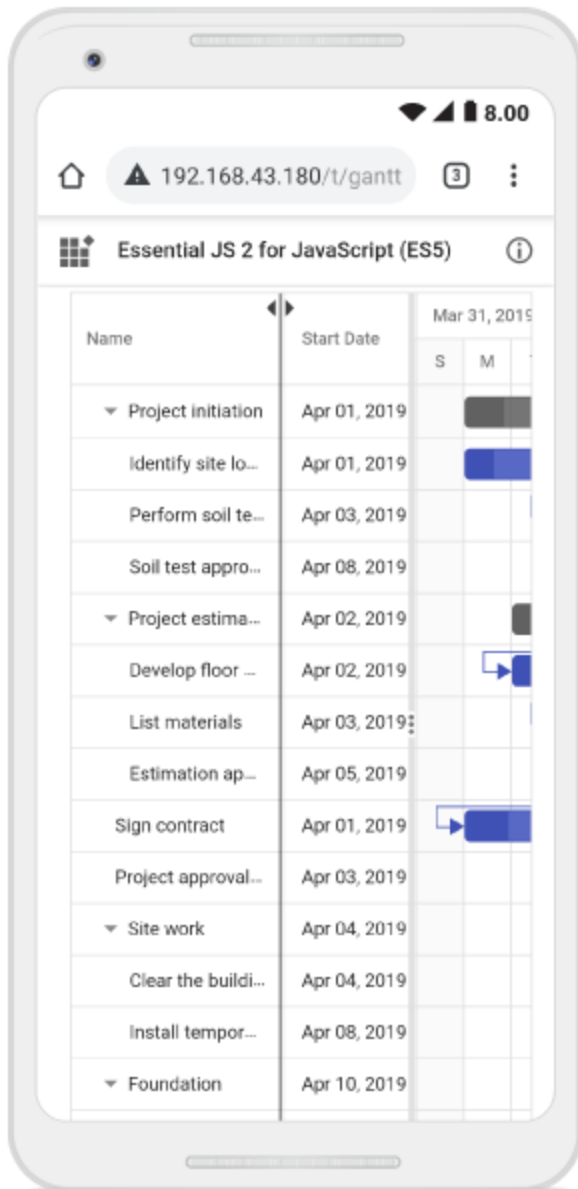
The following screenshot shows Gantt touch sorting,



### Column Resize

When the right edge of the column header cell is **tapped**, a floating handler will be visible over the right border of the column. To [Resize](#) the column, drag the floating handler as needed.

The following screenshot represents the Gantt column resizing in touch device.



### Editing

The Gantt Chart component editing actions can be achieved using the double tap and tap and drag actions on a element.

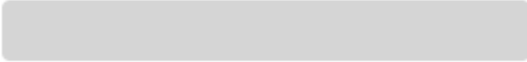
The following table describes different types of editing modes available in Gantt.

Action | Description

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



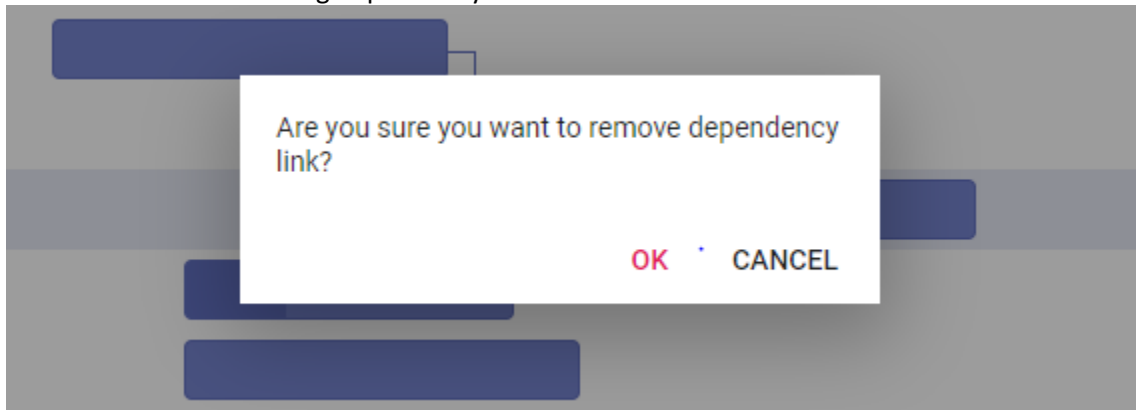
**Taskbar without dependency** | If you tap a valid child taskbar, it will create FS type dependency line between tasks, otherwise exits from task dependency edit mode. <br>



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



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



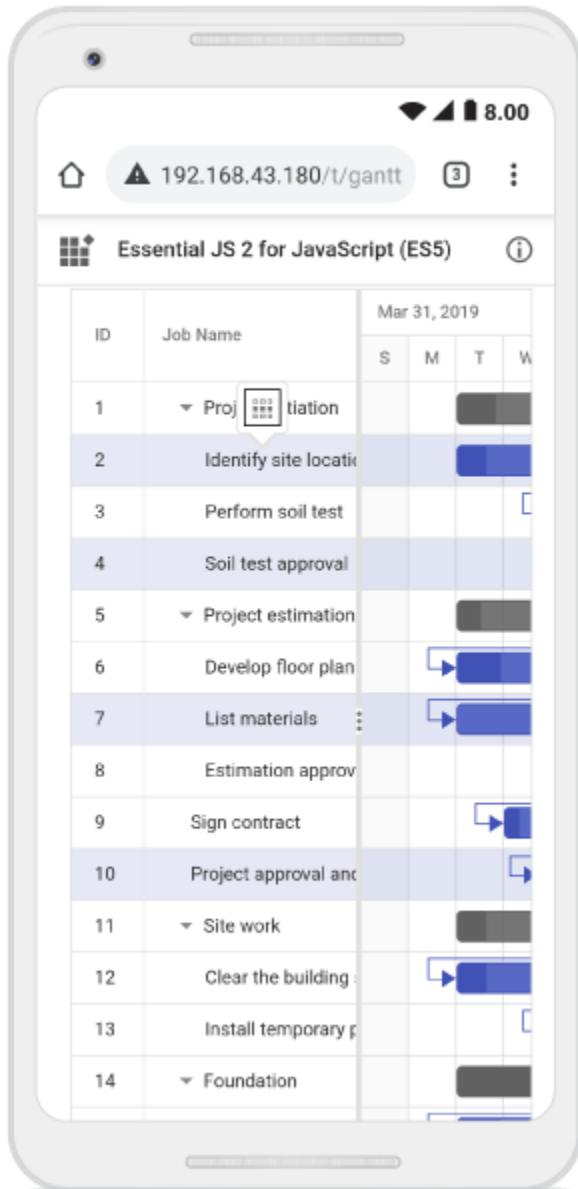
In mobile device, you cannot create dependency other than FS by taskbar editing. By using cell/dialog editing, you can add all type of dependencies. -->

### Selection

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

[Single selection](#) : To select a single row or cell, perform single tap on it.

[Multiple selection](#) : To perform multiple selection, tap on the multiple selection popup, and then tap the desired rows or cells.



You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

### Style And Appearance in Blazor Gantt Chart Component

To modify the Gantt Chart appearance, you need to override the default CSS of gantt chart. Please find the list of CSS classes and its corresponding section in Gantt Chart. Also, you have an option to create your own custom theme for all the JavaScript controls using our [Theme Studio](#).

|Section | CSS Class | Purpose of Class |

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

|**Root**|e-gantt|This class is in the root element (div) of the gantt chart control. |

**|Header|** e-gridheader| This class is added in the root element of header element. In this class, You can override thin line between header and content of the gantt chart. |

| | e-table | This class is added at 'table' of the gantt chart header. This CSS class makes table width as 100 %. |

| | e-columnheader| This class is added at 'tr' of the gantt chart header. |

**|Grid Content|** e-gridcontent| This class is added at root of body content. This is to override background color of the body.

| | e-table| This class is added to table of content. This CSS class makes table width as 100 %.

| | e=row| This class is added to rows of gantt chart.

| | e-altrow| This class is added to alternate rows of gantt chart. This is to override alternate row color of the gantt chart.

| | e-rowcell| This class is added to all cells in the gantt chart. This is to override cells appearance and styling.

**|Chart Content|** e-gantt-chart| This class is added to the chart side of the gantt chart.

| | e-chart-row| This class is added to rows of gantt chart.

**|Timeline|** e-timeline-header-container| This class is added to timeline of the gantt chart.

| | e-header-cell-label| This class is added to the header cell of the timeline.

| | e-weekend-header-cell| This class is added to the weekend cells.

**|Taskbar|** e-taskbar-main-container| This class is added to taskbar of the gantt chart.

| | e-gantt-parent-taskbar| This class is added to the parent task bar of the gantt chart.

| | e-gantt-milestone| This class is added to the milestone tasks of the gantt chart.

| | e-gantt-unscheduled-taskbar| This class is added to the unscheduled tasks.

| | e-gantt-manualparenttaskbar| This class is added to the manual scheduled parent taskbar.

| | e-gantt-child-manualtaskbar| This class is added to the manual scheduled child taskbar.

| | e-gantt-unscheduled-manualtask| This class is added to the manual unscheduled tasks.

**|Splitter|** e-split-bar| This class is added to the gantt chart splitter.

| | e-resize-handler| This class is added to the resize handler of the gantt chart splitter.

| | e-arrow-left| This class is added to the left arrow of the resize handler.

| | e-arrow=right| This class is added to the right arrow of the resize handler.

**|Connector Lines|** e-line| This class is added to the connector lines.

| | e-connector-line-right-arrow| This class is added to the right arrow of the connector line.

| | e-connector-line-left-arrow| This class is added to the left arrow of the connector line.

**|Labels|** e-task-label| This class is added to the task labels.

| | e-right-label-container| This class is added to the right label.

| | e-left-label-container| This class is added to the left label.

**|Event Markers** |e-event-markers| This class is added to the event markers.

**|Baseline** |e-baseline-bar| This class is added to the baseline.

**|** |e-baseline-gantt-milestone-container| This class is added to the baseline of milestone tasks.

**|Tooltip** |e-gantt-tooltip| This class is added to the tooltip.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="1000px"
RenderBaseline="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress"
Dependency="Predecessor" Child="SubTasks"
BaselineStartDate="BaselineStartDate"
BaselineEndDate="BaselineEndDate">
</GanttTaskFields>
<GanttLabelSettings RightLabel="TaskName"
TVValue="TaskData"></GanttLabelSettings>
<GanttEventMarkers>
<GanttEventMarker Day="@Event" Label="Project approval and kick-
off"></GanttEventMarker>
</GanttEventMarkers>
</SfGantt>
<style>
.e-split-bar, .e-headercell {
background: #add8e6 !important;
}
.e-timeline-header-container, .e-weekend-header-cell {
background: #add8e6 !important;
}
.e-gantt-parent-taskbar-inner-div {
background-color: #7ab748 !important;
}
.e-gantt-parent-progressbar-inner-div {
background-color: #4b732a !important;
}
.e-milestone-top {
border-bottom-color: #ad7a66 !important;
}
.e-milestone-bottom {
border-top-color: #ad7a66 !important;
}
.e-gantt-child-taskbar-inner-div {
background-color: #6d619b !important;
}
.e-gantt-child-progressbar-inner-div {
background-color: #4e466e !important;
}
.e-tooltip-wrap {
background: #a9e0f4 !important;
}
.e-event-marker {
border-left-color: #05088f !important;
}
.e-baseline-bar {
```

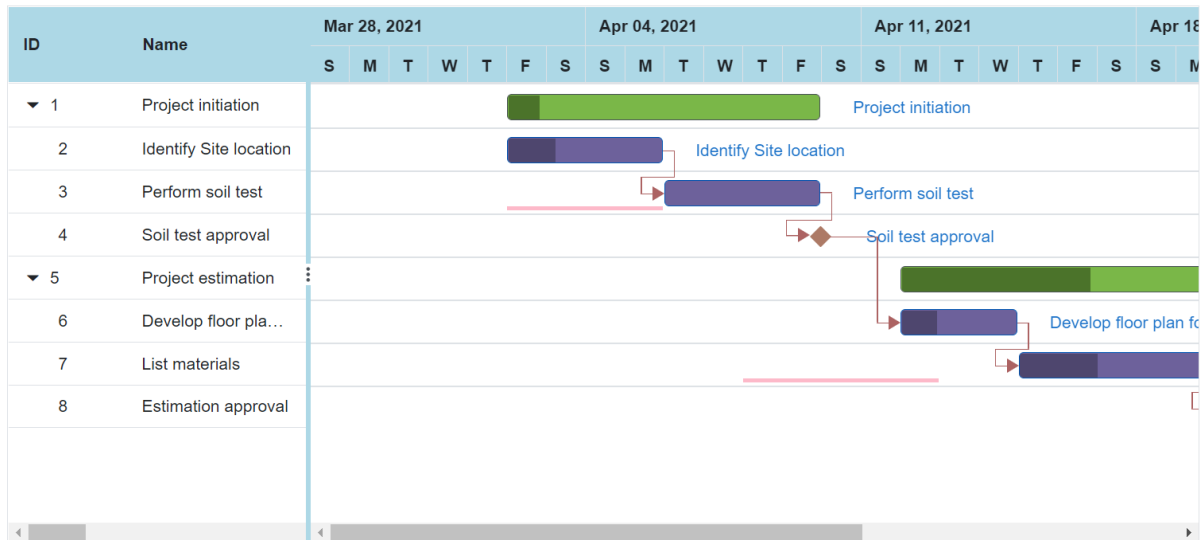


```

background-color: #fdb9c9 !important;
}
.e-label {
color: #1e74ca !important;
}
.e-line {
border-color: #ab6060fc !important;
}
.e-connector-line-right-arrow {
border-left-color: #ab6060fc !important;
}
</style>
@code{
public List<TaskData> TaskCollection { get; set; }
public DateTime Event = new DateTime(2021, 04, 27);
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public DateTime BaselineStartDate { get; set; }
public DateTime BaselineEndDate { get; set; }
public string Predecessor { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() { TaskId = 1, TaskName = "Project initiation", StartDate =
new DateTime(2021, 04, 02), EndDate = new DateTime(2021, 04, 21), SubTasks =
(new List <TaskData> () { new TaskData() { TaskId = 2, TaskName = "Identify
Site location", StartDate = new DateTime(2021, 04, 02), Duration = "2",
Progress = 30, }, new TaskData() { TaskId = 3, TaskName = "Perform soil
test", StartDate = new DateTime(2021, 04, 02), Duration = "4",
BaselineStartDate = new DateTime(2021, 04, 02), BaselineEndDate = new
DateTime(2021, 04, 05), Predecessor = "2" }, new TaskData() { TaskId = 4,
TaskName = "Soil test approval", StartDate = new DateTime(2021, 04, 02),
Duration = "0", Progress = 30, Predecessor = "3" }}}},
new TaskData() { TaskId = 5, TaskName = "Project estimation", StartDate =
new DateTime(2021, 04, 02), EndDate = new DateTime(2021, 04, 21), SubTasks =
(new List <TaskData> () { new TaskData() { TaskId = 6, TaskName = "Develop
floor plan for estimation", StartDate = new DateTime(2021, 04, 04), Duration
= "3", Progress = 30, Predecessor = "4" }, new TaskData() { TaskId = 7,
TaskName = "List materials", StartDate = new DateTime(2021, 04, 04),
Duration = "3", Progress = 40, BaselineStartDate = new DateTime(2021, 04,
08), BaselineEndDate = new DateTime(2021, 04, 12), Predecessor = "6"}, new
TaskData() { TaskId = 8, TaskName = "Estimation approval", StartDate = new
DateTime(2021, 04, 04), Duration = "5",Progress = 30,Predecessor = "7"}}}
};
return Tasks;
}

```

```
}
}
```



You can refer to our [Blazor Gantt Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Gantt Chart example](#) to know how to render and configure the gantt.

### Events in Blazor GanttChart Component

In this section, the list of events of the Gantt Chart component has been provided which will be triggered for appropriate Gantt Chart actions.

The events should be provided to the Gantt Chart using the GanttChartEvents component. When using events of the Gantt Chart, TValue must be provided in the GanttChartEvents component.

#### OnActionBegin

[OnActionBegin](#) event triggers when Gantt Chart actions such as sorting, filtering, editing etc., starts.

#### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttEditSettings AllowEditing="true"
  Mode="Syncfusion.Blazor.Gantt.EditMode.Auto"
  AllowTaskbarEditing="true"></GanttEditSettings>
  <GanttEvents TValue="TaskData" OnActionBegin="actionBegin"></GanttEvents>
</SfGantt>

@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
```

```
public void actionBegin(GanttActionEventArgs<TaskData> args)
{
    //Here you can customize your code
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
```

```

Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### OnActionComplete

[OnActionComplete](#) event triggers when Gantt Chart actions such as sorting, filtering, editing etc. are completed.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowEditing="true"
Mode="Syncfusion.Blazor.Gantt.EditMode.Auto"
AllowTaskbarEditing="true"></GanttEditSettings>
<GanttEvents TValue="TaskData"
OnActionComplete="OnActionComplete"></GanttEvents>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public void OnActionComplete(GanttActionEventArgs<TaskData> args)
{
//Here you can customize your code
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
}
}

```

```
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 40
                },
            })
        },
    }
}
```

```

TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### OnActionFailure

[OnActionFailure](#) event trigger when any Gantt Chart action failed to achieve the desired results. By using this event the error details and their cause is achieved. In the following sample, the wrong field name has been provided for the IdMapping property, so that it will throw the OnActionFailure event.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowEditing="true"
Mode="Syncfusion.Blazor.Gantt.EditMode.Auto"></GanttEditSettings>
<GanttEvents TValue="TaskData"
OnActionFailure="ActionFailureHandler"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void ActionFailureHandler(FailureEventArgs args)
{
// Here you can get the error details in the args
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {

```

```
new TaskData() {
    TaskId = 1,
    TaskName = "Project initiation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 3,
            TaskName = "Perform soil test",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "4",
            Progress = 40,
        },
        new TaskData() {
            TaskId = 4,
            TaskName = "Soil test approval",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "0",
            Progress = 30
        },
    })
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
})
```

```

    }
    };
    return Tasks;
    }
}

```

### Created

[Created](#) event triggers when the Gantt Chart component is created. The Gantt Chart properties can be modified by using this event.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEvents TValue="TaskData" Created="CreatedHandler"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void CreatedHandler(object args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,

```



```
    },
    new TaskData() {
        TaskId = 3,
        TaskName = "Perform soil test",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "4",
        Progress = 40,
    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "0",
        Progress = 30
    },
    })
    },
    new TaskData() {
        TaskId = 5,
        TaskName = "Project estimation",
        StartDate = new DateTime(2019, 04, 02),
        EndDate = new DateTime(2019, 04, 21),
        SubTasks = (new List <TaskData> () {
            new TaskData() {
                TaskId = 6,
                TaskName = "Develop floor plan for estimation",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "3",
                Progress = 30,
            },
            new TaskData() {
                TaskId = 7,
                TaskName = "List materials",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "3",
                Progress = 40
            },
            new TaskData() {
                TaskId = 8,
                TaskName = "Estimation approval",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "0",
                Progress = 30,
            }
        })
    },
    });
    return Tasks;
}
```

### OnLoad

[OnLoad](#) event triggers before the rendering process starts which allows customization of Gantt Chart properties before the Gantt Chart rendering.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEvents TValue="TaskData" OnLoad="LoadHandler"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void LoadHandler(object args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
}
}
}
}
}

```

```

Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

## Destroyed

[Destroyed](#) event triggers when the Gantt Chart component is destroyed. By using this event, confirm that the Gantt Chart gets destroyed.

## ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttEvents TValue="TaskData" Destroyed="DestroyHandler"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void DestroyHandler(object args)
{

```

```
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
```

```

TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
}))
};
return Tasks;
}
}

```

### TaskbarEdited

[TaskbarEdited](#) event triggers when taskbar was dragged and dropped on a new position.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
<GanttEvents TaskbarEdited="TaskbarEdited" TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void TaskbarEdited(TaskbarEditedEventArgs<TaskData> args)
{
// Here you can get the error details in the args
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
}
}

```

```
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 40
                },
            })
        },
    }
}
```

```

new TaskData() {
    TaskId = 8,
    TaskName = "Estimation approval",
    StartDate = new DateTime(2019, 04, 04),
    Duration = "0",
    Progress = 30,
}
}))
}
};
return Tasks;
}
}

```

### RowDropped

[RowDropped](#) event triggers when row elements are dropped on the Gantt Chart.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
AllowRowDragAndDrop="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
<GanttEvents RowDropped="RowDropped" TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void RowDropped(RowDragEventArgs<TaskData> args)
{
    // Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,

```

```
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 2,
        TaskName = "Identify Site location",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "0",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 3,
        TaskName = "Perform soil test",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "4",
        Progress = 40,
    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "0",
        Progress = 30
    },
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
}
};
```



```
return Tasks;
}
}
```

### RowDataBound

[RowDataBound](#) event triggers every time a request is made to access row information, element, or data and before the row element is appended to the Gantt Chart element.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
<GanttEvents RowDataBound="RowDataBoundHandler"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void RowDataBoundHandler(RowDataBoundEventArgs<TaskData> args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
```

```

Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### OnRowDragStart

[OnRowDragStart](#) event triggers when row elements drag starts.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
AllowRowDragAndDrop="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
<GanttEvents OnRowDragStart="OnRowDragStart"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void OnRowDragStart(RowDragEventArgs<TaskData> args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {

```

```

TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### QueryChartRowInfo

[QueryChartRowInfo](#) event triggers during the rendering of Taskbar in the Gantt Chart so that the Chart rows can be customized.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>

```

```

<GanttEvents QueryChartRowInfo ="QueryChartRowInfo"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void QueryChartRowInfo(QueryChartRowInfoEventArgs<TaskData> args)
{
    // Here you can get the error details in the args
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {

```

```

TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 6,
        TaskName = "Develop floor plan for estimation",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 7,
        TaskName = "List materials",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 40
    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "0",
        Progress = 30,
    }
})
};
return Tasks;
}
}

```

### QueryCellInfo

[QueryCellInfo](#) event triggers every time a request is made to access cell information, element, or data and before the cell element is appended to the Gantt Chart element.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
    <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
    EndDate="EndDate"
    Duration="Duration" Progress="Progress" Child="SubTasks">
    </GanttTaskFields>
    <GanttEditSettings AllowEditing="true"
    Mode="Syncfusion.Blazor.Gantt.EditMode.Auto"></GanttEditSettings>
    <GanttEvents TValue="TaskData"
    QueryCellInfo="QueryCellInfoHandler"></GanttEvents>
</SfGantt>
@code{
    public SfGantt<TaskData> Gantt;
    public void QueryCellInfoHandler(QueryCellInfoEventArgs<TaskData> args)
    {
        // Here you can customize your code
    }
}

```

```
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
```

```

StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
};
return Tasks;
}
}

```

### EndEdit

[EndEdit](#) event triggers when a task gets saved by cell edit.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
<GanttEvents EndEdit="EndEdit" TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void EndEdit(TaskbarEditedEventArgs<TaskData> args)
{
// Here you can get the error details in the args
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
}
}

```



```
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 40
                },
            })
        },
        new TaskData() {
```

```

TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### BeforeTooltipRender

[BeforeTooltipRender](#) event triggers before tooltip gets rendered.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
<GanttEvents BeforeTooltipRender = "BeforeTooltipRender"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void BeforeTooltipRender(BeforeTooltipRenderEventArgs<TaskData> args)
{
// Here you can get the error details in the args
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),

```

```
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 2,
        TaskName = "Identify Site location",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "0",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 3,
        TaskName = "Perform soil test",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "4",
        Progress = 40,
    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "0",
        Progress = 30
    },
    },
    new TaskData() {
        TaskId = 5,
        TaskName = "Project estimation",
        StartDate = new DateTime(2019, 04, 02),
        EndDate = new DateTime(2019, 04, 21),
        SubTasks = (new List <TaskData> () {
            new TaskData() {
                TaskId = 6,
                TaskName = "Develop floor plan for estimation",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "3",
                Progress = 30,
            },
            new TaskData() {
                TaskId = 7,
                TaskName = "List materials",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "3",
                Progress = 40
            },
            new TaskData() {
                TaskId = 8,
                TaskName = "Estimation approval",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "0",
                Progress = 30,
            }
        })
    },
    },
    };
return Tasks;
}
```

```
}

```

### SplitterResizeStart

[SplitterResizeStart](#) event triggers when column resize starts.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Layouts
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
AllowRowDragAndDrop="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
<GanttEvents SplitterResizeStart="Resizestart"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void Resizestart(Syncfusion.Blazor.Layouts.ResizeEventArgs args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},

```

```

new TaskData() {
    TaskId = 3,
    TaskName = "Perform soil test",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "4",
    Progress = 40,
},
new TaskData() {
    TaskId = 4,
    TaskName = "Soil test approval",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "0",
    Progress = 30
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
};
return Tasks;
}

```

### SplitterResized

[SplitterResized](#) event triggers when column resize ends.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
```

```

@using Syncfusion.Blazor.Layouts
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
AllowRowDragAndDrop="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
<GanttEvents SplitterResized="Resized" TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void Resized(Syncfusion.Blazor.Layouts.ResizingEventArgs args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",

```

```

StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
}))
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
}))
}
};
return Tasks;
}
}

```

### OnCellEdit

[OnCellEdit](#) event triggers when the cell is being edited.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowEditing="true"
Mode="Syncfusion.Blazor.Gantt.EditMode.Auto"></GanttEditSettings>
<GanttEvents TValue="TaskData" OnCellEdit="CellEditHandler"></GanttEvents>
</SfGantt>

```

```
@code{
public SfGantt<TaskData> Gantt;
public void CellEditHandler(CellEditArgs<TaskData> args)
{
    // Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
```



```

EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 6,
        TaskName = "Develop floor plan for estimation",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 7,
        TaskName = "List materials",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 40
    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "0",
        Progress = 30,
    }
})
};
return Tasks;
}
}

```

### SplitterResizing

[SplitterResizing](#) event triggers when the splitter bar is dragged.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Layouts
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
AllowRowDragAndDrop="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
<GanttEvents SplitterResizing ="Resized" TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void Resized(Syncfusion.Blazor.Layouts.ResizingEventArgs args)
{
    // Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
}

```

```
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
```

```

TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### RowSelecting

[RowSelecting](#) event triggers before row selection occurs.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSelectionSettings Mode="SelectionMode.Row"></GanttSelectionSettings>
<GanttEvents TValue="TaskData"
RowSelecting="RowSelectingHandler"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void RowSelectingHandler(RowSelectingEventArgs<TaskData> args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
}

```

```
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 40
                },
                new TaskData() {
                    TaskId = 8,
                    TaskName = "Estimation approval",
                    StartDate = new DateTime(2019, 04, 04),
```

```

Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### RowSelected

[RowSelected](#) event triggers when a row is selected.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSelectionSettings Mode="SelectionMode.Row"
Type="Syncfusion.Blazor.Grids.SelectionType.Multiple"></GanttSelectionSettin
gs>
<GanttEvents TValue="TaskData" RowSelected="rowSelect"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void rowSelect(RowSelectEventArgs<TaskData> args)
{
//Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),

```

```
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 2,
        TaskName = "Identify Site location",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "3",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 3,
        TaskName = "Perform soil test",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "4",
        Progress = 40,
    },
    new TaskData() {
        TaskId = 4,
        TaskName = "Soil test approval",
        StartDate = new DateTime(2019, 04, 02),
        Duration = "0",
        Progress = 30
    },
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
},
});
return Tasks;
}
```

## RowDeselecting

[RowDeselecting](#) event triggers before a selected row is being deselected.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate"
  Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttSelectionSettings Mode="SelectionMode.Row"></GanttSelectionSettings>
  <GanttEvents TValue="TaskData"
  RowDeselecting="RowDeselectingHandler"></GanttEvents>
</SfGantt>

@code{
public SfGantt<TaskData> Gantt;
public void RowDeselectingHandler(RowDeselectEventArgs<TaskData> args)
{
  // Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
  this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
  public int TaskId { get; set; }
  public string TaskName { get; set; }
  public DateTime StartDate { get; set; }
  public DateTime EndDate { get; set; }
  public string Duration { get; set; }
  public int Progress { get; set; }
  public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
  List<TaskData> Tasks = new List<TaskData>() {
    new TaskData() {
      TaskId = 1,
      TaskName = "Project initiation",
      StartDate = new DateTime(2019, 04, 02),
      EndDate = new DateTime(2019, 04, 21),
      SubTasks = (new List<TaskData> () {
        new TaskData() {
          TaskId = 2,
          TaskName = "Identify Site location",
          StartDate = new DateTime(2019, 04, 02),
          Duration = "0",
          Progress = 30,
        },
        new TaskData() {
          TaskId = 3,
```

```

TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### RowDeselected

[RowDeselected](#) event triggers when a selected row is deselected.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px" >

```



```

<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSelectionSettings Mode="SelectionMode.Cell"></GanttSelectionSettings>
<GanttEvents TValue="TaskData"
RowDeselected="RowDeselectHandler"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void RowDeselectHandler(RowDeselectEventArgs<TaskData> args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
}
}
}
}
}

```

```

Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### CellSelecting

[CellSelecting](#) event triggers before a cell selection occurs.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSelectionSettings
Mode="Syncfusion.Blazor.Grids.SelectionMode.Cell"></GanttSelectionSettings>
<GanttEvents CellSelecting="CellSelecting" TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public void
CellSelecting(Syncfusion.Blazor.Grids.CellSelectingEventArgs<TaskData> args)
{

```

```
//Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
    List <TaskData> Tasks = new List <TaskData> () {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
```

```

StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
};
return Tasks;
}
}

```

### CellSelected

[CellSelected](#) event triggers after a cell is selected.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
EnableContextMenu="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSelectionSettings Mode="SelectionMode.Cell"></GanttSelectionSettings>
<GanttEvents TValue="TaskData"
CellSelected="CellSelectedHandler"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void CellSelectedHandler(CellSelectEventArgs<TaskData> args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
}
}

```

```
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
```

```

Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### CellDeselecting

[CellDeselecting](#) event triggers before a cell is deselected.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
EnableContextMenu="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSelectionSettings Mode="SelectionMode.Cell"></GanttSelectionSettings>
<GanttEvents TValue="TaskData"
CellDeselecting="CellDeselectingHandler"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void CellDeselectingHandler(CellDeselectEventArgs<TaskData> args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{

```

```
List<TaskData> Tasks = new List<TaskData>() {
    new TaskData() {
        TaskId = 1,
        TaskName = "Project initiation",
        StartDate = new DateTime(2019, 04, 02),
        EndDate = new DateTime(2019, 04, 21),
        SubTasks = (new List <TaskData> () {
            new TaskData() {
                TaskId = 2,
                TaskName = "Identify Site location",
                StartDate = new DateTime(2019, 04, 02),
                Duration = "0",
                Progress = 30,
            },
            new TaskData() {
                TaskId = 3,
                TaskName = "Perform soil test",
                StartDate = new DateTime(2019, 04, 02),
                Duration = "4",
                Progress = 40,
            },
            new TaskData() {
                TaskId = 4,
                TaskName = "Soil test approval",
                StartDate = new DateTime(2019, 04, 02),
                Duration = "0",
                Progress = 30
            },
        })
    },
    new TaskData() {
        TaskId = 5,
        TaskName = "Project estimation",
        StartDate = new DateTime(2019, 04, 02),
        EndDate = new DateTime(2019, 04, 21),
        SubTasks = (new List <TaskData> () {
            new TaskData() {
                TaskId = 6,
                TaskName = "Develop floor plan for estimation",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "3",
                Progress = 30,
            },
            new TaskData() {
                TaskId = 7,
                TaskName = "List materials",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "3",
                Progress = 40
            },
            new TaskData() {
                TaskId = 8,
                TaskName = "Estimation approval",
                StartDate = new DateTime(2019, 04, 04),
                Duration = "0",
                Progress = 30,
            }
        })
    }
}
```

```

    })
  }
};
return Tasks;
}
}

```

### CellDeselected

[CellDeselected](#) event triggers after a cell is deselected.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
EnableContextMenu="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttSelectionSettings Mode="SelectionMode.Cell"></GanttSelectionSettings>
<GanttEvents TValue="TaskData"
CellDeselected="CellDeselectedHandler"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void CellDeselectedHandler(CellDeselectEventArgs<TaskData> args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,

```



```
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}
```

## OnToolbarClick

[OnToolbarClick](#) event triggers when toolbar item is clicked.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Navigations
<SfGantt DataSource="@TaskCollection" Toolbar="Toolbaritems" Height="450px"
Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEvents OnToolbarClick="ToolbarClickHandler"
TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public List<Object> Toolbaritems = new List<Object>() { "ExpandAll",
"CollapseAll", new ItemModel() { Text = "Test", TooltipText = "Test", Id =
"Test" } };
public void ToolbarClickHandler(ClickEventArgs args)
{
//Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
```

```
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}
```

### ColumnMenuClicked

[ColumnMenuClicked](#) event triggers while clicking on the column menu.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
```

```

<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
EnableContextMenu="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEvents TValue="TaskData"
ColumnMenuClicked="ColumnMenuClickedHandler" ></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void ColumnMenuClickedHandler(ColumnMenuClickEventArgs args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),

```

```

Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### ContextMenuItemClick

[ContextMenuItemClick](#) event triggers while clicking on the context menu.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
EnableContextMenu="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEvents TValue="TaskData"
ContextMenuItemClick="ContextMenuItemClickHandler" ></GanttEvents>
</SfGantt>
@code{

```

```
public SfGantt<TaskData> Gantt;
public void
ContextMenuClickedHandler(ContextMenuClickEventArgs<TaskData> args)
{
    // Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
```

```

EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 6,
        TaskName = "Develop floor plan for estimation",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 7,
        TaskName = "List materials",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 40
    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "0",
        Progress = 30,
    }
})
};
return Tasks;
}
}

```

### ContextMenuOpen

[ContextMenuOpen](#) event triggers before opening the context menu.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
EnableContextMenu="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEvents TValue="TaskData" ContextMenuOpen="ContextMenuOpenHandler"
></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void ContextMenuOpenHandler(ContextMenuOpenEventArgs<TaskData> args)
{
    // Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
}

```

```
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List <TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
```



```

TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### Collapsed

[Collapsed](#) event triggers after row get collapsed.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.TreeGrid
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
AllowRowDragAndDrop="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
<GanttEvents Collapsed="Collapsed" TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void Collapsed(RowCollapsedEventArgs<TaskData> args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
}

```

```
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {
            TaskId = 5,
            TaskName = "Project estimation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 6,
                    TaskName = "Develop floor plan for estimation",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 7,
                    TaskName = "List materials",
                    StartDate = new DateTime(2019, 04, 04),
                    Duration = "3",
                    Progress = 40
                },
                new TaskData() {
                    TaskId = 8,
                    TaskName = "Estimation approval",
                    StartDate = new DateTime(2019, 04, 04),
```

```

Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

## Collapsing

[Collapsing](#) event triggers before row get collapsed.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.TreeGrid
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px"
AllowRowDragAndDrop="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowTaskbarEditing="true"></GanttEditSettings>
<GanttEvents Collapsing="Collapsed" TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void Collapsed(RowCollapsingEventArgs<TaskData> args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {

```

```
new TaskData() {
    TaskId = 2,
    TaskName = "Identify Site location",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "0",
    Progress = 30,
},
new TaskData() {
    TaskId = 3,
    TaskName = "Perform soil test",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "4",
    Progress = 40,
},
new TaskData() {
    TaskId = 4,
    TaskName = "Soil test approval",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "0",
    Progress = 30
},
})
},
new TaskData() {
    TaskId = 5,
    TaskName = "Project estimation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 6,
            TaskName = "Develop floor plan for estimation",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 30,
        },
        new TaskData() {
            TaskId = 7,
            TaskName = "List materials",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "3",
            Progress = 40
        },
        new TaskData() {
            TaskId = 8,
            TaskName = "Estimation approval",
            StartDate = new DateTime(2019, 04, 04),
            Duration = "0",
            Progress = 30,
        }
    })
},
};
return Tasks;
}
```

## Expanding

[Expanding](#) event triggers when a row is expanding.

**ASPX-CS**

```
using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEvents Expanding="Expanding" TValue="TaskData"></GanttEvents>
</SfGantt>
@code{
public void
Expanding(Syncfusion.Blazor.TreeGrid.RowExpandingEventArgs<TaskData> args)
{
// Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
```

```

Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
Progress = 30,
}
})
}
};
return Tasks;
}
}

```

### Expanded

[Expanded](#) event triggers when a row is expanded.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.TreeGrid
<SfGantt DataSource="@TaskCollection" Height="450px" Width="900px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate"
Duration="Duration" Progress="Progress" Child="SubTasks">

```

```

</GanttTaskFields>
<GanttEvents TValue="TaskData" Expanded="ExpandedHandler" ></GanttEvents>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void ExpandedHandler(RowExpandedEventArgs<TaskData> args)
{
    // Here you can customize your code
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30,
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 40,
                },
                new TaskData() {
                    TaskId = 4,
                    TaskName = "Soil test approval",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "0",
                    Progress = 30
                },
            })
        },
        new TaskData() {

```

```

TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
    new TaskData() {
        TaskId = 6,
        TaskName = "Develop floor plan for estimation",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 30,
    },
    new TaskData() {
        TaskId = 7,
        TaskName = "List materials",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "3",
        Progress = 40
    },
    new TaskData() {
        TaskId = 8,
        TaskName = "Estimation approval",
        StartDate = new DateTime(2019, 04, 04),
        Duration = "0",
        Progress = 30,
    }
})
};
return Tasks;
}
}

```

We are not going to limit Gantt Chart with these events, we will be adding new events 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

### Hide chart part in Blazor Gantt Chart Component

In the Gantt Chart component, you can hide chart part and display Tree Grid part alone by setting the value of `GanttSplitterSettings.View` property as `Grid`.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="230px" Width="700px">
    <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
    EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
    </GanttTaskFields>
    <GanttSplitterSettings View="SplitterView.Grid"></GanttSplitterSettings>
</SfGantt>
@code{
    public List<TaskData> TaskCollection { get; set; }
    protected override void OnInitialized()
    {
        this.TaskCollection = GetTaskCollection();
    }
}

```



```
}  
public class TaskData  
{  
    public int TaskId { get; set; }  
    public string TaskName { get; set; }  
    public DateTime StartDate { get; set; }  
    public DateTime EndDate { get; set; }  
    public string Duration { get; set; }  
    public int Progress { get; set; }  
    public List<TaskData> SubTasks { get; set; }  
}  
public static List<TaskData> GetTaskCollection()  
{  
    List<TaskData> Tasks = new List<TaskData>() {  
        new TaskData() {  
            TaskId = 1,  
            TaskName = "Project initiation",  
            StartDate = new DateTime(2019, 04, 02),  
            EndDate = new DateTime(2019, 04, 21),  
            SubTasks = (new List<TaskData> () {  
                new TaskData() {  
                    TaskId = 2,  
                    TaskName = "Identify Site location",  
                    StartDate = new DateTime(2019, 04, 02),  
                    Duration = "4",  
                    Progress = 50,  
                },  
                new TaskData() {  
                    TaskId = 3,  
                    TaskName = "Perform soil test",  
                    StartDate = new DateTime(2019, 04, 02),  
                    Duration = "4",  
                    Progress = 50,  
                }  
            })  
        }  
    };  
    return Tasks;  
}
```

The following screenshot shows the output of the above code snippet.

ID	Name	Start Date	End Date	Duration
1	Project initiation	04/02/2019	04/05/2019	4 days
2	Identify Site location	04/02/2019	04/05/2019	4 days
3	Perform soil test	04/02/2019	04/05/2019	4 days

### Open Add Edit Dialog Dynamically in Blazor Gantt Chart Component

Gantt Chart add and edit dialogs can be opened dynamically by using `OpenAddDialog` and `OpenEditDialog` methods. The following code example shows how to open add and edit dialog on separate button click actions.

#### ASPX-CS

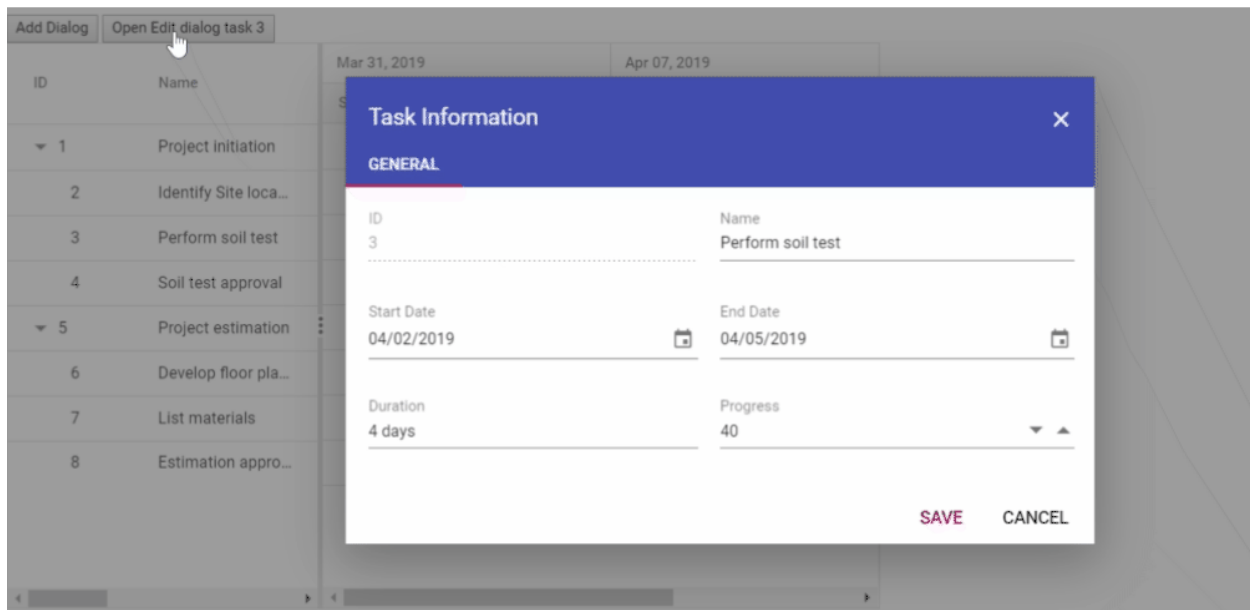
```
@using Syncfusion.Blazor.Gantt
<button @onclick="AddDialog">Add Dialog</button>
<button @onclick="EditDialog">Open Edit dialog task 3</button>
<SfGantt @ref="Gantt" DataSource="@TaskCollection" Height="450px"
Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
<GanttEditSettings AllowEditing="true"
AllowAdding="true"></GanttEditSettings>
</SfGantt>
@code{
public SfGantt<TaskData> Gantt;
public void AddDialog()
{
this.Gantt.OpenAddDialogAsync();
}
public void EditDialog()
{
this.Gantt.OpenEditDialogAsync(3);
}
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
```

```
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 40,
},
new TaskData() {
TaskId = 4,
TaskName = "Soil test approval",
StartDate = new DateTime(2019, 04, 02),
Duration = "0",
Progress = 30
},
})
},
new TaskData() {
TaskId = 5,
TaskName = "Project estimation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 6,
TaskName = "Develop floor plan for estimation",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 30,
},
new TaskData() {
TaskId = 7,
TaskName = "List materials",
StartDate = new DateTime(2019, 04, 04),
Duration = "3",
Progress = 40
},
new TaskData() {
TaskId = 8,
TaskName = "Estimation approval",
StartDate = new DateTime(2019, 04, 04),
Duration = "0",
```

```

Progress = 30,
}
})
}
};
return Tasks;
}
}

```



### WebAssembly Gantt Chart in Blazor Gantt Chart Component

This article provides a step-by-step instructions to configure Syncfusion [Blazor Gantt Chart](#) 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.

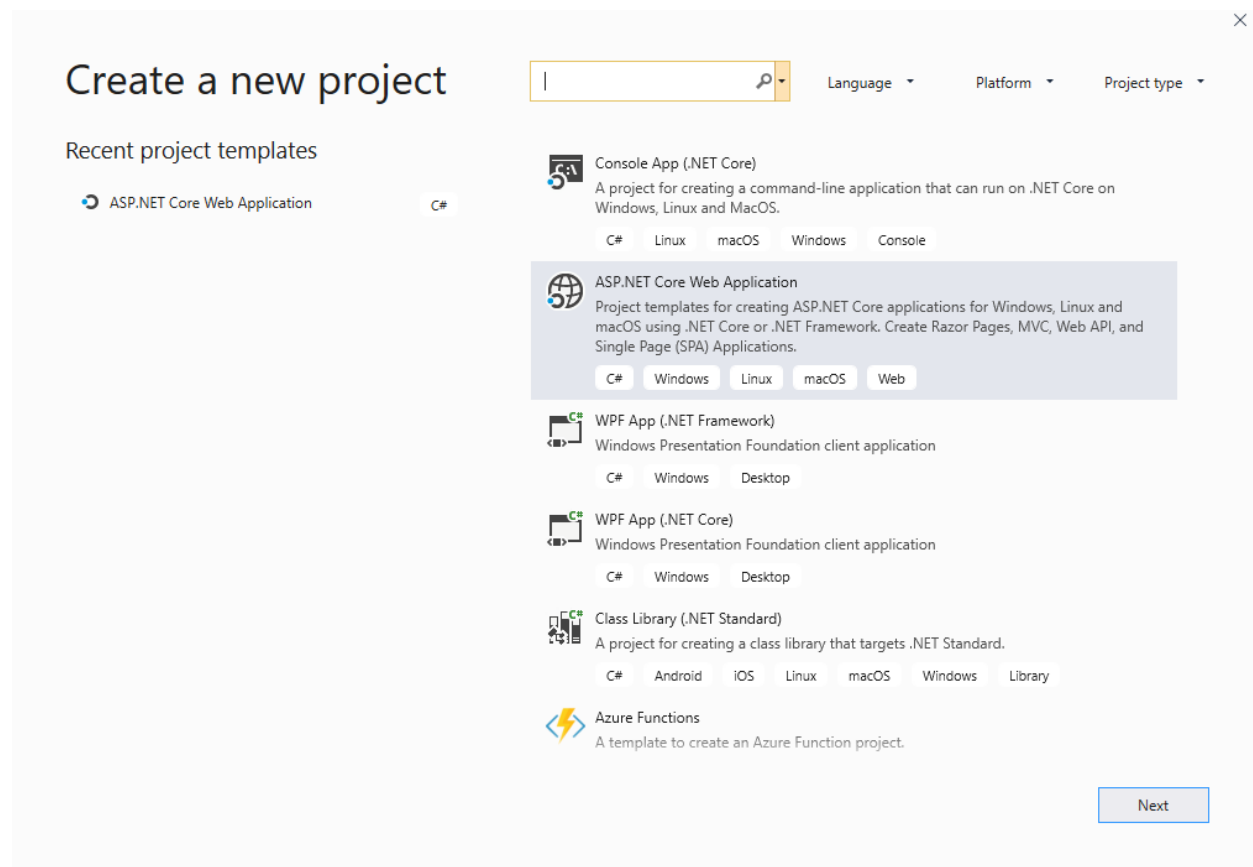
#### Create a Blazor WebAssembly project in Visual Studio 2019

1. Install the essential project templates in the Visual Studio 2019 by running the below command line in the command prompt.

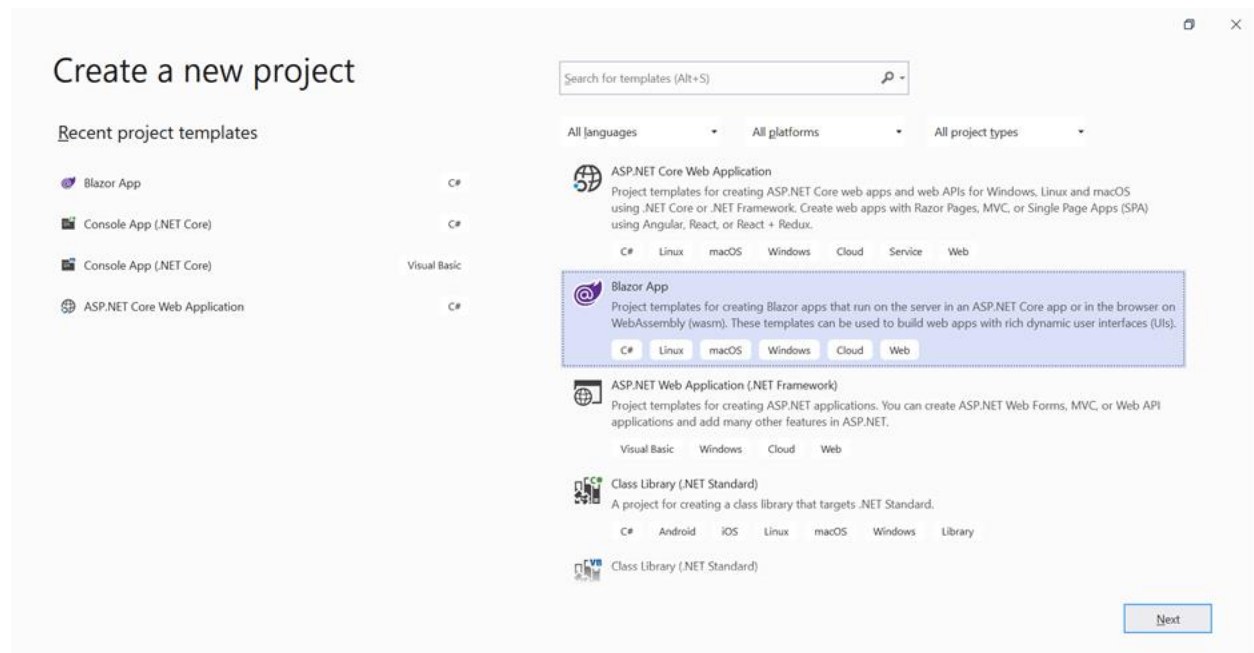
#### BASH

```
dotnet new -i Microsoft.AspNetCore.Components.WebAssembly.Templates::3.2.0-rc1.20223.4
```

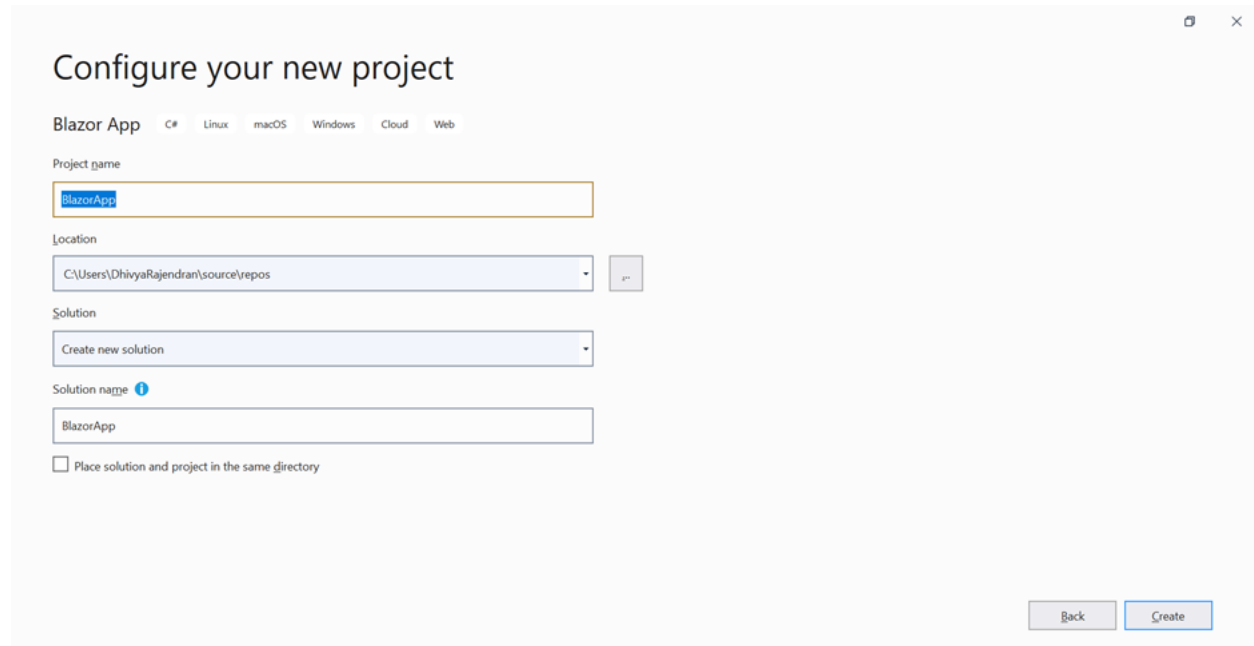
2. Choose **Create a new project** from the Visual Studio dashboard.



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




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




- Choose **Blazor WebAssembly App** from the dashboard, and then click **Create** button to create a new Blazor WebAssembly application.

## 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

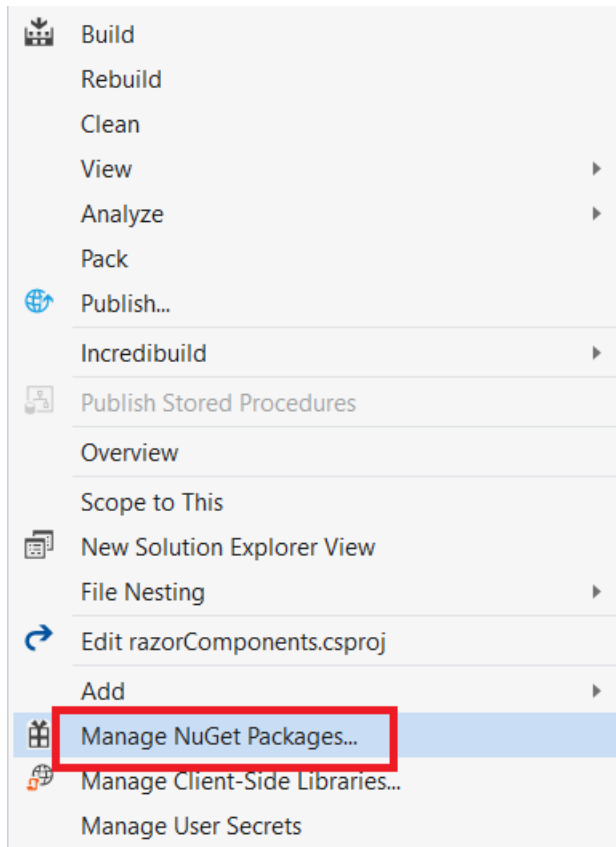
---

ASP.NET Core 3.1 available in Visual Studio 2019 version.

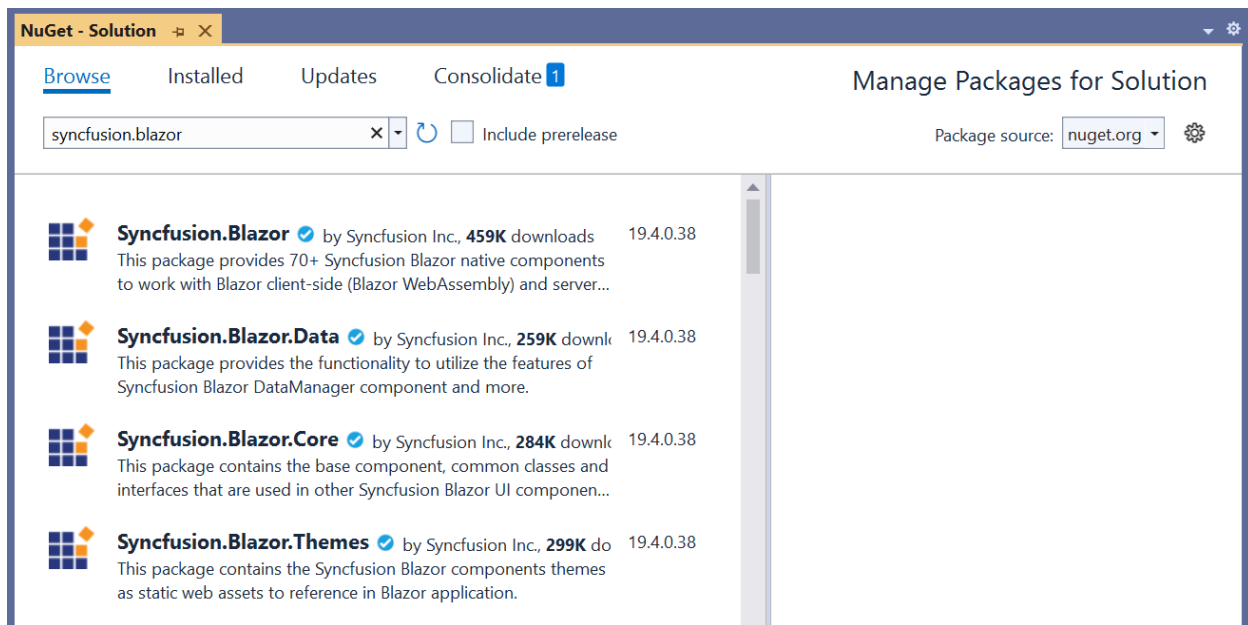
---

### *Importing Syncfusion Blazor component in the application*

1. 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.Gantt` packages.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
```

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();  
}  
}  
}
```

### *Adding Gantt Chart component to the application*

Now, add the Syncfusion [Blazor Gantt Chart component](#) in any web page (razor) in the **Pages** folder. For example, the Gantt Chart component is added in the **~/Pages/Index.razor** page.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Gantt  
<SfGantt TValue="TaskData">  
</SfGantt>  
@code{  
public class TaskData  
{  
public int TaskId { get; set; }  
public string TaskName { get; set; }  
public DateTime StartDate { get; set; }  
public DateTime EndDate { get; set; }  
public string Duration { get; set; }  
public int Progress { get; set; }  
public List<TaskData> SubTasks { get; set; }  
}  
}
```

### *Binding Gantt Chart with Data*

Bind data with the Gantt Chart component by using the **DataSource** property. It accepts an list objects or the DataManager instance.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Gantt  
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">  
</SfGantt>  
@code{  
public List<TaskData> TaskCollection { get; set; }  
protected override void OnInitialized()  
{  
this.TaskCollection = GetTaskCollection();  
}  
public class TaskData  
{  
public int TaskId { get; set; }  
public string TaskName { get; set; }  
public DateTime StartDate { get; set; }  
public DateTime EndDate { get; set; }  
public string Duration { get; set; }  
public int Progress { get; set; }  
public List<TaskData> SubTasks { get; set; }  
}  
public static List <TaskData> GetTaskCollection() {  
List <TaskData> Tasks = new List <TaskData> () {
```

```

new TaskData() {
    TaskId = 1,
    TaskName = "Project initiation",
    StartDate = new DateTime(2019, 04, 02),
    EndDate = new DateTime(2019, 04, 21),
    SubTasks = (new List <TaskData> () {
        new TaskData() {
            TaskId = 2,
            TaskName = "Identify Site location",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "4",
            Progress = 50,
        },
        new TaskData() {
            TaskId = 3,
            TaskName = "Perform soil test",
            StartDate = new DateTime(2019, 04, 02),
            Duration = "4",
            Progress = 50,
        }
    })
};
return Tasks;
}
}

```

### Mapping Task Fields

The data source fields that are required to render the tasks are mapped to the Gantt Chart component using the `GanttTaskFields` property.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
    <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
    EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
    </GanttTaskFields>
</SfGantt>
@code{
    public List<TaskData> TaskCollection { get; set; }
    protected override void OnInitialized()
    {
        this.TaskCollection = GetTaskCollection();
    }
    public class TaskData
    {
        public int TaskId { get; set; }
        public string TaskName { get; set; }
        public DateTime StartDate { get; set; }
        public DateTime EndDate { get; set; }
        public string Duration { get; set; }
        public int Progress { get; set; }
        public List<TaskData> SubTasks { get; set; }
    }
    public static List <TaskData> GetTaskCollection() {

```

```

List <TaskData> Tasks = new List <TaskData> () {
    new TaskData() {
        TaskId = 1,
        TaskName = "Project initiation",
        StartDate = new DateTime(2019, 04, 02),
        EndDate = new DateTime(2019, 04, 21),
        SubTasks = (new List <TaskData> () {
            new TaskData() {
                TaskId = 2,
                TaskName = "Identify Site location",
                StartDate = new DateTime(2019, 04, 02),
                Duration = "4",
                Progress = 50,
            },
            new TaskData() {
                TaskId = 3,
                TaskName = "Perform soil test",
                StartDate = new DateTime(2019, 04, 02),
                Duration = "4",
                Progress = 50,
            }
        })
    }
};
return Tasks;
}

```

### Defining Columns

Gantt Chart has an option to define columns as an array. You can customize the Gantt Chart columns using the following properties:

- **Field**: Maps the data source fields to the columns.
- **HeaderText**: Changes the title of columns.
- **TextAlign**: Changes the alignment of columns. By default, columns will be left aligned. To change the columns to right align, set **TextAlign** to right.
- **Format**: Formats the number and date values to standard or custom formats. Here, it is defined for the conversion of numeric values to currency.

### ASPX-CS

```

@using Syncfusion.Blazor.Gantt
@using Syncfusion.Blazor.Grids
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
    <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
    EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
    </GanttTaskFields>
    <GanttColumns>
        <GanttColumn Field="TaskId" HeaderText="Task ID" TextAlign="TextAlign.Right"
        Width="100"></GanttColumn>
        <GanttColumn Field="TaskName" HeaderText="Task Name"
        Width="250"></GanttColumn>
    </GanttColumns>
</SfGantt>

```

```

<GanttColumn Field="StartDate" HeaderText="Start Date"
Width="250"></GanttColumn>
<GanttColumn Field="Duration" HeaderText="Duration"
Width="250"></GanttColumn>
<GanttColumn Field="Progress" HeaderText="Progress" Format="@NumberFormat"
Width="250"></GanttColumn>
</GanttColumns>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
public string NumberFormat = "C";
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50,
}
})
}
};
return Tasks;
}
}

```

For further details regarding Columns, Please refer [here](#)

### Enable Editing

The editing feature enables you to edit the tasks in the Gantt Chart component. It can be enabled by using the `EditSettings.AllowEditing` and `EditSettings.AllowTaskbarEditing` properties.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
  <GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
  EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
  </GanttTaskFields>
  <GanttEditSettings AllowEditing="true"
  Mode="Syncfusion.Blazor.Gantt.EditMode.Auto"
  AllowTaskbarEditing="true"></GanttEditSettings>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List <TaskData> GetTaskCollection() {
List <TaskData> Tasks = new List <TaskData> () {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List <TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50,
}
})
}
};
return Tasks;
}
```

```
}
}
```

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

You can find the full information regarding Editing from [here](#)

### Enable Filtering

The filtering feature enables you to view the reduced amount of records based on filter criteria. Gantt Chart provides the menu filtering support for each column. It can be enabled by setting the **AllowFiltering** property to **true**. Filtering feature can also be customized using the **FilterSettings** property.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowFiltering="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
public int Progress { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
List<TaskData> Tasks = new List<TaskData>() {
new TaskData() {
TaskId = 1,
TaskName = "Project initiation",
StartDate = new DateTime(2019, 04, 02),
EndDate = new DateTime(2019, 04, 21),
SubTasks = (new List<TaskData> () {
new TaskData() {
TaskId = 2,
TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50,
},
},
},
}
```

```
new TaskData() {
    TaskId = 3,
    TaskName = "Perform soil test",
    StartDate = new DateTime(2019, 04, 02),
    Duration = "4",
    Progress = 50,
}
}))
}
};
return Tasks;
}
}
```

You can find the full information regarding Filtering from [here](#)

### Enable Sorting

The sorting feature enables you to order the records. It can be enabled by setting the `AllowSorting` property to `true`. The sorting feature can be customized using the `SortSettings` property.

### ASPX-CS

```
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px"
AllowSorting="true">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
    this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
    public int TaskId { get; set; }
    public string TaskName { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public string Duration { get; set; }
    public int Progress { get; set; }
    public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
```



```

TaskName = "Identify Site location",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50,
},
new TaskData() {
TaskId = 3,
TaskName = "Perform soil test",
StartDate = new DateTime(2019, 04, 02),
Duration = "4",
Progress = 50,
}
})
}
};
return Tasks;
}
}

```

You can find the full information regarding Sorting from [here](#)

#### *Enabling Predecessors or Task Relationships*

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

- Start to Start (SS): You cannot start a task until the dependent task starts.
- Start to Finish (SF): You cannot finish a task until the dependent task finishes.
- Finish to Start (FS): You cannot start a task until the dependent task completes.
- Finish to Finish (FF): You cannot finish a task until the dependent task completes.

You can show the relationship in tasks by using the **Dependency** property as shown in the following code example.

#### **ASPX-CS**

```

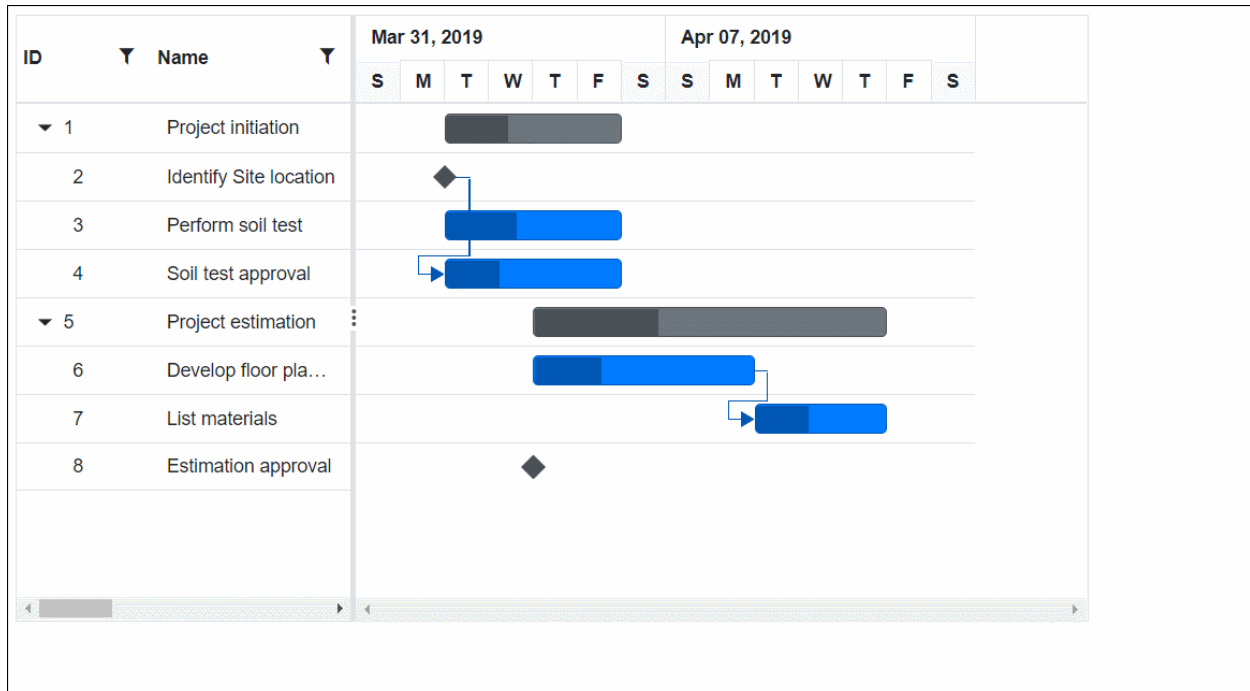
@using Syncfusion.Blazor.Gantt
<SfGantt DataSource="@TaskCollection" Height="450px" Width="700px">
<GanttTaskFields Id="TaskId" Name="TaskName" StartDate="StartDate"
EndDate="EndDate" Duration="Duration" Progress="Progress" Child="SubTasks"
Dependency="Predecessor">
</GanttTaskFields>
</SfGantt>
@code{
public List<TaskData> TaskCollection { get; set; }
protected override void OnInitialized()
{
this.TaskCollection = GetTaskCollection();
}
public class TaskData
{
public int TaskId { get; set; }
public string TaskName { get; set; }
public DateTime StartDate { get; set; }
public DateTime EndDate { get; set; }
public string Duration { get; set; }
}
}

```

```
public int Progress { get; set; }
public string Predecessor { get; set; }
public List<TaskData> SubTasks { get; set; }
}
public static List<TaskData> GetTaskCollection()
{
    List<TaskData> Tasks = new List<TaskData>() {
        new TaskData() {
            TaskId = 1,
            TaskName = "Project initiation",
            StartDate = new DateTime(2019, 04, 02),
            EndDate = new DateTime(2019, 04, 21),
            SubTasks = (new List<TaskData> () {
                new TaskData() {
                    TaskId = 2,
                    TaskName = "Identify Site location",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 50
                },
                new TaskData() {
                    TaskId = 3,
                    TaskName = "Perform soil test",
                    StartDate = new DateTime(2019, 04, 02),
                    Duration = "4",
                    Progress = 50,
                    Predecessor = "2"
                }
            })
        }
    };
    return Tasks;
}
```

You can find the full information regarding Predecessors from [here](#)

The following image represents Gantt with Editing, Sorting, Filtering and Predecessors.



## HeatMap Chart

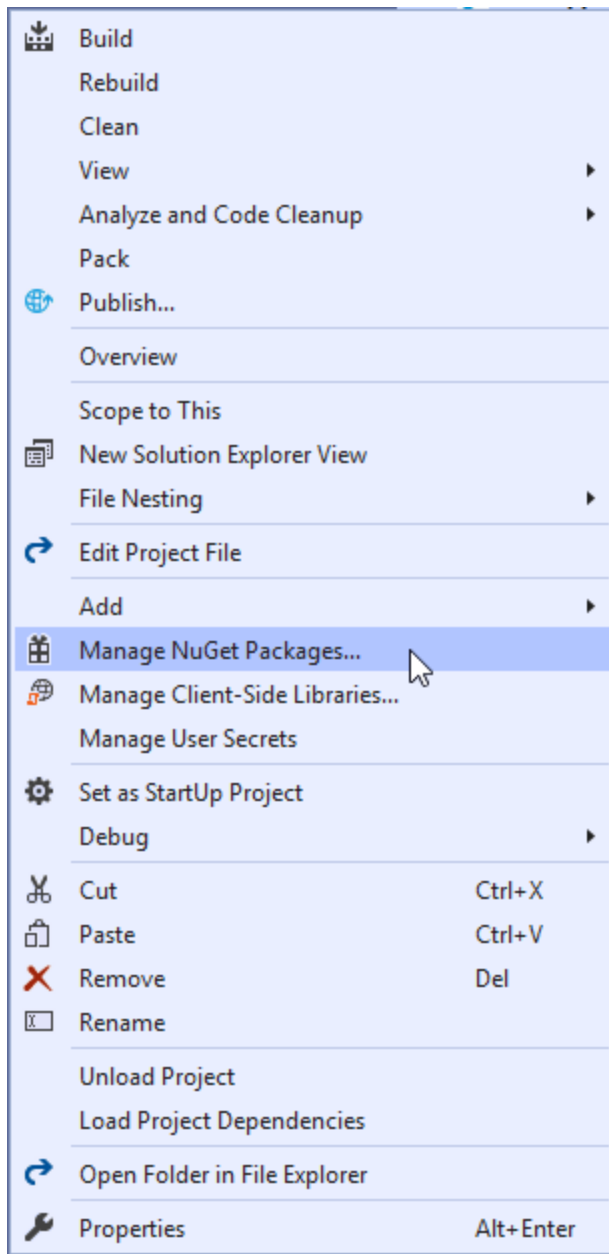
### Getting Started with Blazor HeatMap Chart Component

This section briefly explains how to include a **HeatMap Chart** 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.

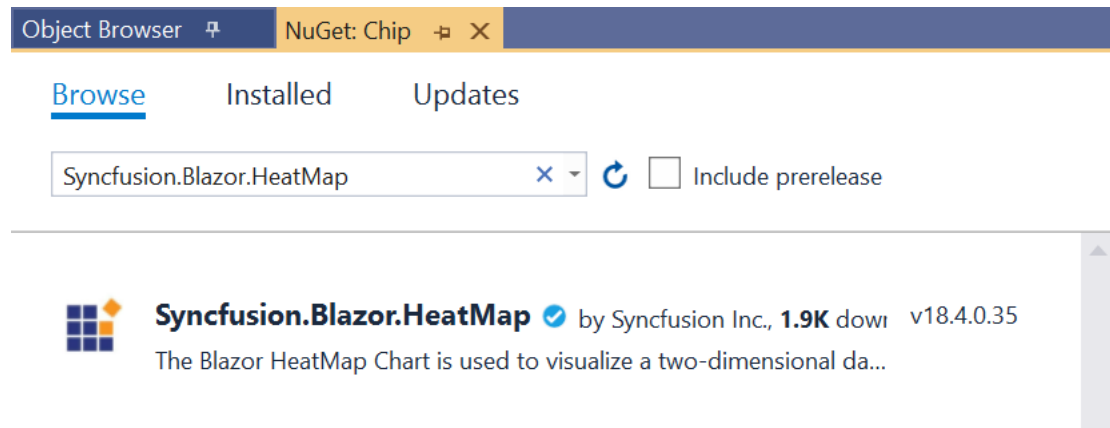
Importing Syncfusion Blazor component in the application

*Using Syncfusion.Blazor NuGet Package [New standard]*

1. Install **Syncfusion.Blazor.HeatMap** 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.HeatMap keyword in the Browse tab and install Syncfusion.Blazor.HeatMap NuGet package in the application.



3. Once the installation process is completed, the Syncfusion Blazor HeatMap 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 element of the `~/wwwroot/index.html` page in Blazor WebAssembly app or `~/Pages/_Host.cshtml` page in Blazor Server app.

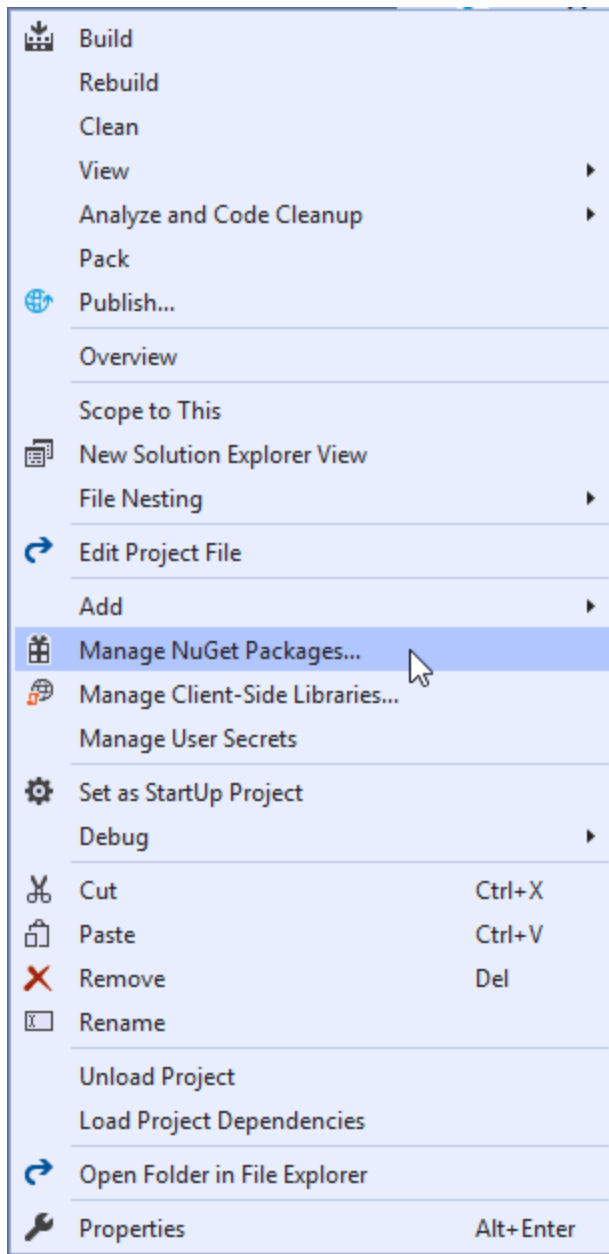
### HTML

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

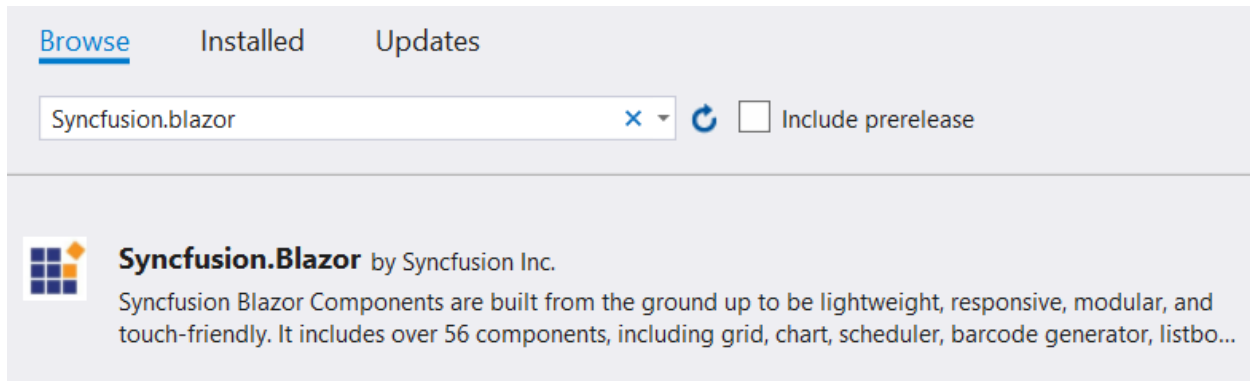
**Warning:** If you prefer the above new standard (individual NuGet packages), then skip this section. Using both old and new standards in the same application will throw ambiguous compilation errors.

### *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.

### 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>
```

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.

#### ASPX-CS

```
<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.HeatMap` package.

#### ASPX-CS

```
@using Syncfusion.Blazor  
@using Syncfusion.Blazor.HeatMap
```



### Adding HeatMap component to the application

Now, add the Syncfusion Blazor HeatMap component in any web page razor in the Pages folder. For example, the HeatMap component is added in the ~/Pages/Index.razor page.

#### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000
US$)"></HeatMapTitleSettings>
<HeatMapCellSettings ShowLabel="true"
TileType="CellType.Rect"></HeatMapCellSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{52, 65, 67, 45, 37, 52},
{68, 52, 63, 51, 30, 51},
{7, 16, 47, 47, 88, 6},
{66, 64, 46, 40, 47, 41},
{14, 46, 97, 69, 69, 3},
{54, 46, 61, 46, 40, 39}
};
return dataSource;
}
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}
```



Run the application

After successful compilation of your application, simply press **F5** to run the application.

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](#)

[Working with data in Blazor HeatMap Chart Component](#)

Heat map visualizes the JSON data and two-dimensional array data. Using the data adaptor support, data can be bound to the heat map.

Data adaptor

Heat map supports the following types of data binding with the adaptor support.

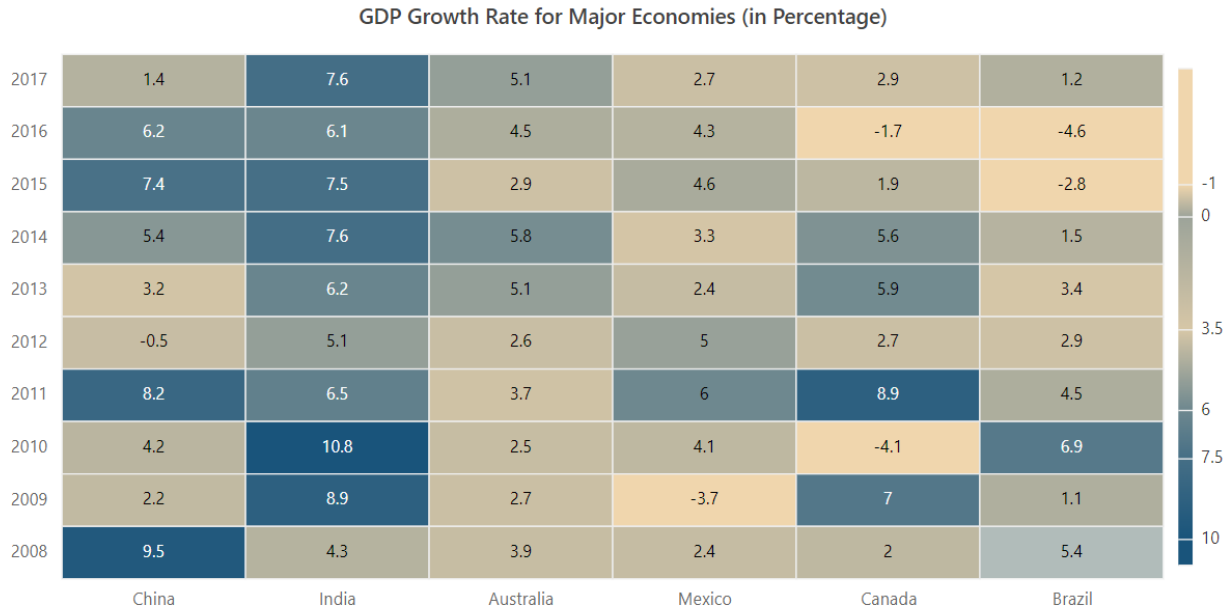
- Array - Table Binding

*Array - Table binding*

This data type is a collection of one dimensional array objects, at which each inner array contains data points for an X-axis data label. This is the default data binding type for heat map. You can also directly bind the array object to the `DataSource` property.

**ASPX-CS**

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
<HeatMapTitleSettings Text="GDP Growth Rate for Major Economies (in
Percentage)">
</HeatMapTitleSettings>
<HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
<HeatMapPaletteSettings Type="PaletteType.Gradient">
<HeatMapPalettes>
<HeatMapPalette Color="#F0D6AD" Value=-1></HeatMapPalette>
<HeatMapPalette Color="#9da49a" Value=0></HeatMapPalette>
<HeatMapPalette Color="#d7c7a7" Value=3.5></HeatMapPalette>
<HeatMapPalette Color="#6e888f" Value=6.0></HeatMapPalette>
<HeatMapPalette Color="#466f86" Value=7.5></HeatMapPalette>
<HeatMapPalette Color="#19547B" Value=10></HeatMapPalette>
</HeatMapPalettes>
</HeatMapPaletteSettings>
</SfHeatMap>
@code{
double[,] GetDefaultData()
{
double[,] dataSource = new double[6, 10]
{
{9.5, 2.2, 4.2, 8.2, -0.5, 3.2, 5.4, 7.4, 6.2, 1.4 },
{4.3, 8.9, 10.8, 6.5, 5.1, 6.2, 7.6, 7.5, 6.1, 7.6},
{3.9, 2.7, 2.5, 3.7, 2.6, 5.1, 5.8, 2.9, 4.5, 5.1},
{2.4, -3.7, 4.1, 6.0, 5.0, 2.4, 3.3, 4.6, 4.3, 2.7},
{2.0, 7.0, -4.1, 8.9, 2.7, 5.9, 5.6, 1.9, -1.7, 2.9},
{5.4, 1.1, 6.9, 4.5, 2.9, 3.4, 1.5, -2.8, -4.6, 1.2}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "China", "India", "Australia",
"Mexico", "Canada", "Brazil" };
string[] YAxisLabels = new string[] { "2008", "2009", "2010", "2011",
"2012", "2013", "2014", "2015", "2016", "2017" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}
```



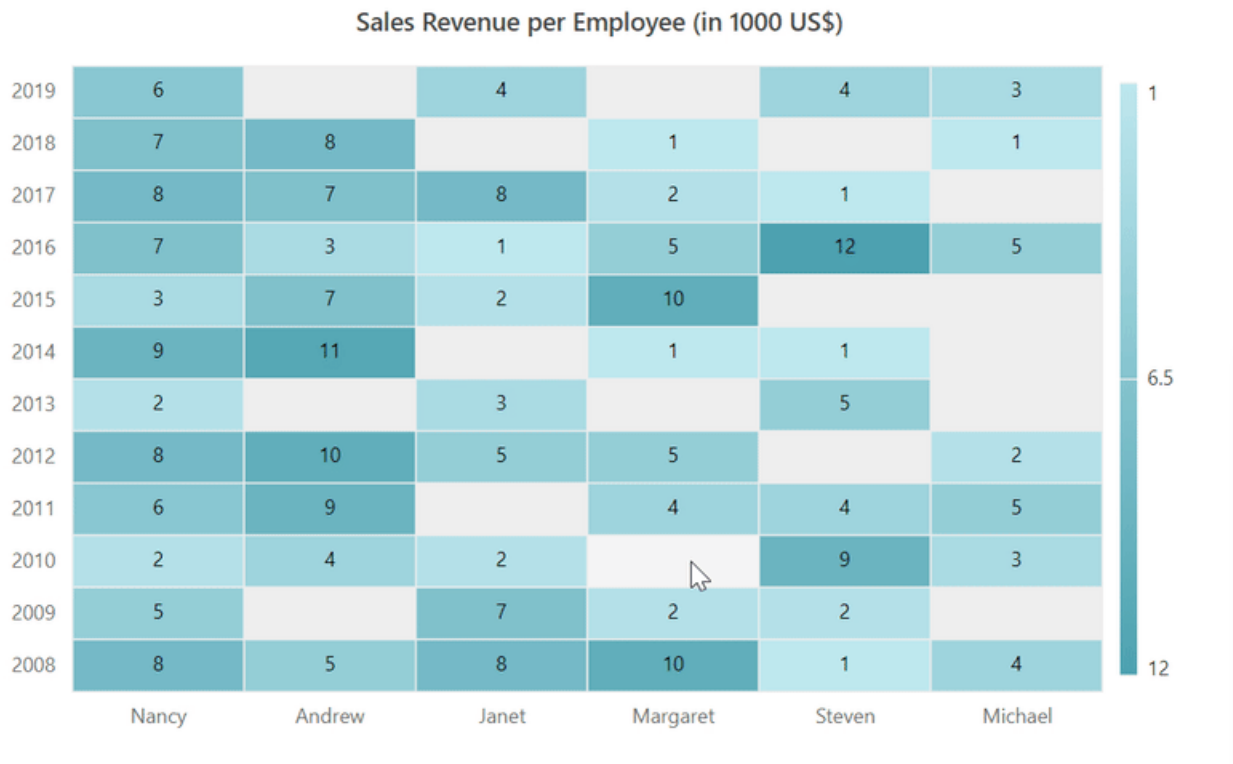
### Empty points

The data points that use the `null` or `""` or `undefined` as value are considered as empty points. Empty data points are ignored and not displayed in the heat map, and these points are rendered with default palette. You can customize the empty data point color value using the `EmptyPointColor` property.

### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
</HeatMapTitleSettings>
<HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
</SfHeatMap>
@code{
int?[,] GetDefaultData()
{
int?[,] dataSource = new int?[6, 12]
{
{8, 5, 2, 6, 8, 2, 9, 3, 7, 8, 7, 6},
{5, null, 4, 9, 10, null, 11, 7, 3, 7, 8, null},
{8, 7, 2, null, 5, 3, null, 2, 1, 8, null, 4},
{10, 2, null, 4, 5, null, 1, 10, 5, 2, 1, null},
{1, 2, 9, 4, null, 5, 1, null, 12, 1, null, 4},
{4, null, 3, 5, 2, null, null, null, 5, null, 1, 3},
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael", "Robert", "Laura", "Anne", "Paul", "Karin", "Mario" };
string[] YAxisLabels = new string[] { "2008", "2009", "2010", "2011",
"2012", "2013", "2014", "2015", "2016", "2017", "2018", "2019" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
}
```

```
HeatMapData = GetDefaultData();
}
}
```



### Bubble heat map in Blazor HeatMap Chart Component

Data points represent the data source values with **Gradient** or **Fixed** colors in the heat map. You can customize the appearance of these data points by changing the **Color** and **Shape** attributes. The data points can be represented in color fill or bubble shape by defining the **TileType** property. By default, the data points are color filled with **Gradient** or **Fixed** and this depiction of data points is defined as **Rect** in the **TileType** property.

The cell customizations and color mapping for **Rect** tile type is defined in **Appearance** and **Palette** sections in detail.

#### Bubble attributes

The data points can be represented in the bubble along with its attributes by setting the **TileType** property to **Bubble**.

In bubble heat map, you can display the data points with bubble size, bubble colors, and sector attributes of the bubble.

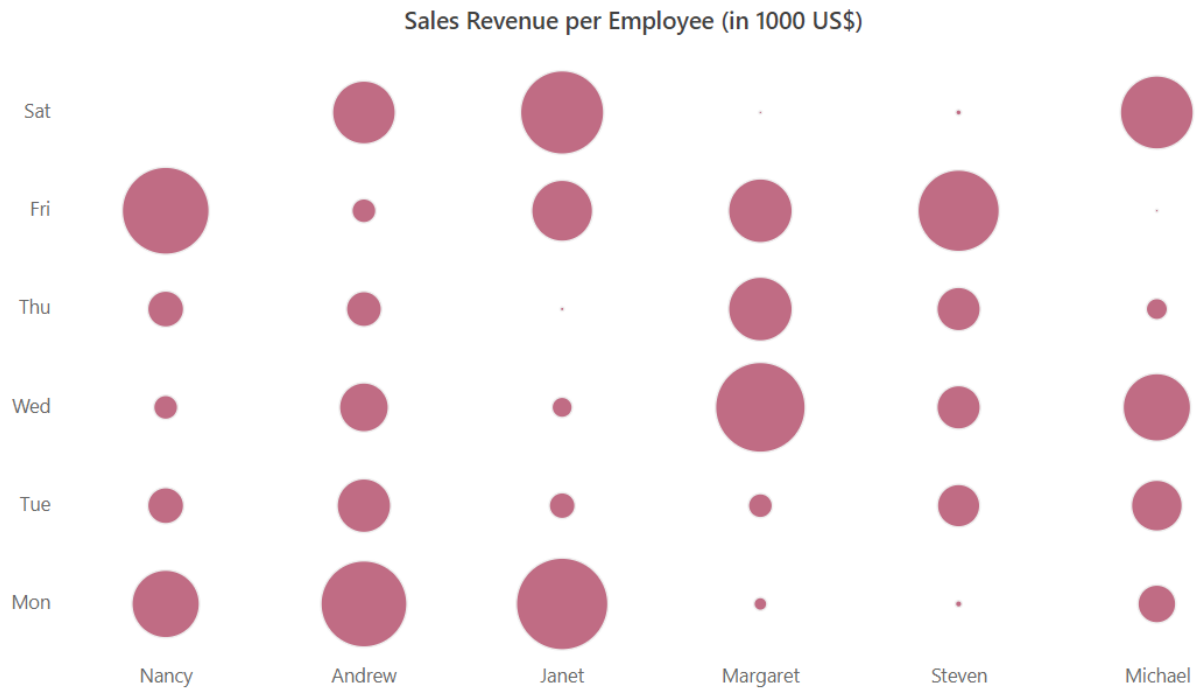
#### Bubble size

In this bubble heat map type, the size factor of the bubble is used to denote the data variations. The radius of the bubble varies according to data values. By default, the bubble with small size denotes the data value with small magnitude and the larger bubble size denotes the data value with larger magnitude. This behavior can be inverted by using the **IsInversedBubbleSize** property.

To render a bubble heat map with size series, set the `BubbleType` property to `Size`.

#### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
</HeatMapTitleSettings>
<HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
<HeatMapCellSettings ShowLabel="false" TileType="CellType.Bubble"
BubbleType="BubbleType.Size"></HeatMapCellSettings>
<HeatMapPaletteSettings Type="PaletteType.Gradient">
<HeatMapPalettes>
<HeatMapPalette Color="#C06C84"></HeatMapPalette>
<HeatMapPalette Color="#6C5B7B"></HeatMapPalette>
<HeatMapPalette Color="#355C7D"></HeatMapPalette>
</HeatMapPalettes>
</HeatMapPaletteSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]{
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael", "Robert", "Laura", "Anne", "Paul", "Karin", "Mario" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}
```



#### Bubble color

In heat map, defined with this tile type, the data points will be represented with same sized bubbles and the data value variations are represented with the bubble colors.

To represent the data points with variations in bubble colors, set the `BubbleType` property to `Color`.

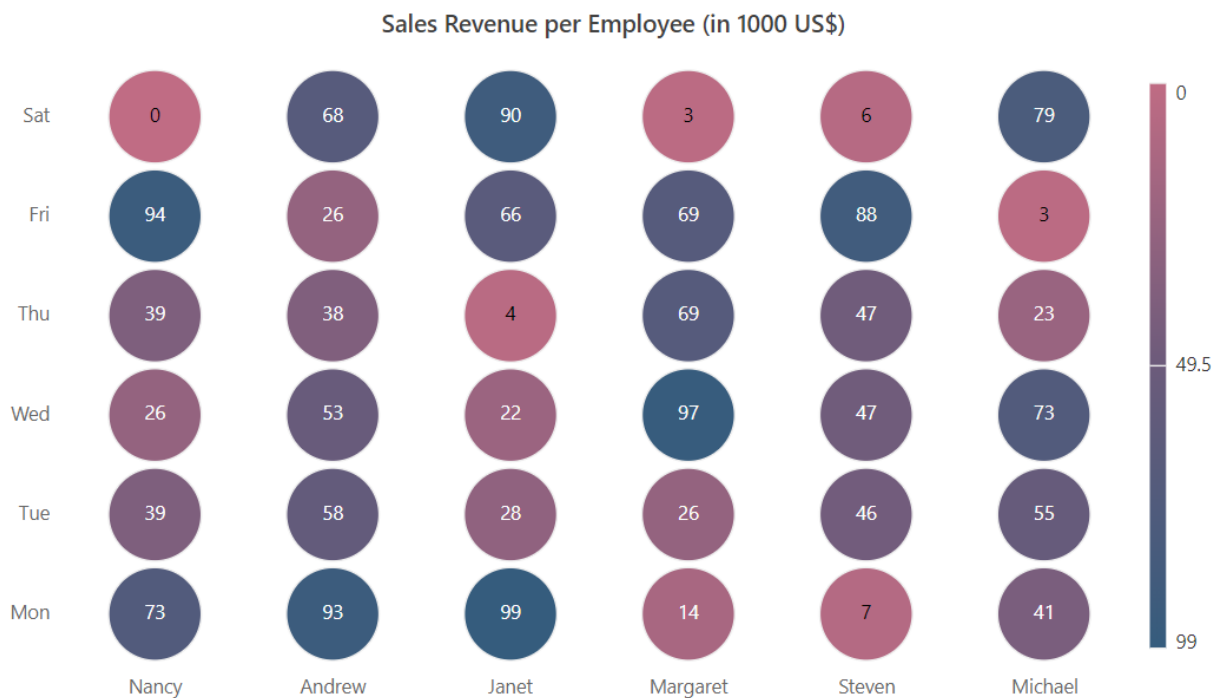
#### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapCellSettings ShowLabel="true" TileType="CellType.Bubble"
  BubbleType="BubbleType.Color"></HeatMapCellSettings>
  <HeatMapPaletteSettings Type="PaletteType.Gradient">
  <HeatMapPalettes>
    <HeatMapPalette Color="#C06C84"></HeatMapPalette>
    <HeatMapPalette Color="#6C5B7B"></HeatMapPalette>
    <HeatMapPalette Color="#355C7D"></HeatMapPalette>
  </HeatMapPalettes>
  </HeatMapPaletteSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
}
```

```

{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



### Bubble sector

In this bubble heat map type, the sector of the bubble decides the magnitude of data point. If the sector is large, then the data point value will be high.

To render the data points with bubble sector, set the **BubbleType** property to **Sector**.

### ASPX-CS

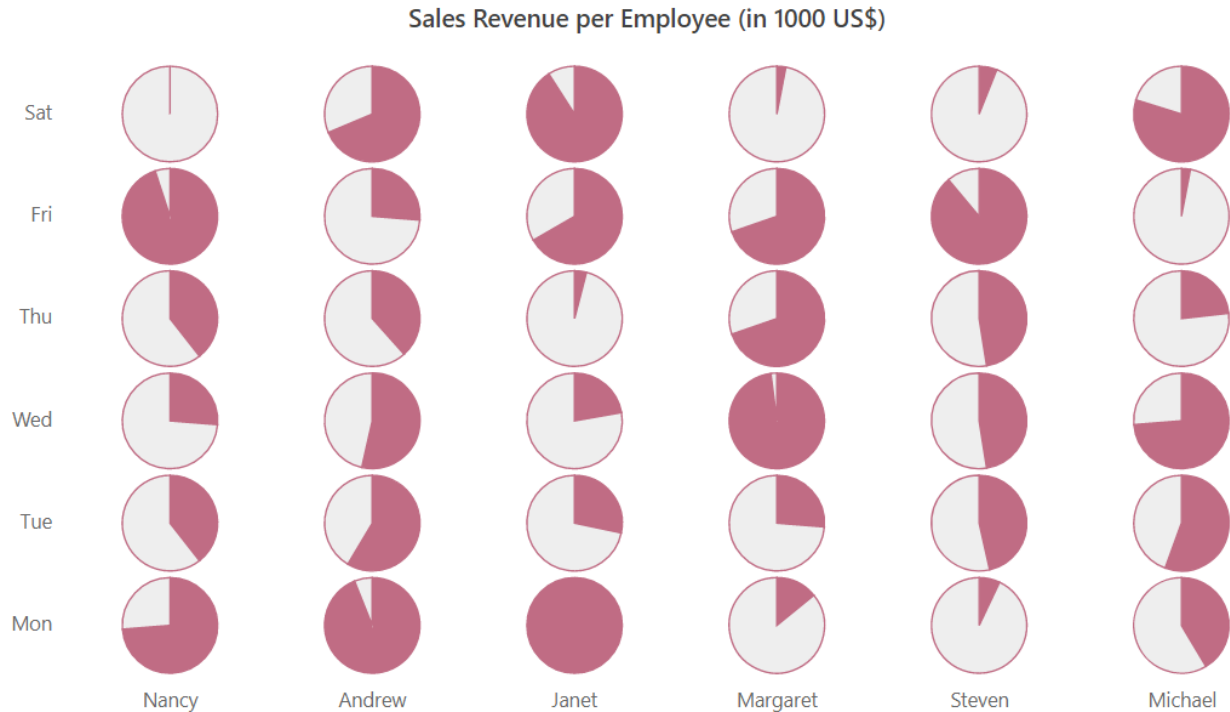
```

@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
</HeatMapTitleSettings>
<HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>

```



```
<HeatMapCellSettings ShowLabel="true" TileType="CellType.Bubble"
BubbleType="BubbleType.Sector"></HeatMapCellSettings>
<HeatMapPaletteSettings Type="PaletteType.Gradient">
<HeatMapPalettes>
<HeatMapPalette Color="#C06C84"></HeatMapPalette>
<HeatMapPalette Color="#6C5B7B"></HeatMapPalette>
<HeatMapPalette Color="#355C7D"></HeatMapPalette>
</HeatMapPalettes>
</HeatMapPaletteSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}
```



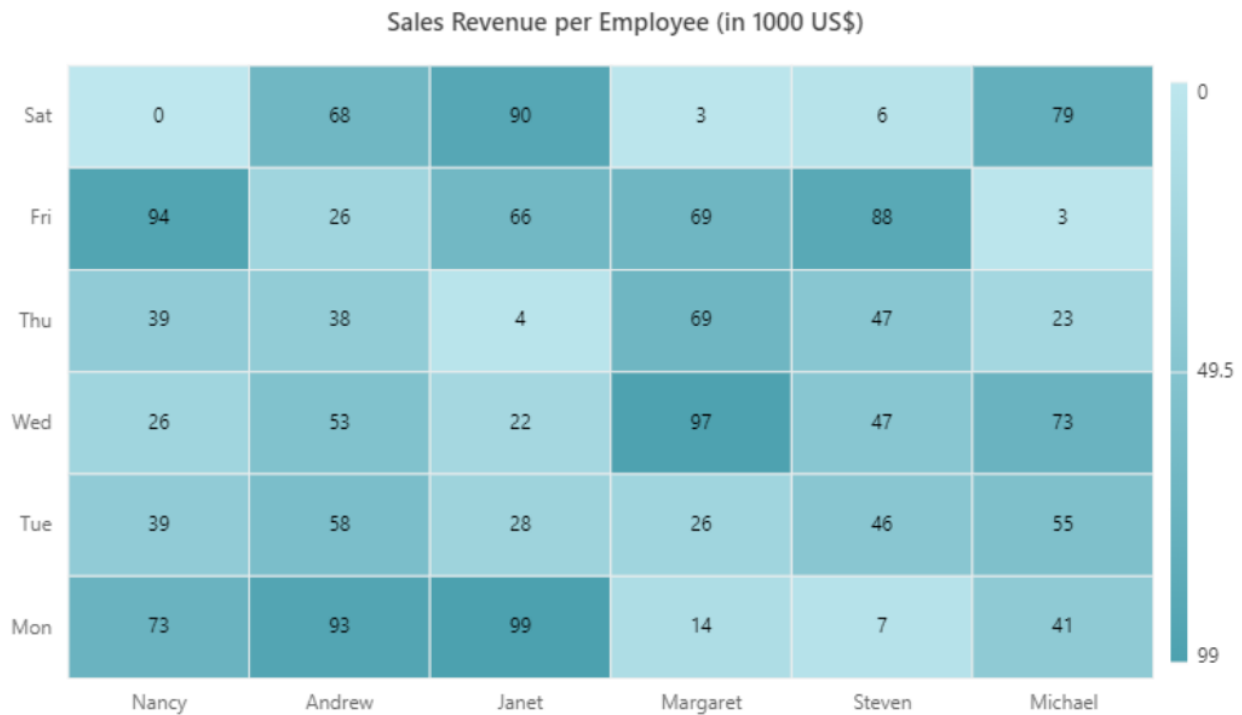
### Rendering mode in Blazor HeatMap Chart Component

Heat map can be displayed using **Scalable Vector Graphics (SVG)** rendering logic to improve the initial load performance and scalability.

#### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData" RenderingMode="DrawType.SVG">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapCellSettings ShowLabel="true"
    TileType="CellType.Rect"></HeatMapCellSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
  int[,] dataSource = new int[,]{
    {73, 39, 26, 39, 94, 0},
    {93, 58, 53, 38, 26, 68},
    {99, 28, 22, 4, 66, 90},
    {14, 26, 97, 69, 69, 3},
    {7, 46, 47, 47, 88, 6},
    {41, 55, 73, 23, 3, 79}
  };
  return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
  "Steven", "Michael" };
```

```
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}
```



### Axis in Blazor HeatMap Chart Component

Heat map consists of two axes namely, X-axis and Y-axis that displays the row headers and column headers to plot the data points respectively. You can define the type, format, and other customizing options for both axes in the heat map.

#### Types

There are three different axis types available in the heat map, which defines the data type of the axis labels. You can define the axis type by using the `ValueType` property in the heat map.

#### Category axis

Category axis type is used to represent the string values in axis labels.

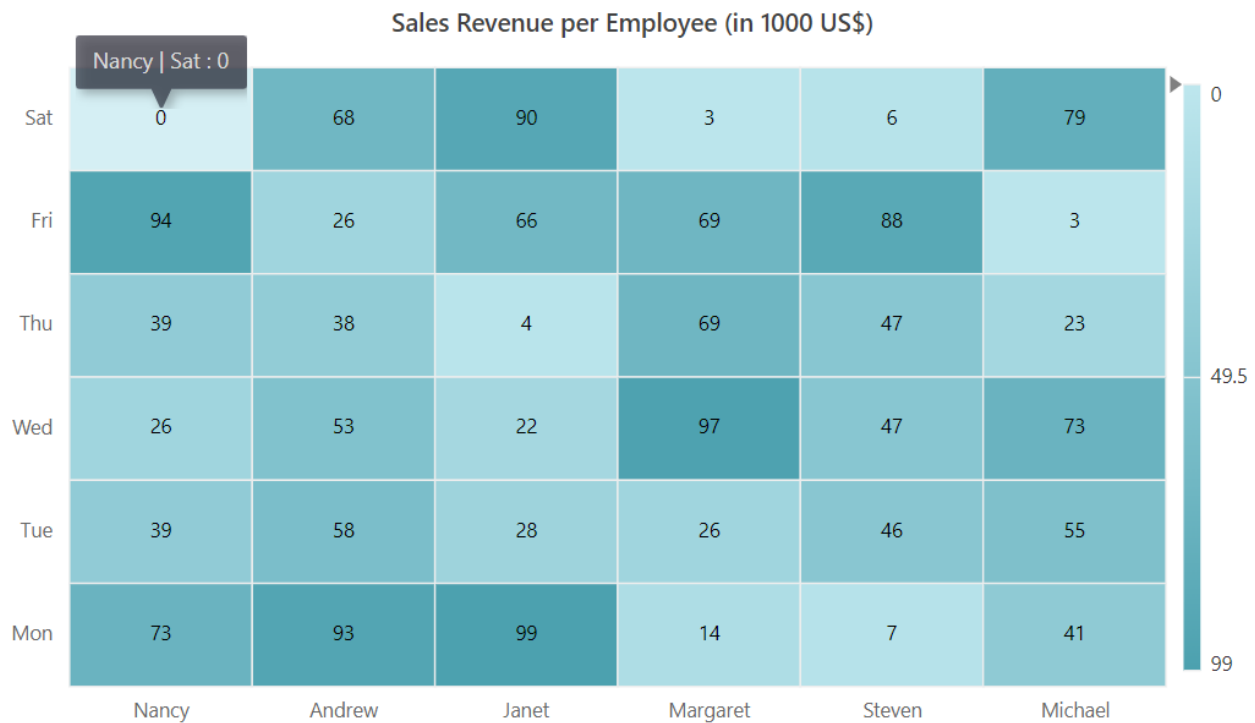
#### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"
  ValueType="Syncfusion.Blazor.HeatMap.ValueType.Category"></HeatMapXAxis>
```

```

<HeatMapYAxis Labels="@YAxisLabels"
ValueType="Syncfusion.Blazor.HeatMap.ValueType.Category"></HeatMapYAxis>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```

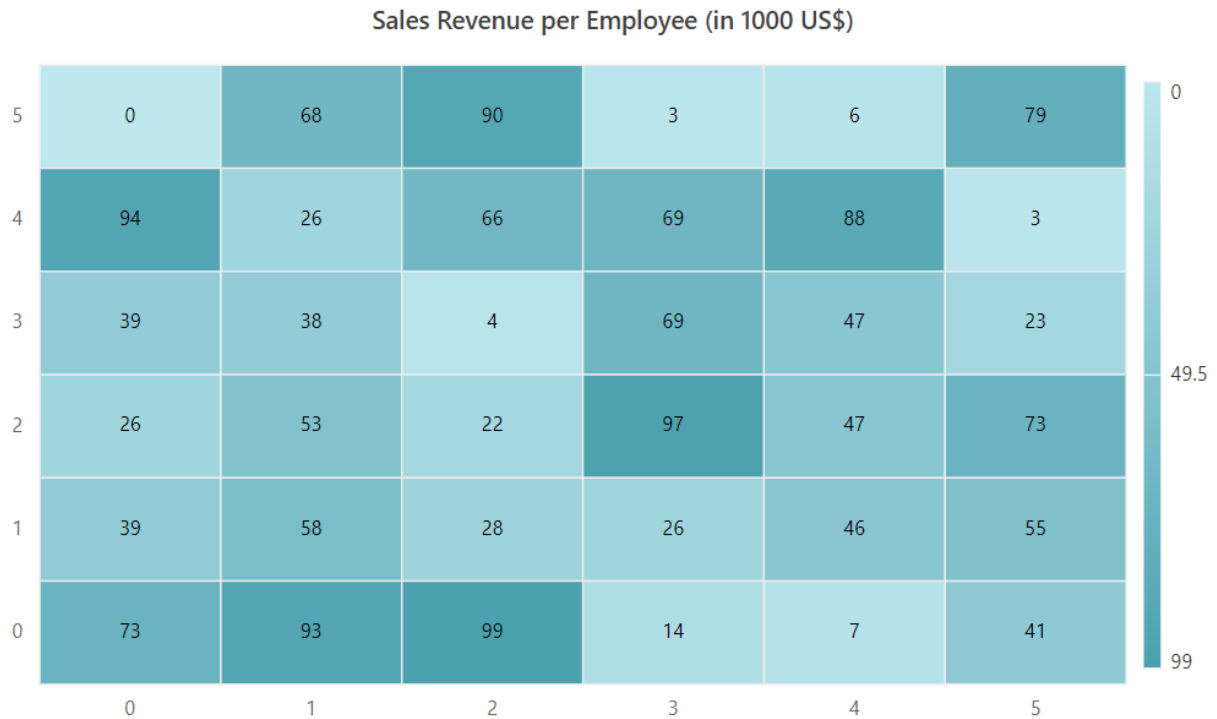


#### Numeric axis

Numeric axis type is used to represent the numeric values in axis labels.

**ASPX-CS**

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000
  US$)"></HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"
  ValueType="Syncfusion.Blazor.HeatMap.ValueType.Numeric"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"
  ValueType="Syncfusion.Blazor.HeatMap.ValueType.Numeric"></HeatMapYAxis>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}
```



#### Date-time axis

Date-time axis type is used to represent the date-time values in axis labels with a specific format. In date-time axis, you can define the start and end date/time using the `Minimum` and `Maximum` properties.

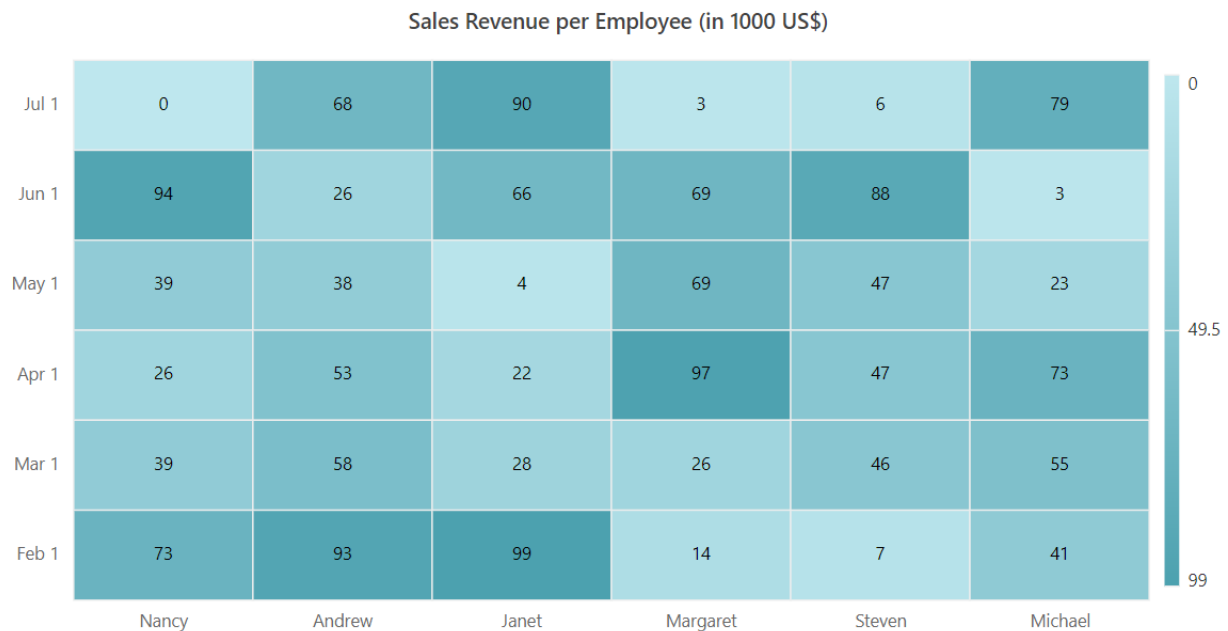
#### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"
  ValueType="Syncfusion.Blazor.HeatMap.ValueType.Category"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"
  ValueType="Syncfusion.Blazor.HeatMap.ValueType.DateTime" Minimum="@Minimum"
  Maximum="@Maximum" IntervalType="IntervalType.Months"></HeatMapYAxis>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
```

```

}
public object Minimum = new DateTime(2007, 2, 1);
public object Maximum = new DateTime(2007, 7, 1);
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



### Inversed axis

Heat map supports inverting the axis origin for both axes, where the axis labels are placed in an inversed manner. You can enable axis inverting by enabling the `IsInversed` property.

### ASPX-CS

```

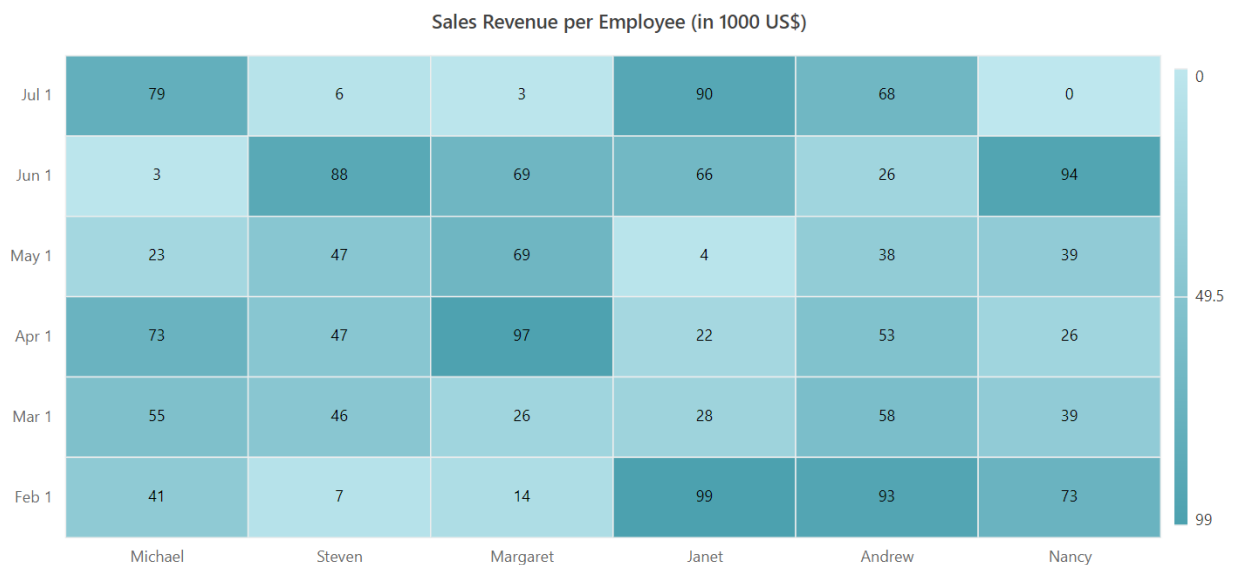
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"
  ValueType="Syncfusion.Blazor.HeatMap.ValueType.Category"
  IsInversed="true"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"
  ValueType="Syncfusion.Blazor.HeatMap.ValueType.DateTime" Minimum="@Minimum"
  Maximum="@Maximum" IntervalType="IntervalType.Months"></HeatMapYAxis>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{

```

```

int[,] dataSource = new int[,]
{
    {73, 39, 26, 39, 94, 0},
    {93, 58, 53, 38, 26, 68},
    {99, 28, 22, 4, 66, 90},
    {14, 26, 97, 69, 69, 3},
    {7, 46, 47, 47, 88, 6},
    {41, 55, 73, 23, 3, 79}
};
return dataSource;
}
public object Minimum = new DateTime(2007, 2, 1);
public object Maximum = new DateTime(2007, 7, 1);
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
    HeatMapData = GetDefaultData();
}
}

```



### Opposed axis

In heat map, you can place the axis label in an opposite position of its default axis label position by using the `OpposedPosition` property.

### ASPX-CS

```

@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
</HeatMapTitleSettings>

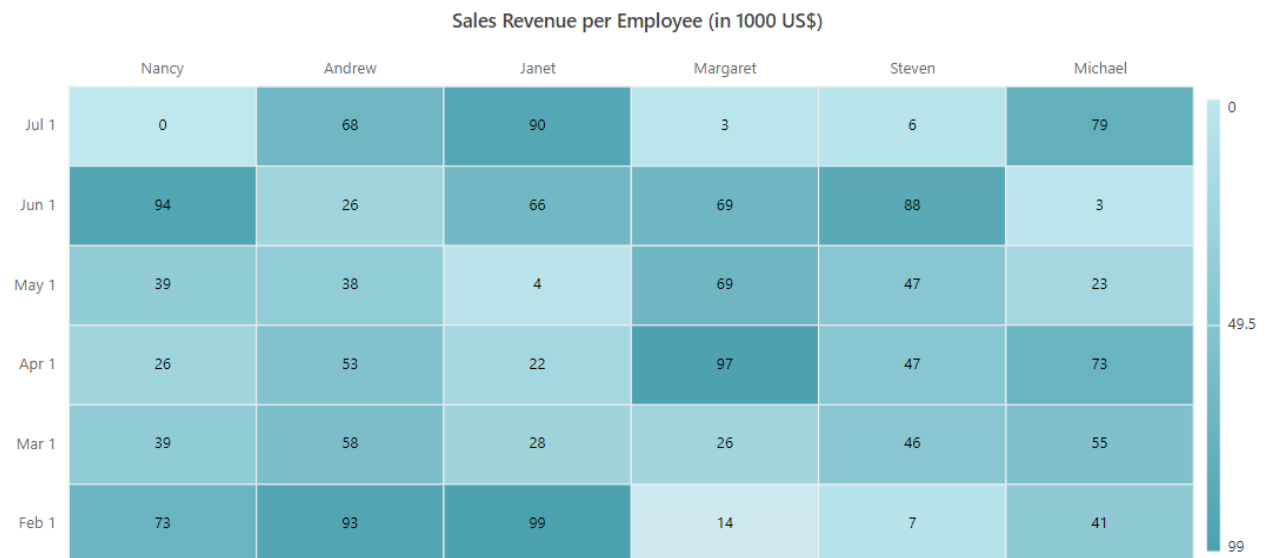
```



```

<HeatMapXAxis Labels="@XAxisLabels"
ValueType="Syncfusion.Blazor.HeatMap.ValueType.Category"
OpposedPosition="true"></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"
ValueType="Syncfusion.Blazor.HeatMap.ValueType.DateTime" Minimum="@Minimum"
Maximum="@Maximum" IntervalType="IntervalType.Months"></HeatMapYAxis>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
public object Minimum = new DateTime(2007, 2, 1);
public object Maximum = new DateTime(2007, 7, 1);
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```

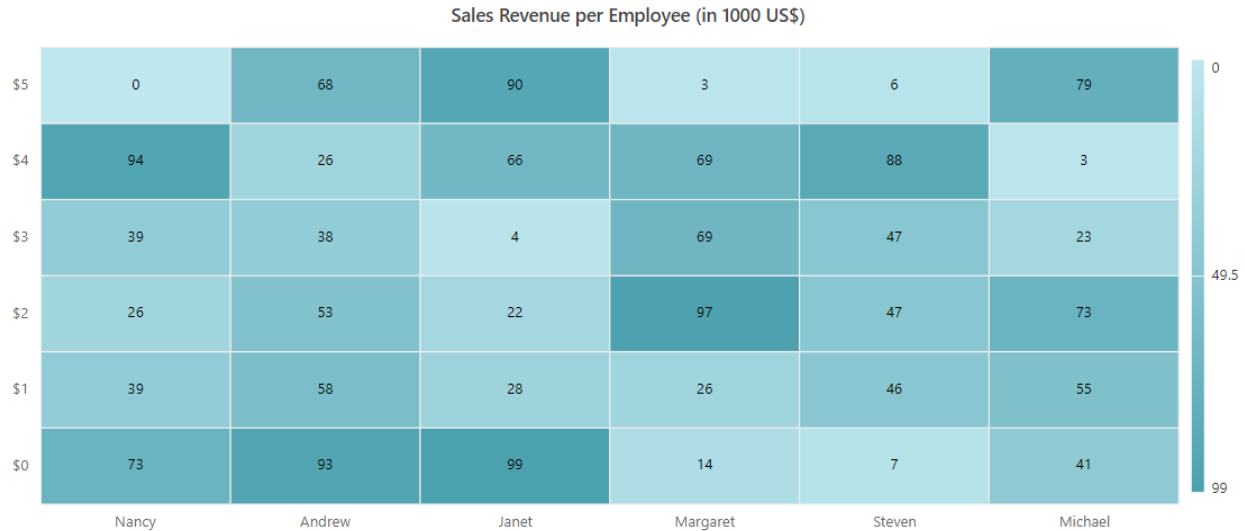


### Label formatting

Heat map supports formatting the axis labels by using the `LabelFormat` property. Using this property, you can customize the axis label by global string format ('P', 'C', etc) or customized format like '{value}'C'.

### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
</HeatMapTitleSettings>
<HeatMapXAxis Labels="@XAxisLabels"
ValueType="Syncfusion.Blazor.HeatMap.ValueType.Category"></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"
ValueType="Syncfusion.Blazor.HeatMap.ValueType.Numeric"
LabelFormat="{value}'C"></HeatMapYAxis>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
public object Minimum = new DateTime(2007, 2, 1);
public object Maximum = new DateTime(2007, 7, 1);
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}
```



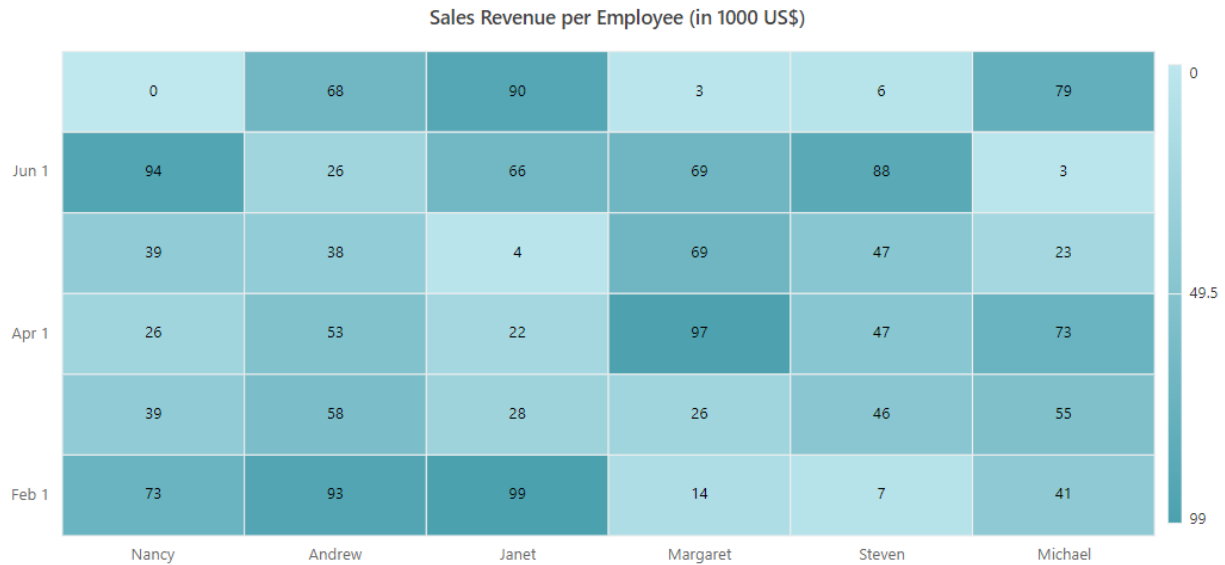
### Axis intervals

In heat map, you can define an interval between the axis labels using the `Interval` property. In date-time axis, you can change the interval mode by using the `IntervalType` property. The date-time axis supports the following interval types.

### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"
    ValueType="Syncfusion.Blazor.HeatMap.ValueType.Category"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"
    ValueType="Syncfusion.Blazor.HeatMap.ValueType.DateTime" Minimum="@Minimum"
    Interval=2 Maximum="@Maximum"
    IntervalType="IntervalType.Months"></HeatMapYAxis>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
public object Minimum = new DateTime(2007, 2, 1);
public object Maximum = new DateTime(2007, 7, 1);
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet",
"Margaret", "Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
```

```
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
    HeatMapData = GetDefaultData();
}
}
```



### Axis label increment

Axis label increment in the heat map is used to display the axis labels with regular interval values in numeric and date-time axes. The labels will be displayed with tick gaps when you set the label interval. But, to achieve the same behavior without tick gaps, use the label increment. You can set the axis label increment using the `Increment` property and the default value of this property is 1.

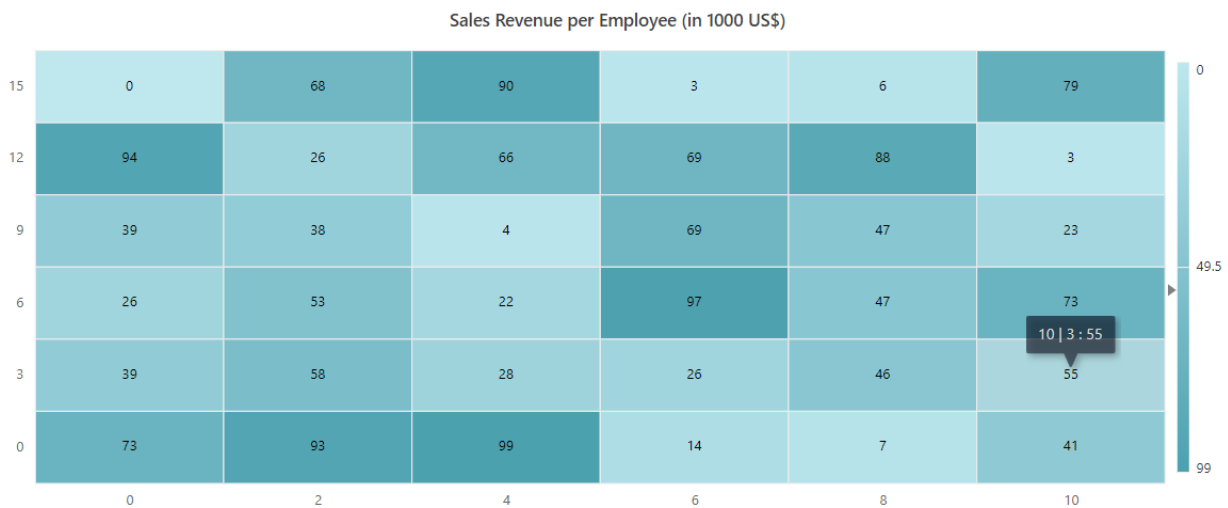
### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
</HeatMapTitleSettings>
<HeatMapXAxis Labels="@XAxisLabels"
ValueType="Syncfusion.Blazor.HeatMap.ValueType.Numeric"
Increment=2></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"
ValueType="Syncfusion.Blazor.HeatMap.ValueType.Numeric"
Increment=3></HeatMapYAxis>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
    int[,] dataSource = new int[,]
    {
        {73, 39, 26, 39, 94, 0},
        {93, 58, 53, 38, 26, 68},
        {99, 28, 22, 4, 66, 90},
        {14, 26, 97, 69, 69, 3},
    }
}
```

```

{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet",
"Margaret", "Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



### Palette in Blazor HeatMap Chart Component

In heat map, each data point is displayed as a cell with applied color based on the data value. The palette in the heat map is used to define the color range for cells and gradient type for colors. You can define the colors either in RGB or hex codes using the **Color** property in the **HeatMapPalette**. The defined colors are applied to the cell background based on the palette type and cell value.

#### Palette types

You can display the heat map cells either in gradient colors or fixed colors.

##### Gradient

The smooth transition between the given palette colors can be applied for the heat map cells based on value. The heat map calculates all the gradient colors between the start and end colors for all distinct data values. Default start color and end color will be considered for gradient calculation, if the colors are not defined. The palette type must be defined as **Gradient** for the **Type** property in the **HeatMapPaletteSettings** property.

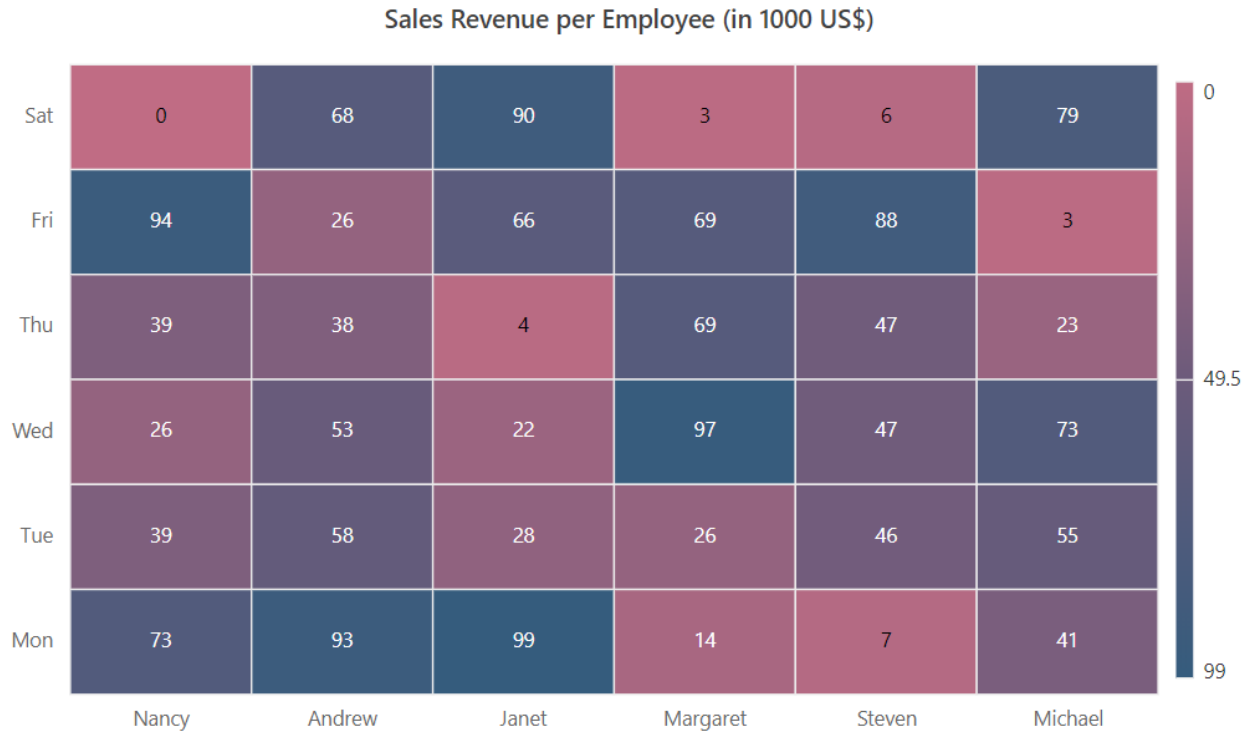
#### ASPX-CS

```

@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData" RenderingMode="DrawType.SVG">
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">

```

```
</HeatMapTitleSettings>
<HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
<HeatMapCellSettings ShowLabel="true"
TileType="CellType.Rect"></HeatMapCellSettings>
<HeatMapPaletteSettings Type="PaletteType.Gradient">
<HeatMapPalettes>
<HeatMapPalette Color="#C06C84"></HeatMapPalette>
<HeatMapPalette Color="#6C5B7B"></HeatMapPalette>
<HeatMapPalette Color="#355C7D"></HeatMapPalette>
</HeatMapPalettes>
</HeatMapPaletteSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}
```



#### Fixed

In fixed palette type, solid colors are applied to the heat map cells. The data values can be grouped based on the number of colors defined for the heat map. The palette type should be defined as **Fixed** for the **Type** property in the **HeatMapPaletteSettings** property.

#### ASPX-CS

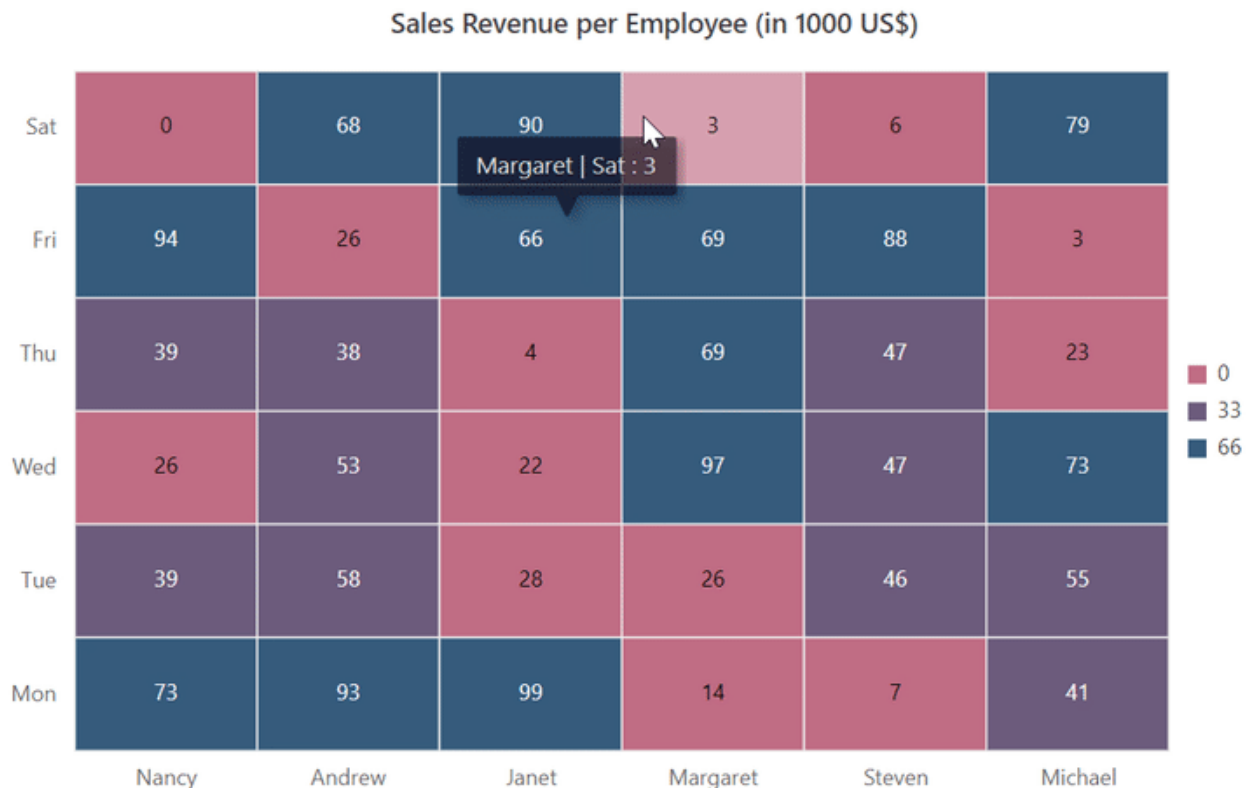
```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData" RenderingMode="DrawType.SVG">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapCellSettings ShowLabel="true"
  TileType="CellType.Rect"></HeatMapCellSettings>
  <HeatMapPaletteSettings Type="PaletteType.Fixed">
  <HeatMapPalettes>
    <HeatMapPalette Color="#C06C84"></HeatMapPalette>
    <HeatMapPalette Color="#6C5B7B"></HeatMapPalette>
    <HeatMapPalette Color="#355C7D"></HeatMapPalette>
  </HeatMapPalettes>
  </HeatMapPaletteSettings>
</SfHeatMap>

@code{
int[,] GetDefaultData()
{
  int[,] dataSource = new int[,]{
    {73, 39, 26, 39, 94, 0},
    {93, 58, 53, 38, 26, 68},
    {99, 28, 22, 4, 66, 90},
  }
}
```

```

{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



<!-- ## Defining color stops

You can define the colors ranges or color stops for data values in both gradient and fixed palette types. You need to define the data value in the `Value` property for `HeatMapPalette` property to calculate the color stops. The heat map automatically calculates the color stops if the `Value` property is not defined. The `Label` property is used to provide the additional information about the color that is to be displayed in the legend. If the label is not provided, the value will be displayed in the legend. The labels can be automatically calculated based on data values, if both the values and labels are not defined.

#### ASPX-CS



```

@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData" RenderingMode="DrawType.SVG">
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
</HeatMapTitleSettings>
<HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
<HeatMapCellSettings ShowLabel="true"
TileType="CellType.Rect"></HeatMapCellSettings>
<HeatMapPaletteSettings Type="PaletteType.Fixed">
<HeatMapPalettes>
<HeatMapPalette Color="#C06C84" Label="Low" Value=3></HeatMapPalette>
<HeatMapPalette Color="#6C5B7B" Label="Moderate"
Value=33.3></HeatMapPalette>
<HeatMapPalette Color="#355C7D" Label="High" Value=75></HeatMapPalette>
</HeatMapPalettes>
</HeatMapPaletteSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```

--&gt;

### Legend in Blazor HeatMap Chart Component

The legend is used to provide the information about the heat map cell. You can enable the legend by setting the **Visible** property to true.

#### ASPX-CS

```

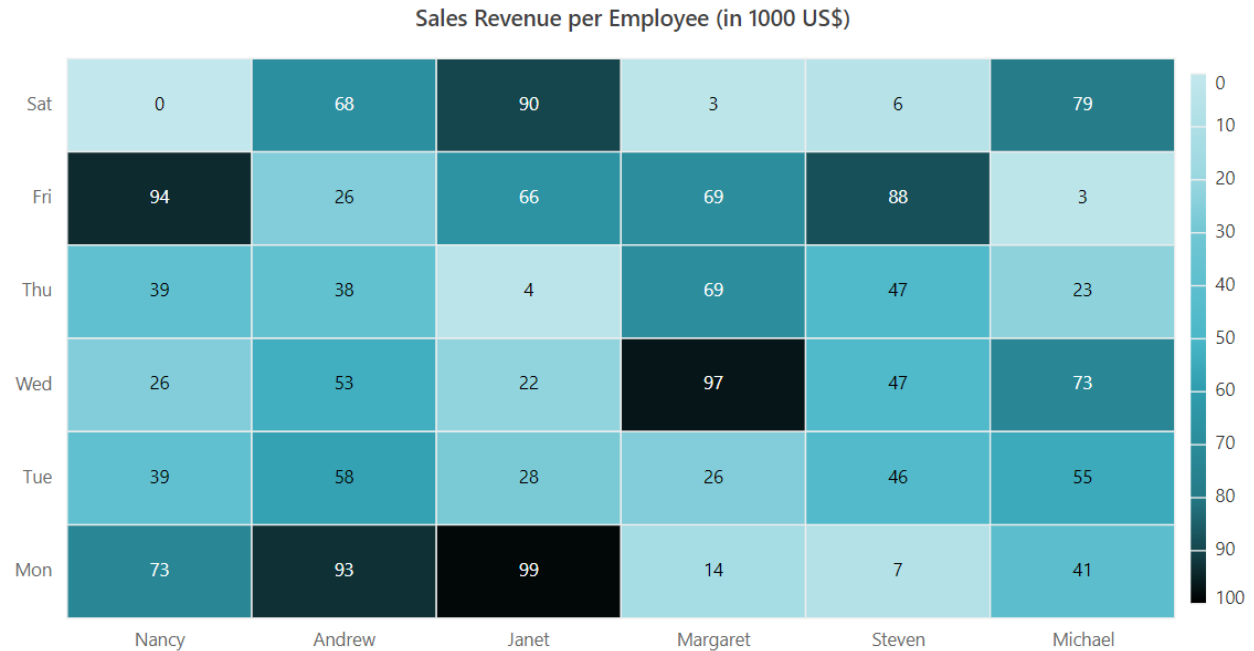
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
</HeatMapTitleSettings>
<HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>

```

```

<HeatMapCellSettings ShowLabel="true"
TileType="CellType.Rect"></HeatMapCellSettings>
<HeatMapPaletteSettings Type="PaletteType.Gradient">
<HeatMapPalettes>
<HeatMapPalette Value="0" Color="#C2E7EC"></HeatMapPalette>
<HeatMapPalette Value="10" Color="#AEDFE6"></HeatMapPalette>
<HeatMapPalette Value="20" Color="#9AD7E0"></HeatMapPalette>
<HeatMapPalette Value="30" Color="#72C7D4"></HeatMapPalette>
<HeatMapPalette Value="40" Color="#5EBFCE"></HeatMapPalette>
<HeatMapPalette Value="50" Color="#4AB7C8"></HeatMapPalette>
<HeatMapPalette Value="60" Color="#309DAE"></HeatMapPalette>
<HeatMapPalette Value="70" Color="#2B8C9B"></HeatMapPalette>
<HeatMapPalette Value="80" Color="#257A87"></HeatMapPalette>
<HeatMapPalette Value="90" Color="#15464D"></HeatMapPalette>
<HeatMapPalette Value="100" Color="#000000"></HeatMapPalette>
</HeatMapPalettes>
</HeatMapPaletteSettings>
<HeatMapLegendSettings Visible="true"></HeatMapLegendSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



### Legend types

Heat map supports two legend types: Gradient and list type.

- Gradient - This is a continuous color legend with smooth color transition between palette color values.
- List - List is a fixed color legend. Each palette color information is shown separately in the list item.

You can change the legend type by using the `Type` property in the `HeatMapPaletteSettings` property.

### ASPX-CS

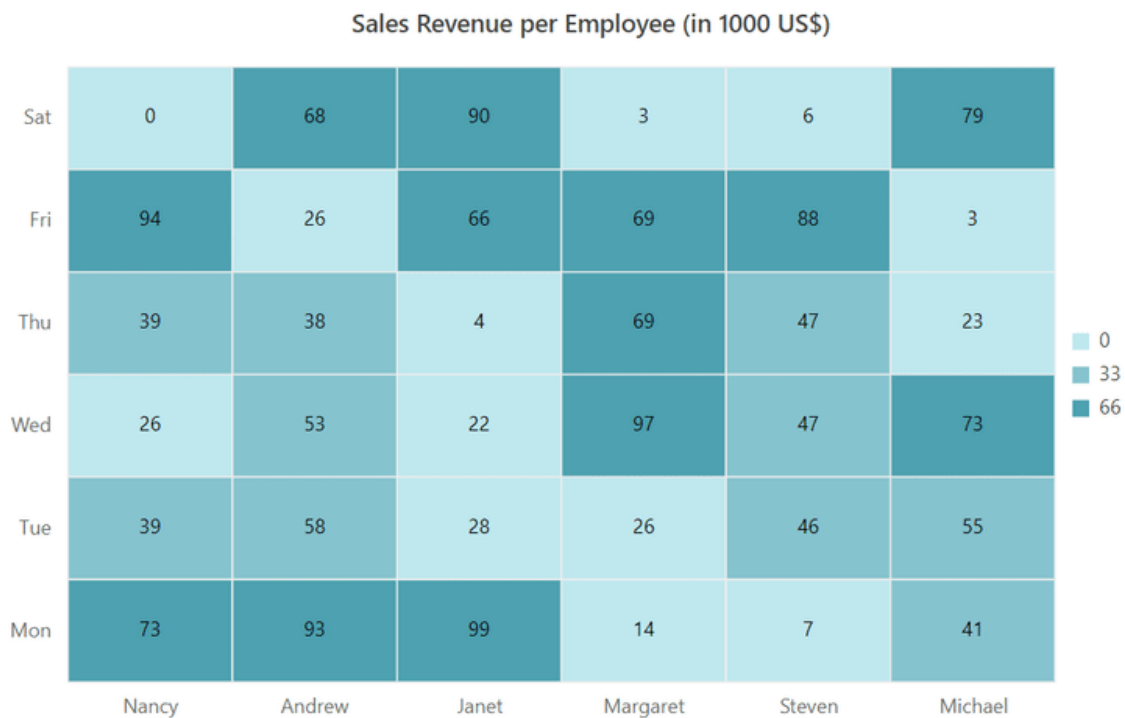
```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapCellSettings ShowLabel="true"
  TileType="CellType.Rect"></HeatMapCellSettings>
  <HeatMapPaletteSettings Type="PaletteType.Fixed"></HeatMapPaletteSettings>
  <HeatMapLegendSettings ShowLabel="true"></HeatMapLegendSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},

```

```

{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



### Placement

You can place the legend at left, right, top, or bottom to the heat map layout by using the **Position** property. The legend is positioned at the right to the heat map by default.

### ASPX-CS

```

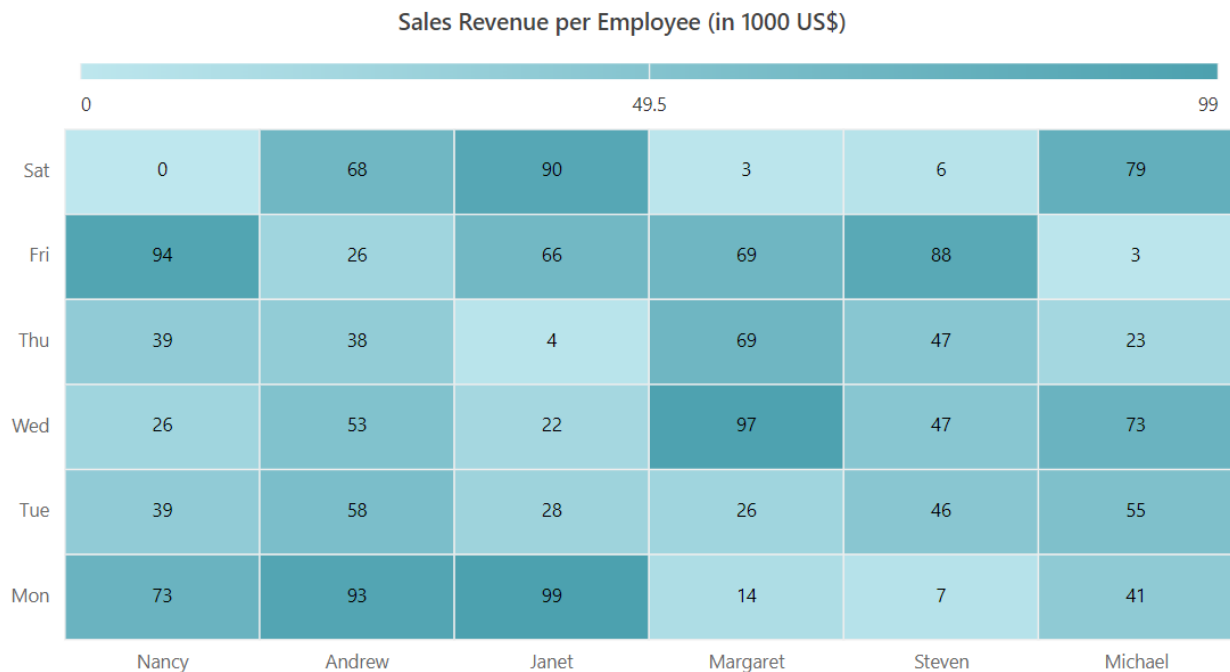
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>

```

```

<HeatMapCellSettings ShowLabel="true"
TileType="CellType.Rect"></HeatMapCellSettings>
<HeatMapLegendSettings ShowLabel="true"
Position="LegendPosition.Top"></HeatMapLegendSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```

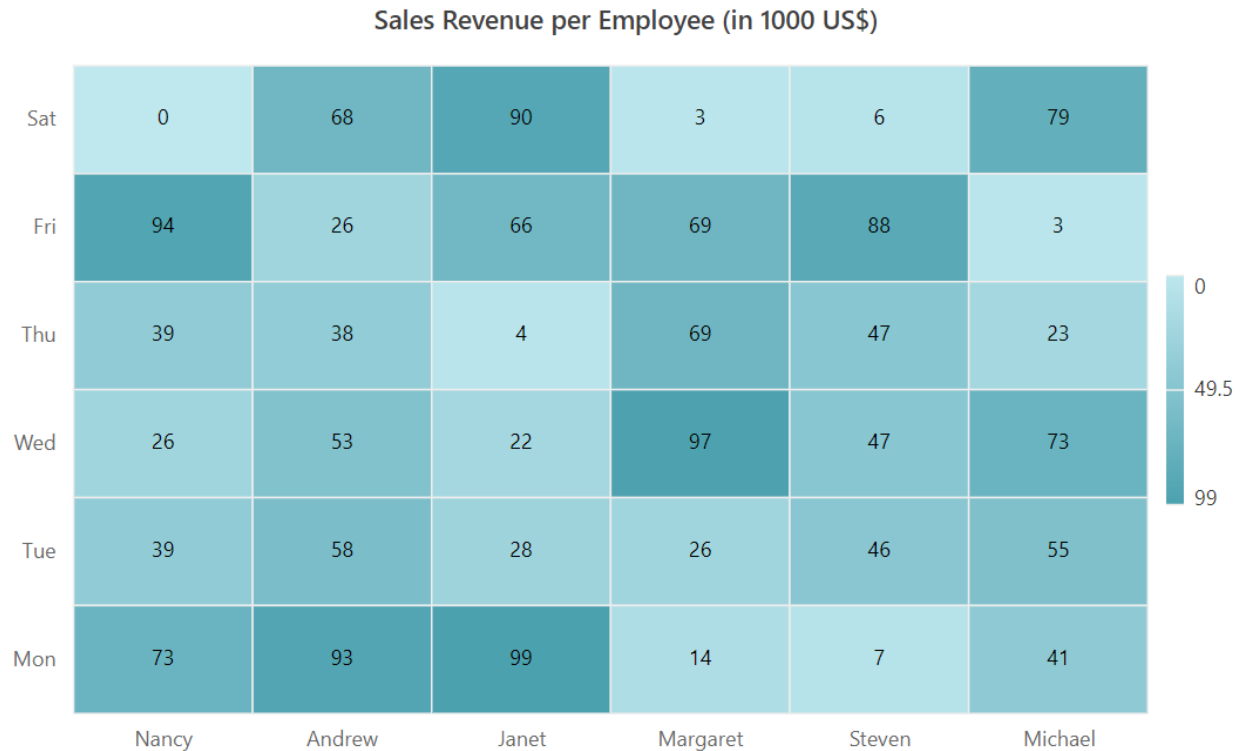


### Alignment

You can align the legend as center, far, or near to the heat map using the `Alignment` property.

### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapCellSettings ShowLabel="true"
  TileType="CellType.Rect"></HeatMapCellSettings>
  <HeatMapLegendSettings ShowLabel="true" Position="LegendPosition.Right"
  Height="150px" Alignment="Alignment.Center"></HeatMapLegendSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}
```



### Legend dimensions

You can change the legend dimensions with values in pixels or percentage by using the **Width** and **Height** properties.

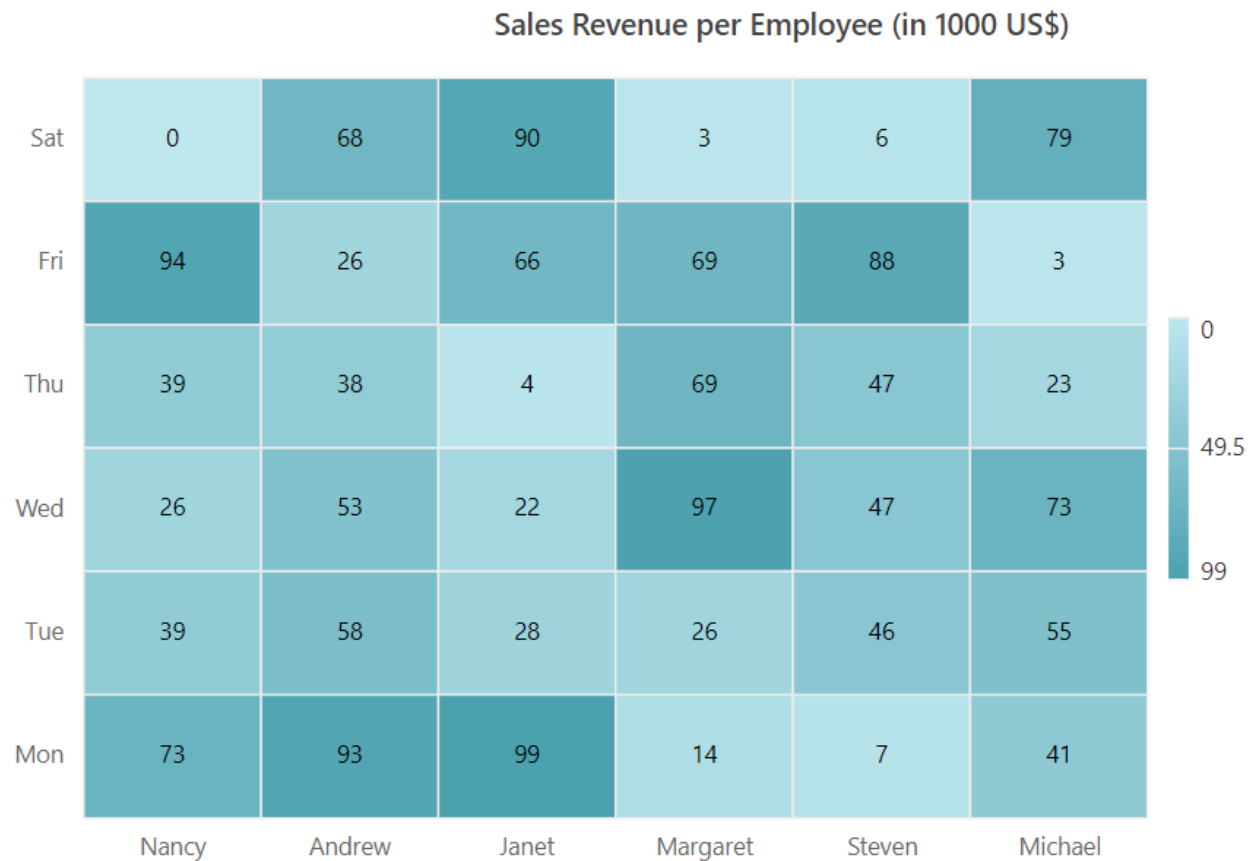
### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapCellSettings ShowLabel="true"
  TileType="CellType.Rect"></HeatMapCellSettings>
  <HeatMapLegendSettings ShowLabel="true" Position="LegendPosition.Right"
  Width="200px" Height="150px"></HeatMapLegendSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
```

```

}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
    HeatMapData = GetDefaultData();
}
}

```



<!-- ## Paging for legend

Paging is available only for the list type legend in the heat map, and it can be enabled by default, when the legend items exceed the legend bounds. You can view each legend items by navigating between the pages using navigation buttons.

#### ASPX-CS

```

@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>

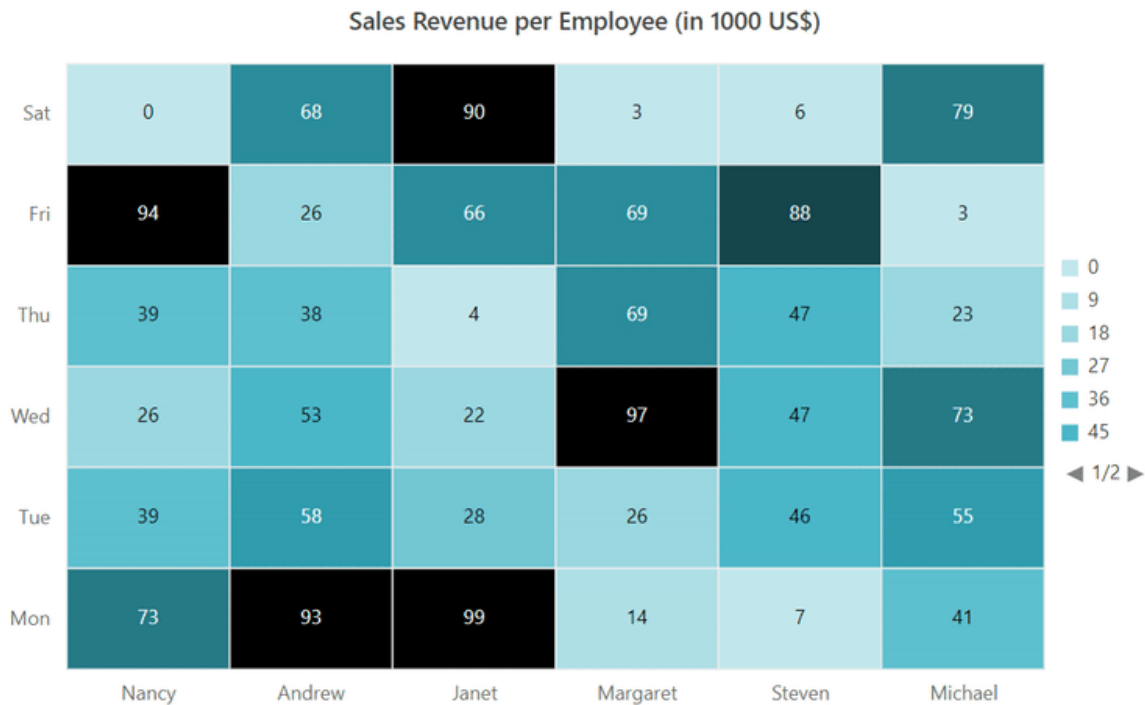
```



```

<HeatMapCellSettings ShowLabel="true"
TileType="CellType.Rect"></HeatMapCellSettings>
<HeatMapPaletteSettings Type="PaletteType.Fixed">
<HeatMapPalettes>
<HeatMapPalette Value="0" Color="#C2E7EC"></HeatMapPalette>
<HeatMapPalette Value="10" Color="#AEDFE6"></HeatMapPalette>
<HeatMapPalette Value="20" Color="#9AD7E0"></HeatMapPalette>
<HeatMapPalette Value="30" Color="#72C7D4"></HeatMapPalette>
<HeatMapPalette Value="40" Color="#5EBFCE"></HeatMapPalette>
<HeatMapPalette Value="50" Color="#4AB7C8"></HeatMapPalette>
<HeatMapPalette Value="60" Color="#309DAE"></HeatMapPalette>
<HeatMapPalette Value="70" Color="#2B8C9B"></HeatMapPalette>
<HeatMapPalette Value="80" Color="#257A87"></HeatMapPalette>
<HeatMapPalette Value="90" Color="#15464D"></HeatMapPalette>
<HeatMapPalette Value="100" Color="#000000"></HeatMapPalette>
</HeatMapPalettes>
</HeatMapPaletteSettings>
<HeatMapLegendSettings ShowLabel="true" Position="LegendPosition.Right"
Alignment="Alignment.Center" Height="150px"></HeatMapLegendSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



### Smart Legend

Smart legend is another way of showing list type legend with responsiveness and readability, when the palette has more number of items. You can enable this smart legend by using the `EnableSmartLegend` property when the palette type is set to `Fixed`.

In smart legend, you can change the display type of legend labels by using the `LabelDisplayType` property.

The following are the legend label display types:

- All: Displays all labels in the legend.
- Edge: Displays the legend labels only at extreme ends.
- None: None of the labels are displayed. The tooltip will appear for this type of label display when hovering over the legend item.

### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapCellSettings ShowLabel="true"
  TileType="CellType.Rect"></HeatMapCellSettings>
  <HeatMapPaletteSettings Type="PaletteType.Fixed">
  <HeatMapPalettes>
```

```

<HeatMapPalette Value="0" Color="#C2E7EC"></HeatMapPalette>
<HeatMapPalette Value="10" Color="#AEDFE6"></HeatMapPalette>
<HeatMapPalette Value="20" Color="#9AD7E0"></HeatMapPalette>
<HeatMapPalette Value="30" Color="#72C7D4"></HeatMapPalette>
<HeatMapPalette Value="40" Color="#5EBFCE"></HeatMapPalette>
<HeatMapPalette Value="50" Color="#4AB7C8"></HeatMapPalette>
<HeatMapPalette Value="60" Color="#309DAE"></HeatMapPalette>
<HeatMapPalette Value="70" Color="#2B8C9B"></HeatMapPalette>
<HeatMapPalette Value="80" Color="#257A87"></HeatMapPalette>
<HeatMapPalette Value="90" Color="#15464D"></HeatMapPalette>
<HeatMapPalette Value="100" Color="#000000"></HeatMapPalette>
</HeatMapPalettes>
</HeatMapPaletteSettings>
<HeatMapLegendSettings ShowLabel="true" Position="LegendPosition.Right"
EnableSmartLegend="true"
Alignment="Alignment.Center"></HeatMapLegendSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```

### Legend Selection

In the HeatMap, the legend selection is used to toggle the visibility of cell for viewing the specific range value. You can enable the legend selection using the `ToggleVisibility` property.

### ASPX-CS

```

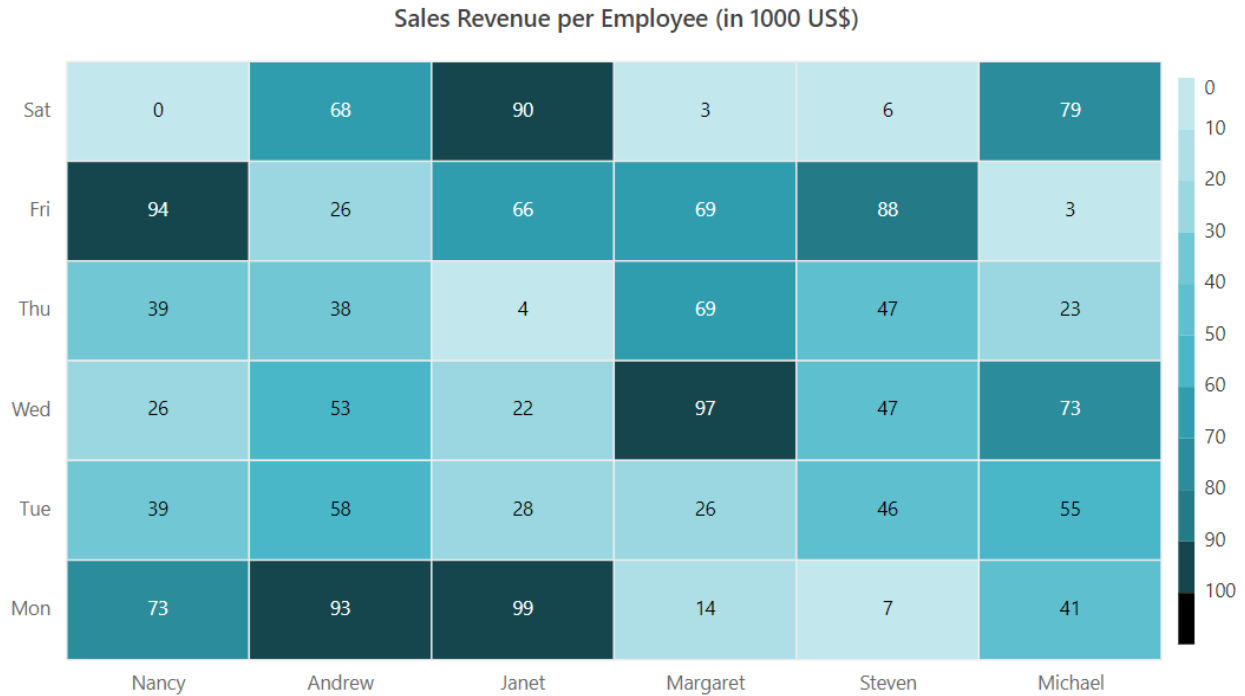
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
</HeatMapTitleSettings>
<HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
<HeatMapCellSettings ShowLabel="true"
TileType="CellType.Rect"></HeatMapCellSettings>

```

```

<HeatMapPaletteSettings Type="PaletteType.Fixed">
  <HeatMapPalettes>
    <HeatMapPalette Value="0" Color="#C2E7EC"></HeatMapPalette>
    <HeatMapPalette Value="10" Color="#AEDFE6"></HeatMapPalette>
    <HeatMapPalette Value="20" Color="#9AD7E0"></HeatMapPalette>
    <HeatMapPalette Value="30" Color="#72C7D4"></HeatMapPalette>
    <HeatMapPalette Value="40" Color="#5EBFCE"></HeatMapPalette>
    <HeatMapPalette Value="50" Color="#4AB7C8"></HeatMapPalette>
    <HeatMapPalette Value="60" Color="#309DAE"></HeatMapPalette>
    <HeatMapPalette Value="70" Color="#2B8C9B"></HeatMapPalette>
    <HeatMapPalette Value="80" Color="#257A87"></HeatMapPalette>
    <HeatMapPalette Value="90" Color="#15464D"></HeatMapPalette>
    <HeatMapPalette Value="100" Color="#000000"></HeatMapPalette>
  </HeatMapPalettes>
</HeatMapPaletteSettings>
<HeatMapLegendSettings ShowLabel="true" Position="LegendPosition.Right"
  EnableSmartLegend="true" ToggleVisibility="true"></HeatMapLegendSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



--&gt;

## Appearance in Blazor HeatMap Chart Component

### Cell customizations

You can customize the cell by using the `CellSettings`

#### Border

Change the width, color, and radius of the heat map cells by using the `HeatMapCellBorder` tag.

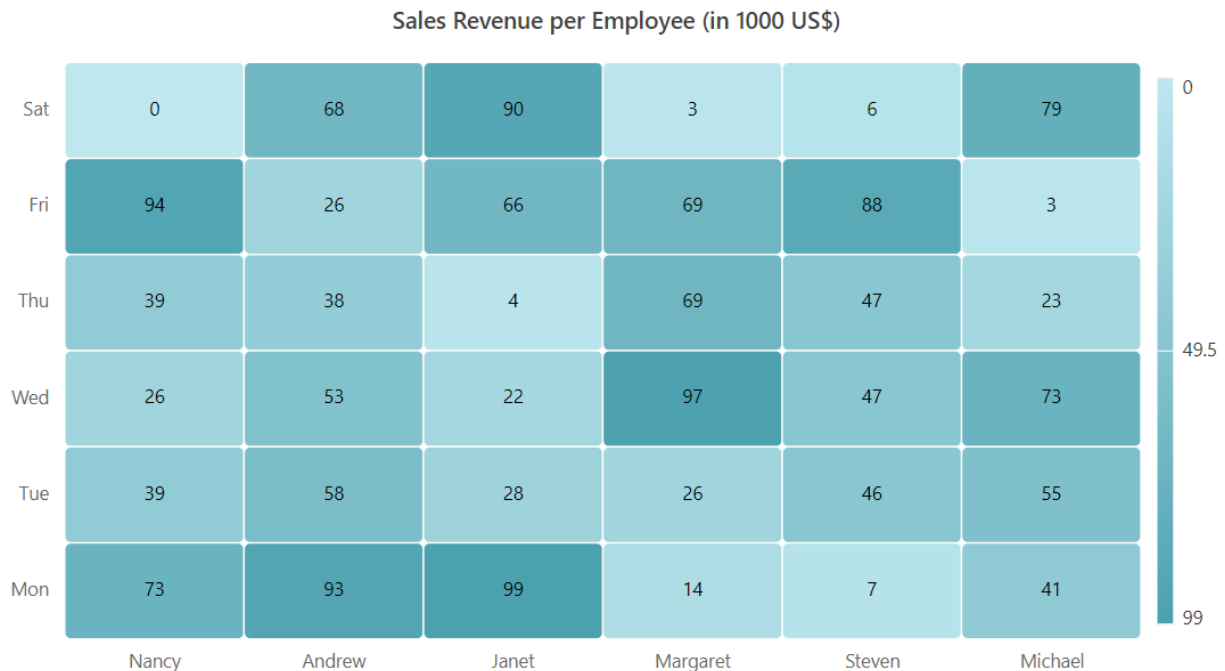
#### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapCellSettings ShowLabel="true" TileType="CellType.Rect">
  <HeatMapCellBorder Width = "1" Radius = "4" Color = "White"
  </HeatMapCellBorder>
  </HeatMapCellSettings>
  <HeatMapLegendSettings ShowLabel="true"></HeatMapLegendSettings>
</SfHeatMap>
@code{
int[, ] GetDefaultData()
{
int[, ] dataSource = new int[, ]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
```

```

{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79},
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



### Cell highlighting

Enable or disable the cell highlighting while hover over the heat map cells by using the `EnableCellHighlighting` property.

The cell highlighting only works in a SVG rendering mode.

### ASPX-CS

```

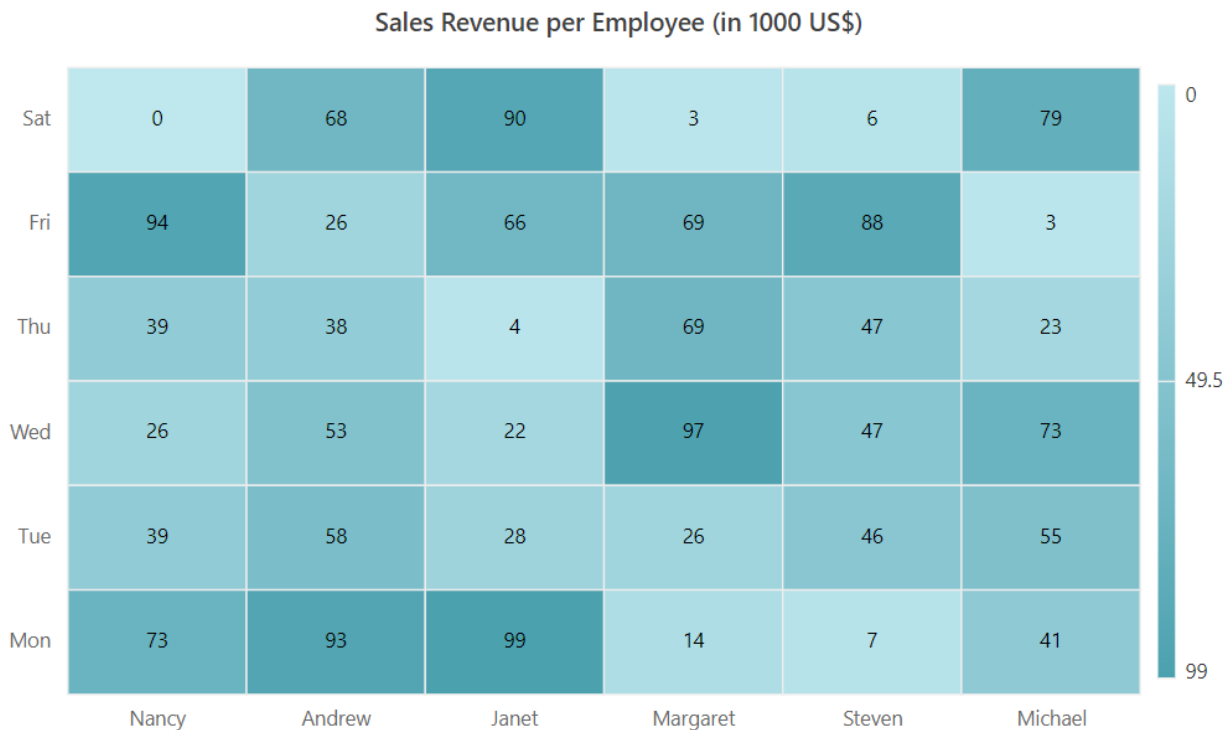
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
<HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
</HeatMapTitleSettings>
<HeatMapCellSettings ShowLabel="true" TileType="CellType.Rect"
EnableCellHighlighting="true"></HeatMapCellSettings>

```

```

<HeatMapLegendSettings ShowLabel="true" Position="LegendPosition.Right"
EnableSmartLegend="true" ToggleVisibility="true"></HeatMapLegendSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



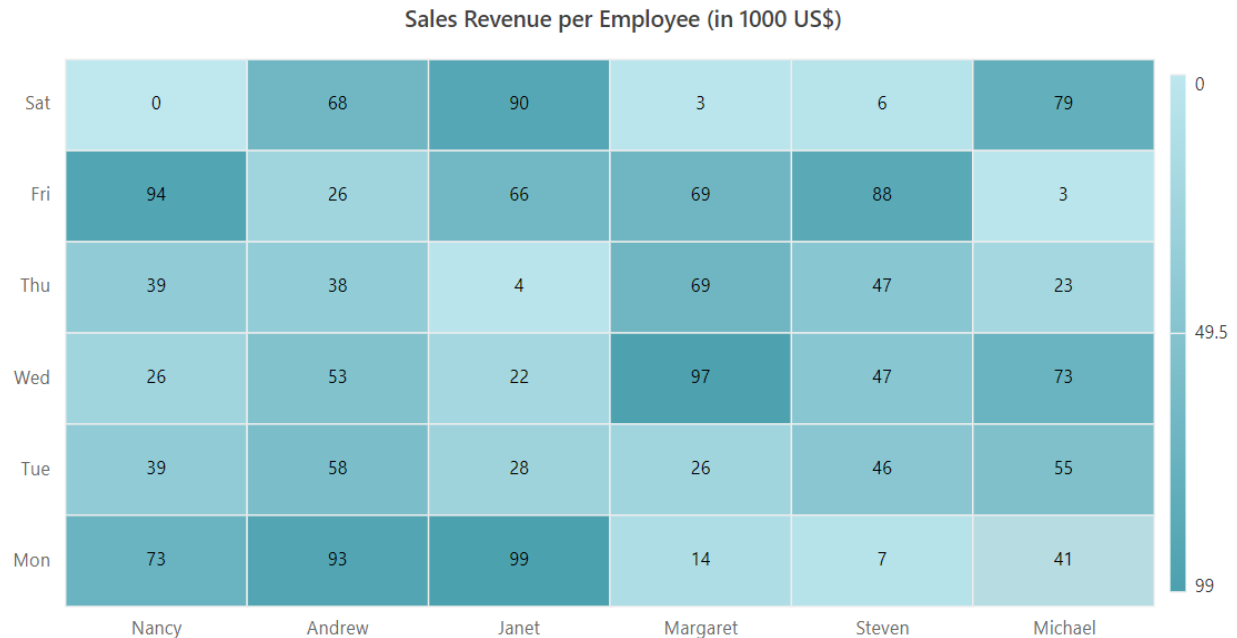
## Margin

Set the margin for the heat map from its container by using the `HeatMapMargin` property.

### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapCellSettings ShowLabel="true"
  TileType="CellType.Rect"></HeatMapCellSettings>
  <HeatMapMargin Left="15" Right="15" Top="15" Bottom="15"></HeatMapMargin>
  <HeatMapLegendSettings ShowLabel="true" Position="LegendPosition.Right"
  EnableSmartLegend="true" ToggleVisibility="true"></HeatMapLegendSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}
```





### Title

The title is used to provide a quick information about the data plotted in heat map. The `Text` property is used to set the title for heat map. You can also customize text style of a title by using the `HeatMapTitleTextStyle` tag.

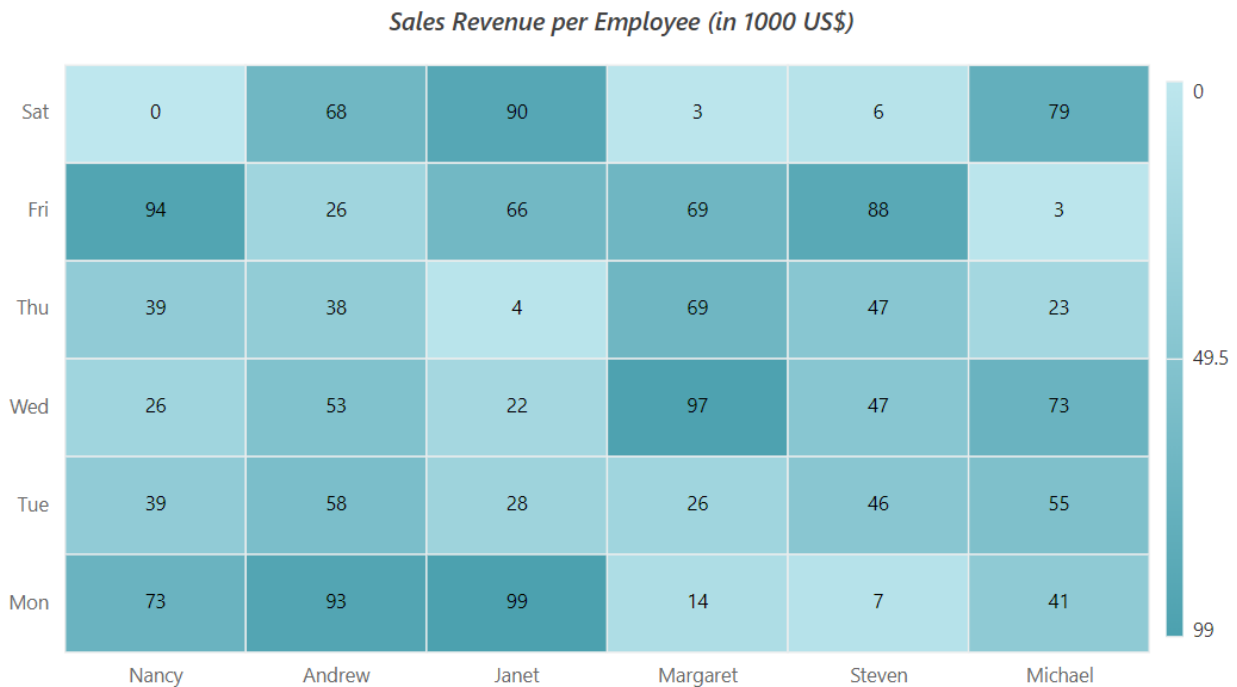
### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
<HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
<HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
<HeatMapTitleTextStyle Size = "15px" FontWeight = "500" FontStyle = "Italic"
FontFamily = "Segoe UI"></HeatMapTitleTextStyle>
</HeatMapTitleSettings>
<HeatMapCellSettings ShowLabel="true"
TileType="CellType.Rect"></HeatMapCellSettings>
<HeatMapMargin Left="15" Right="15" Top="15" Bottom="15"></HeatMapMargin>
<HeatMapLegendSettings ShowLabel="true" Position="LegendPosition.Right"
EnableSmartLegend="true" ToggleVisibility="true"></HeatMapLegendSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
```

```

}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



### Data label

You can toggle the visibility of data labels by using the `ShowLabel` property. By default, the data label will be visible.

### ASPX-CS

```

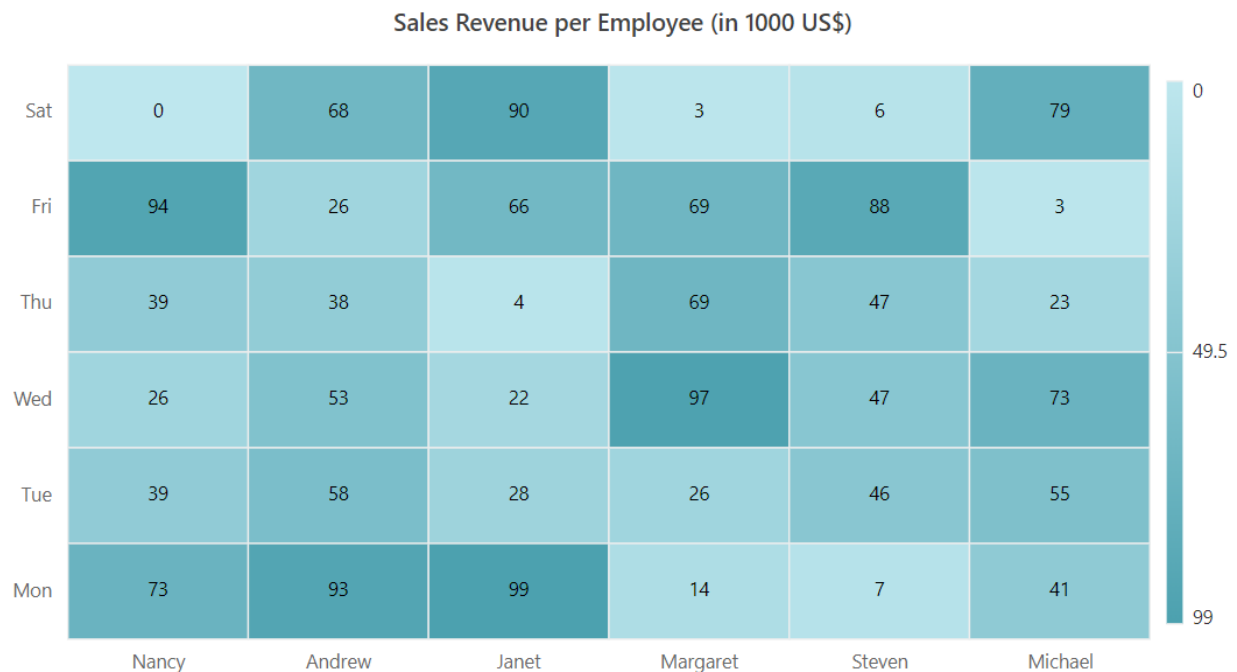
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapCellSettings ShowLabel="true"
  TileType="CellType.Rect"></HeatMapCellSettings>
  <HeatMapMargin Left="15" Right="15" Top="15" Bottom="15"></HeatMapMargin>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{

```

```

int[,] dataSource = new int[,]
{
    {73, 39, 26, 39, 94, 0},
    {93, 58, 53, 38, 26, 68},
    {99, 28, 22, 4, 66, 90},
    {14, 26, 97, 69, 69, 3},
    {7, 46, 47, 47, 88, 6},
    {41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
    HeatMapData = GetDefaultData();
}
}

```



### Text style

You can customize the font family, font size, and color of the data label by using the `HeatMapCellTextStyle` tag in the `HeatMapCellSettings` tag.

### ASPX-CS

```

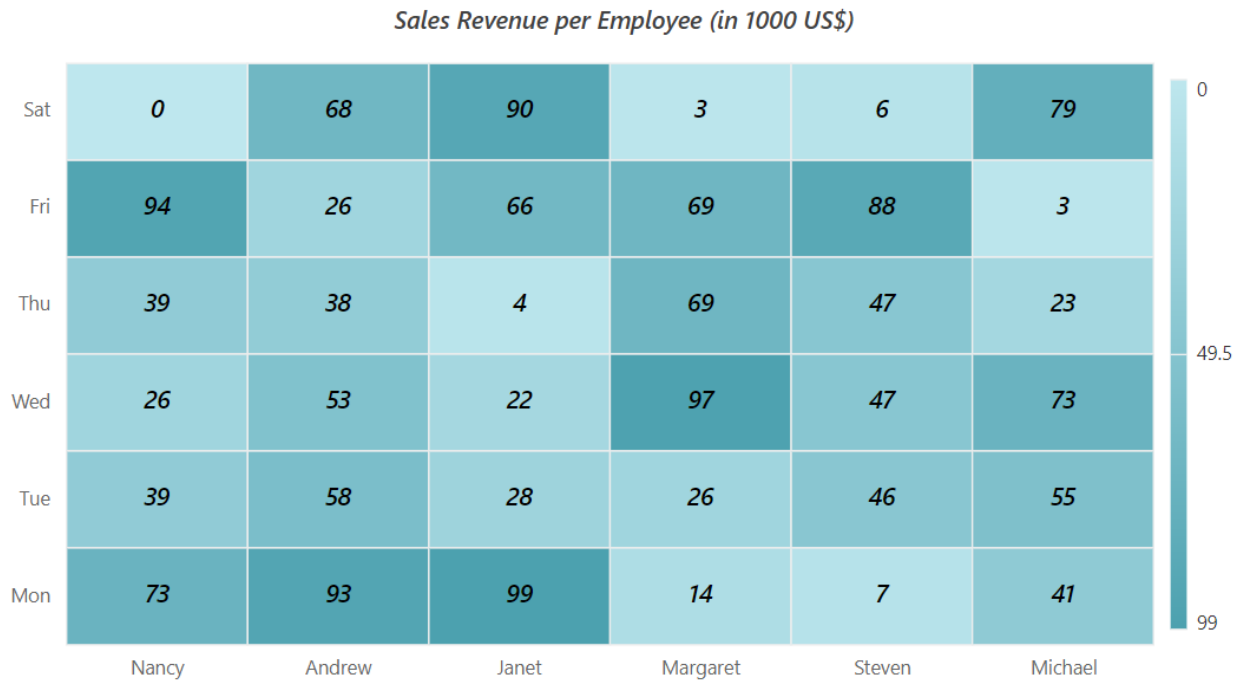
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>

```

```

<HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
<HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
<HeatMapTitleTextStyle Size="15px" FontWeight="500" FontStyle="Italic"
FontFamily="Segoe UI"></HeatMapTitleTextStyle>
</HeatMapTitleSettings>
<HeatMapCellSettings ShowLabel="true" TileType="CellType.Rect">
<HeatMapCellTextStyle Size="15px" FontWeight="500" FontStyle="Italic"
FontFamily="Segoe UI"></HeatMapCellTextStyle>
</HeatMapCellSettings>
<HeatMapMargin Left="15" Right="15" Top="15" Bottom="15"></HeatMapMargin>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



### Format

You can change the format of the data label, such as currency, decimal, percent etc. by using `Format` property.

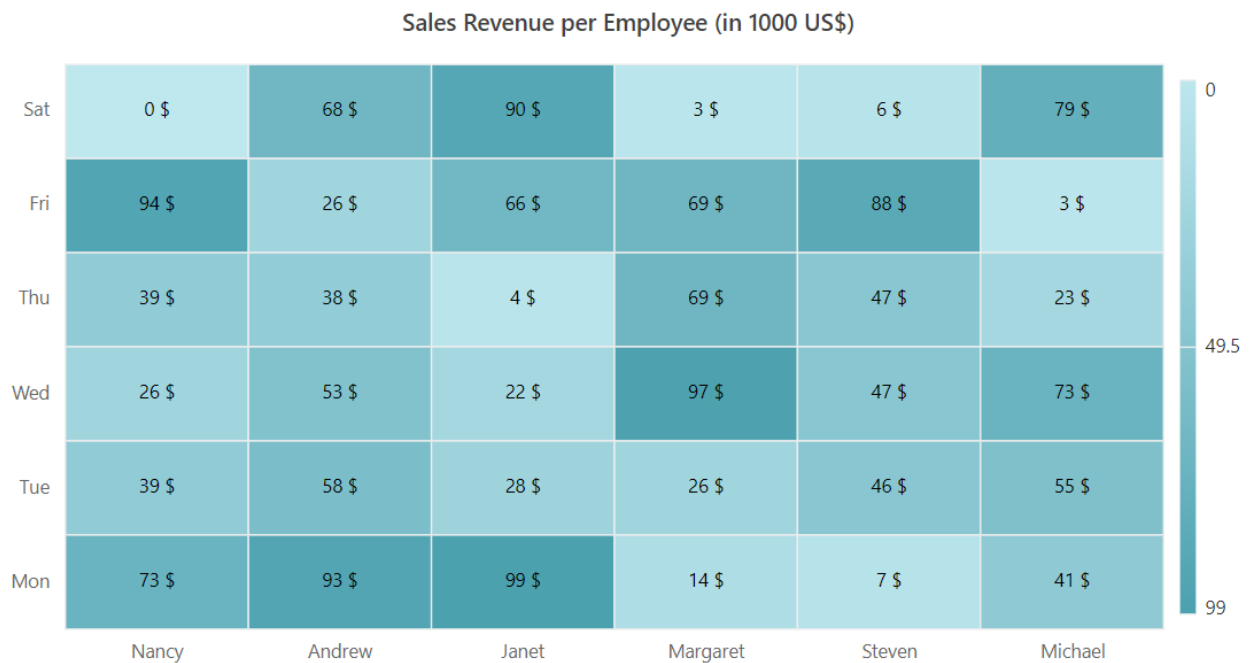
### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapCellSettings ShowLabel="true" TileType="CellType.Rect"
  Format="{value} $"></HeatMapCellSettings>
  <HeatMapMargin Left="15" Right="15" Top="15" Bottom="15"></HeatMapMargin>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
```

```

string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
    "Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
    HeatMapData = GetDefaultData();
}
}

```



## Dimensions in Blazor HeatMap Chart Component

### Size for heat map

You can set the size of heat map directly by using the **Width** and **Height** properties.

*In pixel*

You can set the size for heat map in a pixel.

### ASPX-CS

```

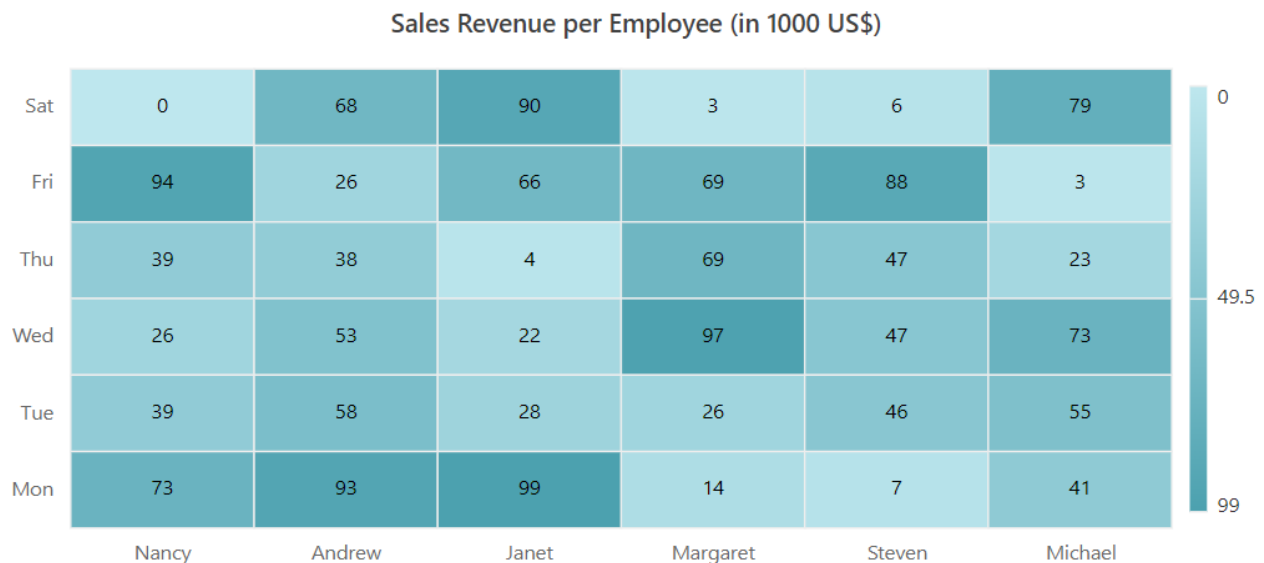
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData" Width="750px" Height="350px">
    <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
    </HeatMapTitleSettings>
    <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
    <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
    <HeatMapCellSettings ShowLabel="true"
        TileType="CellType.Rect"></HeatMapCellSettings>
</SfHeatMap>
@code{
    int[,] GetDefaultData()
    {
        int[,] dataSource = new int[,]
        {

```

```

{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}

```



#### *In percentage*

By setting value in percentage, heat map gets its dimension with respect to its container. For example, when the height is '50%', heat map rendered to half of the container height.

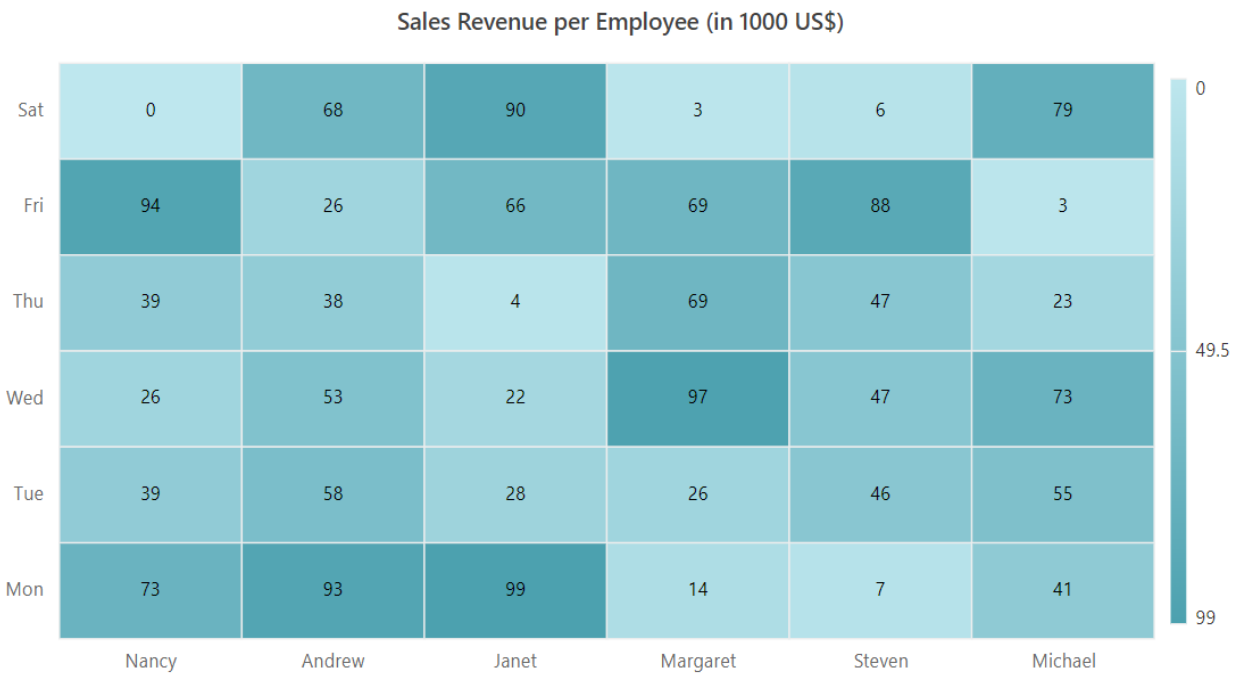
#### **ASPX-CS**

```

@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData" Width=75% Height=75%>
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapCellSettings ShowLabel="true"
  TileType="CellType.Rect"></HeatMapCellSettings>
</SfHeatMap>

```

```
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}
```



### Tooltip in Blazor HeatMap Chart Component

Tooltip is used to provide the details of the heatmap cell, and this can be displayed, while hovering the cursor over the cell or performing tap action in touch devices.

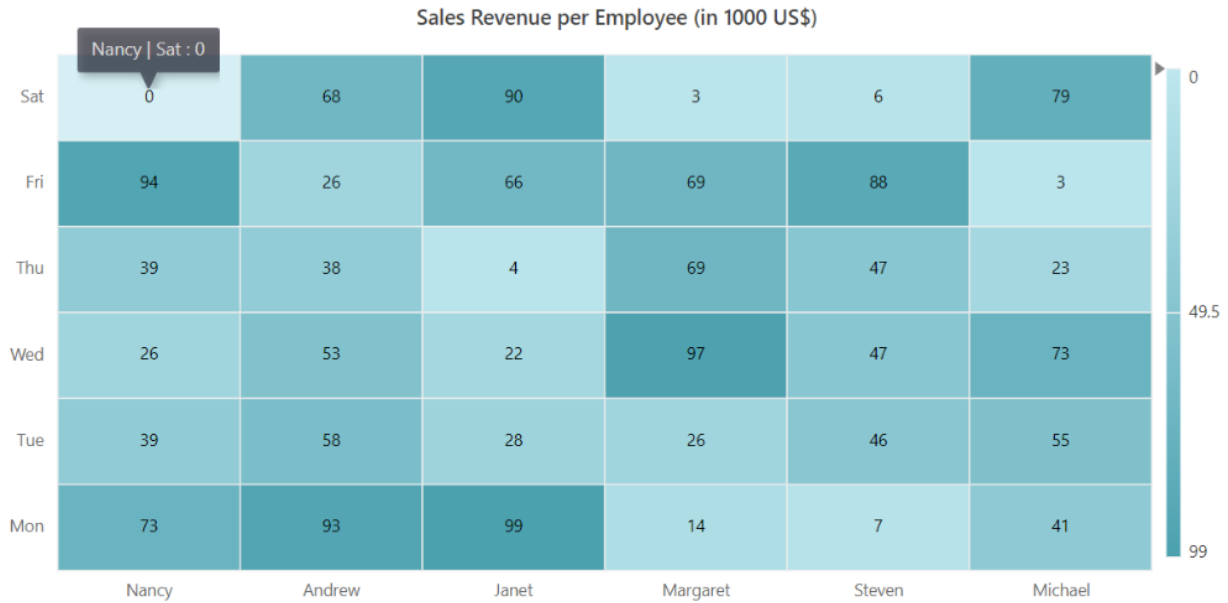


### Default tooltip

You can enable the tooltip by setting the `Enable` property to true inside the `HeatMapTooltipSettings` tag.

### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapCellSettings ShowLabel="true"
  TileType="CellType.Rect"></HeatMapCellSettings>
  <HeatMapTooltipSettings Enable="true"></HeatMapTooltipSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
int[,] dataSource = new int[,]{
{
{73, 39, 26, 39, 94, 0},
{93, 58, 53, 38, 26, 68},
{99, 28, 22, 4, 66, 90},
{14, 26, 97, 69, 69, 3},
{7, 46, 47, 47, 88, 6},
{41, 55, 73, 23, 3, 79}
};
return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
protected override void OnInitialized()
{
HeatMapData = GetDefaultData();
}
}
```



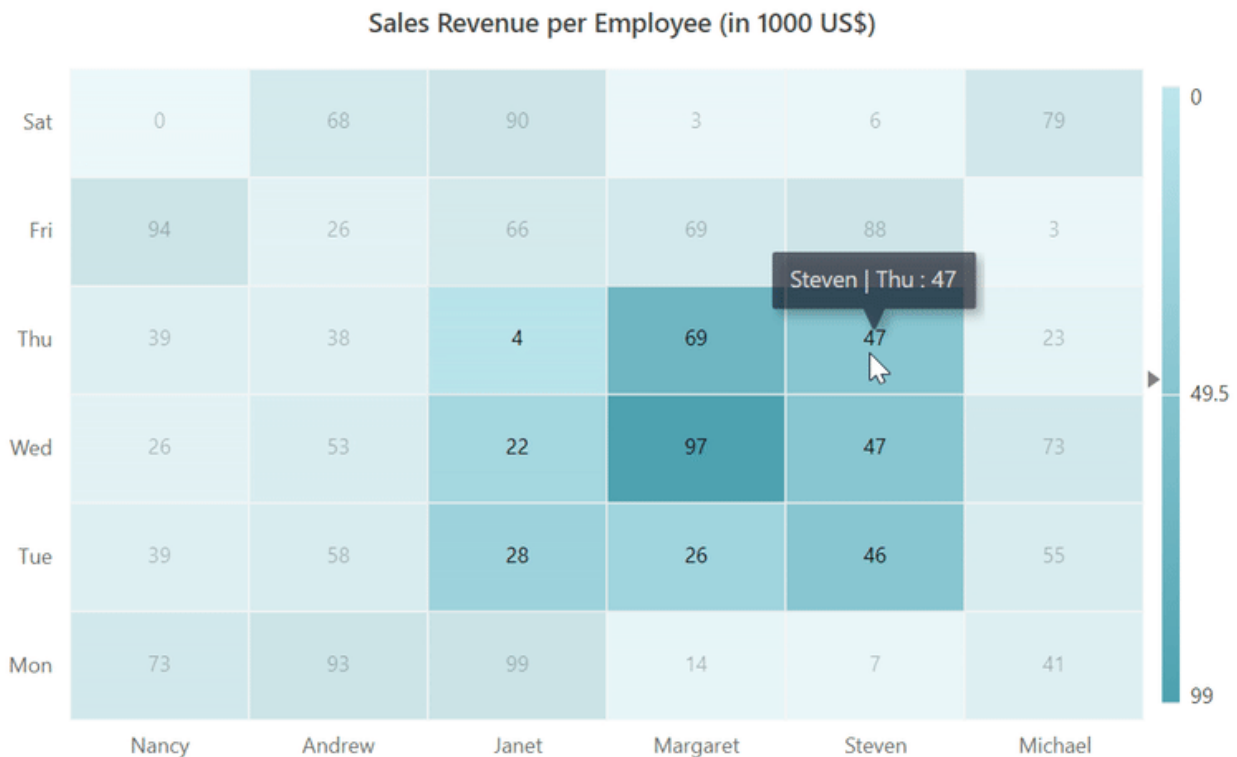
### Selection in Blazor HeatMap Chart Component

In the HeatMap, the cell selection is used to select the single or multiple heat map cells at runtime and get the selected cell details using the `CellSelected` event. You can enable this cell selection using the `AllowSelection` property.

#### ASPX-CS

```
@using Syncfusion.Blazor.HeatMap
<SfHeatMap AllowSelection="true" DataSource="@HeatMapData">
  <HeatMapTitleSettings Text="Sales Revenue per Employee (in 1000 US$)">
  </HeatMapTitleSettings>
  <HeatMapXAxis Labels="@XAxisLabels"></HeatMapXAxis>
  <HeatMapYAxis Labels="@YAxisLabels"></HeatMapYAxis>
  <HeatMapCellSettings ShowLabel="true"
  TileType="CellType.Rect"></HeatMapCellSettings>
</SfHeatMap>
@code{
int[,] GetDefaultData()
{
  int[,] dataSource = new int[,]{
    {73, 39, 26, 39, 94, 0},
    {93, 58, 53, 38, 26, 68},
    {99, 28, 22, 4, 66, 90},
    {14, 26, 97, 69, 69, 3},
    {7, 46, 47, 47, 88, 6},
    {41, 55, 73, 23, 3, 79}
  };
  return dataSource;
}
string[] XAxisLabels = new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael" };
string[] YAxisLabels = new string[] { "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
public object HeatMapData { get; set; }
```

```
protected override void OnInitialized()
{
    HeatMapData = GetDefaultData();
}
}
```



## In-place Editor

<!-- markdownlint-disable MD024 -->

### Getting Started with Blazor In-place Editor Component

This section briefly explains how to include a In-Place Editor 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 the introduction and configuring the common specifications.

#### Importing Syncfusion Blazor component in the application

- Install **Syncfusion.Blazor.InPlaceEditor** 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.

#### 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/blazor.polyfill.min.js"></script>
</environment>
</head>
```

Adding component package to the application

Open ~/\_Imports.razor file and import the **Syncfusion.Blazor.InPlaceEditor** package.

### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
```

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();
        }
    }
}
```

Add In-Place Editor component

To initialize the In-Place Editor component, add the below code to your **Index.razor** view page which is present under ~/Pages folder.

The following code explains how to initialize a simple In-place Editor with TextBox in the Razor page.

#### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.Inputs
<table>
<tr>
<td>
<label class="control-label" style="text-align: left;font-size: 14px;font-weight: 400">
TextBox
</label>
</td>
<td>
<SfInPlaceEditor @bind-Value="@TextValue" TValue="string">
<EditorComponent>
<SfTextBox @bind-Value="@TextValue" Placeholder="Enter employee name"></SfTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
@code {
public string TextValue = "Andrew";
}
```

The type of component editor must be configured in the 'Type' Editor In-place property. Also, should configure the two-way binding between the In-place Editor and its EditorComponent. It's used to update the editor component value into the In-place Editor component.

#### Render In-place Editor with popup

The following code explains how to initialize a simple In-place Editor with popup in the Blazor page.

#### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.DropDowns
<table>
<tr>
<td>
<label class="control-label">
Choose a Country:
</label>
</td>
<td>
<SfInPlaceEditor
Type="Syncfusion.Blazor.InPlaceEditor.InputType.AutoComplete" @bind-Value="@AutoValue" Mode="RenderMode.Popup" TValue="string">
<EditorComponent>
<SfAutoComplete TValue="string" TItem="Countries" @bind-Value="@AutoValue" Placeholder="e.g. Australia" DataSource="@LocalData">
<AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
</EditorComponent>
</td>
</tr>
</table>
```

```

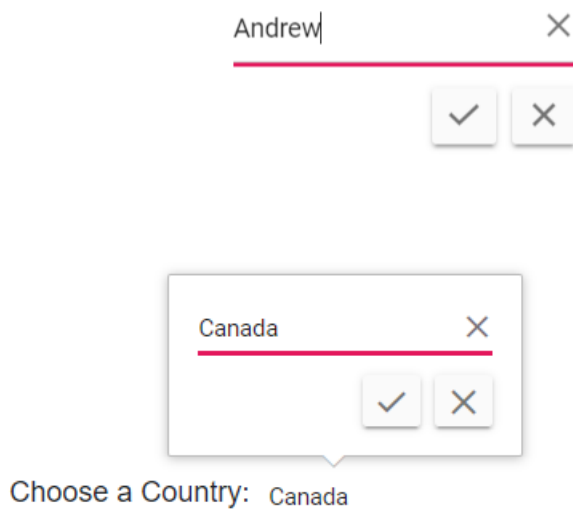
</SfInPlaceEditor>
</td>
</tr>
</table>
@code {
public string AutoValue = "Australia";
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> LocalData = new List<Countries> {
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { 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, run the application.

Output be like the below.



### Configuring DropDownList

You can render the Blazor DropDownList by changing the `Type` property as `DropDownList` and configure `DropDownList` component inside the Editor component.

The following sample demonstrates how to render the `DropDownList` component in the In-place Editor,

### ASPX-CS

```

@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.InPlaceEditor
<SfInPlaceEditor @bind-Value="@DropDownValue"
Type="Syncfusion.Blazor.InPlaceEditor.InputType.DropDownList"
TValue="string">
<EditorComponent>
<SfDropDownList TValue="string" TItem="Games" @bind-Value="@DropDownValue"
Placeholder="Select a game" DataSource="@LocalData">
<DropDownListFieldSettings Value="ID"
Text="Text"></DropDownListFieldSettings>
</SfDropDownList>
</EditorComponent>
</SfInPlaceEditor>
@code {
public string DropDownValue = "Cricket";
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" },
};
}

```

### Integrate DatePicker

You can render the Blazor `DatePicker` by changing the `Type` property as `Date` and configure `DatePicker` component inside the Editor component. Also, configure its properties directly in the `DatePicker` component

The following sample demonstrates how to render the `DatePicker` component in the In-place Editor,

### ASPX-CS

```

@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.Calendars
<SfInPlaceEditor Type="Syncfusion.Blazor.InPlaceEditor.InputType.Date"
TValue="DateTime?" @bind-Value="@DateValue">
<EditorComponent>
<SfDatePicker TValue="DateTime?" @bind-Value="@DateValue"
Placeholder="Choose a Date"></SfDatePicker>
</EditorComponent>
</SfInPlaceEditor>
@code {
public DateTime? DateValue { get; set; } = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, DateTime.Now.Day);
}

```

In the following code, it is configured to render the `DatePicker`, `Dropdownlist` and `Textbox` components.

#### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.Calendars
@using Syncfusion.Blazor.DropDowns
<div id="container" class="control-group">
<h3> Modify Basic Details </h3>
<table style="margin: 10px auto;">
<tr>
<td>Name</td>
<td class='left'>
<SfInPlaceEditor @bind-Value="@TextValue" TValue="string">
<EditorComponent>
<SfTextBox @bind-Value="@TextValue" Placeholder="Enter your
name"></SfTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td>Date of Birth</td>
<td class='left'>
<SfInPlaceEditor Type="Syncfusion.Blazor.InPlaceEditor.InputType.Date"
TValue="DateTime?" @bind-Value="@DateValue">
<EditorComponent>
<SfDatePicker TValue="DateTime?" @bind-Value="@DateValue"
Placeholder="Select date"></SfDatePicker>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td>Gender</td>
<td class='left'>
<SfInPlaceEditor @bind-Value="@DropdownValue"
Type="Syncfusion.Blazor.InPlaceEditor.InputType.DropDownList"
TValue="string">
<EditorComponent>
<SfDropDownList Width="90%" TItem="Gender" TValue="string"
DataSource="@dropdownData" @bind-Value="@DropdownValue">
<DropDownListFieldSettings Text="text"
Value="text"></DropDownListFieldSettings>
</SfDropDownList>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
</div>
<style>
#container {
```



```
text-align: center;
margin-top: 50px;
}
#container table {
width: 400px;
margin: auto;
}
#container table td {
height: 70px;
width: 150px;
}
#container table .left {
text-align: left;
}
</style>
@code {
public DateTime? DateValue { get; set; } = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, DateTime.Now.Day);
public string TextValue = "Andrew";
public string DropdownValue = "Male";
public class Gender
{
public string value { get; set; }
public string text { get; set; }
}
List<Gender> dropdownData = new List<Gender>()
{
new Gender(){ text= "Male" },
new Gender(){ text= "Female" }
};
}
```

Output be like the below.

#### Modify Basic Details

Name	<u>Empty</u>
Date of Birth	<u>4/12/2018</u>
Gender	<u>Male</u>

### Submitting data to the server (save)

You can submit editor value to the server by configuring the **SaveUrl**, **Adaptor** and **PrimaryKey** property.

Property	Usage
<b>SaveUrl</b>	Gets the URL for server submit action.
<b>Adaptor</b>	Specifies the adaptor type that is used by DataManager to communicate with DataSource.
<b>PrimaryKey</b>	Defines the unique primary key of editable field which can be used for saving data in the data-base.

The **PrimaryKey** property is mandatory. If it is not set, edited data are not sent to the server.

### Refresh In-place Editor with modified value

The edited data is submitted to the server and you can see the new values getting reflected in the In-place Editor.

### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.InPlaceEditor
<div id="container">
<div class="control-group">
Best Employee of the year:
<SfInPlaceEditor @ref="InPlaceObj" PrimaryKey="Employee" Name="Employee"
Adaptor="Adaptors.UrlAdaptor"
SaveUrl="https://ej2services.syncfusion.com/production/web-
services/api/Editor/UpdateData"
Type="Syncfusion.Blazor.InPlaceEditor.InputType.DropDownList" @bind-
Value="@DropDownValue" TValue="string">
<EditorComponent>
<SfDropDownList TValue="string" TItem="Employees" Placeholder="Select
employee" PopupHeight="200px" DataSource="@LocalData">
<DropDownListFieldSettings Value="ID"
Text="Text"></DropDownListFieldSettings>
</SfDropDownList>
</EditorComponent>t
<InPlaceEditorEvents Created="@OnCreate" OnActionSuccess="@OnSuccess"
TValue="string"></InPlaceEditorEvents>
</SfInPlaceEditor>
</div>
<table style="margin:60px auto;width:25%">
<tr>
<td style="text-align: left">
Old Value :
</td>
<td id="oldValue" style="text-align: left">
@PreviousValue
</td>
</tr>
<tr>
<td style="text-align: left">
New Value :
```

```

</td>
<td id="newValue" style="text-align: left">
@CurrentValue
</td>
</tr>
</table>
</div>
<style>
.e-inplaceeditor {
min-width: 200px;
text-align: left;
}
#container .control-group {
text-align: center;
margin: 100px auto;
}
</style>
@code {
SfInPlaceEditor<string> InPlaceObj;
public string PreviousValue { get; set; }
public string DropdownValue = "Andrew";
public string CurrentValue { get; set; }
public class Employees
{
public string ID { get; set; }
public string Text { get; set; }
}
List<Employees> LocalData = new List<Employees> {
new Employees() { ID= "Andrew", Text= "Andrew" },
new Employees() { ID= "Margaret Hamilit", Text= "Margaret Hamilit" },
new Employees() { ID= "Fuller", Text= "Fuller" },
new Employees() { ID= "John Smith", Text= "John Smith" },
new Employees() { ID= "Victoria", Text= "Victoria" },
new Employees() { ID= "David", Text= "David" },
new Employees() { ID= "Johnson", Text= "Johnson" },
new Employees() { ID= "Rosy", Text= "Rosy"}
};
public void OnCreate(Object args)
{
this.CurrentValue = this.DropdownValue;
}
public void OnSuccess(ActionEventArgs<string> args)
{
this.PreviousValue = this.CurrentValue;
this.CurrentValue = args.Value;
}
}

```

The output will be as follows.

Best Employee of the year: Margaret Hamilt

Old Value : Andrew Fuller  
New Value : Margaret Hamilt

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](#)

### List of Components in Blazor In-place Editor Component

In-place Editor renders various components based on the **Type** property and the Editor components should be rendered inside the In-place Editor. Also, need to configure the two-way binding between the In-place Editor and its EditorComponent. It is used to update the EditorComponent value into the In-place Editor component.

The following table explains Editor components name and their configurations.

<a href="#">AutoComplete</a> (AutoComplete)	<a href="#">TextBox</a> (Text)	
<a href="#">ComboBox</a> (ComboBox)	<a href="#">DatePicker</a> (Date)	
<a href="#">MultiSelect</a> (MultiSelect)	<a href="#">DateTimePicker</a> (DateTime)	
<a href="#">TimePicker</a> (Time)	<a href="#">DropDownList</a> (DropDownList)	
<a href="#">DateRangePicker</a> (DateRange)	<a href="#">MaskedTextBox</a> (Mask)	
<a href="#">Slider</a> (Slider)	<a href="#">NumericTextBox</a> (Numeric)	
<a href="#">RichTextEditor</a> (RichTextEditor)	<a href="#">ColorPicker</a> (Color)	

The following example demonstrates how to render the Editor components in the In-place Editor,

#### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.Calendars;
@using Syncfusion.Blazor.DropDowns;
@using Syncfusion.Blazor.RichTextEditor;
<h3> Built-in Controls </h3>
<table class="table-section">
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> DatePicker
</td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
```

```

<SfInPlaceEditor Type="Syncfusion.Blazor.InPlaceEditor.InputType.Date"
TValue="DateTime?" @bind-Value="@DateValue1">
<EditorComponent>
<SfDatePicker TValue="DateTime?" @bind-Value="@DateValue1"
Placeholder="Select a date"></SfDatePicker>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
DateTimePicker </td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor Type="Syncfusion.Blazor.InPlaceEditor.InputType.DateTime"
TValue="DateTime?" @bind-Value="@DateValue3">
<EditorComponent>
<SfDateTimePicker Placeholder="Select a date and time" TValue="DateTime?"
@bind-Value="@DateValue3"></SfDateTimePicker>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> DropDownList
</td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor @bind-Value="@DropdownValue"
Type="Syncfusion.Blazor.InPlaceEditor.InputType.DropDownList" Mode="@Mode"
TValue="string">
<EditorComponent>
<SfDropDownList TValue="string" TItem="Countries" @bind-
Value="@DropdownValue" DataSource="@Country">
<DropDownListFieldSettings Text="Name"
Value="Code"></DropDownListFieldSettings>
</SfDropDownList>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> MaskedTextBox
</td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor Type="Syncfusion.Blazor.InPlaceEditor.InputType.Mask"
@bind-Value="@MaskValue" TValue="string">
<EditorComponent>
<SfMaskedTextBox Mask="000-000-0000" @bind-
Value="@MaskValue"></SfMaskedTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
NumericTextBox </td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">

```

```

<SfInPlaceEditor Type="Syncfusion.Blazor.InPlaceEditor.InputType.Numeric"
@bind-Value=@NumericValue TValue="double">
<EditorComponent>
<SfNumericTextBox TValue="double" @bind-Value=@NumericValue Format="c2"
Placeholder="Currency format"></SfNumericTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> TextBox </td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor @bind-Value="@TextValue" TValue="string"
Type="Syncfusion.Blazor.InPlaceEditor.InputType.Text">
<EditorComponent>
<SfTextBox @bind-Value="@TextValue" Placeholder="Enter employee
name"></SfTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
<h3> Injectable Controls </h3>
<table class="table-section">
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> AutoComplete
</td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor
Type="Syncfusion.Blazor.InPlaceEditor.InputType.AutoComplete" @bind-
Value="@AutoValue" Mode="@Mode" TValue="string">
<EditorComponent>
<SfAutoComplete TValue="string" TItem="Countries" @bind-Value="@AutoValue"
DataSource="@Country" Autofill=true>
<AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> ColorPicker
</td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor Type="Syncfusion.Blazor.InPlaceEditor.InputType.Color"
@bind-Value="colorValue" TValue="string">
<EditorComponent>
<SfColorPicker @bind-Value="colorValue"></SfColorPicker>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> ComboBox
</td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">

```

```

<SfInPlaceEditor Type="Syncfusion.Blazor.InPlaceEditor.InputType.ComboBox"
@bind-Value="@ComboValue" TValue="string">
<EditorComponent>
<SfComboBox TValue="string" @bind-Value="@ComboValue" TItem="Countries"
DataSource="@Country">
<ComboBoxFieldSettings Text="Name" Value="Code"></ComboBoxFieldSettings>
</SfComboBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
DateRangePicker </td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor Type="Syncfusion.Blazor.InPlaceEditor.InputType.DateRange"
TValue="DateTime[]" @bind-Value="@DateRangeValue">
<EditorComponent>
<SfDateRangePicker StartDate="@DateValue2"
EndDate="@DateValue3"></SfDateRangePicker>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> MultiSelect
</td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor
Type="Syncfusion.Blazor.InPlaceEditor.InputType.MultiSelect" @bind-
Value="@MultiValue" TValue="string[]">
<EditorComponent>
<SfMultiSelect TValue="string[]" @bind-Value="@MultiValue" TItem="Countries"
DataSource="@Country" AllowFiltering="true">
<MultiSelectFieldSettings Text="Name"
Value="Code"></MultiSelectFieldSettings>
</SfMultiSelect>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
RichTextEditor </td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor
Type="Syncfusion.Blazor.InPlaceEditor.InputType.RichTextEditor" @bind-
Value="@value" TValue="string">
<EditorComponent>
<SfRichTextEditor @bind-Value="@value">
</SfRichTextEditor>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> Slider </td>

```

```

<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor @bind-Value="@sliderValue"
Type="Syncfusion.Blazor.InPlaceEditor.InputType.Slider" TValue="double">
<EditorComponent>
<SfSlider @bind-Value="@sliderValue"></SfSlider>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> TimePicker
</td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor Mode="@Mode"
Type="Syncfusion.Blazor.InPlaceEditor.InputType.Time" TValue="DateTime?"
@bind-Value="@DateValue2">
<EditorComponent>
<SfTimePicker TValue="DateTime?" @bind-Value="@DateValue2"
Placeholder="Select a time"></SfTimePicker>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
<style>
body {
padding: 20px 0
}
.control-title {
font-weight: 600;
padding-right: 20px;
}
.control-title {
width: 50%;
}
td {
height: 80px;
}
tr td:first-child {
text-align: right;
}
tr td:last-child {
text-align: left;
}
.table-section {
margin: 0 auto;
}
h3 {
text-align: center;
}
</style>
@code {
private string value { get; set; } = "syncfusion";
private DateTime? DateValue1 = new DateTime(2017, 05, 23);
private DateTime? DateValue2 = new DateTime(2017, 05, 23);
private DateTime? DateValue3 = new DateTime(2017, 05, 23);

```



```

private DateTime[] DateRangeValue = new DateTime[2] { new DateTime(2017, 05, 23), new DateTime(2017, 07, 05) };
private object DateData = new { placeholder = "Select a date" };
private object TimeData = new { placeholder = "Select a time" };
private object DateTimeData = new { placeholder = "Select a date and time" };
private object DateRangeData = new { placeholder = "Select a date range" };
private string TextValue = "Andrew";
private string MaskValue = "012-345-6789";
private double NumericValue = 100;
private double sliderValue { get; set; } = 30;
private string DropdownValue = "CA";
private string AutoValue = "Australia";
private string ComboValue = "Finland";
private string[] MultiValue = new string[] { "AU", "BM" };
public string DropMode { get; set; } = "Inline";
private string colorValue { get; set; } = "#0db1e7";
private RenderMode Mode = RenderMode.Inline;
public class Countries
{
    public string Name { get; set; }
    public string Code { get; set; }
}
private 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.

**Built-in Controls**

DatePicker	11/22/2018
DateTimePicker	11/21/2018 1:30 PM
DropDownList	logis
MaskedTextBox	242-423-423
NumericTextBox	2.00
TextBox	Andrew ffsafsf

**Injectable Controls**

AutoComplete	React
ColorPicker	#244a8d
ComboBox	ionic
DateRangePicker	12/12/2018 - 12/20/2018
MultiSelect	Android,React,Ionic,TypeScript
RTE	afasfsfsf
Slider	34
TimePicker	2:30 AM

See Also

- [HTML5 components](#)

## Configuration in Blazor In-place Editor Component

### Rendering modes

This section explains the supported rendering modes of the In-place Editor. The possible Rendering modes are:

- Popup
- Inline

---

By default, **Inline** mode will be rendered when opening an editor.

---

- For **Popup** mode, the editable container displays as like tooltip or popover above the element.
- For **Inline** mode, the editable container displays instead of the element. To render **Inline** mode while opening the editor, specify **Mode** as **Inline**.

In the following code block, the In-place Editor renders with **Inline** mode. You can dynamically switch into another mode by changing the drop-down item value.

#### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Inputs
<table class="table-section">
<tr>
<td> Mode: </td>
<td>
<SfDropDownList Width="90%" TItem="InplaceModes" TValue="string"
DataSource="@ModeData" @bind-Value="@DropDownValue">
<DropDownListEvents TValue="string" TItem="InplaceModes"
ValueChange="@OnChange"></DropDownListEvents>
<DropDownListFieldSettings Text="text"
Value="value"></DropDownListFieldSettings>
</SfDropDownList>
</td>
</tr>
<tr>
<td class="sample-td">Enter your name: </td>
<td class="sample-td">
<SfInPlaceEditor @bind-Value="@TextValue" Mode="@InplaceMode"
TValue="string">
<EditorComponent>
<SfTextBox @bind-Value="@TextValue" Placeholder="Enter some
text"></SfTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
<style>
.table-section {
margin: 0 auto;
}
tr td:first-child {
text-align: right;
padding-right: 20px;
}
.sample-td {
padding-top: 10px;
min-width: 230px;
height: 100px;
}
</style>
@code {
public RenderMode InplaceMode = RenderMode.Inline;
public string TextValue = "Andrew";
public string DropdownValue = "Inline";
```

```

public class InplaceModes
{
    public string value { get; set; }
    public string text { get; set; }
}
List<InplaceModes> ModeData = new List<InplaceModes>()
{
    new InplaceModes() { value= "Inline", text= "Inline" },
    new InplaceModes() { value= "Popup", text= "Popup" }
};
private void OnChange(Syncfusion.Blazor.DropDowns.ChangeEventArgs<string,
InplaceModes> args)
{
    this.InplaceMode = (args.Value.ToString() == "Popup" ? RenderMode.Popup :
RenderMode.Inline);
    this.StateHasChanged();
}
}

```

### Pop-up customization

In-place Editor popup mode can be customized by using the `InPlaceEditorPopupSettings` tag.

Popup mode is rendered by using the Blazor Tooltip component, so you can use the tooltip properties and events to customize the behavior of popup via the `InPlaceEditorPopupSettings` by configuring tooltip properties.

---

For more details, refer to the tooltip documentation [section](#).

---

### Event actions for editing

The event action of the editor will be enabled in the edit mode based on the `EditableOn` property. By default, `Click` is enabled.

The following options are also supported:

- **Click:** The editor will be opened on single-click actions.
- **DoubleClick:** The editor will be opened on double-click actions and it is not applicable for the edit icon.
- **EditIconClick:** Disables the editing of event action of input and allows the users to edit only through edit icon.

---

In-place Editor gets focus by pressing the `tab` key from the previous focusable DOM element. The editor can be opened by pressing the `enter` key.

---

In the following code block, when switching the drop-down item, the selected value is assigned to the `EditableOn` property. The editor will be opened when you double click on the input.

### ASPX-CS

```

@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Inputs
<table class="table-section">
<tr>
<td>Choose Editable Type: </td>

```

```

<td>
<SfDropDownList Width="90%" TValue="string" TItem="InplaceEditableModes"
DataSource="@EditableData" @bind-Value="@DropDownValue">
<DropDownListEvents TValue="string" TItem="InplaceEditableModes"
ValueChange="@OnChange"></DropDownListEvents>
<DropDownListFieldSettings Text="text"
Value="value"></DropDownListFieldSettings>
</SfDropDownList>
</td>
</tr>
<tr>
<td class="sample-td">Enter your name: </td>
<td class="sample-td">
<SfInPlaceEditor @bind-Value="@TextValue" EditableOn="EditableOn"
SubmitOnEnter="true" TValue="string">
<EditorComponent>
<SfTextBox @bind-Value="@TextValue" Placeholder="Enter some
text"></SfTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
<style>
.table-section {
margin: 0 auto;
}
tr td:first-child {
text-align: right;
padding-right: 20px;
}
.sample-td {
padding-top: 10px;
min-width: 230px;
height: 100px;
}
</style>
@code {
public string TextValue = "Andrew";
public string DropDownValue = "Click";
public EditableType EditableOn = EditableType.Click;
public class InplaceEditableModes
{
public string value { get; set; }
public string text { get; set; }
}
private List<InplaceEditableModes> EditableData = new
List<InplaceEditableModes>()
{
new InplaceEditableModes(){ value= "Click", text= "Click" },
new InplaceEditableModes(){ value= "Double Click", text= "Double Click" },
new InplaceEditableModes(){ value= "Edit Icon Click", text= "Edit Icon
Click" }
};
private void OnChange(Syncfusion.Blazor.DropDowns.ChangeEventArgs<string,
InplaceEditableModes> args)
{

```

```

if (args.Value != null)
{
    if (args.Value.ToString() == "Click")
    {
        this.EditableOn = EditableType.Click;
    }
    else if (args.Value.ToString() == "Double Click")
    {
        this.EditableOn = EditableType.DoubleClick;
    }
    else
    {
        this.EditableOn = EditableType.EditIconClick;
    }
    this.StateHasChanged();
}
}
}

```

### Action on focus out

Action will be performed when the user clicks outside the container. That means, focusing out of the editable content can be handled by the `ActionOnBlur` property. By default, `Submit` is enabled.

It also has the following options.

- **Cancel:** Cancels the editing and resets the old content.
- **Submit:** Submits the edited content to the server.
- **Ignore:** No action will be performed with this type.

In the following code block, when switching drop-down item, the selected value assigned to the `ActionOnBlur` property.

### ASPX-CS

```

@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Inputs
<table class="table-section">
<tr>
<td> ActionOnBlur: </td>
<td>
<SfDropDownList Width="90%" TItem="DropDownFields" TValue="string"
DataSource="@Modes" @bind-Value="@DropdownValue">
<DropDownListEvents TValue="string" TItem="DropDownFields"
ValueChange="@OnChange"></DropDownListEvents>
<DropDownListFieldSettings Text="Text"
Value="Text"></DropDownListFieldSettings>
</SfDropDownList>
</td>
</tr>
<tr>
<td class="sample-td">Enter your name: </td>
<td class="sample-td">
<SfInPlaceEditor @bind-Value="@TextValue" ActionOnBlur="OnBlur"
SubmitOnEnter="true" TValue="string">

```

```

<EditorComponent>
<SfTextBox @bind-Value="@TextValue" Placeholder="Enter some
text"></SfTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
<style>
.table-section {
margin: 0 auto;
}
tr td:first-child {
text-align: right;
padding-right: 20px;
}
.sample-td {
padding-top: 10px;
min-width: 230px;
height: 100px;
}
</style>
@code {
public string TextValue = "Andrew";
public string DropdownValue = "Submit";
public ActionBlur OnBlur = ActionBlur.Ignore;
public class DropDownFields
{
public string Text { get; set; }
}
public List<DropDownFields> Modes = new List<DropDownFields>
{
new DropDownFields() { Text= "Submit" },
new DropDownFields() { Text= "Cancel" },
new DropDownFields() { Text= "Ignore" }
};
private void OnChange(Syncfusion.Blazor.DropDowns.ChangeEventArgs<string,
DropDownFields> args)
{
if (args.Value.ToString() == "Submit")
{
this.OnBlur = ActionBlur.Submit;
}
else if (args.Value.ToString() == "Cancel")
{
this.OnBlur = ActionBlur.Cancel;
}
else
{
this.OnBlur = ActionBlur.Ignore;
}
this.StateHasChanged();
}
}

```

### Display modes

By default, the In-place Editor input element is highlighted with a dotted underline. To remove dotted underline from input element, add {`"data-underline", "false"`} attribute at In-place Editor root element.

In the following code block, indicates the intractable and normal display modes with different examples.

### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.Inputs
<h4>Example of data-underline attribute</h4>
<table class="table-section">
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> Intractable
UI </td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor @bind-Value="@TextValue" TValue="string"
SubmitOnEnter="true">
<EditorComponent>
<SfTextBox @bind-Value="@TextValue" Placeholder="Enter some
text"></SfTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
<tr>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> Normal UI
</td>
<td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<SfInPlaceEditor @bind-Value="@TextValue" TValue="string"
SubmitOnEnter="true" @attributes="htmlAttribute">
<EditorComponent>
<SfTextBox @bind-Value="@TextValue" Placeholder="Enter some
text"></SfTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
<style>
.table-section {
margin: 0 auto;
}
td {
padding: 20px 0;
min-width: 230px;
height: 100px;
}
.control-title {
font-weight: 600;
padding-right: 20px;
text-align: right;
width: 50%;
}
h4 {
text-align: center;
```



```
}
</style>
@code {
    public string TextValue = "Andrew";
    Dictionary<string, object> htmlAttribute = new Dictionary<string, object>()
    {
        {"data-underline", "false" }
    };
}
```

The output will be as follows.

### Example of data-underline attribute

**Intractable UI**      Andrew  
-----

**Normal UI**      Andrew

See Also

- [Disable the editor](#)
- [Animate the editor during popup mode](#)

### Buttons in Blazor In-place Editor Component

The In-place Editor has an option to save and cancel using buttons. The `InPlaceEditorSaveButton` and `InPlaceEditorCancelButton` tags accept the button properties for customizing the save and cancel button.

Buttons can be shown or hidden by setting a Boolean value to the `ShowButtons` property.

---

Without buttons, the value will be processed in the following ways.

---

- **ActionOnBlur**: By clicking outside, the editor component gets focus out and do an action based on this property value.
- **SubmitOnEnter**: Pressing the `Enter` key performs the submit action if this property is set to `true`.

In the following sample, the `Content` and `CssClass` properties of the `Button` value are assigned to the `InPlaceEditorSaveButton` and `InPlaceEditorCancelButton` properties to customize its appearance. Also, check or uncheck the checkbox buttons rendered or removed from the editor.

---

For more details about buttons, refer this documentation [section](#).

---

**ASPX-CS**

```

@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.Buttons
<table class="table-section">
<tr>
<td> Submit on Enter: </td>
<td>
<SfCheckBox @bind-Checked="SubmitOnEnter" Label="Show"
ValueChange="OnChange" TChecked="bool"></SfCheckBox>
</td>
</tr>
<tr>
<td class="sample-td">Enter your name: </td>
<td class="sample-td">
<SfInPlaceEditor @bind-Value="@TextValue" SubmitOnEnter="SubmitOnEnter"
TValue="string">
<EditorComponent>
<SfTextBox @bind-Value="@TextValue" Placeholder="Enter some
text"></SfTextBox>
</EditorComponent>
<InPlaceEditorSaveButton Content="OK" CssClass="e-
outline"></InPlaceEditorSaveButton>
<InPlaceEditorCancelButton Content="Cancel" CssClass="e-
outline"></InPlaceEditorCancelButton>
</SfInPlaceEditor>
</td>
</tr>
</table>
<style>
.table-section {
margin: 0 auto;
}
tr td:first-child {
text-align: right;
padding-right: 20px;
}
.sample-td {
padding-top: 10px;
min-width: 230px;
height: 100px;
}
</style>
@code {
public bool SubmitOnEnter { get; set; } = true;
public string TextValue = "Andrew";
private void OnChange(Syncfusion.Blazor.Buttons.ChangeEventArgs<bool>
args)
{
this.SubmitOnEnter = args.Checked;
this.StateHasChanged();
}
}

```

The output will be as follows.

Enter your name:  ✕

See Also

- [In-place editor buttons](#)

### Server Actions in Blazor In-place Editor Component

By passing In-place Editor component value to the server, the `PrimaryKey` property value must require, otherwise action not performed for remote data.

If the `SaveURL` property value is empty, data passing will handled at local and also the `OnActionSuccess` event will trigger with `null` as argument value.

---

The following arguments are passed to the server when the submit actions are performed.

---

Arguments	Explanations	
Value	For processing edited value, like DB value updating.	
PrimaryKey	For value mapping to the server, like selecting DB.	

Find the following sample server codes for defining models and controller functions to configure processing data.

#### CSHARP

```
public class SubmitModel
{
    public string Name { get; set; }
    public string PrimaryKey { get; set; }
    public string Value { get; set; }
}
```

#### CSHARP

```
public IEnumerable<SubmitModel> UpdateData([FromBody] SubmitModel value)
{
    // User can process data
    return value;
}
```

- Server actions successfully done, the `OnActionSuccess` event will be fired with returned server data.
- If the server is not responding, the `OnActionFailure` event will be fired with data, but value not updated in the Editor.

In the following sample, the `OnActionSuccess` event will trigger once the value submitted successfully into the server.

### CSHARP

```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor;
<table class="table-section">
<tr>
<td class="sample-td"> Enter your name: </td>
<td class="sample-td">
<SfInPlaceEditor
Type="Syncfusion.Blazor.InPlaceEditor.InputType.MultiSelect" @bind-
Value="@MultiSelectValue" SubmitOnEnter="true" Name="Skill"
SaveUrl="https://ej2services.syncfusion.com/production/web-
services/api/Editor/UpdateData" PrimaryKey="FrameWork"
Adaptor="Adaptors.UrlAdaptor" TValue="string[]">
<EditorComponent>
<SfMultiSelect Placeholder="Select skill" Mode="VisualMode.Box" @bind-
Value="@MultiSelectValue" DataSource="@DataSource">
<MultiSelectFieldSettings Text="Text" Value="ID"></MultiSelectFieldSettings>
</SfMultiSelect>
</EditorComponent>
<InPlaceEditorEvents OnActionSuccess="OnSuccess"
TValue="string"></InPlaceEditorEvents>
</SfInPlaceEditor>
</td>
</tr>
</table>
<style>
.table-section {
margin: 0 auto;
}
tr td:first-child {
text-align: right;
padding-right: 20px;
}
.sample-td {
padding-top: 10px;
min-width: 230px;
height: 100px;
}
</style>
@code {
public string[] MultiSelectValue = new string[] { "JavaScript", "jQuery" };
public string[] DataSource = new string[] { "Android", "JavaScript",
"jQuery", "TypeScript", "Angular", "React", "Vue", "Ionic" };
public class Program
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<Program> Games = new List<Program>()
{
new Program() { ID= "Ad", Text= "Android" },
new Program() { ID= "Js", Text= "JavaScript" },
}
```

```

new Program() { ID= "Jq", Text= "jQuery" },
new Program() { ID= "Ts", Text= "TypeScript" },
new Program() { ID= "Ag", Text= "Angular" },
new Program() { ID= "Re", Text= "React" },
new Program() { ID= "Vu", Text= "Vue" },
new Program() { ID= "Io", Text= "Ionic" }
};
public void OnSuccess (ActionEventArgs<string> args)
{
    Console.WriteLine("Event is triggered");
}
}

```

## Data Binding in Blazor In-place Editor Component

The Razor components load the data either from local data sources or remote data services using the `DataSource` property and it supports the data type of an array or `DataManager`. Also supports different kind of data services such as OData, OData V4, Web API, and data formats such as XML, JSON, JSONP with the help of `DataManager` adaptors.

### Local

To bind local data to the Razor components, you can assign an array of object or string to the `DataSource` property. The local data source can also be provided as an instance of the `DataManager`.

### CSHARP

```

@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.InPlaceEditor
<div id="container">
<span class="content-title"> Select customer name: </span>
<SfInPlaceEditor @bind-Value="@DropDownValue"
Type="Syncfusion.Blazor.InPlaceEditor.InputType.DropDownList"
TValue="string">
<EditorComponent>
<SfDropDownList TItem="string" TValue="string" Placeholder="Select a
customer" @bind-Value="@DropDownValue" DataSource="@DataManager">
</SfDropDownList>
</EditorComponent>
</SfInPlaceEditor>
</div>
<style>
#container {
display: flex;
justify-content: center;
align-items: center;
height: 80px;
}
#element {
width: 150px;
}
.content-title {
font-weight: 500;
margin-right: 20px;
display: flex;
align-items: center;
}

```

```

</style>
@code {
    public string DropDownValue = "Maria Anders";
    public static string[] DataManager = new string[] { "Maria Anders", "Ana Trujillo", "Antonio Moreno", "Thomas Hardy", "Chiristina Berglund", "Hanna Moos" };
}

```

The output will be as follows.

Select customer name: Maria Anders

## Integrate HTML5 Components in Blazor In-place Editor Component

The In-place Editor supports adding HTML5 input components using the `InPlaceEditorTemplate` property. The Template property can be given as follows.

### ASPX-CS

```

<InPlaceEditorTemplate>
<input id="date" type="text" />
</InPlaceEditorTemplate>

```

In Template mode, the `Value` property cannot be handled by the In-place Editor component. So, before sending a value to the server, you need to modify the `OnActionSuccess` event, otherwise, an empty string will be passed.

In the following template sample, before submitting data to the server, the event argument and `Value` property contents are updated in the `OnActionSuccess` event handler.

### ASPX-CS

```

@using Syncfusion.Blazor.InPlaceEditor
<div id='container'>
    <span class="content-title"> Select date: </span>
    <SfInPlaceEditor @ref="InplaceditorObj" EmptyText="Value" TValue="string"
    @bind-Value="@inplaceValue" Mode="RenderMode.Inline"
    Type="InputType.Template">
        <InPlaceEditorTemplate>
            <input @bind-value="@inplaceValue" id="date" type="text" />
        </InPlaceEditorTemplate>
        <InPlaceEditorEvents TValue="string"
        OnActionSuccess="OnSuccess"></InPlaceEditorEvents>
    </SfInPlaceEditor>
</div>
<style>
#container {
display: flex;
justify-content: center;
}
#InplaceDate {
width: 150px;
}

```

```
.content-title {
font-weight: 500;
margin-right: 20px;
display: flex;
align-items: center;
}
</style>
@code {
SfInPlaceEditor<string> InplaceditorObj;
public string inplaceValue { get; set; } = "syncfusion";
private void OnSuccess(ActionEventArgs<string> args)
{
inplaceValue = args.Value;
}
}
```

The output will be as follows.

Select date: 2018-05-23

See Also

- [Built-in Controls](#)

### Validation in Blazor In-place Editor Component

Now, validation can be done by using the `EditForm` validation on the server-side. We need to handle the validation from the application level and the custom validation can also be achieved by using this.

Please refer to the following link for more details, [EditForm Validation](#). Validation for the `TextBox` is achieved in the following sample using the `EditForm` validation with a custom error message and validation rules.

#### ASPX-CS



```
@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.Calendars
@using System.ComponentModel.DataAnnotations;
<table class="table-section">
<tr>
<td class="sample-td"> Enter your name: </td>
<td class="sample-td">
<SfInPlaceEditor @bind-Value="exampleModel.Name"
Type="Syncfusion.Blazor.InPlaceEditor.InputType.Text" TValue="string">
<EditorComponent>
<EditForm Model="@exampleModel">
<DataAnnotationsValidator />
<SfTextBox @bind-Value="exampleModel.Name"></SfTextBox>
<ValidationMessage For="@(() => exampleModel.Name)" />
</EditForm>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
```

```

</table>
<style>
.table-section {
margin: 0 auto;
}
tr td:first-child {
text-align: right;
padding-right: 20px;
}
.sample-td {
padding-top: 10px;
min-width: 230px;
height: 100px;
}
</style>
@code {
private string primaryKey { get; set; } = "editor1";
private ExampleModel exampleModel = new ExampleModel();
public class ExampleModel
{
[Required]
[StringLength(10, ErrorMessage = "Name is too long.")]
public string Name { get; set; } = "sync";
}
}

```

The output will be as follows.

<p><b>Default Error Message</b></p>	<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;"> Select date  </div> <div style="border: 1px solid red; padding: 5px; margin-bottom: 10px;"> This field is required. </div> <div style="display: flex; justify-content: flex-end; gap: 10px;"> <div style="border: 1px solid #ccc; padding: 5px 10px;">✓</div> <div style="border: 1px solid #ccc; padding: 5px 10px;">✗</div> </div>
<p><b>Customized Error Message</b></p>	<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;"> Select date  </div> <div style="border: 1px solid red; padding: 5px; margin-bottom: 10px;"> Field should not be empty </div> <div style="display: flex; justify-content: flex-end; gap: 10px;"> <div style="border: 1px solid #ccc; padding: 5px 10px;">✓</div> <div style="border: 1px solid #ccc; padding: 5px 10px;">✗</div> </div>

### Style and appearance in Blazor In-place Editor Component

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

#### Customizing the In-place Editor text

Use the following CSS to customize the default In-place Editor's text content properties like font-family, font-size, color and border bottom.

#### CSS

```

/* To change color, font family and font size */
.e-inplaceeditor .e-editable-value-wrapper .e-editable-value {
border-bottom: 2px dotted green;
color: red;
}

```



```
font-size: 12px;
font-family: Segoe UI
}
```

### Customizing the In-place Editor action buttons

Use the following CSS to customize the default In-place Editor's action buttons.

#### CSS

```
/* To change icon color for save button */
.e-inplaceeditor .e-editable-action-buttons .e-btn-save.e-icon-btn .e-btn-
icon.e-icons,
.e-inplaceeditor-tip .e-editable-action-buttons .e-btn-save.e-icon-btn .e-
btn-icon.e-icons{
color: green;
}
/* To change icon color for cancel button */
.e-inplaceeditor .e-editable-action-buttons .e-btn-cancel.e-icon-btn .e-btn-
icon.e-icons, .e-inplaceeditor-tip .e-editable-action-buttons .e-btn-
cancel.e-icon-btn .e-btn-icon.e-icons {
color: red;
}
/* To change background color for save button */
.e-inplaceeditor .e-editable-action-buttons .e-btn-save.e-icon-btn,
.e-inplaceeditor-tip .e-editable-action-buttons .e-btn-save.e-icon-btn {
background-color: antiquewhite;
}
/* To change background color for cancel button */
.e-inplaceeditor .e-editable-action-buttons .e-btn-cancel.e-icon-btn,
.e-inplaceeditor-tip .e-editable-action-buttons .e-btn-cancel.e-icon-btn {
background-color: antiquewhite;
}
```

## Globalization in Blazor In-place Editor Component

### Localization

Localization library allows you to localize the default text content of the In-place Editor to different cultures using the `Locale` property. In-place Editor following keys will be localize based on culture.

Use `Resource` file to translate the static text of the In-place Editor. 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 Blazor Server and WebAssembly project for Syncfusion Blazor components.

	Locale key		en-US (default)	
	-----		-----	
	InPlaceEditor_Save		Save	
	InPlaceEditor_Cancel		Cancel	
	InPlaceEditor_LoadingText		Loading...	
	InPlaceEditor_EditIcon		Click to edit	
	InPlaceEditor_EditAreaClick		Click to edit	

| InPlaceEditor\_EditAreaDoubleClick | Double click to edit |

In the following sample, **French** culture is set to In-place Editor and change the tooltip text.

### ASPX-CS

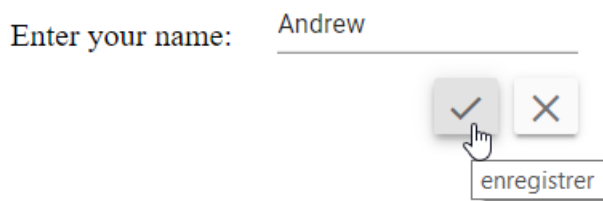
```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.Inputs
<table class="table-section">
<tr>
<td> Choose Editable Type: </td>
<td>
<SfDropDownList Width="90%" TValue="string" TItem="InplaceEditableModes"
DataSource="@EditableData" @bind-Value="@DropDownValue">
<DropDownListEvents TValue="string" TItem="InplaceEditableModes"
ValueChanged="@OnChange"></DropDownListEvents>
<DropDownListFieldSettings Text="text"
Value="value"></DropDownListFieldSettings>
</SfDropDownList>
</td>
</tr>
<tr>
<td class="sample-td"> Enter your name: </td>
<td class="sample-td">
<SfInPlaceEditor @bind-Value="@TextValue" EditableOn="@EditableOn"
TValue="string" Locale="fr-BE">
<EditorComponent>
<SfTextBox @bind-Value="@TextValue"></SfTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
<style>
.table-section {
margin: 0 auto;
}
tr td:first-child {
text-align: right;
padding-right: 20px;
}
.sample-td {
padding-top: 10px;
min-width: 230px;
height: 100px;
}
</style>
@code {
private string TextValue = "Andrew";
public string DropDownValue = "Click";
private EditableType EditableOn = EditableType.Click;
public class InplaceEditableModes
{
public string value { get; set; }
public string text { get; set; }
}
```

```

private List<InplaceEditableModes> EditableData = new
List<InplaceEditableModes>()
{
    new InplaceEditableModes(){ value= "Click", text= "Click" },
    new InplaceEditableModes(){ value= "Double Click", text= "Double Click" },
    new InplaceEditableModes(){ value= "Edit Icon Click", text= "Edit Icon
Click" }
};
private void OnChange(Syncfusion.Blazor.DropDowns.ChangeEventArgs<string,
InplaceEditableModes> args)
{
    if (args.Value != null)
    {
        if (args.Value.ToString() == "Click")
        {
            this.EditableOn = EditableType.Click;
        }
        else if (args.Value.ToString() == "Double Click")
        {
            this.EditableOn = EditableType.DoubleClick;
        }
        else
        {
            this.EditableOn = EditableType.EditIconClick;
        }
        this.StateHasChanged();
    }
}

```

The output will be as follows.



### Right to left

Specifies the direction of the In-place Editor component using the `EnableRtl` property. For writing systems that requires Arabic, Hebrew, and more. The direction can be switched to right-to-left.

It will not change based on the locale property.

### ASPX-CS

```

@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.Inputs
<table>
<tr>
<td class="control-title content-title"> Enter your name: </td>
<td>

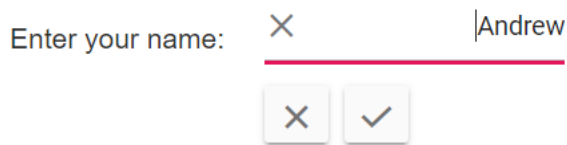
```

```

<SfInPlaceEditor @bind-Value="@TextValue" EnableRtl="true" TValue="string">
  <EditorComponent>
    <SfTextBox @bind-Value="@TextValue" Placeholder="Enter some
text"></SfTextBox>
  </EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
@code {
public string TextValue = "Andrew";
}

```

The output will be as follows.



### Format

Formatting is a way of representing the value in different formats. You can format the following mentioned components with its `format` property when it is directly configured in the Editor component.

- [DatePicker](#)
- [DateRangePicker](#)
- [DateTimePicker](#)
- [NumericTextBox](#)
- [Slider](#)
- [TimePicker](#)

### ASPX-CS

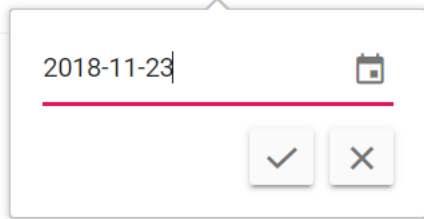
```

@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.Calendars
<table>
<tr>
<td class="control-title"> DatePicker </td>
<td>
<SfInPlaceEditor Type="Syncfusion.Blazor.InPlaceEditor.InputType.Date"
TValue="DateTime?" @bind-Value="@DateValue">
  <EditorComponent>
    <SfDatePicker TValue="DateTime?" @bind-Value="@DateValue" Format="yyyy-MM-
dd" Placeholder="Choose a Date"></SfDatePicker>
  </EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
@code {
public DateTime? DateValue { get; set; } = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, DateTime.Now.Day);
}

```

The output will be as follows.

Select date: 11/23/2018



## Events in Blazor In-place Editor Component

This section explains the list of events of the In-place Editor's component which will be triggered for appropriate In-place Editor's actions.

### Created

**Created** event triggers once the component rendering is completed.

#### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
<SfInPlaceEditor>
  <InPlaceEditorEvents Created="@CreatedHandler" >/InPlaceEditorEvents>
</SfInPlaceEditor>
@code{
public void CreatedHandler(Object args)
{
  // Here you can customize your code
}
}
```

### OnActionBegin

**OnActionBegin** event triggers before the data submitted to the server.

#### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
<SfInPlaceEditor>
  <InPlaceEditorEvents OnActionBegin="@OnActionBeginHandler"
  >/InPlaceEditorEvents>
</SfInPlaceEditor>
@code{
public void OnActionBeginHandler(ActionBeginEventArgs args)
{
  // Here you can customize your code
}
}
```

### OnActionSuccess

**OnActionSuccess** event triggers when data submitted successfully to the server.

#### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
<SfInPlaceEditor>
<InPlaceEditorEvents OnActionSuccess="@OnActionSuccessHandler"
></InPlaceEditorEvents>
</SfInPlaceEditor>
@code{
public void OnActionSuccessHandler(ActionEventArgs args)
{
// Here you can customize your code
}
}
```

### OnActionFailure

**OnActionFailure** event triggers when data submission failed.

#### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
<SfInPlaceEditor>
<InPlaceEditorEvents OnActionFailure="@OnActionFailureHandler"
></InPlaceEditorEvents>
</SfInPlaceEditor>
@code{
public void OnActionFailureHandler(ActionEventArgs args)
{
// Here you can customize your code
}
}
```

### ValueChange

**ValueChange** event triggers when the integrated component value has changed that render based on the **type** property in the In-place editor.

#### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
<SfInPlaceEditor>
<InPlaceEditorEvents ValueChange="@ValueChangeHandler"
></InPlaceEditorEvents>
</SfInPlaceEditor>
@code{
public void ValueChangeHandler(ChangeEventArgs args)
{
// Here you can customize your code
}
}
```

### Destroyed

**Destroyed** event triggers when the component gets destroyed.

#### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
<SfInPlaceEditor>
```

```
<InPlaceEditorEvents Destroyed="@DestroyedHandler" >/InPlaceEditorEvents>
</SfInPlaceEditor>
@code{
public void DestroyedHandler(Object args)
{
// Here you can customize your code
}
}
```

## How To

### Dynamically move In-place Editor to edit mode in Blazor

At component initial load, you can open the editor state without interacting with the In-place Editor input element by configuring the `EnableEditMode` property to `true`.

In the following example, editor opened at initial load and when toggling a checkbox, it will remove or open the editor.

### ASPX-CS

```
@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Inputs
<table class="table-section">
<tr>
<td> EnableEditMode: </td>
<td>
<SfCheckBox @bind-Checked="@EditModeEnable" Label="Disable"
ValueChange="OnChange" TChecked="bool">/SfCheckBox>
</td>
</tr>
<tr>
<td class="sample-td"> Enter your name: </td>
<td class="sample-td">
<SfInPlaceEditor @bind-Value="@TextValue" EnableEditMode="EditModeEnable"
TValue="string" ActionOnBlur="ActionBlur.Ignore">
<EditorComponent>
<SfTextBox @bind-Value="@TextValue" Placeholder="Enter some
text">/SfTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
<style>
.table-section {
margin: 0 auto;
}
tr td:first-child {
text-align: right;
padding-right: 20px;
}
.sample-td {
padding-top: 10px;
min-width: 230px;
height: 100px;
}
```

```

}
</style>
@code {
public bool EditModeEnable { get; set; } = true;
public string TextValue { get; set; } = "Andrew";
private void OnChange(Syncfusion.Blazor.Buttons.ChangeEventArgs<bool> args)
{
this.EditModeEnable = args.Checked;
this.StateHasChanged();
}
}

```

### Disable the edit mode specifically in Blazor In-place Editor Component

The edit mode of the In-place Editor can be disabled by setting the **Disabled** property value to **true**. In the following example, when you check or uncheck the checkbox, the In-place Editor component will disable or enable the edit mode respectively.

#### ASPX-CS

```

@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Inputs
<table class="table-section">
<tr>
<td> Disabled: </td>
<td>
<SfCheckBox @bind-Checked="Checked" Label="Disable" ></SfCheckBox>
</td>
</tr>
<tr>
<td class="sample-td"> Enter your name: </td>
<td class="sample-td">
<SfInPlaceEditor @bind-Value="@TextValue" TValue="string"
Disabled="Checked">
<EditorComponent>
<SfTextBox @bind-Value="@TextValue" Placeholder="Enter some
text"></SfTextBox>
</EditorComponent>
</SfInPlaceEditor>
</td>
</tr>
</table>
<style>
.table-section {
margin: 0 auto;
}
tr td:first-child {
text-align: right;
padding-right: 20px;
}
.sample-td {
padding-top: 10px;
min-width: 230px;
height: 100px;
}
</style>

```



```
@code {
public string TextValue { get; set; } = "Andrew";
public bool Checked { get; set; } = true;
}
```

### Custom Animation for popup mode in Blazor In-place Editor Component

In popup mode, the In-place Editor is rendered with the Blazor **Tooltip** component. You can use the tooltip properties and events to customize the popup by configuring properties using the **InPlaceEditorPopupSettings** tag.

In the following example, popup animation can be customized by passing animation effect using the **InPlaceEditorPopupSettings** tag and the dynamic animation effect changes configured from the Blazor **DropDownList** component **ValueChange** event.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.InPlaceEditor
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Popups
<div id='container'>
<table class="table-section">
<tr>
<td> Open Animation: </td>
<td>
<SfDropDownList PopupHeight="150px" @bind-Value="DropdownValue"
Placeholder="Select a animate type" DataSource="OpenAnimateData">
<DropDownListEvents ValueChange="@OnChange" TValue="string"
TItem="DropDownFields"></DropDownListEvents>
<DropDownListFieldSettings Value="Text"
Text="Text"></DropDownListFieldSettings>
</SfDropDownList>
</td>
</tr>
<tr>
<td class="sample-td"> Enter your name: </td>
<td class="sample-td">
<SfInPlaceEditor @bind-Value="@TextValue" Mode="RenderMode.Popup"
TValue="string">
<EditorComponent>
<SfTextBox @bind-Value="@TextValue" Placeholder="Enter some
text"></SfTextBox>
</EditorComponent>
<InPlaceEditorPopupSettings
Animation="@Animation"></InPlaceEditorPopupSettings>
</SfInPlaceEditor>
</td>
</tr>
</table>
</div>
<style>
#container {
display: flex;
justify-content: center;
}
```

```

.table-section {
margin: 0 auto;
}
tr td:first-child {
text-align: right;
padding-right: 20px;
}
.sample-td {
padding-top: 10px;
min-width: 230px;
height: 100px;
}
</style>
@code {
public class DropDownFields
{
public string Text { get; set; }
}
public List<DropDownFields> OpenAnimateData = new List<DropDownFields>()
{
new DropDownFields(){ Text= "None" },
new DropDownFields(){ Text= "FadeIn" },
new DropDownFields(){ Text= "FadeZoomIn" },
new DropDownFields() { Text= "ZoomIn" }
};
public string DropdownValue { get; set; } = "ZoomIn";
public string TextValue { get; set; } = "Andrew";
public AnimationModel Animation { get; set; } = new AnimationModel
{
Open = new TooltipAnimationSettings { Delay = 0, Duration = 150, Effect =
Effect.ZoomIn }
};
public void OnChange(Syncfusion.Blazor.DropDowns.ChangeEventArgs<string,
DropDownFields> args)
{
Animation = new AnimationModel
{
Open = new TooltipAnimationSettings { Delay = 0, Duration = 150, Effect =
(Effect)System.Enum.Parse(typeof(Effect), args.Value) }
};
}
}
}

```

## Input Mask

### Getting Started with Blazor Input Mask Component

This section briefly explains about how to include a [Blazor MaskedTextBox](#) Component in your Blazor Server-Side and Client-Side application. You can refer to our Getting Started with [Blazor Server-Side MaskedTextBox](#) and [Blazor WebAssembly MaskedTextBox](#) documentation pages for configuration specifications.

To get start quickly with Blazor MaskedTextBox component, you can check on this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=QQwlnHTmBUY"%}

### Importing Syncfusion Blazor component in the application

- Install Syncfusion.Blazor.Inputs NuGet package to the application by using the NuGet Package Manager.

---

Please ensure to check the Include prerelease option for our Beta release.

---

- 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.Inputs package.

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
```

### Add SyncfusionBlazor service in Program.cs

Open the **Program.cs** file and add services required by Syncfusion components using **builder.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 MaskedTextBox component to the application

To initialize the MaskedTextBox component add the below code to your `Index.razor` view page which is present under `~/Pages` folder.

### ASPX-CS

```
<SfMaskedTextBox></SfMaskedTextBox>
```

The output will be as follows.

Customer Id

---

### Set the mask

You can set the mask to the MaskedTextBox to validate the user input by using the [Mask](#) property.

The following example demonstrates the usage of mask element `0` that allows any single digit from `0` to `9`.

### ASPX-CS

```
<SfMaskedTextBox Mask='000-000-0000'></SfMaskedTextBox>
```

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 Input Mask Component

Data binding can be achieved by using the `bind-Value` attribute and it supports string type. If component value has been changed, it will affect the all places where you bind the variable for the `bind-Value` attribute.

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<p>MaskedTextBox value is: @MaskValue</p>
<SfMaskedTextBox @bind-Value="@MaskValue"></SfMaskedTextBox>
@code {
    public string MaskValue { get; set; } = "12345";
}
```

## Dynamic Value Binding

You can bind the value to the MaskedTextBox component dynamically for `bind-Value` attribute as mentioned in the following code.

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfMaskedTextBox Mask="00000" @bind-Value="@MaskValue"></SfMaskedTextBox>
<button @onclick="@UpdateValue">Update Value</button>
@code {
    public string MaskValue { get; set; } = "12345";
    public void UpdateValue()
    {
        MaskValue = "67890";
    }
}
```

## Mask Configuration in Blazor Input Mask Component

The [Mask](#) is a combination of standard and custom mask elements that validates the user input based on its behavior.

---

When the mask value is empty, the MaskedTextBox behaves as an input element with text type.

---

### Standard mask elements

The following table shows the list of mask elements and its behavior based on [MSDN](#) standard.

The mask can be formed by combining any one or more of these mask elements.

Mask Element	Description
-----	-----
0	Digit required. This element will accept any single digit from 0 to 9.
9	Digit or space, optional.
#	Digit or space, optional, Plus(+) and minus(-) signs are allowed.
L	Letter required. It will accept letters a-z and A-Z.

- | ? | Letter or space, optional. |
- | & | Requires a character. |
- | C | Character or space, optional. |
- | A | Alphanumeric (**A-Za-z0-9**) required. |
- | a | Alphanumeric (**A-Za-z0-9**) or space, optional. |
- | < | Shift down. Converts all characters to lower case. |
- | > | Shift up. Converts all characters to upper case. |
- | &#124; | Disable a previous shift up or shift down. |
- | \\\ | Escapes a mask character, turning it into a literal. |
- | All other characters | Literals. All non-mask elements (literals) will appear as themselves within MaskedTextBox. |

The following example demonstrates the usage of standard mask elements.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfMaskedTextBox Mask="#####" Placeholder="Mask ##### (ex: 012+-)"
FloatLabelType="@FloatLabelType.Always"></SfMaskedTextBox>
<SfMaskedTextBox Mask="LLLLLL" Placeholder="Mask LLLLLL (ex: Sample)"
FloatLabelType="@FloatLabelType.Always"></SfMaskedTextBox>
<SfMaskedTextBox Mask="&&&&&" Placeholder="Mask &&&&& (ex: A12#)"
FloatLabelType="@FloatLabelType.Always"></SfMaskedTextBox>
<SfMaskedTextBox Mask=">LLL<LLL" Placeholder="Mask >LLL<LL (ex: SAMple)"
FloatLabelType="@FloatLabelType.Always"></SfMaskedTextBox>
<SfMaskedTextBox Mask="\A999" Placeholder="Mask \A999 (ex: A321)"
FloatLabelType="@FloatLabelType.Always"></SfMaskedTextBox>
```

The output will be as follows.

Mask ##### (ex: 012+-)

\_\_\_\_\_

Mask LLLLLL (ex: Sample)

\_\_\_\_\_

Mask &&&&& (ex: A12#)

\_\_\_\_\_

Mask >LLL<LLL

\_\_\_\_\_

Mask \A999 (ex: A321)

A\_\_\_\_\_

### Custom mask elements

Other than the above standard mask elements, the mask can be configured with the custom characters or regular expression to define a custom behavior.

#### Custom characters

You can define any of the non-mask element as the mask element and its behavior through the [CustomCharacters](#) property.

In the following example, non-mask element **P** accepts the values **P, A, p, a**, and **M** accepts the values **M, m** as mentioned in the custom characters collection.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfMaskedTextBox Mask="00:00 >PM" Placeholder="Time (ex: 10:00 PM, 10:00 AM)" CustomCharacters="@CustomMask"></SfMaskedTextBox>
@code {
public Dictionary<string, string> CustomMask = new Dictionary<string, string>()
{
    { "P" , "P,p,A,a" },
    { "M" , "m,M" }
};
}
```

The output will be as follows.

Time (ex: 10:00 PM, 10:00 AM)

#### Regular expression

Instead of the mask element, you can define your own regular expression to validate the input of a particular input place. The regular expressions should be wrapped by the square brackets (e.g., [Regex]).

In the following example, regular expression has been set for each input places.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfMaskedTextBox Placeholder="Enter value" Mask="[0-2][0-9][0-9].[0-2][0-9][0-9].[0-2][0-9][0-9].[0-2][0-9][0-9]"
FloatLabelType="@FloatLabelType.Auto"></SfMaskedTextBox>
```

The output will be as follows.

Enter value  
23\_.\_.\_

#### Prompt character

The Prompt character is a prompting symbol in the MaskedTextBox for the mask elements. The symbol is used to show the input positions in the MaskedTextBox. You can customize the prompt character of MaskedTextBox by using the [PromptChar](#) property.

The following example demonstrates the MaskedTextBox with customized prompt character as #.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfMaskedTextBox Mask="999-999-9999"
PromptChar="@PromptCharacter"></SfMaskedTextBox>
@code{
public char PromptCharacter { get; set; } = '#';
}
```

The output will be as follows,

### ### ###

#### Accessibility in Blazor Input Mask Component

The MaskedTextBox is characterized with complete ARIA Accessibility support that helps to access using on-screen readers and other assistive technology devices. This component is designed with the reference of the guidelines document given in [WAI ARAI Accessibility practices](#).

The MaskedTextBox uses the `textbox` role and following ARIA properties for its element based on its state.

##### | Property | Functionality |

| --- | --- |

| aria-live | Indicates the priority of updates to a live region. |

| aria-disabled | Indicates the disabled state of the MaskedTextBox. |

| aria-valuenow | Specifies the current value of the MaskedTextBox. |

| aria-invalid | Indicates that the user input is incorrect or not within the acceptable ranges. |

| aria-placeholder | It is a short hint to help the users with data entry when the MaskedTextBox has no value. |

| aria-labelledby | Indicates the floating label element of the MaskedTextBox. |

#### Native Events in Blazor Input Mask Component

The following section explains the steps to include native events and pass data to event handler in MaskedTextBox component.

##### Bind native events to MaskedTextBox

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.Inputs
<SfMaskedTextBox Mask='000-000-0000'
@onkeypress='@KeyPressed'></SfMaskedTextBox>
@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.Inputs
<SfMaskedTextBox Mask='000-000-0000' @onkeypress="@(() =>
Console.WriteLine("Key Pressed!"))"></SfMaskedTextBox>
```

#### 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 "m" key.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfMaskedTextBox Mask='000-000-0000' @onkeypress='@(e => KeyPressed(e))'
></SfMaskedTextBox>
@code {
public void KeyPressed(KeyboardEventArgs args) {
if (args.Key == "m") {
Console.WriteLine("M 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 |

## Events in Blazor Input Mask Component

This section explains the list of events of the Input Mask component which will be triggered for appropriate Input Mask actions.

### Blur

**Blur** event triggers when the Input Mask component has focus-out.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfMaskedTextBox Blur="@BlurHandler"></SfMaskedTextBox>
@code {
private void BlurHandler(MaskBlurEventArgs args)
{
// Here you can customize your code
}
}
```

### Created

**Created** event triggers when the Input Mask component is created.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfMaskedTextBox Created="@CreatedHandler"></SfMaskedTextBox>
@code {
private void CreatedHandler(Object args)
{
// Here you can customize your code
}
}
```

### Destroyed

**Destroyed** event triggers when the Input Mask component is destroyed.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfMaskedTextBox Destroyed="@DestroyedHandler"></SfMaskedTextBox>
@code {
private void DestroyedHandler(Object args)
{
// Here you can customize your code
}
}
```

### Focus

**Focus** event triggers when the Input Mask gets focus.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfMaskedTextBox Focus="@FocusHandler"></SfMaskedTextBox>
@code {
private void FocusHandler(MaskFocusEventArgs args)
```

```
{  
  // Here you can customize your code  
}
```

### ValueChanged

**ValueChanged** event triggers when the content of Input Mask has changed and gets focus-out.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs  
<SfMaskedTextBox ValueChange="@ValueChangeHandler"></SfMaskedTextBox>  
@code {  
  private void ValueChangeHandler(MaskChangeEventArgs args)  
  {  
    // Here you can customize your code  
  }  
}
```

### ValueChanged

**ValueChanged** event Specifies the callback to trigger when the value changes.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs  
<SfMaskedTextBox ValueChanged="@ValueChangedHandler"></SfMaskedTextBox>  
@code {  
  private void ValueChangedHandler(String args)  
  {  
    // Here you can customize your code  
  }  
}
```

Input Mask 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

Customize the UI appearance of the Blazor Input Mask Component

The appearance of the MaskedTextBox can be changed by adding custom [CssClass](#) to the component and enabling styles.

Refer to the following example to change the appearance of the MaskedTextBox.

#### ASPX-CS

```
@using Syncfusion.Blazor.Inputs  
<SfMaskedTextBox Mask="00000" Value="34523" CssClass="e-style"  
  Placeholder="Enter user ID" FloatLabelType="@FloatLabelType.Always">  
</SfMaskedTextBox>  
<style>  
  .e-mask.e-style .e-control.e-maskedtextbox {  
    color: #00ffff;  
    letter-spacing: 10px;  
    font-size: xx-large;  
  }  
</style>
```

```
border: 1px;
border-color: #ffffff;
}
.e-control-wrapper.e-mask.e-float-input.e-style .e-float-line::before {
background: #ffffff;
}
.e-control-wrapper.e-mask.e-float-input.e-style .e-float-line::after {
background: #ffffff;
}
.e-control-wrapper.e-mask.e-float-input.e-style .e-float-text.e-label-top {
color: #00ffff;
font-size: medium;
}
</style>
```

The output will be as follows,

Enter user ID  
3 4 5 2 3

### Model Binding in Blazor Input Mask Component

This section demonstrates the Strongly typed extension support in MaskedTextBox. 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 click the submit button to post the selected value in the MaskedTextBox. When posting the null value, validation error message will be shown below the MaskedTextBox.

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
@using System.ComponentModel.DataAnnotations
<EditForm Model="@User">
<DataAnnotationsValidator />
<div asp-validation-summary="All" class="text-danger"></div>
<div class="form-group">
<SfMaskedTextBox Mask="00000" Placeholder='Provide user ID' @bind-
Value="@User.ID"></SfMaskedTextBox>
<ValidationMessage For="@(() => User.ID)" />
</div>
<button type="submit" class="btn btn-primary">Submit</button>
</EditForm>
@code {
public Customer User = new Customer();
public class Customer
{
[Required(ErrorMessage = "User ID is required")]
public string ID { get; set; }
}
}
```

The output will be as follows.

Provide user ID

User ID is required

Submit

## Kanban

<!-- markdownlint-disable MD024 -->

### Getting Started with Blazor Kanban Component

This section briefly explains how to include a Kanban 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 the introduction and configuring the common specifications.

To get start quickly with Blazor Kanban component, you can check on this video

{% youtube

"youtube:https://www.youtube.com/watch?v=EYDC0RCMZKg" %}

### Importing Syncfusion Blazor component in the application

1. Install **Syncfusion.Blazor.Kanban** 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.Kanban** package.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
```

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();
        }
    }
}
```

Add Kanban component

To initialize the Kanban component, add the below code to your **Index.razor** view page which is present under `~/Pages` folder.

### ASPX-CS

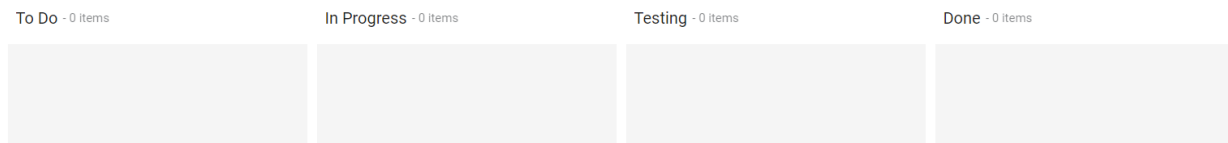
```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel">
<KanbanColumns>
<KanbanColumn HeaderText="To Do" KeyField="@ (new List<string>()
{"Open"}) "></KanbanColumn>
<KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{"InProgress"}) "></KanbanColumn>
<KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
{"Testing"}) "></KanbanColumn>
<KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
{"Close"}) "></KanbanColumn>
</KanbanColumns>
</SfKanban>
@code {
```

```
public class TasksModel
{
    public string Id { get; set; }
    public string Title { get; set; }
    public string Status { get; set; }
    public string Summary { get; set; }
}
}
```

### Run the application

After successful compilation of your application, run the application.

The output will be as follows.



### Populating cards

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

### ASPX-CS

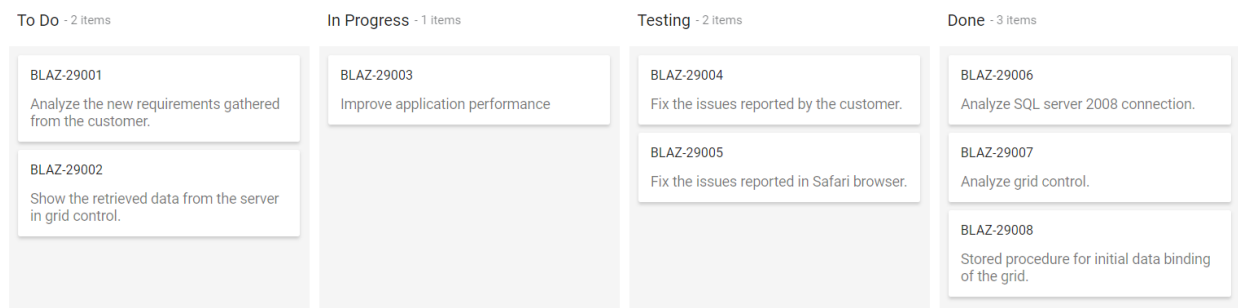
```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
    <KanbanColumns>
        <KanbanColumn HeaderText="To Do" KeyField="@ (new List<string>()
        { "Open" }) "></KanbanColumn>
        <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
        { "InProgress" }) "></KanbanColumn>
        <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
        { "Testing" }) "></KanbanColumn>
        <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
        { "Close" }) "></KanbanColumn>
    </KanbanColumns>
    <KanbanCardSettings HeaderField="Title"
    ContentField="Summary"></KanbanCardSettings>
</SfKanban>
@code {
    public class TasksModel
    {
        public string Id { get; set; }
        public string Title { get; set; }
        public string Status { get; set; }
        public string Summary { get; set; }
    }
    public List<TasksModel> Tasks = new List<TasksModel>()
    {
        new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
        Summary = "Analyze the new requirements gathered from the customer." },
    }
```

```

new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "Open",
Summary = "Show the retrieved data from the server in grid control." },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "InProgress",
Summary = "Improve application performance" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "Testing",
Summary = "Fix the issues reported by the customer." },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Testing",
Summary = "Fix the issues reported in Safari browser." },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Close",
Summary = "Analyze SQL server 2008 connection." },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Analyze grid control." },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid." }
};
}

```

The output will be as follows.



### Enable Swimlane

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

### ASPX-CS

```

@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="ToDo" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Title"
  ContentField="Summary"></KanbanCardSettings>
  <KanbanSwimlaneSettings KeyField="Assignee"></KanbanSwimlaneSettings>
</SfKanban>
@code {
public class TasksModel
{

```



```
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
    new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
        Summary = "Analyze the new requirements gathered from the customer.",
        Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
        Summary = "Improve application performance", Assignee = "Andrew Fuller" },
    new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
        Summary = "Arrange a web meeting with the customer to get new
        requirements.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
        Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
        Leverling" },
    new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
        Summary = "Fix the issues reported by the customer.", Assignee = "Steven
        walker" },
    new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
        Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
        Davloio" },
    new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
        Summary = "Test the application in the IE browser.", Assignee = "Margaret
        hamilt" },
    new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
        Summary = "Validate the issues reported by the customer.", Assignee =
        "Steven walker" },
    new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
        Summary = "Show the retrieved data from the server in grid control.",
        Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
        "InProgress", Summary = "Fix cannot open user's default database SQL
        error.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
        Summary = "Fix the issues reported in data binding.", Assignee = "Janet
        Leverling" },
    new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
        Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
    },
    new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
        Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
        Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
        Summary = "Stored procedure for initial data binding of the grid.", Assignee
        = "Steven walker" },
    new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
        Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
        Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
        Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
}
```

```

new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}

```

The output will be as follows.

To Do - 5 items	In Progress - 4 items	Testing - 5 items	Done - 5 items
<b>Andrew Fuller - 3 items</b> BLAZ-29019 Enhance editing functionality.	BLAZ-29002 Improve application performance		BLAZ-29012 Analyze SQL server 2008 connection.
<b>Janet Leverling - 6 items</b> BLAZ-29003 Arrange a web meeting with the customer to get new requirements.	BLAZ-29004 Fix the issues reported in the IE browser.  BLAZ-29010 Fix cannot open user's default database SQL error.	BLAZ-29025 Fix the issues reported in data binding.  BLAZ-29027 Test editing feature in the IE browser.	BLAZ-29016 Analyze stored procedures.
<b>Margaret hamilt - 3 items</b> BLAZ-29009 Show the retrieved data from the server in grid control.			BLAZ-29007 Test the application in the IE browser.  BLAZ-29014 Analyze grid control.

## 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 Kanban Component

The Kanban uses [SfDataManager](#), which supports both RESTFUL JSON data service binding and IEnumerable binding. The [DataSource](#) property of Kanban can be assigned either with the instance of DataManager or a list of DataSource collection.

It supports the following types of data binding:

- 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, the TValue must be provided explicitly in the Kanban component.

---

## List binding

In list binding, you can assign an IEnumerable object to the [DataSource](#) property. The list data source can also be provided as an instance of the [SfDataManager](#) or by using the [SfDataManager](#) component.

## ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="@Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="To Do" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
</SfKanban>

@code {
    public class TasksModel
    {
        public int Id { get; set; }
        public string Status { get; set; }
        public string Summary { get; set; }
    }

    public List<TasksModel> Tasks { get; set; }
    protected override void OnInitialized()
    {
        Tasks = Enumerable.Range(1, 10).Select(x => new TasksModel()
        {
            Id = 1000 + x,
```

```
Status = (new string[] { "Open", "InProgress", "Testing", "Close" })[new
Random().Next(4)],
Summary = (new string[] { "Analyze SQL server 2008 connection.", "Fix the
issues reported in Safari browser.", "Improve application performance",
"Analyze grid control." })[new Random().Next(4)],
}).ToList();
}
}
```

By default, [SfDataManager](#) uses BlazorAdaptor for list data-binding.

#### *ExpandableObject binding*

Kanban is a generic component that is strongly bound to a model type. In some cases, the model type may be unknown during compile time. In such cases, you can bind data to the Kanban as a list of `ExpandableObject`.

`ExpandableObject` can be bound to Kanban by assigning to the [DataSource](#) property. Kanban can also perform all kinds of supported data operations and editing in `ExpandableObject`.

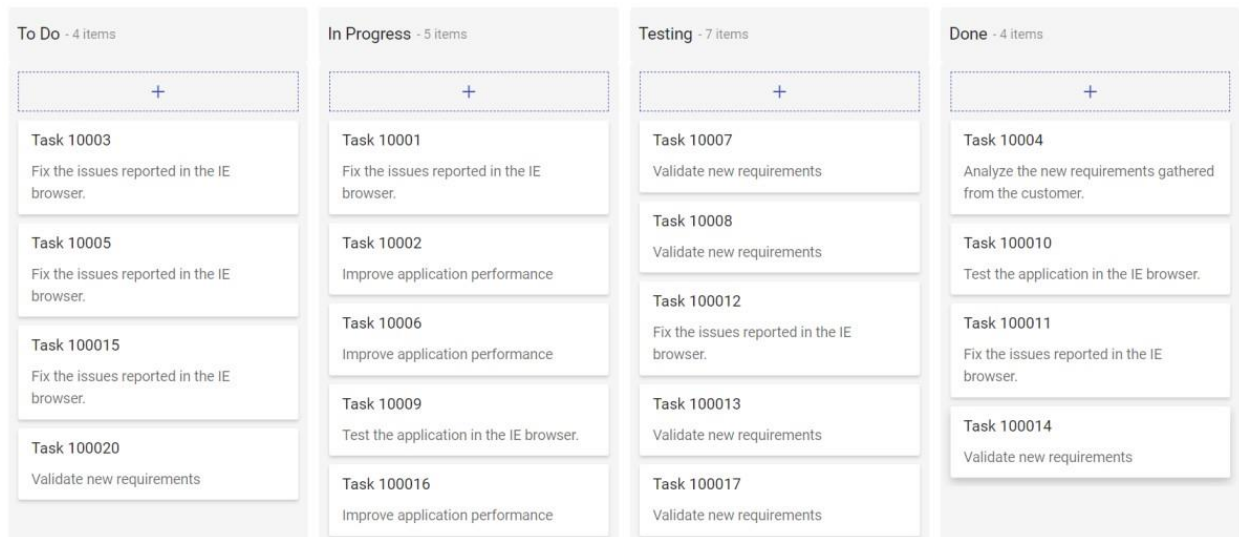
#### **ASPX-CS**

```
@using Syncfusion.Blazor.Kanban
@using System.Dynamic
<div class="col-lg-12 control-section">
<SfKanban KeyField="Status" DataSource="@Tasks">
<KanbanColumns>
@foreach (ColumnModel item in columnData)
{
<KanbanColumn HeaderText="@item.HeaderText" KeyField="@item.KeyField"
AllowAdding="true"></KanbanColumn>}
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
</SfKanban>
</div>
@code{
public List<ExpandableObject> Tasks { get; set; } = new List<ExpandableObject>();
private List<ColumnModel> columnData = new List<ColumnModel>() {
new ColumnModel(){ HeaderText= "To Do", KeyField= new List<string>() {
"Open" } },
new ColumnModel(){ HeaderText= "In Progress", KeyField= new List<string>() {
"In Progress" } },
new ColumnModel(){ HeaderText= "Testing", KeyField= new List<string>() {
"Testing" } },
new ColumnModel(){ HeaderText= "Done", KeyField=new List<string>() { "Close"
} }
};
protected override void OnInitialized()
{
Tasks = Enumerable.Range(1, 20).Select((x) =>
{
dynamic d = new ExpandableObject();
d.Id = "Task 1000" + x;
d.Status = (new string[] { "Open", "In Progress", "Testing", "Close" })[new
Random().Next(4)];
```

```

d.Summary = (new string[] { "Analyze the new requirements gathered from the
customer.", "Improve application performance", "Fix the issues reported in
the IE browser.", "Validate new requirements", "Test the application in the
IE browser." })[new Random().Next(5)];
d.Assignee = (new string[] { "Nancy Davloio", "Andrew Fuller", "Janet
Leverling", "Steven walker", "Margaret hamilt", "Michael Suyama", "Robert
King" })[new Random().Next(7)];
return d;
}).Cast<ExpandoObject>().ToList<ExpandoObject>();
}
}

```



### DynamicObject binding

DynamicObject can be bound to Kanban by assigning DynamicObject to the [DataSource](#) property. Kanban can also perform all kinds 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 operations and editing while using DynamicObject.

### ASPX-CS

```

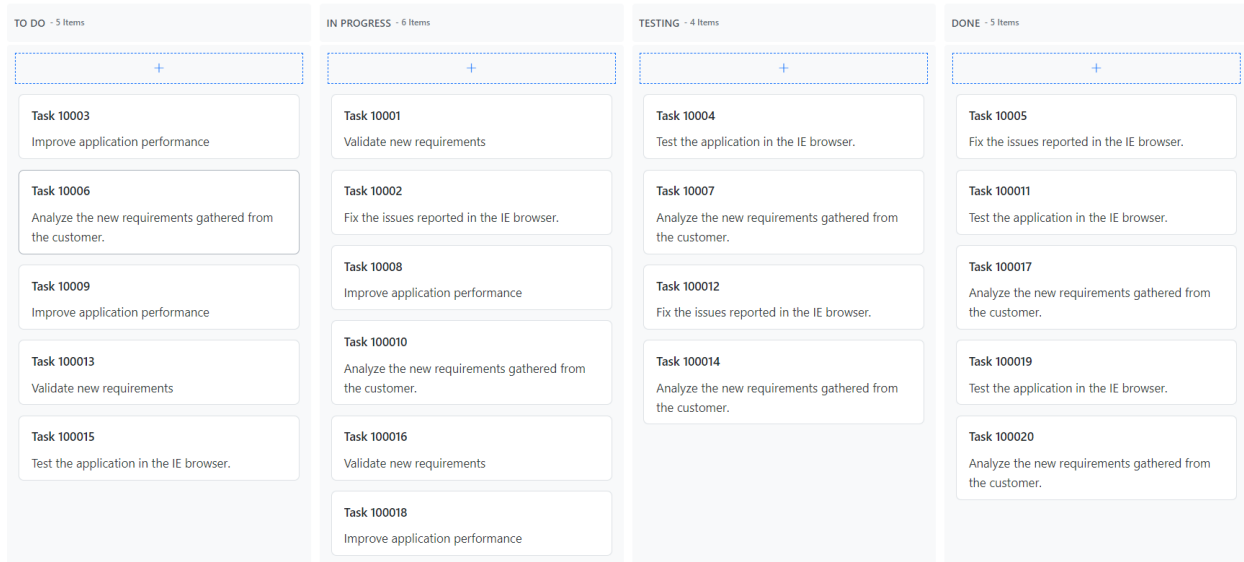
@using Syncfusion.Blazor.Kanban
@using System.Dynamic
<div class="col-lg-12 control-section">
<SfKanban KeyField="Status" DataSource="@Tasks">
<KanbanColumns>
@foreach (ColumnModel item in columnData)
{
<KanbanColumn HeaderText="@item.HeaderText" KeyField="@item.KeyField"
AllowAdding="true"></KanbanColumn>
}
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
</SfKanban>
</div>
@code{
private List<ColumnModel> columnData = new List<ColumnModel>() {

```

```

new ColumnModel(){ HeaderText= "To Do", KeyField= new List<string>() {
"Open" } },
new ColumnModel(){ HeaderText= "In Progress", KeyField= new List<string>() {
"In Progress" } },
new ColumnModel(){ HeaderText= "Testing", KeyField= new List<string>() {
"Testing" } },
new ColumnModel(){ HeaderText= "Done", KeyField=new List<string>() { "Close"
} }
};
public List<DynamicDictionary> Tasks = new List<DynamicDictionary>() { };
protected override void OnInitialized()
{
Tasks = Enumerable.Range(1, 20).Select((x) =>
{
dynamic d = new DynamicDictionary();
d.Id = "Task 1000" + x;
d.Status = (new string[] { "Open", "In Progress", "Testing", "Close" })[new
Random().Next(4)];
d.Summary = (new string[] { "Analyze the new requirements gathered from the
customer.", "Improve application performance", "Fix the issues reported in
the IE browser.", "Validate new requirements", "Test the application in the
IE browser." })[new Random().Next(5)];
d.Assignee = (new string[] { "Nancy Davloio", "Andrew Fuller", "Janet
Leverling", "Steven walker", "Margaret hamilt", "Michael Suyama", "Robert
King" })[new Random().Next(7)];
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;
}
}
}

```



### Remote data

Bind the remote data services to Kanban component by assigning service data as an instance of [SfDataManager](#) to the [DataSource](#) property or by using [SfDataManager](#) component.

By default, SfDataManager uses ODataAdaptor for remote data-binding.

TValue must be provided in the Kanban component when using SfDataManager.

### Binding with OData services

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

### ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Kanban
<div class="col-lg-12 control-section">
  <SfKanban TValue="Order" KeyField="ShipCountry" AllowDragAndDrop="false">
    <SfDataManager
      Url="https://js.syncfusion.com/ejServices/Wcf/Northwind.svc/Orders"
      Adaptor="@Syncfusion.Blazor.Adaptors.ODataAdaptor"></SfDataManager>
    <KanbanColumns>
      @foreach (ColumnModel item in columnData)
      {
        <KanbanColumn HeaderText="@item.HeaderText"
          KeyField="@item.KeyField"></KanbanColumn>
      }
    </KanbanColumns>
    <KanbanCardSettings HeaderField="OrderID"
      ContentField="ShipName"></KanbanCardSettings>
    <KanbanEvents TValue="Order" DialogOpen="@((args)=> { args.Cancel = true;
      }) "></KanbanEvents>
  </SfKanban>
</div>
@code {
  public class Order
  {
```

```

public int? OrderID { get; set; }
public string ShipName { get; set; }
public string ShipCountry { get; set; }
}
private List<ColumnModel> columnData = new List<ColumnModel>() {
    new ColumnModel(){ HeaderText= "Denmark", KeyField= new List<string>() {
        "Denmark" } },
    new ColumnModel(){ HeaderText= "Brazil", KeyField= new List<string>() {
        "Brazil" } },
    new ColumnModel(){ HeaderText= "Switzerland", KeyField= new List<string>() {
        "Switzerland" } },
    new ColumnModel(){ HeaderText= "Germany", KeyField=new List<string>() {
        "Germany" } }
};
}

```

### *Binding with OData v4 services*

The ODataV4 is an improved version of OData protocols to 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.Data
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="Order" KeyField="ShipCountry" AllowDragAndDrop="false">
    <SfDataManager
        Url="https://services.odata.org/V4/Northwind/Northwind.svc/Orders/"
        Adaptor="@Syncfusion.Blazor.Adaptors.ODataV4Adaptor"></SfDataManager>
    <KanbanColumns>
        @foreach (ColumnModel item in columnData)
        {
            <KanbanColumn HeaderText="@item.HeaderText"
                KeyField="@item.KeyField"></KanbanColumn>
        }
    </KanbanColumns>
    <KanbanCardSettings HeaderField="OrderID"
        ContentField="ShipName"></KanbanCardSettings>
    <KanbanEvents TValue="Order" DialogOpen="@((args)=> { args.Cancel = true;
        })"></KanbanEvents>
</SfKanban>

@code {
    public class Order
    {
        public int? OrderID { get; set; }
        public string ShipName { get; set; }
        public string ShipCountry { get; set; }
    }
    private List<ColumnModel> columnData = new List<ColumnModel>() {
        new ColumnModel(){ HeaderText= "Denmark", KeyField= new List<string>() {
            "Denmark" } },
        new ColumnModel(){ HeaderText= "Brazil", KeyField= new List<string>() {
            "Brazil" } },
        new ColumnModel(){ HeaderText= "Switzerland", KeyField= new List<string>() {
            "Switzerland" } },
        new ColumnModel(){ HeaderText= "Germany", KeyField=new List<string>() {
            "Germany" } }
    }
}

```



```
};
}
```

### Web API

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

### ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" AllowDragAndDrop="false">
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Kanban"
Adaptor="@Syncfusion.Blazor.Adaptors.WebApiAdaptor"></SfDataManager>
<KanbanColumns>
@foreach (ColumnModel item in columnData)
{
<KanbanColumn HeaderText="@item.HeaderText"
KeyField="@item.KeyField"></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
<KanbanEvents TValue="TasksModel" DialogOpen="@((args) => { args.Cancel =
true; })"></KanbanEvents>
</SfKanban>
@code {
public class TasksModel
{
public int Id { get; set; }
public string Status { get; set; }
public string Assignee { get; set; }
public string Summary { get; set; }
}
private List<ColumnModel> columnData = new List<ColumnModel>() {
new ColumnModel(){ HeaderText= "To Do", KeyField= new List<string>() {
"Open" } },
new ColumnModel(){ HeaderText= "In Progress", KeyField= new List<string>() {
"InProgress" } },
new ColumnModel(){ HeaderText= "Testing", KeyField= new List<string>() {
"Testing" } },
new ColumnModel(){ HeaderText= "Done", KeyField=new List<string>() { "Close"
} }
};
}
```

### Enable Data Manager after Initial Rendering

It is possible to render the data source in Kanban after initial rendering. This can be achieved by conditionally enabling the [SfDataManager](#) component after Kanban rendering.

The following sample code demonstrates enabling data manager condition in the Kanban component on button click.

### ASPX-CS

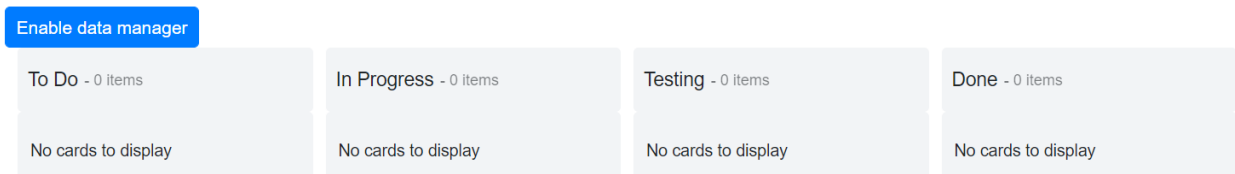
```
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Data
```

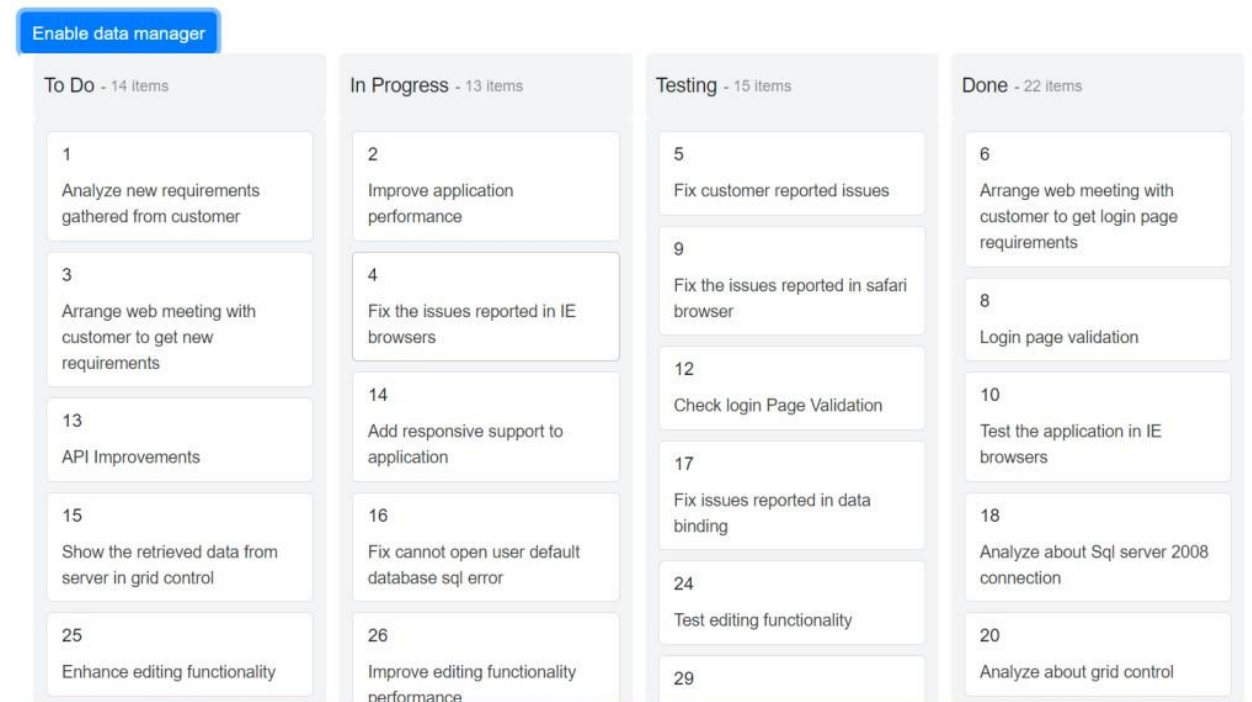
```

@using Syncfusion.Blazor.Kanban
<SfButton OnClick="Enable" CssClass="e-primary" IsPrimary="true"
Content="Enable data manager"></SfButton>
<SfKanban TValue="TasksModel" KeyField="Status" AllowDragAndDrop="false">
@if (IsInitialRender)
{
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Kanban"
Adaptor="@Syncfusion.Blazor.Adaptors.WebApiAdaptor"></SfDataManager>
}
<KanbanColumns>
@foreach (ColumnModel item in columnData)
{
<KanbanColumn HeaderText="@item.HeaderText"
KeyField="@item.KeyField"></KanbanColumn>
}
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
<KanbanEvents TValue="TasksModel" DialogOpen="@((args) => { args.Cancel =
true; })"></KanbanEvents>
</SfKanban>
@code{
public bool IsInitialRender = false;
public class TasksModel
{
public int Id { get; set; }
public string Status { get; set; }
public string Assignee { get; set; }
public string Summary { get; set; }
}
private List<ColumnModel> columnData = new List<ColumnModel>() {
new ColumnModel(){ HeaderText= "To Do", KeyField= new List<string>() {
"Open" } },
new ColumnModel(){ HeaderText= "In Progress", KeyField= new List<string>() {
"InProgress" } },
new ColumnModel(){ HeaderText= "Testing", KeyField= new List<string>() {
"Testing" } },
new ColumnModel(){ HeaderText= "Done", KeyField=new List<string>() { "Close"
} }
};
public void Enable()
{
// Enabling condition to render the data manager
this.IsInitialRender = true;
}
}

```

### Before Button click



**After Button click***Sending additional parameters to the server*

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

The following sample code demonstrates sending additional parameters using the Query property,

**ASPX-CS**

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" AllowDragAndDrop="false"
Query=@KanbanQuery>
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Kanban"
Adaptor="@Syncfusion.Blazor.Adaptors.WebApiAdaptor"></SfDataManager>
<KanbanColumns>
@foreach (ColumnModel item in columnData)
{
<KanbanColumn HeaderText="@item.HeaderText"
KeyField="@item.KeyField"></KanbanColumn>
}
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
<KanbanEvents TValue="TasksModel" DialogOpen="@((args) => { args.Cancel =
true; })"></KanbanEvents>
</SfKanban>
@code{
public string ParamValue = "true";
public Query KanbanQuery { get; set; }
protected override void OnInitialized()
{

```

```
KanbanQuery = new Query().AddParams("BlazorKanban", ParamValue);
}
public class TasksModel
{
    public int Id { get; set; }
    public string Status { get; set; }
    public string Assignee { get; set; }
    public string Summary { get; set; }
}
private List<ColumnModel> columnData = new List<ColumnModel>() {
    new ColumnModel(){ HeaderText= "To Do", KeyField= new List<string>() {
        "Open" } },
    new ColumnModel(){ HeaderText= "In Progress", KeyField= new List<string>() {
        "InProgress" } },
    new ColumnModel(){ HeaderText= "Testing", KeyField= new List<string>() {
        "Testing" } },
    new ColumnModel(){ HeaderText= "Done", KeyField=new List<string>() { "Close"
    } }
};
}
```

#### *Change Query parameter value dynamically*

It is possible to dynamically modify Kanban [Query](#) property value.

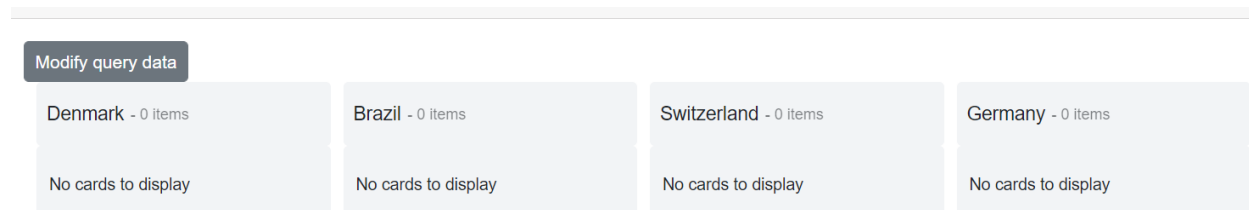
The following sample code demonstrates achieving this,

#### **ASPX-CS**

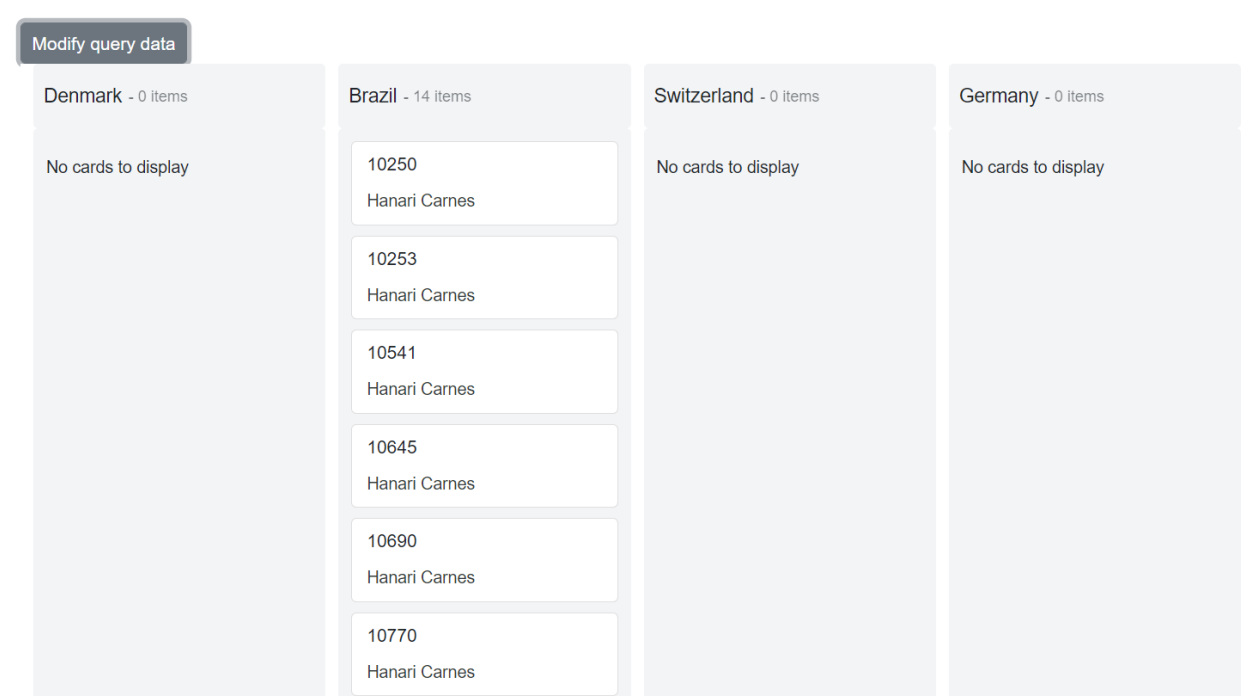
```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Kanban
@using Syncfusion.Blazor.Buttons
<SfButton Content="Modify query data" OnClick="BtnClick"></SfButton>
<SfKanban TValue="Order" @ref="KanbanObj" KeyField="ShipCountry"
AllowDragAndDrop="false" Query=@QueryData>
<SfDataManager
Url="https://services.odata.org/V4/Northwind/Northwind.svc/Orders/"
Adaptor="@Syncfusion.Blazor.Adaptors.ODataV4Adaptor"></SfDataManager>
<KanbanColumns>
@foreach (ColumnModel item in columnData)
{
    <KanbanColumn HeaderText="@item.HeaderText"
    KeyField="@item.KeyField"></KanbanColumn>
}</KanbanColumns>
<KanbanCardSettings HeaderField="OrderID"
ContentField="ShipName"></KanbanCardSettings>
<KanbanEvents TValue="Order" DialogOpen="@((args) => { args.Cancel = true;
})"></KanbanEvents>
</SfKanban>
@code {
    public SfKanban<Order> KanbanObj;
    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 string ShipName { get; set; }
public string ShipCountry { get; set; }
}
private List<ColumnModel> columnData = new List<ColumnModel>() {
    new ColumnModel(){ HeaderText= "Denmark", KeyField= new List<string>() {
        "Denmark" } },
    new ColumnModel(){ HeaderText= "Brazil", KeyField= new List<string>() {
        "Brazil" } },
    new ColumnModel(){ HeaderText= "Switzerland", KeyField= new List<string>() {
        "Switzerland" } },
    new ColumnModel(){ HeaderText= "Germany", KeyField=new List<string>() {
        "Germany" } }
};
public void BtnClick()
{
    QueryData = UpdatedQueryData;
}
}
```

### Before button Click



### After button Click



*Using Custom Adaptor*

It is possible to create your own **CustomAdaptor** by extending the built-in available adaptors.

The following example demonstrates the custom adaptor usage and how to bind the data with custom service and the CRUD operations for custom bound data are performed using the methods of [DataAdaptor](#) abstract class.

**ASPX-CS**

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Kanban
@using System.Collections
@inject OrderService _craftService;
<SfKanban ID="Kanban" TValue="Order" KeyField="ShipCity">
  <SfDataManager AdaptorInstance="@typeof(CustomAdaptor)"
  Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
  <KanbanColumns>
    <KanbanColumn HeaderText="Moscow" KeyField=@(new List<string>{"Moscow"})
    AllowAdding=true></KanbanColumn>
    <KanbanColumn HeaderText="London" KeyField=@(new
    List<string>{"London"})></KanbanColumn>
    <KanbanColumn HeaderText="Singapore" KeyField=@(new
    List<string>{"Singapore"})></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="EmployeeID"
  ContentField="ShipName"></KanbanCardSettings>
</SfKanban>
@code {
public class CustomAdaptor : DataAdaptor
{
  OrderContext db = new OrderContext();
  public CustomAdaptor()
  {
  }
  public async override Task<object> ReadAsync(DataManagerRequest dm, string
  key = null)
  {
    OrderService _craftService = new OrderService();
    IEnumerable DataSource = await _craftService.GetOrdersAsync();
    int count = DataSource.Cast<Order>().Count();
    return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
    count } : (object)DataSource;
  }
  public override Task<object> InsertAsync(DataManager dm, object value,
  string key)
  {
    (value as Order).EmployeeID = db.Orders.Select(x => x.EmployeeID).Max() + 1;
    db.Orders.Add((Order)value);
    db.SaveChangesAsync();
    return Task.Run(() =>
    {
      return value;
    });
  }
  public override async Task<object> UpdateAsync(DataManager dataManager,
  object value, string keyField, string key)
```

```
{
var data = db.Orders.Where(or => or.EmployeeID == (value as
Order).EmployeeID).FirstOrDefault();
if (data != null)
{
data.EmployeeID = (value as Order).EmployeeID;
data.ShipCity = (value as Order).ShipCity;
data.ShipName = (value as Order).ShipName;
}
db.SaveChangesAsync();
return Task.Run(() =>
{
return value;
});
}
public override async Task<object> RemoveAsync(DataManager dataManager,
object value, string keyField, string key)
{
Order ord = db.Orders.Find(Convert.ToInt32(value));
db.Orders.Remove(ord);
db.SaveChangesAsync();
return Task.Run(() =>
{
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 = db.Orders.Where(or => or.EmployeeID ==
rec.EmployeeID).FirstOrDefault();
val.EmployeeID = (rec as Order).EmployeeID;
val.ShipCity = (rec as Order).ShipCity;
val.ShipName = (rec as Order).ShipName;
}
}
if (Added != null)
{
foreach (var rec in (IEnumerable<Order>)Added)
{
db.Orders.Add(rec);
}
}
if (Deleted != null)
{
foreach (var rec in (IEnumerable<Order>)Deleted)
{
db.Orders.Remove(db.Orders.Where(or => or.EmployeeID ==
rec.EmployeeID).FirstOrDefault());
}
}
db.SaveChanges();
}
```

```
return db.Orders;
}
}
}
```

You can find the fully working sample [here](#).

### Observable collection

This [ObservableCollection](#) (dynamic data collection) shows notifications when the items are added, removed, and moved. Implementing the [INotifyCollectionChanged](#) will notify when there is any dynamic change (add, remove, move, and clear) in the collection. Implementing the [INotifyPropertyChanged](#) will notify when the property value has been changed on the client side.

Here, the Order class implements the interface of **INotifyPropertyChanged** and it raises the event when the Status property value was changed.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Notifications
@using System.Collections.ObjectModel;
@using System.ComponentModel;
<div class="col-lg-12 control-section">
<div class="content-wrapper" id="toast-kanban-observable">
<div class="row">
<div class="btn" style="margin: 0 0 7px 7px;">
<SfButton @onclick="AddRecord">Add Card</SfButton>
<SfButton @onclick="DeleteRecord">Delete Card</SfButton>
<SfButton @onclick="UpdateRecord">Update Card</SfButton>
</div>
<SfKanban KeyField="Status" DataSource="@ObservableData">
<KanbanColumns>
@foreach (ColumnModel item in columnData)
{
<KanbanColumn HeaderText="@item.HeaderText" KeyField="@item.KeyField"
AllowAdding="true"></KanbanColumn>
}
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
</SfKanban>
<SfToast @ref="ToastObj" ID="toast_type" Content="@ToastContent"
Timeout=2000 Target="@ToastTarget">
<ToastPosition X="Right" Y="Top"></ToastPosition>
</SfToast>
</div>
</div>
</div>
@code{
SfToast ToastObj;
private List<ColumnModel> columnData = new List<ColumnModel>() {
new ColumnModel(){ HeaderText= "To Do", KeyField= new List<string>() {
"Open" } },
new ColumnModel(){ HeaderText= "In Progress", KeyField= new List<string>() {
"In Progress" } },
```



```

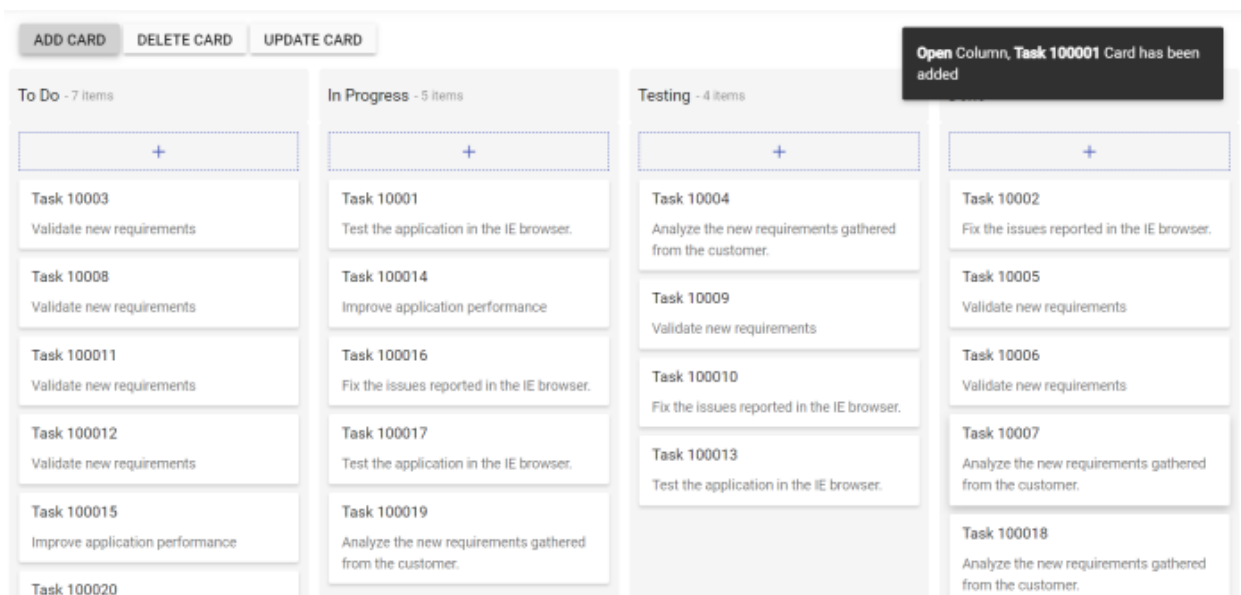
new ColumnModel(){ HeaderText= "Testing", KeyField= new List<string>() {
"Testing" } },
new ColumnModel(){ HeaderText= "Done", KeyField=new List<string>() { "Close"
} }
};
public ObservableCollection<ObservableDatas> ObservableData { get; set; }
private string ToastContent { get; set; }
List<ObservableDatas> Tasks = new List<ObservableDatas>();
private int AddUniqueId { get; set; }
private int UpdateUniqueId { get; set; }
private string ToastTarget { get; set; } = "#toast-kanban-observable";
protected override void OnInitialized()
{
Tasks = Enumerable.Range(1, 20).Select(x => new ObservableDatas()
{
Id = "Task 1000" + x,
Status = (new string[] { "Open", "In Progress", "Testing", "Close" })[new
Random().Next(4)],
Summary = (new string[] { "Analyze the new requirements gathered from the
customer.", "Improve application performance", "Fix the issues reported in
the IE browser.", "Validate new requirements", "Test the application in the
IE browser." })[new Random().Next(5)],
Assignee = (new string[] { "Nancy Davloio", "Andrew Fuller", "Janet
Leverling", "Steven walker", "Margaret hamilt", "Michael Suyama", "Robert
King" })[new Random().Next(7)],
}).ToList();
ObservableData = new ObservableCollection<ObservableDatas>(Tasks);
}
public async Task AddRecord()
{
var TaskId = "Task 10000" + ++AddUniqueId;
this.ToastContent = "<b>Open</b> Column, <b>" + TaskId + "</b> Card has been
added";
await Task.Delay(100);
ObservableData.Add(new ObservableDatas() { Id = TaskId, Status = "Open",
Summary = "Improve application performance", Assignee = "Janet Leverling"
});
await this.ToastObj.ShowAsync();
}
public async Task DeleteRecord()
{
if (ObservableData.Count() != 0)
{
this.ToastContent = "<b>" + ObservableData.First().Status + "</b> Column,
<b>" + ObservableData.First().Id + "</b> Card has been deleted";
await Task.Delay(100);
ObservableData.Remove(ObservableData.First());
await this.ToastObj.ShowAsync();
}
}
public async Task UpdateRecord()
{
if (ObservableData.Count() != 0)
{
var updateId = ++UpdateUniqueId;
var data = ObservableData[updateId];

```

```

this.ToastContent = "<b>" + data.Status + "</b> Column, <b>" + data.Id +
"</b> Card has been updated";
await Task.Delay(100);
data.Summary = "Card Updated";
await this.ToastObj.ShowAsync();
}
}
public class ObservableDatas : INotifyPropertyChanged
{
    public string Id { get; set; }
    private string status { get; set; }
    public string Status
    {
        get { return status; }
        set
        {
            this.status = value;
            NotifyPropertyChanged("Status");
        }
    }
    public string Summary { get; set; }
    public string Assignee { get; set; }
    public event PropertyChangedEventHandler PropertyChanged;
    private void NotifyPropertyChanged(string propertyName)
    {
        var handler = PropertyChanged;
        if (handler != null)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
}

```



## Columns in Blazor Kanban Component

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

### Single-key mapping

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

---

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

---

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
</SfKanban>

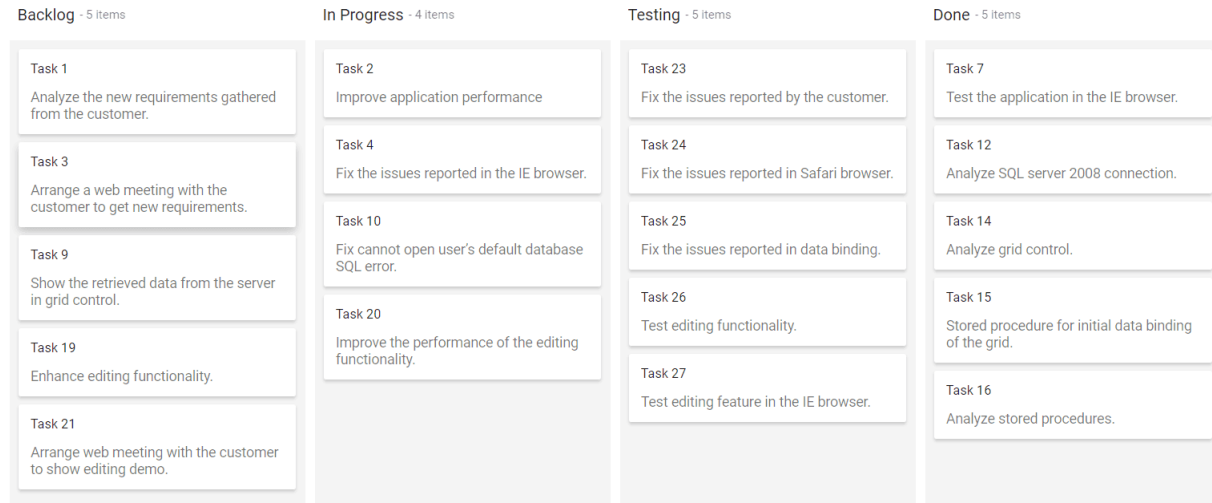
@code {
public class TasksModel
{
    public string Id { get; set; }
    public string Title { get; set; }
    public string Status { get; set; }
    public string Summary { get; set; }
    public string Assignee { get; set; }
}

public List<TasksModel> Tasks = new List<TasksModel>()
{
    new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
    Summary = "Analyze the new requirements gathered from the customer.",
    Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
    Summary = "Improve application performance", Assignee = "Andrew Fuller" },
    new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
    Summary = "Arrange a web meeting with the customer to get new
    requirements.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
    Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
    Leverling" },
    new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
    Summary = "Fix the issues reported by the customer.", Assignee = "Steven
    walker" },
}
```

```
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
```

```
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}
```

Output be like the below.



### Multi-key mapping

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

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>() { "Open",
    "Validate" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
</SfKanban>

@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
}
```

```
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
    new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
        Summary = "Analyze the new requirements gathered from the customer.",
        Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
        Summary = "Improve application performance", Assignee = "Andrew Fuller" },
    new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
        Summary = "Arrange a web meeting with the customer to get new
        requirements.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
        Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
        Leverling" },
    new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
        Summary = "Fix the issues reported by the customer.", Assignee = "Steven
        walker" },
    new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
        Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
        Davloio" },
    new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
        Summary = "Test the application in the IE browser.", Assignee = "Margaret
        hamilt" },
    new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
        Summary = "Validate the issues reported by the customer.", Assignee =
        "Steven walker" },
    new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
        Summary = "Show the retrieved data from the server in grid control.",
        Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
        "InProgress", Summary = "Fix cannot open user's default database SQL
        error.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
        Summary = "Fix the issues reported in data binding.", Assignee = "Janet
        Leverling" },
    new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
        Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
    },
    new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
        Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
        Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
        Summary = "Stored procedure for initial data binding of the grid.", Assignee
        = "Steven walker" },
    new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
        Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
        Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
        Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
        Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
}
```

```

new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}

```

Output be like the below.

The image shows a Kanban board with four columns. Each column has a header indicating the status and the number of items. The tasks are distributed as follows:

- Backlog - 8 items:** Task 1 (Analyze the new requirements gathered from the customer.), Task 3 (Arrange a web meeting with the customer to get new requirements.), Task 9 (Show the retrieved data from the server in grid control.), Task 19 (Enhance editing functionality.), Task 21 (Arrange web meeting with the customer to show editing demo.), Task 8 (Validate the issues reported by the customer.), Task 13 (Validate databinding issues.), Task 17 (Validate editing issues.).
- In Progress - 4 items:** Task 2 (Improve application performance), Task 4 (Fix the issues reported in the IE browser.), Task 10 (Fix cannot open user's default database SQL error.), Task 20 (Improve the performance of the editing functionality.).
- Testing - 5 items:** Task 23 (Fix the issues reported by the customer.), Task 24 (Fix the issues reported in Safari browser.), Task 25 (Fix the issues reported in data binding.), Task 26 (Test editing functionality.), Task 27 (Test editing feature in the IE browser.).
- Done - 5 items:** Task 7 (Test the application in the IE browser.), Task 12 (Analyze SQL server 2008 connection.), Task 14 (Analyze grid control.), Task 15 (Stored procedure for initial data binding of the grid.), Task 16 (Analyze stored procedures.).

### Header text

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

### Header template

You can customize the column header with `Template` property as shown in the following code.

To get start quickly with Blazor Kanban component using Templates, you can check on this video

{% youtube

"youtube:https://www.youtube.com/watch?v=PjTgXuibeI8" %}

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) ">
      <Template>
        @ {
          KanbanColumn column = (KanbanColumn)context;
          <div class="header-template-wrap">
            <div class="header-icon e-icons @column.KeyField[0]"></div>
            <div class="header-text">@column.HeaderText</div>
          </div>
        }
      </Template>
    </KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) ">
      <Template>
        @ {
          KanbanColumn column = (KanbanColumn)context;
          <div class="header-template-wrap">
            <div class="header-icon e-icons @column.KeyField[0]"></div>
            <div class="header-text">@column.HeaderText</div>
          </div>
        }
      </Template>
    </KanbanColumn>
    <KanbanColumn HeaderText="Review" KeyField="@ (new List<string>()
    { "Review" }) ">
      <Template>
        @ {
          KanbanColumn column = (KanbanColumn)context;
          <div class="header-template-wrap">
            <div class="header-icon e-icons @column.KeyField[0]"></div>
            <div class="header-text">@column.HeaderText</div>
          </div>
        }
      </Template>
    </KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>() { "Close" }) ">
      <Template>
        @ {
          KanbanColumn column = (KanbanColumn)context;
          <div class="header-template-wrap">
            <div class="header-icon e-icons @column.KeyField[0]"></div>
            <div class="header-text">@column.HeaderText</div>
          </div>
        }
      </Template>
    </KanbanColumn>
  </KanbanColumns>
</SfKanban>
```



```

</Template>
</KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
</SfKanban>
<style type="text/css">
.e-kanban .header-template-wrap {
display: inline-flex;
font-size: 15px;
font-weight: 400;
}
.e-kanban .header-template-wrap .header-icon {
font-family: "KanbanHeaderIcons";
margin-top: 3px;
width: 10%;
}
.e-kanban .header-template-wrap .header-text {
margin-left: 15px;
}
.e-kanban .card-header {
padding-left: 12px;
}
.e-kanban .Open::before {
content: "\e700";
color: #0251cc;
font-size: 16px;
}
.e-kanban .InProgress::before {
content: "\e703";
color: #ea9713;
font-size: 16px;
}
.e-kanban .Review::before {
content: "\e701";
color: #8e4399;
font-size: 16px;
}
.e-kanban .Close::before {
content: "\e702";
color: #63ba3c;
font-size: 16px;
}
@@font-face {
font-family: "KanbanHeaderIcons";
src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj1tSfUAAAEoAAAAVmNtYXDnE+dkAAABlAAAADxnbHlmg4w
eAgAAAdwAAAhQaGVhZBfH57sAAADQAAAAANmhoZWEIVQQGAAAArAAAACRobXR4FAAAAAAAYAAAAA
UbG9jYQNeBi4AAAAHQAAAADG1heHABGAfGAAABCAAAACBuYw1lH65UOQAACiWAAALNcG9zdFsyKlE
AAAz8AAAAUgABAAAEAAAAAFwEAAAAAAD+AAABAAAAAAAAAAAAAAAAABQABAAAAAQAA7pb8lF8
PPPUACwQAAAAANpY0WMAAAAAA2ljRYwAAAAAD+AP4AAAAACAACAAAAAAAAAAAAEAAAFAVQACQAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAAAQQA ZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EAAAAABAAAAAQAAAAAAAAAAAAAaAAAAAwAAABQAAAwABAAAAFAAEACgAAAAEEAAQAAQAA5wP//wAA5wD//wA
AAAEABAAAAAEAAgADAAQAAAAAMwCBgKSBCgABAAAAAAD+AP4ACEAQwBlAKkAAAEfBw8HIS8HPwc
lHwcPByEvBz8HJR8HDwchLwc/BycRHw8hPw8RLw8hDw4CXgcGBQUEAwEBAQEDBAUFBGf+hqYGBQU

```

```

EAWEB AQEDBAUFBgYCOAYGBQUEAwEBAQEDBAUFBg9yAYGBQUEAwEBAQEDBAUFBgYCOAYGBQUEAwE
BAQEDBAUFBg9yAYGBQUEAwEBAQEDBAUFBgbcAQIDBQUHCAkKCgSMDQ0ODQLgDQ4NDQwLCgoJCAc
FBQMCAQECaAwUFBwgJCgoLDA0NDg39IA0ODQ0MCwoKCQgHBQUdAgFDAQEDBAUFBgYHBgUFBAMBAQE
BAwQFBQYHBgYFBQQDAQg9AQEDBAUFBgCGBgUFBAMBAQEBAwQFBQYGBwYFBQQDAQg9AQEDBAUFBgY
HBgUFBAMBAQEBAwQFBQYHBgYFBQQDAQgZ / SANDg0NDAsKCgkIBwUFAwIBAQIDBQUHCAkKCgSMDQ0
ODQLgDQ4NDQwLCgoJCAcFBQMCAQECaAwUFBwgJCgoLDA0NDgAABAAAAAAD+AP4AD8AggDUARgAAAE
fBw8PLw41Pw8fBicPDx8PMz8OLxAHNzMfEhUPESsBLx9AT8UJREfDyE / DxEvDyEPDgJlCacGBgQ
CAGEBAGMEBQCHCAkJCwsMDAwnDgwnDAsLCgkICAYFAwMBAQMDBQUHBwgJCQoLCwwMDA4MDAwLCgq
EDg8PDw4PDw8VFBQUEXmTEhUWFhYXfXgYehMSERISEREUEBEREBESERkZGRgXFxcXEA8QEBARERE
WFxYVFhUWFhIeFASXGBkYGRkYGSATEXISEhIRBQMBAGICHBkaGhscGx0UEXmTEhUWFhUFRQVFBu
VHBoaGhkYGRkeAgIDGBQVFhYXfXcREREQEREQE8ODv4aAQIDBQUHCAkKCgSMDQ0ODQLgDQ4NDQw
LCgoJCAcFBQMCAQECaAwUFBwgJCgoLDA0NDg39IA0ODQ0MCwoKCQgHBQUdAgJXCQoKCwsMDAwnDAw
MCgSJCQgHBgUEAwIBAQIDBQUHCAkJCgSMCw0MDQwLDAoLCQkJBwcGBQQAQgEBAgMEBQYIwQMEBQY
GBwgJDg4PERETEXUYFhUTEhAPDgkIBwUFAwEBAgIEBQYHCA0QEBMUfHcaEREQDw8NDQ0PDQsJCAy
EAWEBMAIEBggJDA4PFg8PERESFBQBwYGBgUEIBsZFhUTERAJCAyGBAMCAGQFBggJChAREhUWGBBo
eCAUFBAyHGxcVFBMREQ8KCQgHBgYEBAMCAyT9IA0ODQ0MCwoKCQgHBQUdAgEBAgMFBQcICQoKCww
NDQ4NAuANDg0NDAsKCgkIBwUFAwIBAQIDBQUHCAkKCgSMDQ0OAAIAAAAAA / gD+AArAG8AAAEfAhU
PAwEPay8INT8GMx8DAT8DHwILER8PIT8PES8PIQ8OAvMEAwIBAQME / r8FBQYGBgYFBXkeAwEBAgM
EBQUGBgYGBgViASoFBgYGBgYF / RoBAgMFBQcICQoKCwwNDQ4NAuANDg0NDAsKCgkIBwUFAwIBAQI
DBQUHCAkKCgSMDQ0ODf0gDQ4NDQwLCgoJCAcFBQMCArQFBgYGBgYFBf7FBAMBAQEBAwR2BQUGBgY
GBgUEAwEBAgMEYAE1BAMBAQEBA7j9IA0ODQ0MCwoKCQgHBQUdAgEBAgMFBQcICQoKCwwNDQ4NAuA
NDg0NDAsKCgkIBwUFAwIBAQIDBQUHCAkKCgSMDQ0OAAAJAAAAAP4A / gAIQBDAGUAhwCpAMsA7QE
PAVMAAAEVDwcvBzU / Bx8GNx8EDwYrAS8GPQE / BTsBHwEFHwMPBysBLwU9AT8GOWeFASUfBw8HIY8
HPwchHwcPByMvBz8HJR8DDwcrAS8FPQE / BjsBHwEFHwMDAQ8FKwEvBz8GOWeFASUVDwcvBzU / Bx8
GJREfDyE / DxEvDyEPDgIgAQIDBAQGBgYGBgYEBAMCAQECaAwQEBgYGBgYGBAQDAopiBAMCAQECaAwQ
FBQYGBgYFBWIEAwICaWQFBQYGBgYF / t8EAWIBAQIDBGIFBQYGBgYFBQQDAgIDBGIFBQYGBgYFAdw
HBgUFBAMBAQEBAwQFBQYHigYGBgQEAWIBAQIDBAQGBgB+YAYGBgQEAWIBAQIDBAQGBgaKBwYFBQQ
DAQEBAQMEBQUGBwJlBAMCAQECaWRiBQUGBgYGBQUEAwICaWRiBQUGBgYGBf4bYgQDAgIDBAUFBgY
GBgUFYgQDAgEBAgMEBQUGBgYGBQEEAQIDBAQGBgYGBgYEBAMCAQECaAwQEBgYGBgYGBAQDAv3pAQI
DBQUHCAkKCgSMDQ0ODQLgDQ4NDQwLCgoJCAcFBQMCAQECaAwUFBwgJCgoLDA0NDg39IA0ODQ0MCwo
KCQgHBQUdAgEwigcGBQUEAwEBAQEDBAUFBgEKBgYGBAQDAgEBAgMEBAYGTWIFBQYGBgYFBQQDAgI
DBGIFBQYGBgYFBQQDAgIDBAUFBgYGBgYFYGQDAgIDBAUFBgYGBgYFYGQDAgIDmQECaWQEBgYGBgY
GBAQDAgEBAgMEBAYGBgYGBgQEAWIBAQIDBAQGBgYGBgYEBAMCAQECaAwQEBgYGBgYGBAQDAgHrBQU
GBgYGBQViBAMCAGMEBQUGBgYGBQViBAMCAGMEYgUFBgYGBgUFBAMCAGMEYgUFBgYGBgUFBAMCAGN
LigYGBgQEAWIBAQIDBAQGBgaKBwYFBQQDAQEBAQMEBQUGD / 0gDQ4NDQwLCgoJCAcFBQMCAQECaAwU
FBwgJCgoLDA0NDg0C4A0ODQ0MCwoKCQgHBQUdAgEBAgMFBQcICQoKCwwNDQ4AAAAAEgDeAAEAAAA
AAAAAAQAAAAEAAAAAAAEAFQABAAEAAAAAAAIABwAWAAEAAAAAAAMAFQAdAAEAAAAAAQAQFQAYAAE
AAAAAAUACwBHAAEAAAAAAAYAFQBSAAEAAAAAAAOALABnAAEAAAAAAASAEgCTAAMAAQQJAAAAAgC
lAAMAAQQJAAEAkGcNaAMAAQQJAAIADgDRAAMAAQQJAAAMAKgDfAAMAAQQJAAQAKgEJAAMAAQQJAAU
AFgEzAAMAAQQJAAAYAKgFJAAMAAQQJAAoAWAFzAAMAAQQJAAAsAJAHLIethbmJhbiBwcmllvcml0eSB
pY29uc1JlZ3VsYXJlYW5iYW4gcHJpb3JpdHkgaWNvbnNlYW5iYW4gcHJpb3JpdHkgaWNvbnNWZXJ
zaW9uIDEuMETHbmJhbiBwcmllvcml0eSBpY29uc0ZvbncgZ2VuZXJhdGVkIHVzaW5nIFN5bmNmdXN
pb24gTWV0cm8gU3RlZGlvd3d3LnN5bmNmdXNpb24uY29tACAASwBhAG4AYgBhAG4AIABwAHIAaQB
vAHIAaQB0AHkAIABpAGMabwBuAHMAUgBlAGcAdQBsaGEAcgBLAGEAbgBiAGEAbgAgAHAACgBpAG8
AcgBpAHQAEQAgAGkAYwBvAG4AcwBLAGEAbgBiAGEAbgAgAHAACgBpAG8AcgBpAHQAEQAgAGkAYwB
vAG4AcwBWAGUAcgBzAGkAbwBuACAAMQAuADAASwBhAG4AYgBhAG4AIABwAHIAaQBvAHIAaQB0AHk
AIABpAGMabwBuAHMARgBvAG4AdAAgAGcAZQBuAGUAcgBhAHQAZQBkACAAdQBzAGkAbgBnACAAUwB
5AG4AYwBmAHUAcwBpAG8AbgAgAE0AZQB0AHIAbwAgAFMAdABlAGQAaQBvAHkAdwB3AC4AcwB5AG4
AYwBmAHUAcwBpAG8AbgAgAuAGMabwBtAAAAAIAAAAAAAACgAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ABQECAQMBBAEFAQYACFRvZG9saXN0BlJldmllldwldB21wbGV0ZWQIUHJvZ3Jlc3MAAAAA)
format("trueType");
font-weight: normal;
font-style: normal;
}
[class^="sf-icon-"],
[class*=" sf-icon-"] {
font-family: "KanbanHeaderIcons" !important;

```

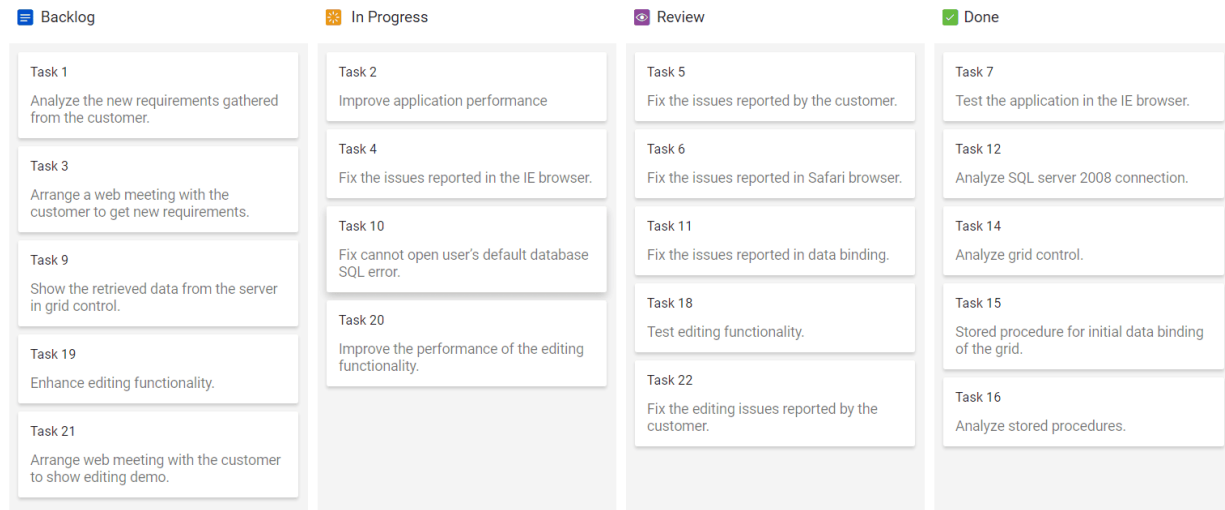
```

speak: none;
font-size: 55px;
font-style: normal;
font-weight: normal;
font-variant: normal;
text-transform: none;
line-height: 1;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
}
</style>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
},

```

```
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}
```

Output be like the below.



### Toggle columns

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

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

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>() { "Open" })"
      AllowToggle="true"></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
      { "InProgress" })" AllowToggle="true"></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
      { "Testing" })" AllowToggle="true"></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>() { "Close" })"
      AllowToggle="true"></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
    ContentField="Summary"></KanbanCardSettings>
</SfKanban>

@code {
public class TasksModel
{
    public string Id { get; set; }
    public string Title { get; set; }
    public string Status { get; set; }
    public string Summary { get; set; }
    public string Assignee { get; set; }
}

public List<TasksModel> Tasks = new List<TasksModel>()
{
    new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
        Summary = "Analyze the new requirements gathered from the customer.",
        Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
        Summary = "Improve application performance", Assignee = "Andrew Fuller" },
}
```

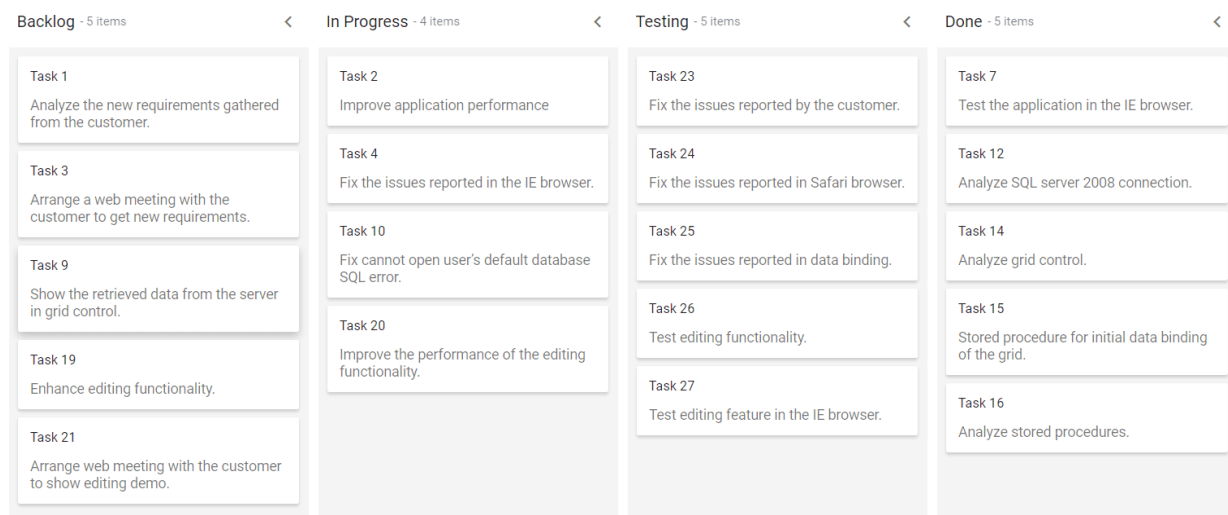
```
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
```

```

new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}

```

Output be like the below.



### Initially collapsed column

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

The `IsExpanded` property only works when enabling the `AllowToggle` property on particular column.

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

### ASPX-CS

```

@using Syncfusion.Blazor.Kanban
<SfKanban KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>() { "Open" })"
AllowToggle="true" IsExpanded="false"></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{ "InProgress" })" AllowToggle="true"></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
{ "Testing" })" AllowToggle="true" IsExpanded="false"></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>() { "Close" })"
AllowToggle="true"></KanbanColumn>
  </KanbanColumns>

```

```
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
}
```

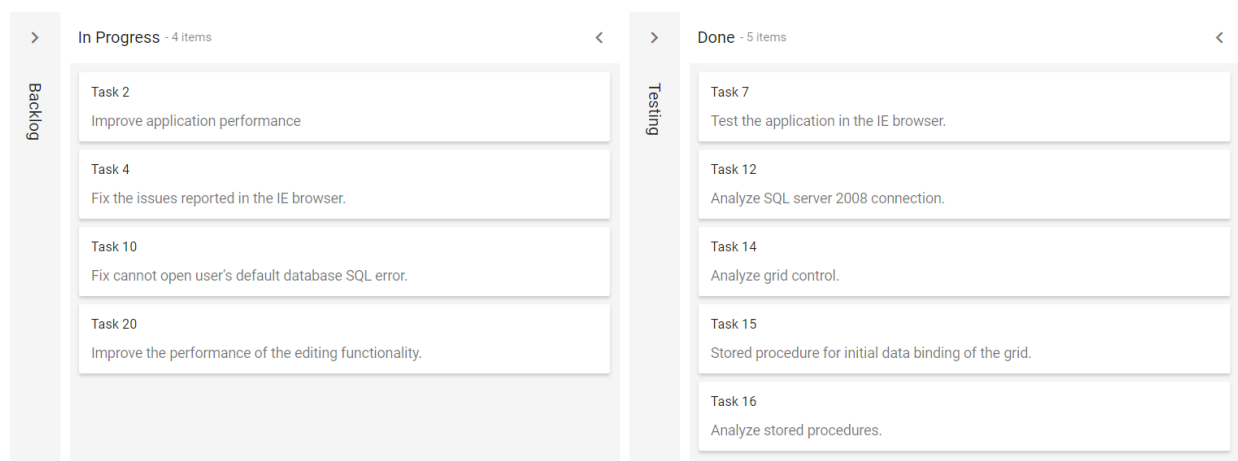


```

new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}

```

Output be like the below.



### Stacked headers

Stacked headers are the additional headers to column header that will group the similar columns. Define the grouping of columns **Key** value to the **KeyField** property and provide the custom header text name to grouped columns using the **Text** property in **KanbanStackedHeaders**.

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

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="To Do" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanStackedHeaders>
    <KanbanStackedHeader Text="To Do" KeyFields="@ (new List<string>()
    { "Open" }) "></KanbanStackedHeader>
    <KanbanStackedHeader Text="Development Phase" KeyFields="@ (new
    List<string>() { "InProgress", "Testing" }) "></KanbanStackedHeader>
    <KanbanStackedHeader Text="Done" KeyFields="@ (new List<string>()
    { "Close" }) "></KanbanStackedHeader>
  </KanbanStackedHeaders>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
</SfKanban>

@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}

public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
```

```
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
```

```
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}
```

Output be like the below.

To Do	Development Phase		Done
To Do - 5 items	In Progress - 4 items	Testing - 5 items	Done - 5 items
<div>Task 1</div> <div>Analyze the new requirements gathered from the customer.</div>	<div>Task 2</div> <div>Improve application performance</div>	<div>Task 23</div> <div>Fix the issues reported by the customer.</div>	<div>Task 7</div> <div>Test the application in the IE browser.</div>
<div>Task 3</div> <div>Arrange a web meeting with the customer to get new requirements.</div>	<div>Task 4</div> <div>Fix the issues reported in the IE browser.</div>	<div>Task 24</div> <div>Fix the issues reported in Safari browser.</div>	<div>Task 12</div> <div>Analyze SQL server 2008 connection.</div>
<div>Task 9</div> <div>Show the retrieved data from the server in grid control.</div>	<div>Task 10</div> <div>Fix cannot open user's default database SQL error.</div>	<div>Task 25</div> <div>Fix the issues reported in data binding.</div>	<div>Task 14</div> <div>Analyze grid control.</div>
<div>Task 19</div> <div>Enhance editing functionality.</div>	<div>Task 20</div> <div>Improve the performance of the editing functionality.</div>	<div>Task 26</div> <div>Test editing functionality.</div>	<div>Task 15</div> <div>Stored procedure for initial data binding of the grid.</div>
<div>Task 21</div> <div>Arrange web meeting with the customer to show editing demo.</div>		<div>Task 27</div> <div>Test editing feature in the IE browser.</div>	<div>Task 16</div> <div>Analyze stored procedures.</div>

## Cards in Blazor Kanban Component

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

### Drag-and-drop

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

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

### Header

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

The `HeaderField` property must be a unique datasource value to avoid the duplication of card data.

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

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
```

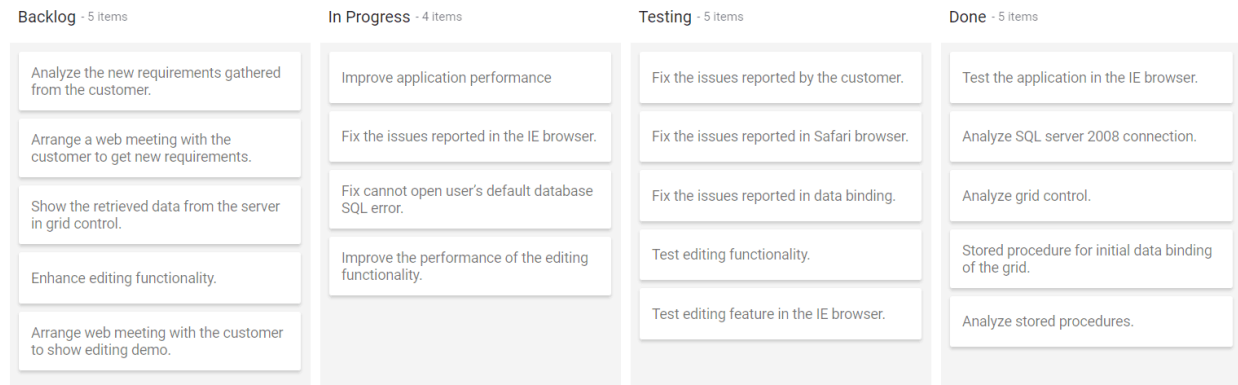
```

<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings ShowHeader="false" HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },

```

```
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}
```

Output be like the below.



## Content

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

## Tags

The card tags are used to display the tag text with the background color. Each tag text is separated and shown below the card content. It can be achieved by mapping the data key to the `TagsField` property, which is placed inside the `KanbanCardSettings` property.

The mapped datasource key value contains single or multiple tags. If it is multiple tags, each tag will be separated by a comma in the datasource.

## ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id" ContentField="Summary"
  TagsField="CardTags"></KanbanCardSettings>
</SfKanban>

@code {
public class TasksModel
{
public string Id { get; set; }
public int ListId { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Type { get; set; }
public string Priority { get; set; }
public List<string> CardTags { get; set; }
public string Tags { get; set; }
public double Estimate { get; set; }
public string Assignee { get; set; }
```

```

public int RankId { get; set; }
public string Color { get; set; }
public string Value { get; set; }
public string OrderID { get; set; }
public string Size { get; set; }
public string ImageURL { get; set; }
public string Description { get; set; }
public string Category { get; set; }
public string Price { get; set; }
public string AssigneeKey { get; set; }
public List<string> ClassName { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
    new TasksModel { Id = "Task 1", Title = "Task - 29001", Status = "Open",
        Summary = "Analyze the new requirements gathered from the customer.", Type =
        "Story", Priority = "Low", CardTags = new List<string>() { "Analyze",
        "Customer" }, Estimate = 3.5, Assignee = "Nancy Davloio", AssigneeKey =
        "Nancy Davloio", RankId = 1, Color = "#8b447a", ClassName = new
        List<string>() { "e-story", "e-low", "e-nancy" } },
    new TasksModel { Id = "Task 2", Title = "Task - 29002", Status =
        "InProgress", Summary = "Improve application performance", Type =
        "Improvement", Priority = "Normal", CardTags = new List<string>() {
        "Improvement" }, Estimate = 6, Assignee = "Andrew Fuller", AssigneeKey =
        "Andrew Fuller", RankId = 1, Color = "#7d7297", ClassName = new
        List<string>() { "e-improvement", "e-normal", "e-andrew" } },
    new TasksModel { Id = "Task 3", Title = "Task - 29003", Status = "Open",
        Summary = "Arrange a web meeting with the customer to get new
        requirements.", Type = "Others", Priority = "Critical", CardTags = new
        List<string>() { "Meeting" }, Estimate = 5.5, Assignee = "Janet Leverling",
        AssigneeKey = "Janet Leverling", RankId = 2, Color = "#27AE60", ClassName =
        new List<string>() { "e-others", "e-critical", "e-janet" } },
    new TasksModel { Id = "Task 4", Title = "Task - 29004", Status =
        "InProgress", Summary = "Fix the issues reported in the IE browser.", Type =
        "Bug", Priority = "Release Breaker", CardTags = new List<string>() { "IE" },
        Estimate = 2.5, Assignee = "Janet Leverling", AssigneeKey = "Janet
        Leverling", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
        "e-bug", "e-release", "e-janet" } },
    new TasksModel { Id = "Task 5", Title = "Task - 29005", Status = "Review",
        Summary = "Fix the issues reported by the customer.", Type = "Bug", Priority
        = "Low", CardTags = new List<string>() { "Customer" }, Estimate = 3.5,
        Assignee = "Steven walker", AssigneeKey = "Steven walker", RankId = 1, Color
        = "#cc0000", ClassName = new List<string>() { "e-bug", "e-low", "e-steven" }
        },
    new TasksModel { Id = "Task 6", Title = "Task - 29007", Status =
        "Validate", Summary = "Validate new requirements", Type = "Improvement",
        Priority = "Low", CardTags = new List<string>() { "Validation" }, Estimate =
        1.5, Assignee = "Robert King", AssigneeKey = "Robert King", RankId = 1,
        Color = "#7d7297", ClassName = new List<string>() { "e-improvement", "e-
        low", "e-robert" } },
    new TasksModel { Id = "Task 7", Title = "Task - 29009", Status = "Review",
        Summary = "Fix the issues reported in Safari browser.", Type = "Bug",
        Priority = "Release Breaker", CardTags = new List<string>() { "Fix",
        "Safari" }, Estimate = 1.5, Assignee = "Nancy Davloio", AssigneeKey = "Nancy
        Davloio", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
        "e-bug", "e-release", "e-nancy" } },

```

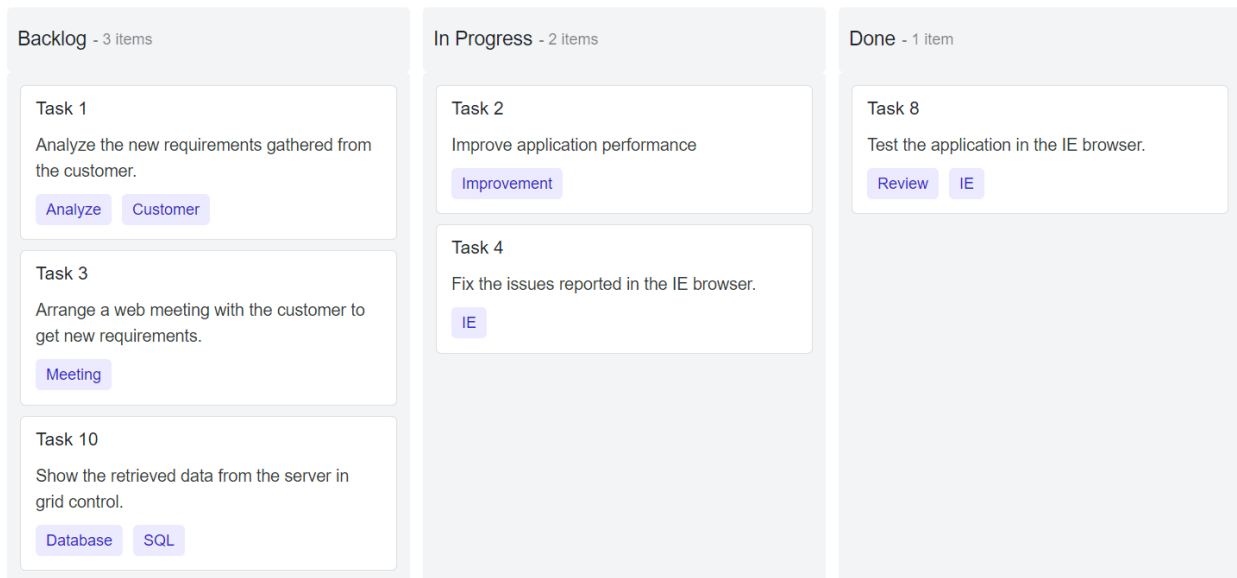


```

new TasksModel { Id = "Task 8", Title = "Task - 29010", Status = "Close",
Summary = "Test the application in the IE browser.", Type = "Story",
Priority = "Low", CardTags = new List<string>() { "Review", "IE" }, Estimate
= 5.5, Assignee = "Margaret hamilt", AssigneeKey = "Margaret hamilt", RankId
= 3, Color = "#8b447a", ClassName = new List<string>() { "e-story", "e-low",
"e-Margaret" } },
new TasksModel { Id = "Task 9", Title = "Task - 29011", Status =
"Validate", Summary = "Validate the issues reported by the customer.", Type =
"Story", Priority = "High", CardTags = new List<string>() { "Validation",
"Fix" }, Estimate = 1, Assignee = "Steven walker", AssigneeKey = "Steven
walker", RankId = 1, Color = "#8b447a", ClassName = new List<string>() { "e-
story", "e-low", "e-nancy" } },
new TasksModel { Id = "Task 10", Title = "Task - 29015", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.", Type =
"Story", Priority = "High", CardTags = new List<string>() { "Database",
"SQL" }, Estimate = 5.5, Assignee = "Margaret hamilt", AssigneeKey =
"Margaret hamilt", RankId = 4, Color = "#8b447a", ClassName = new
List<string>() { "e-story", "e-high", "e-steven" } }
};
}

```

The output will be as follows.



#### Left border color

Kanban card supports to custom the left border color for all the cards. This can be achieved by mapping the data key value to the `GrabberField` property, which is placed inside the `KanbanCardSettings` property. The mapped data key value will be directly assigned to each card element border left color property.

By default, the card border left color width is 3px.

#### ASPX-CS

```

@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>

```

```

<KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
{ "Open" }) "></KanbanColumn>
<KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{ "InProgress" }) "></KanbanColumn>
<KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
{ "Close" }) "></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="Id" ContentField="Summary"
GrabberField="Color"></KanbanCardSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public int ListId { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Type { get; set; }
public string Priority { get; set; }
public List<string> CardTags { get; set; }
public string Tags { get; set; }
public double Estimate { get; set; }
public string Assignee { get; set; }
public int RankId { get; set; }
public string Color { get; set; }
public string Value { get; set; }
public string OrderID { get; set; }
public string Size { get; set; }
public string ImageURL { get; set; }
public string Description { get; set; }
public string Category { get; set; }
public string Price { get; set; }
public string AssigneeKey { get; set; }
public List<string> ClassName { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "Task - 29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.", Type =
"Story", Priority = "Low", CardTags = new List<string>() { "Analyze",
"Customer" }, Estimate = 3.5, Assignee = "Nancy Davloio", AssigneeKey =
"Nancy Davloio", RankId = 1, Color = "#8b447a", ClassName = new
List<string>() { "e-story", "e-low", "e-nancy" } },
new TasksModel { Id = "Task 2", Title = "Task - 29002", Status =
"InProgress", Summary = "Improve application performance", Type =
"Improvement", Priority = "Normal", CardTags = new List<string>() {
"Improvement" }, Estimate = 6, Assignee = "Andrew Fuller", AssigneeKey =
"Andrew Fuller", RankId = 1, Color = "#7d7297", ClassName = new
List<string>() { "e-improvement", "e-normal", "e-andrew" } },
new TasksModel { Id = "Task 3", Title = "Task - 29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Type = "Others", Priority = "Critical", CardTags = new
List<string>() { "Meeting" }, Estimate = 5.5, Assignee = "Janet Leverling",
AssigneeKey = "Janet Leverling", RankId = 2, Color = "#27AE60", ClassName =
new List<string>() { "e-others", "e-critical", "e-janet" } },

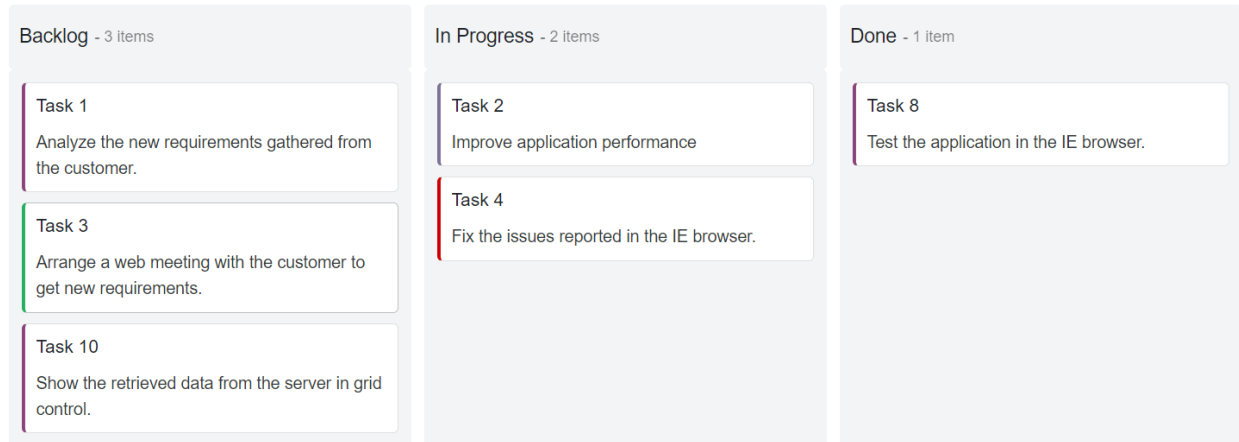
```

```

new TasksModel { Id = "Task 4", Title = "Task - 29004", Status =
"InProgress", Summary = "Fix the issues reported in the IE browser.", Type =
"Bug", Priority = "Release Breaker", CardTags = new List<string>() { "IE" },
Estimate = 2.5, Assignee = "Janet Leverling", AssigneeKey = "Janet
Leverling", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
"e-bug", "e-release", "e-janet" } },
new TasksModel { Id = "Task 5", Title = "Task - 29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Type = "Bug", Priority
= "Low", CardTags = new List<string>() { "Customer" }, Estimate = 3.5,
Assignee = "Steven walker", AssigneeKey = "Steven walker", RankId = 1, Color
= "#cc0000", ClassName = new List<string>() { "e-bug", "e-low", "e-steven" }
},
new TasksModel { Id = "Task 6", Title = "Task - 29007", Status =
"Validate", Summary = "Validate new requirements", Type = "Improvement",
Priority = "Low", CardTags = new List<string>() { "Validation" }, Estimate =
1.5, Assignee = "Robert King", AssigneeKey = "Robert King", RankId = 1,
Color = "#7d7297", ClassName = new List<string>() { "e-improvement", "e-
low", "e-robert" } },
new TasksModel { Id = "Task 7", Title = "Task - 29009", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Type = "Bug",
Priority = "Release Breaker", CardTags = new List<string>() { "Fix",
"Safari" }, Estimate = 1.5, Assignee = "Nancy Davloio", AssigneeKey = "Nancy
Davloio", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
"e-bug", "e-release", "e-nancy" } },
new TasksModel { Id = "Task 8", Title = "Task - 29010", Status = "Close",
Summary = "Test the application in the IE browser.", Type = "Story",
Priority = "Low", CardTags = new List<string>() { "Review", "IE" }, Estimate
= 5.5, Assignee = "Margaret hamilt", AssigneeKey = "Margaret hamilt", RankId
= 3, Color = "#8b447a", ClassName = new List<string>() { "e-story", "e-low",
"e-Margaret" } },
new TasksModel { Id = "Task 9", Title = "Task - 29011", Status =
"Validate", Summary = "Validate the issues reported by the customer.", Type
= "Story", Priority = "High", CardTags = new List<string>() { "Validation",
"Fix" }, Estimate = 1, Assignee = "Steven walker", AssigneeKey = "Steven
walker", RankId = 1, Color = "#8b447a", ClassName = new List<string>() { "e-
story", "e-low", "e-nancy" } },
new TasksModel { Id = "Task 10", Title = "Task - 29015", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.", Type =
"Story", Priority = "High", CardTags = new List<string>() { "Database",
"SQL" }, Estimate = 5.5, Assignee = "Margaret hamilt", AssigneeKey =
"Margaret hamilt", RankId = 4, Color = "#8b447a", ClassName = new
List<string>() { "e-story", "e-high", "e-steven" } }
};
}

```

The output will be as follows.



### Custom class

The card allows to render the custom elements based on the given class names inside the `e-card-footer` element. It can be achieved by mapping the data key to the `FooterCssField` property, which is placed inside the `KanbanCardSettings` property. It will help to create your own class name elements inside the `e-card-footer` element. The mapped datasource key value contains single or multiple class names. If it is multiple class names, each class name will be separated by a comma in the datasource.

In the following demo, images and icons are rendered using the `FooterCssField` property.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id" ContentField="Summary"
  FooterCssField="ClassName"></KanbanCardSettings>
</SfKanban>
<style>
.e-kanban .e-card .e-card-footer {
display: flex;
justify-content: space-between;
}
.e-kanban .e-card .e-card-footer > div:last-child {
margin-left: auto;
}
.e-kanban .e-card .e-card-footer .e-card-footer-css {
background-repeat: no-repeat;
background-size: cover;
background-position: center;
height: 16px;
width: 16px;
margin-right: 8px;
}
```

```
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-improvement {
background-image:
url (https://syncfusion.atlassian.net/secure/viewavatar?size=medium&avatarId=
15507&avatarType=issuetype);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-story {
background-image:
url (https://syncfusion.atlassian.net/secure/viewavatar?size=medium&avatarId=
15515&avatarType=issuetype);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-bug {
background-image:
url (https://syncfusion.atlassian.net/secure/viewavatar?size=medium&avatarId=
15503&avatarType=issuetype);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-others {
background-image:
url (https://syncfusion.atlassian.net/images/icons/issuetypes/documentation.p
ng);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-low {
background-image:
url (https://syncfusion.atlassian.net/images/icons/priorities/trivial.svg);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-high {
background-image:
url (https://syncfusion.atlassian.net/images/icons/priorities/major.svg);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-critical {
background-image:
url (https://syncfusion.atlassian.net/images/icons/priorities/critical.svg);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-release {
background-image:
url (https://syncfusion.atlassian.net/images/icons/priorities/critical.svg);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-normal {
background-image:
url (https://syncfusion.atlassian.net/images/icons/priorities/minor.svg);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-nancy {
background-image:
url (https://ej2.syncfusion.com/demos/src/kanban/images/Nancy%20Davloio.png);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-andrew {
background-image:
url (https://ej2.syncfusion.com/demos/src/kanban/images/Andrew%20Fuller.png);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-janet {
background-image:
url (https://ej2.syncfusion.com/demos/src/kanban/images/Janet%20Leverling.png
);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-steven {
background-image:
url (https://ej2.syncfusion.com/demos/src/kanban/images/Steven%20walker.png);
}
```

```

.e-kanban .e-card .e-card-footer .e-card-footer-css.e-robert {
background-image:
url(https://ej2.syncfusion.com/demos/src/kanban/images/Robert%20King.png);
}
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-nancy,
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-andrew,
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-janet,
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-steven,
.e-kanban .e-card .e-card-footer .e-card-footer-css.e-robert {
border-radius: 72px;
height: 30px;
width: 30px;
}
</style>
@code {
public class TasksModel
{
public string Id { get; set; }
public int ListId { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Type { get; set; }
public string Priority { get; set; }
public List<string> CardTags { get; set; }
public string Tags { get; set; }
public double Estimate { get; set; }
public string Assignee { get; set; }
public int RankId { get; set; }
public string Color { get; set; }
public string Value { get; set; }
public string OrderID { get; set; }
public string Size { get; set; }
public string ImageURL { get; set; }
public string Description { get; set; }
public string Category { get; set; }
public string Price { get; set; }
public string AssigneeKey { get; set; }
public List<string> ClassName { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "Task - 29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.", Type =
"Story", Priority = "Low", CardTags = new List<string>() { "Analyze",
"Customer" }, Estimate = 3.5, Assignee = "Nancy Davloio", AssigneeKey =
"Nancy Davloio", RankId = 1, Color = "#8b447a", ClassName = new
List<string>() { "e-story", "e-low", "e-nancy" } },
new TasksModel { Id = "Task 2", Title = "Task - 29002", Status =
"InProgress", Summary = "Improve application performance", Type =
"Improvement", Priority = "Normal", CardTags = new List<string>() {
"Improvement" }, Estimate = 6, Assignee = "Andrew Fuller", AssigneeKey =
"Andrew Fuller", RankId = 1, Color = "#7d7297", ClassName = new
List<string>() { "e-improvement", "e-normal", "e-andrew" } },
new TasksModel { Id = "Task 3", Title = "Task - 29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Type = "Others", Priority = "Critical", CardTags = new

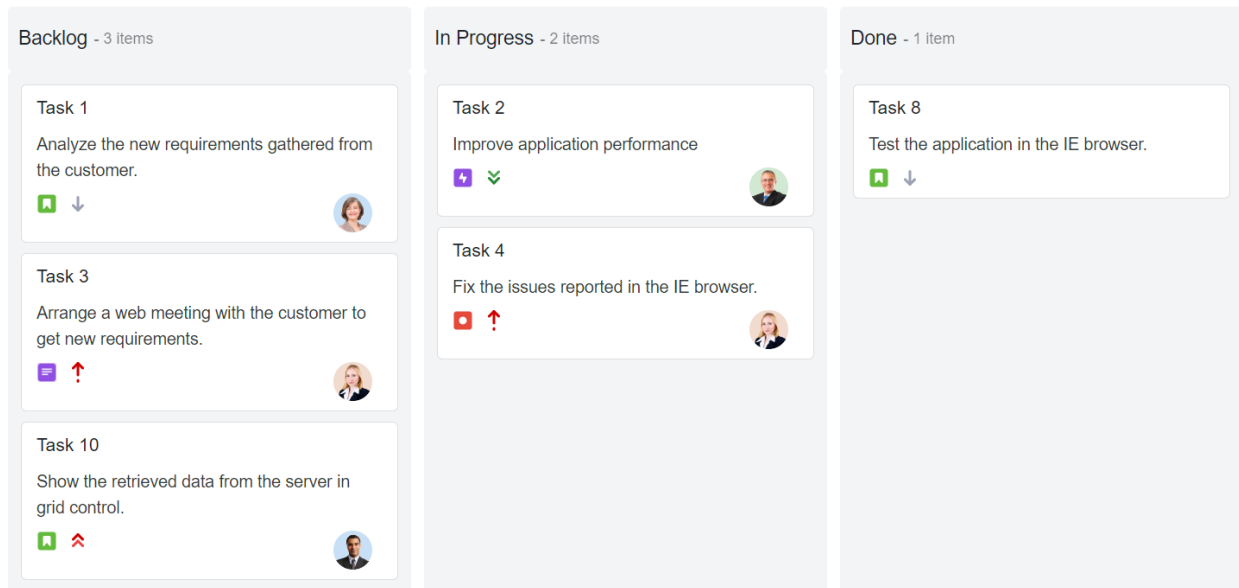
```

```

List<string>() { "Meeting" }, Estimate = 5.5, Assignee = "Janet Leverling",
AssigneeKey = "Janet Leverling", RankId = 2, Color = "#27AE60", ClassName =
new List<string>() { "e-others", "e-critical", "e-janet" } },
new TasksModel { Id = "Task 4", Title = "Task - 29004", Status =
"InProgress", Summary = "Fix the issues reported in the IE browser.", Type =
"Bug", Priority = "Release Breaker", CardTags = new List<string>() { "IE" },
Estimate = 2.5, Assignee = "Janet Leverling", AssigneeKey = "Janet
Leverling", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
"e-bug", "e-release", "e-janet" } },
new TasksModel { Id = "Task 5", Title = "Task - 29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Type = "Bug", Priority
= "Low", CardTags = new List<string>() { "Customer" }, Estimate = 3.5,
Assignee = "Steven walker", AssigneeKey = "Steven walker", RankId = 1, Color
= "#cc0000", ClassName = new List<string>() { "e-bug", "e-low", "e-steven" }
},
new TasksModel { Id = "Task 6", Title = "Task - 29007", Status =
"Validate", Summary = "Validate new requirements", Type = "Improvement",
Priority = "Low", CardTags = new List<string>() { "Validation" }, Estimate =
1.5, Assignee = "Robert King", AssigneeKey = "Robert King", RankId = 1,
Color = "#7d7297", ClassName = new List<string>() { "e-improvement", "e-
low", "e-robert" } },
new TasksModel { Id = "Task 7", Title = "Task - 29009", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Type = "Bug",
Priority = "Release Breaker", CardTags = new List<string>() { "Fix",
"Safari" }, Estimate = 1.5, Assignee = "Nancy Davloio", AssigneeKey = "Nancy
Davloio", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
"e-bug", "e-release", "e-nancy" } },
new TasksModel { Id = "Task 8", Title = "Task - 29010", Status = "Close",
Summary = "Test the application in the IE browser.", Type = "Story",
Priority = "Low", CardTags = new List<string>() { "Review", "IE" }, Estimate
= 5.5, Assignee = "Margaret hamilt", AssigneeKey = "Margaret hamilt", RankId
= 3, Color = "#8b447a", ClassName = new List<string>() { "e-story", "e-low",
"e-Margaret" } },
new TasksModel { Id = "Task 9", Title = "Task - 29011", Status =
"Validate", Summary = "Validate the issues reported by the customer.", Type
= "Story", Priority = "High", CardTags = new List<string>() { "Validation",
"Fix" }, Estimate = 1, Assignee = "Steven walker", AssigneeKey = "Steven
walker", RankId = 1, Color = "#8b447a", ClassName = new List<string>() { "e-
story", "e-low", "e-nancy" } },
new TasksModel { Id = "Task 10", Title = "Task - 29015", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.", Type =
"Story", Priority = "High", CardTags = new List<string>() { "Database",
"SQL" }, Estimate = 5.5, Assignee = "Margaret hamilt", AssigneeKey =
"Margaret hamilt", RankId = 4, Color = "#8b447a", ClassName = new
List<string>() { "e-story", "e-high", "e-steven" } }
};
}

```

The output will be as follows.



### Template

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

To get start quickly with Blazor Kanban component using Templates, you can check on this video

{% youtube

"youtube:https://www.youtube.com/watch?v=PjTgXuibei8" %}

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id" ContentField="Summary">
    <Template>
      @ {
        TasksModel data = (TasksModel)context;
        <div class="e-card-content">
          <table class="card-template-wrap">
            <tbody>
              <tr>
                <td class="CardHeader">Id:</td>
                <td>@data.Id</td>
              </tr>
              <tr>
                <td class="CardHeader">Type:</td>
                <td>@data.Type</td>
              </tr>
            </tbody>
          </table>
        </div>
      }
    </Template>
  </KanbanCardSettings>
</SfKanban>
```



```

<tr>
<td class="CardHeader">Priority:</td>
<td>@data.Priority</td>
</tr>
<tr>
<td class="CardHeader">Summary:</td>
<td>@data.Summary</td>
</tr>
</tbody>
</table>
</div>
}
</Template>
</KanbanCardSettings>
</SfKanban>
<style type="text/css">
.e-kanban .card-template-wrap td {
background: none !important;
}
.e-kanban .card-template-wrap .CardHeader {
font-weight: 500;
}
</style>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Type { get; set; }
public string Priority { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio", Type = "Story", Priority = "Low" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller",
Type = "Improvement", Priority = "Normal" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling", Type = "Others", Priority =
"Critical" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling", Type = "Bug", Priority = "Release Breaker" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker", Type = "Bug", Priority = "Low" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio", Type = "Others", Priority = "Low" },

```

```
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret hamilt", Type = "Improvement", Priority = "Low" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee = "Steven walker", Type = "Story", Priority = "Release Breaker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.", Assignee = "Margaret hamilt", Type = "Bug", Priority = "Release Breaker" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status = "InProgress", Summary = "Fix cannot open user's default database SQL error.", Assignee = "Janet Leverling", Type = "Story", Priority = "Low" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review", Summary = "Fix the issues reported in data binding.", Assignee = "Janet Leverling", Type = "Story", Priority = "High" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close", Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller", Type = "Story", Priority = "Release Breaker" },
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate", Summary = "Validate databinding issues.", Assignee = "Margaret hamilt", Type = "Improvement", Priority = "High" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close", Summary = "Analyze grid control.", Assignee = "Margaret hamilt", Type = "Epic", Priority = "Critical" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close", Summary = "Stored procedure for initial data binding of the grid.", Assignee = "Steven walker", Type = "Story", Priority = "High" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close", Summary = "Analyze stored procedures.", Assignee = "Janet Leverling", Type = "Bug", Priority = "Critical" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate", Summary = "Validate editing issues.", Assignee = "Nancy Davloio", Type = "Story", Priority = "Normal" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review", Summary = "Test editing functionality.", Assignee = "Nancy Davloio", Type = "Story", Priority = "Release Breaker" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open", Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller", Type = "Story", Priority = "Low" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status = "InProgress", Summary = "Improve the performance of the editing functionality.", Assignee = "Nancy Davloio", Type = "Story", Priority = "high" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open", Summary = "Arrange web meeting with the customer to show editing demo.", Assignee = "Steven walker", Type = "Others", Priority = "Release Breaker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review", Summary = "Fix the editing issues reported by the customer.", Assignee = "Janet Leverling", Type = "Story", Priority = "Release Breaker" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing", Summary = "Fix the issues reported by the customer.", Assignee = "Steven walker", Type = "Story", Priority = "Critical" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing", Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy Davloio", Type = "Story", Priority = "Normal" },
```

```

new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling", Type = "Improvement", Priority = "Low" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio", Type =
"Story", Priority = "High" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling", Type = "Bug", Priority = "Normal" }
};
}

```

Output be like the below.

The Kanban board is organized into three columns:

- Backlog - 5 items:**
  - Task 1: Story, Low priority. Summary: Analyze the new requirements gathered from the customer.
  - Task 3: Others, Critical priority. Summary: Arrange a web meeting with the customer to get new requirements.
  - Task 9: Bug, Release Breaker priority. Summary: Show the retrieved data from the server in grid control.
  - Task 19: Story, Low priority. Summary: Enhance editing functionality.
  - Task 21: Others, Release Breaker priority. Summary: Arrange web meeting with the customer to show editing demo.
- In Progress - 4 items:**
  - Task 2: Improvement, Normal priority. Summary: Improve application performance.
  - Task 4: Bug, Release Breaker priority. Summary: Fix the issues reported in the IE browser.
  - Task 10: Story, Low priority. Summary: Fix cannot open user's default database SQL error.
  - Task 20: Story, high priority. Summary: Improve the performance of the editing functionality.
- Done - 5 items:**
  - Task 7: Improvement, Low priority. Summary: Test the application in the IE browser.
  - Task 12: Story, Release Breaker priority. Summary: Analyze SQL server 2008 connection.
  - Task 14: Epic, Critical priority. Summary: Analyze grid control.
  - Task 15: Story, High priority. Summary: Stored procedure for initial data binding of the grid.
  - Task 16: Bug, Critical priority. Summary: Analyze stored procedures.

## Selection

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

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

## Multiple Selection

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

## ASPX-CS

```

@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">

```

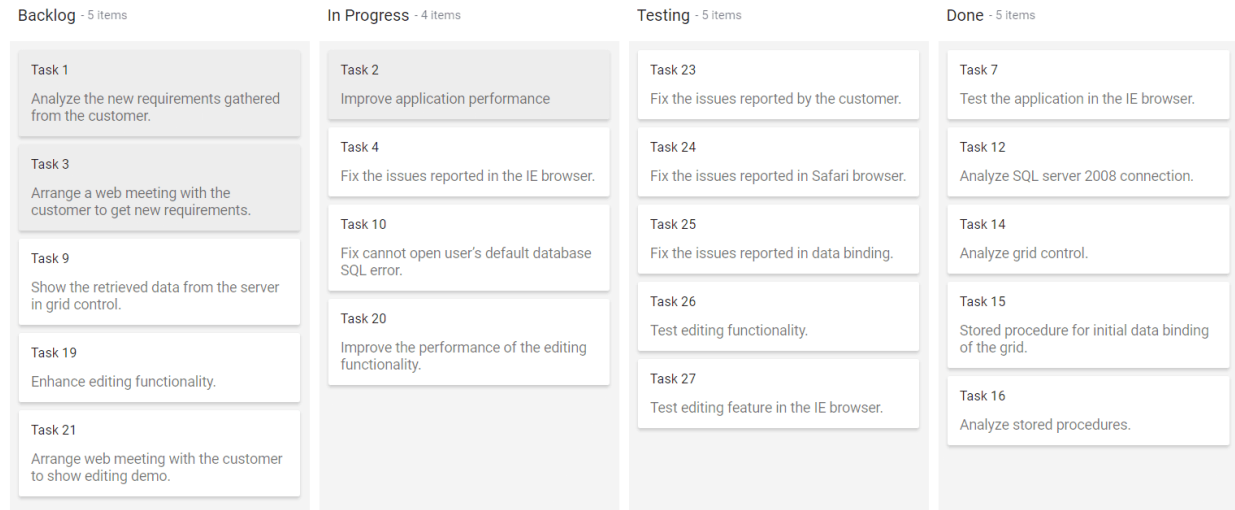
```

<KanbanColumns>
<KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
{"Open"}) "></KanbanColumn>
<KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{"InProgress"}) "></KanbanColumn>
<KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
{"Testing"}) "></KanbanColumn>
<KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
{"Close"}) "></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="Id" ContentField="Summary"
SelectionType="SelectionType.Multiple"></KanbanCardSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },

```

```
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}
```

Output be like the below.



## Swimlane in Blazor Kanban Component

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

### Render swimlane row

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

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>() { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>() { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>() { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>() { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
    ContentField="Summary"></KanbanCardSettings>
  <KanbanSwimlaneSettings KeyField="Assignee"></KanbanSwimlaneSettings>
</SfKanban>

@code {
  public class TasksModel
  {
    public string Id { get; set; }
    public string Title { get; set; }
    public string Status { get; set; }
    public string Summary { get; set; }
    public string Assignee { get; set; }
  }

  public List<TasksModel> Tasks = new List<TasksModel>()
  {
```

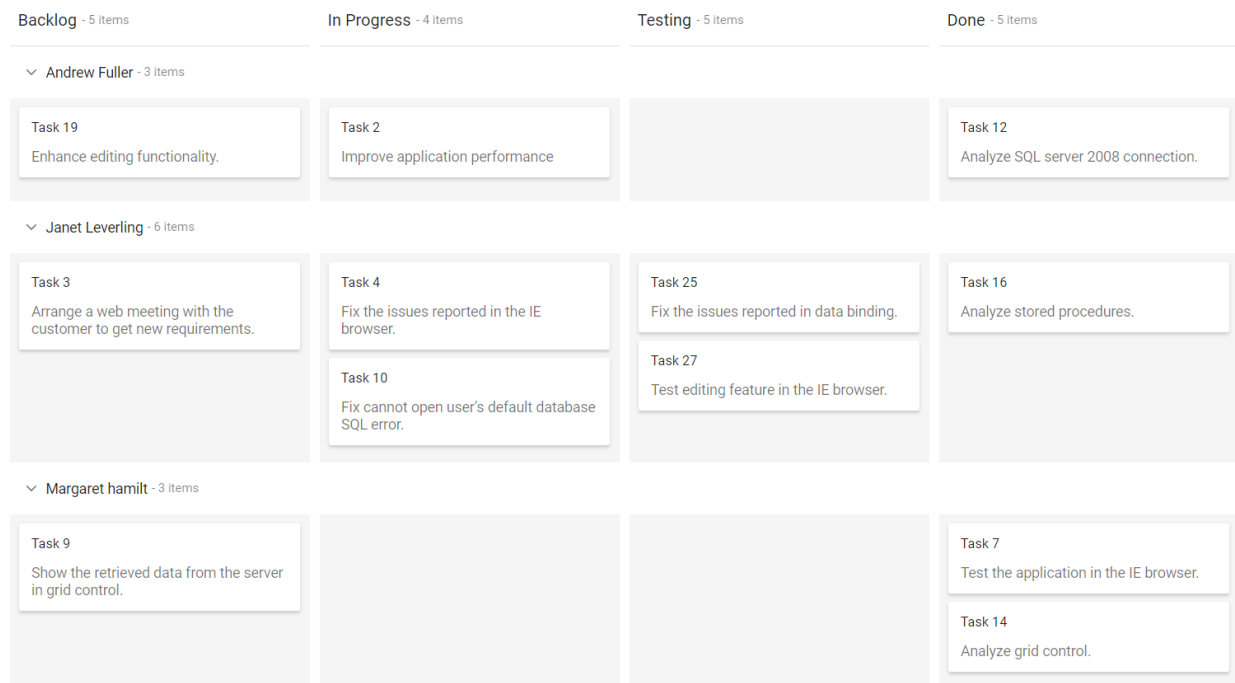
```
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
```

```

new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}

```

Output be like the below.



### Custom row text

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

It is not mandatory to define the `TextField` to `KanbanSwimlaneSettings`. It will automatically consider the `KeyField` to swimlane row header text. If the mapping `TextField` key is not present in the datasource, it will consider the swimlane `KeyField` as swimlane row header text.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
```



```

<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
  <KanbanSwimlaneSettings KeyField="Assignee"
  TextField="AssigneeName"></KanbanSwimlaneSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
public string AssigneeName { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller",
AssigneeName = "Andrew" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt", AssigneeName = "Margaret" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt", AssigneeName = "Margaret" },

```

```
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller",
AssigneeName = "Andrew" },
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt",
AssigneeName = "Margaret" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt",
AssigneeName = "Margaret" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling",
AssigneeName = "Janet" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio",
AssigneeName = "Nancy" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio",
AssigneeName = "Nancy" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller",
AssigneeName = "Andrew" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio",
AssigneeName = "Nancy" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling", AssigneeName = "Janet" }
};
}
```

## Template

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

To get start quickly with Blazor Kanban component using Templates, you can check on this video

{% youtube

"youtube:https://www.youtube.com/watch?v=PjTgXuibeI8" %}

## ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
  <KanbanSwimlaneSettings KeyField="Assignee" TextField="AssigneeName">
    <Template>
      @ {
        SwimlaneSettingsModel swimlane = (SwimlaneSettingsModel)context;
        <div class='swimlane-template e-swimlane-template-table'>
          
          <span>@swimlane.TextField</span>
        </div>
      }
    </Template>
  </KanbanSwimlaneSettings>
</SfKanban>
<style>
.swimlane-template {
display: inline-block;
font-size: 15px;
font-weight: 500;
}
.swimlane-template img {
height: 24px;
width: 24px;
border-radius: 50%;
}
.swimlane-template span {
padding-left: 5px;
vertical-align: middle;
```

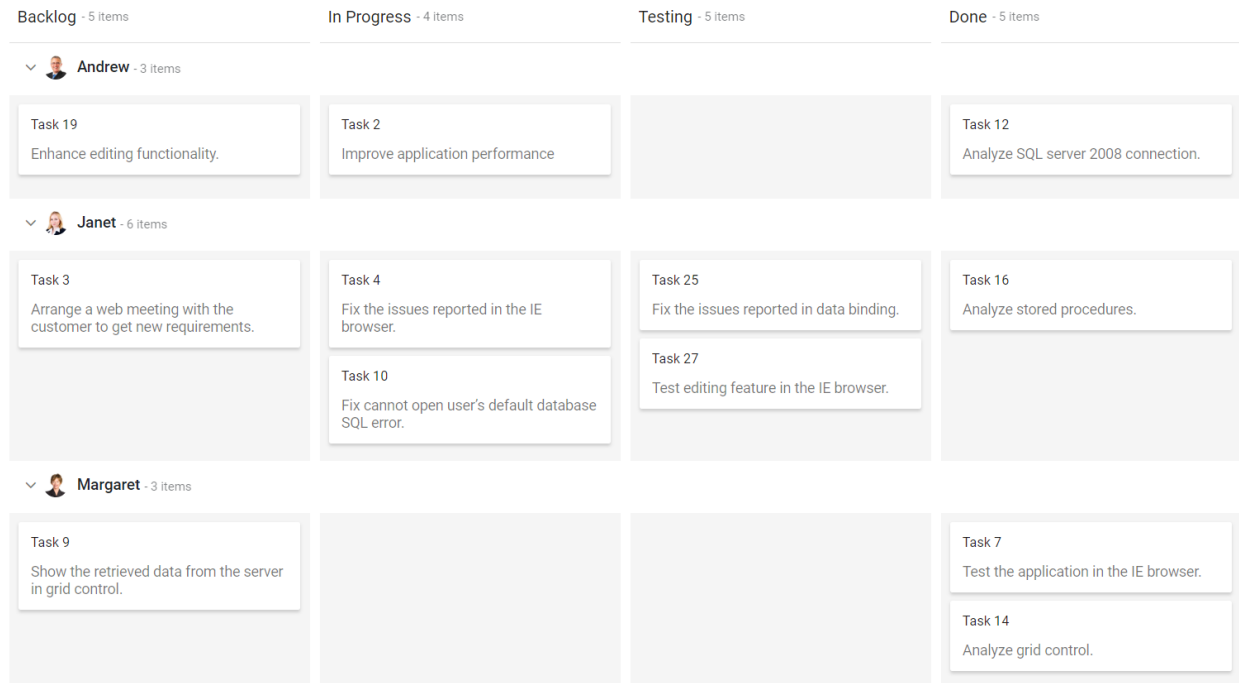
```

}
.e-kanban .e-kanban-content .e-content-row.e-swimlane-row .e-content-cells
.e-swimlane-header .e-item-count {
padding: 4px;
}
</style>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
public string AssigneeName { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller",
AssigneeName = "Andrew" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt", AssigneeName = "Margaret" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt", AssigneeName = "Margaret" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller",
AssigneeName = "Andrew" },
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt",
AssigneeName = "Margaret" },

```

```
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt",
AssigneeName = "Margaret" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling",
AssigneeName = "Janet" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio",
AssigneeName = "Nancy" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio",
AssigneeName = "Nancy" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller",
AssigneeName = "Andrew" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio",
AssigneeName = "Nancy" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling", AssigneeName = "Janet" }
};
}
```

Output be like the below.



### Sorting

Kanban support to sort the swimlane rows in kanban board based on the [TextField](#) property by setting [SortDirection](#) property.

If the [TextField](#) property is not provided, sorting will be performed based on the [KeyField](#) property.

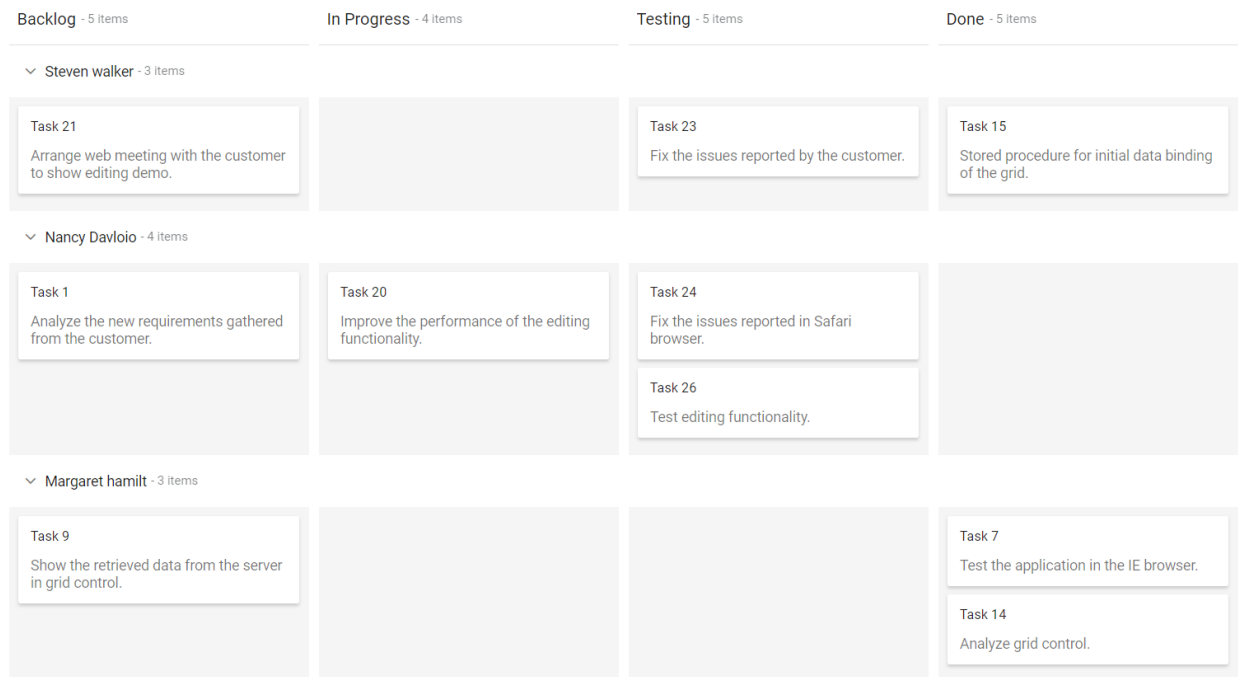
### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban KeyField="Status" DataSource="@Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="To Do" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
  <KanbanSwimlaneSettings KeyField="Assignee" TextField="AssigneeName"
  SortDirection="SortDirection.Descending"></KanbanSwimlaneSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
public string AssigneeName { get; set; }
}
```

```

public List<TasksModel> Tasks = new List<TasksModel>()
{
    new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
        Summary = "Analyze the new requirements gathered from the customer.",
        Assignee = "Nancy Davloio", AssigneeName = "Nancy" },
    new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
        Summary = "Improve application performance", Assignee = "Andrew Fuller",
        AssigneeName = "Andrew" },
    new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "InProgress",
        Summary = "Arrange a web meeting with the customer to get new
        requirements.", Assignee = "Nancy Davloio", AssigneeName = "Nancy" },
    new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "Close",
        Summary = "Fix the issues reported in the IE browser.", Assignee = "Nancy
        Davloio", AssigneeName = "Nancy" },
    new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Close",
        Summary = "Fix the issues reported by the customer.", Assignee = "Andrew
        Fuller", AssigneeName = "Andrew" }
};
}

```



### Custom order

Kanban supports to sort the swimlane rows using custom sort logic by handling [SwimlaneSorting](#) event.

In this event, you can get the argument of `SwimlaneRows` which contains the list of `SwimlaneSettingsModel` and it will align based on the `SortDirection` property. You can change the List of `SwimlaneSettingsModel` as per your wish and assign the changed list to it.

When you refresh the page, the [SwimlaneSorting](#) event will be triggered before Kanban elements append to the DOM element.

In the following code, changed the order of the swimlane rows at positions 2, 0, 1, 3 and assigned to the argument of [SwimlaneRows](#).

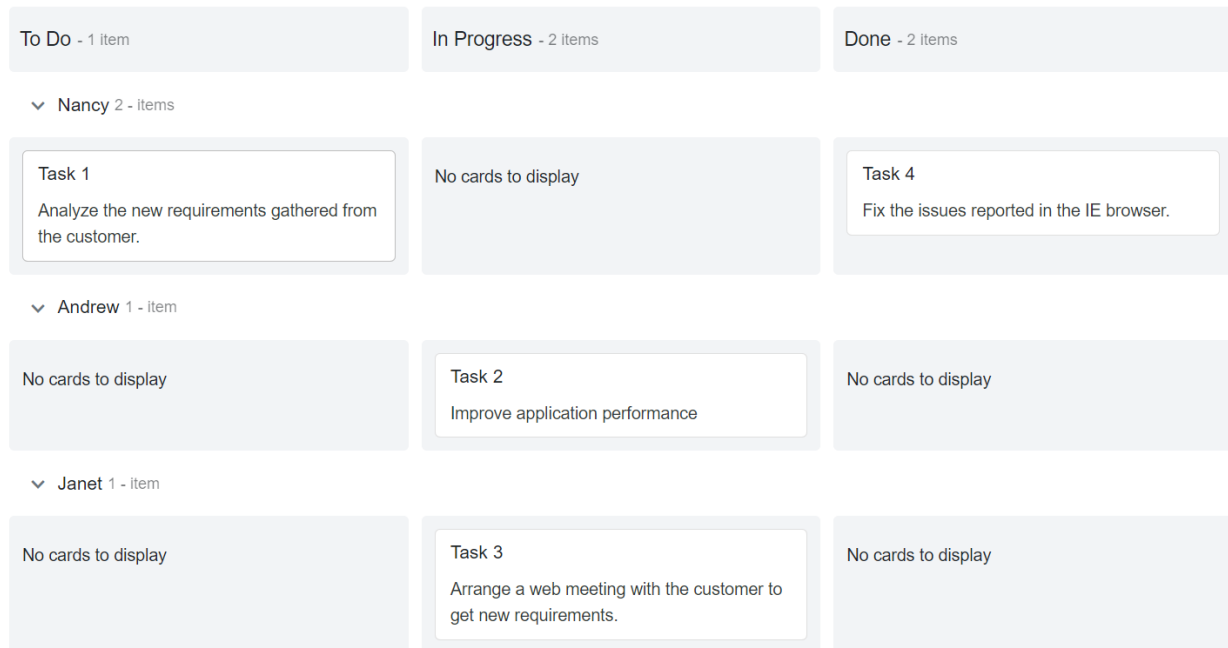
**ASPX-CS**

```

@using Syncfusion.Blazor.Kanban
<SfKanban KeyField="Status" DataSource="@Tasks">
<KanbanColumns>
<KanbanColumn HeaderText="To Do" KeyField="@ (new List<string>()
{"Open"}) "></KanbanColumn>
<KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{"InProgress"}) "></KanbanColumn>
<KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
{"Close"}) "></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
<KanbanSwimlaneSettings KeyField="Assignee"
TextField="AssigneeName"></KanbanSwimlaneSettings>
<KanbanEvents TValue="TasksModel"
SwimlaneSorting="@OnSorting"></KanbanEvents>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
public string AssigneeName { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller",
AssigneeName = "Andrew" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "InProgress",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "Close",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Nancy
Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Close",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker", AssigneeName = "Steven" }
};
public void OnSorting(SwimlaneSortEventArgs args)
{
if (args.SwimlaneRows.Count > 0)
{
var order = new List<int> { 2, 0, 1, 3 };
args.SwimlaneRows = order.Select(i => args.SwimlaneRows[i]).ToList();
}
}
}

```





### Drag-and-drop

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

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
  <KanbanSwimlaneSettings KeyField="Assignee"
  AllowDragAndDrop="true"></KanbanSwimlaneSettings>
</SfKanban>

@code {
  public class TasksModel
  {
    public string Id { get; set; }
    public string Title { get; set; }
    public string Status { get; set; }
    public string Summary { get; set; }
    public string Assignee { get; set; }
  }
  public List<TasksModel> Tasks = new List<TasksModel>()
```

```
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
}
```

```

new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}

```

### Calculate cards count

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

---

Provided localization support for **Items** text.

---

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

### ASPX-CS

```

@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
<KanbanColumns>
<KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
{"Open"})"></KanbanColumn>
<KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{"InProgress"})"></KanbanColumn>
<KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
{"Testing"})"></KanbanColumn>
<KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
{"Close"})"></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
<KanbanSwimlaneSettings KeyField="Assignee"
ShowItemCount="false"></KanbanSwimlaneSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
}

```

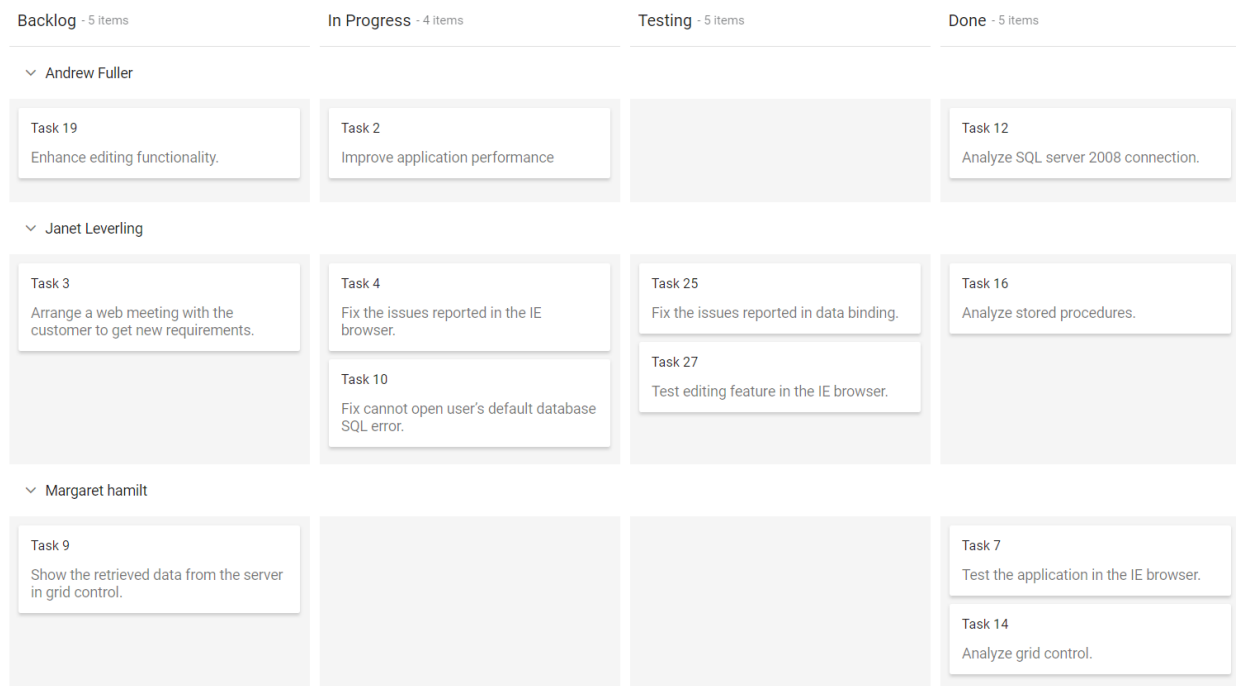
```
public List<TasksModel> Tasks = new List<TasksModel>()
{
    new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
        Summary = "Analyze the new requirements gathered from the customer.",
        Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
        Summary = "Improve application performance", Assignee = "Andrew Fuller" },
    new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
        Summary = "Arrange a web meeting with the customer to get new
        requirements.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
        Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
        Leverling" },
    new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
        Summary = "Fix the issues reported by the customer.", Assignee = "Steven
        walker" },
    new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
        Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
        Davloio" },
    new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
        Summary = "Test the application in the IE browser.", Assignee = "Margaret
        hamilt" },
    new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
        Summary = "Validate the issues reported by the customer.", Assignee =
        "Steven walker" },
    new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
        Summary = "Show the retrieved data from the server in grid control.",
        Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
        "InProgress", Summary = "Fix cannot open user's default database SQL
        error.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
        Summary = "Fix the issues reported in data binding.", Assignee = "Janet
        Leverling" },
    new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
        Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
    },
    new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
        Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
        Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
        Summary = "Stored procedure for initial data binding of the grid.", Assignee
        = "Steven walker" },
    new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
        Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
        Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
        Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
        Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
    new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
        "InProgress", Summary = "Improve the performance of the editing
        functionality.", Assignee = "Nancy Davloio" },
}
```

```

new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}

```

Output be like the below.



### Enable frozen rows

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

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

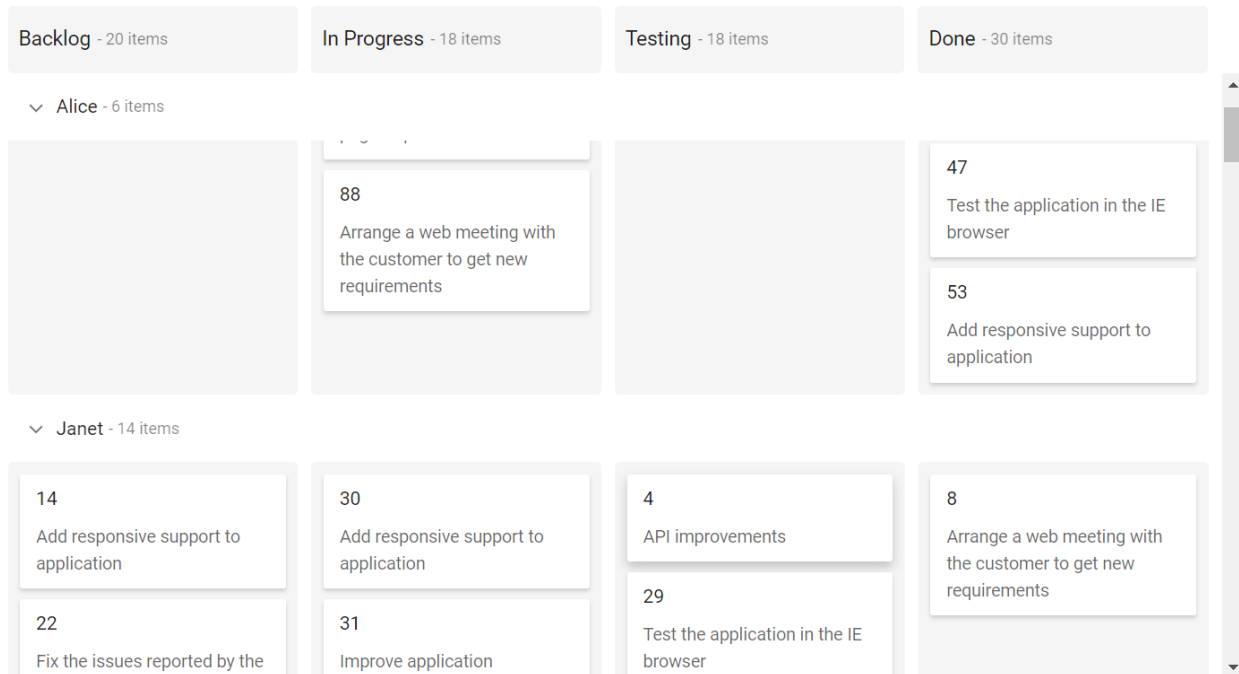
This feature support only when using Kanban content scrolling. The Expand/collapse swimlane icon does not work on frozen rows.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks"
Height="500px">
<KanbanColumns>
<KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
{"Open"}) "></KanbanColumn>
<KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{"InProgress"}) "></KanbanColumn>
<KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
{"Testing"}) "></KanbanColumn>
<KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
{"Close"}) "></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
<KanbanSwimlaneSettings KeyField="Assignee"
EnableFrozenRows="true"></KanbanSwimlaneSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
}
```

```
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}
```

Output be like the below.



## Workflow in Blazor Kanban Component

Kanban workflow allows to set the flow of cards between the columns. It provides restriction on columns when the card is moved from one column to another column. It provides support to prevent the drag and drop action on the column.

### Prevent transition across columns

Provides restriction on columns when performing drag and drop actions while providing the **KeyField** inside the **TransitionColumns** property. If a card is dragged, dotted border line will be shown on the possible drop columns and the not-allowed cursor point will be shown in the restricted dropped columns.

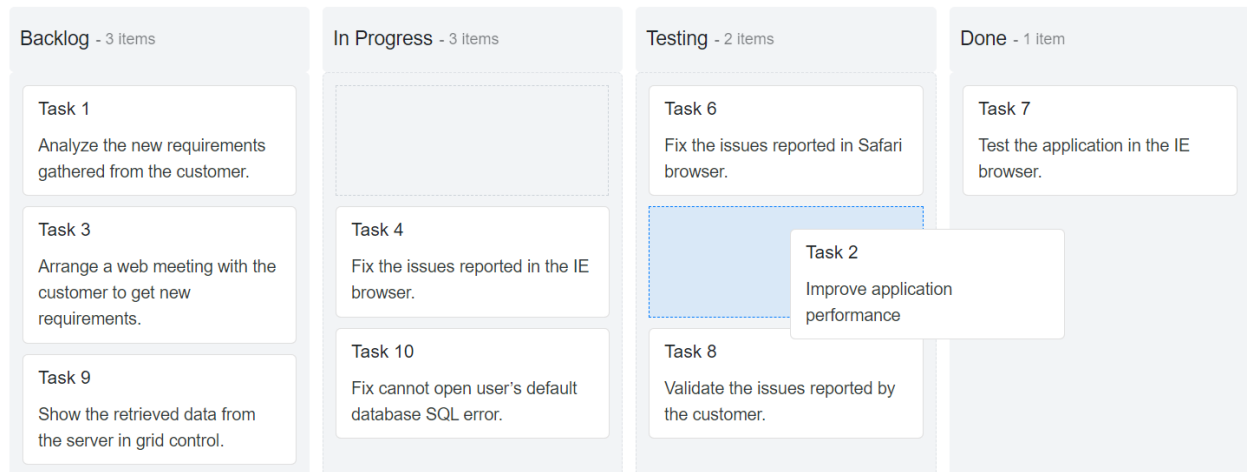
### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>() { "Open" }) "
      TransitionColumns="@ (new List<string>() { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
      { "InProgress" }) " TransitionColumns="@ (new List<string>()
      { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
      { "Testing" }) " TransitionColumns="@ (new List<string>()
      { "Close" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
      { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
    ContentField="Summary"></KanbanCardSettings>
</SfKanban>
@code {
  public class TasksModel
```



```
{
    public string Id { get; set; }
    public string Title { get; set; }
    public string Status { get; set; }
    public string Summary { get; set; }
    public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
    new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
        Summary = "Analyze the new requirements gathered from the customer.",
        Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
        Summary = "Improve application performance", Assignee = "Andrew Fuller" },
    new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
        Summary = "Arrange a web meeting with the customer to get new
        requirements.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
        Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
        Leverling" },
    new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
        Summary = "Fix the issues reported by the customer.", Assignee = "Steven
        walker" },
    new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Testing",
        Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
        Davloio" },
    new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
        Summary = "Test the application in the IE browser.", Assignee = "Margaret
        hamilt" },
    new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Testing",
        Summary = "Validate the issues reported by the customer.", Assignee =
        "Steven walker" },
    new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
        Summary = "Show the retrieved data from the server in grid control.",
        Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
        "InProgress", Summary = "Fix cannot open user's default database SQL
        error.", Assignee = "Janet Leverling" }
};
}
```

The output will be as follows.



### Prevent Drop actions

Column will not allow any dropped card action when you disable the **AllowDrop** properties in the column.

In the following code, the **Backlog** column will not allow any card drop action within the column and any other columns cards.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>() {"Open"})"
      AllowDrop="false"></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>() {"InProgress"})"
      TransitionColumns="@ (new List<string>() {"Testing"})"></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>() {"Testing"})"
      TransitionColumns="@ (new List<string>() {"Close"})"></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>() {"Close"})"></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id" ContentField="Summary"></KanbanCardSettings>
</SfKanban>

@code {
  public class TasksModel
  {
    public string Id { get; set; }
    public string Title { get; set; }
    public string Status { get; set; }
    public string Summary { get; set; }
    public string Assignee { get; set; }
  }

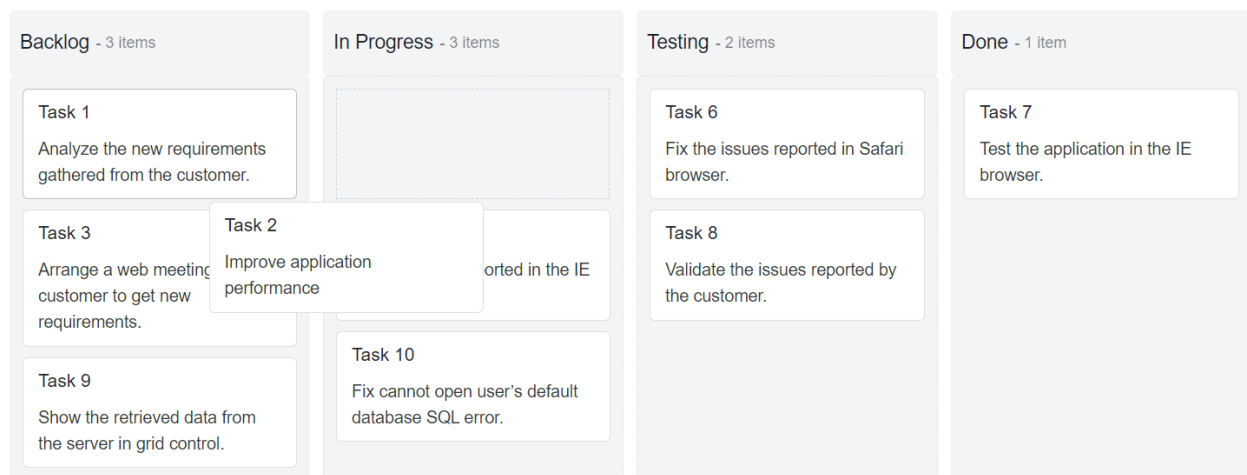
  public List<TasksModel> Tasks = new List<TasksModel>()
  {
    new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
      Summary = "Analyze the new requirements gathered from the customer.",
      Assignee = "Nancy Davloio" },
  }
```

```

new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Testing",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" }
};
}

```

The output will be as follows.



### Prevent Drag actions

Column will not allow any dragged card action when you disable the `AllowDrag` properties in the column.

In the following code, the `Done` column will not allow any card drag action within the column.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
```

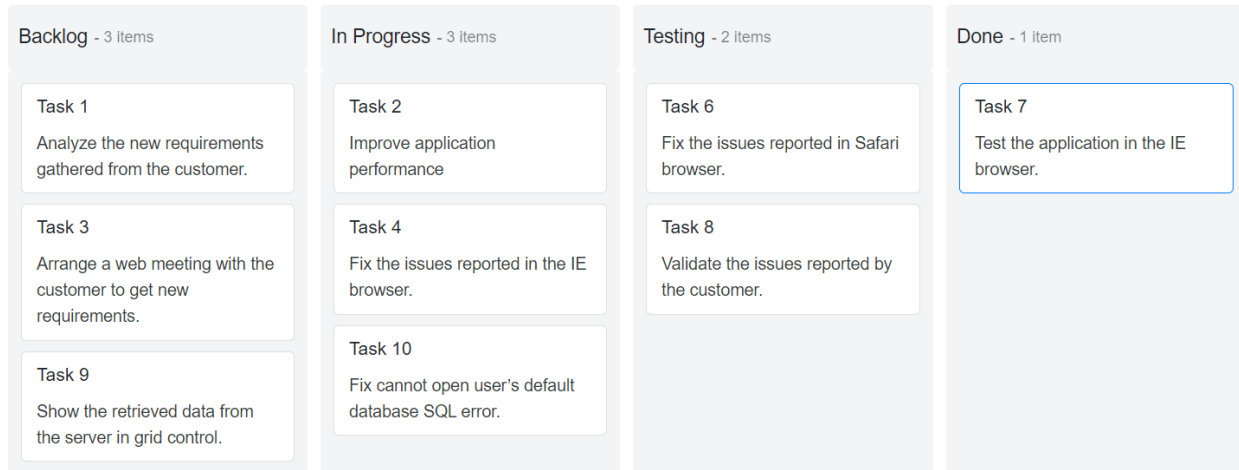
```

<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>() { "Open" })"
      AllowDrop="false"></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
      { "InProgress" })" TransitionColumns="@ (new List<string>()
      { "Testing" })"></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
      { "Testing" })" TransitionColumns="@ (new List<string>()
      { "Close" })"></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>() { "Close" })"
      AllowDrag="false"></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
    ContentField="Summary"></KanbanCardSettings>
  <KanbanSwimlaneSettings KeyField="Assignee"></KanbanSwimlaneSettings>
</SfKanban>
@code {
  public class TasksModel
  {
    public string Id { get; set; }
    public string Title { get; set; }
    public string Status { get; set; }
    public string Summary { get; set; }
    public string Assignee { get; set; }
    public string AssigneeName { get; set; }
  }
  public List<TasksModel> Tasks = new List<TasksModel>()
  {
    new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
      Summary = "Analyze the new requirements gathered from the customer.",
      Assignee = "Nancy Davloio", AssigneeName = "Nancy" },
    new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
      Summary = "Improve application performance", Assignee = "Andrew Fuller",
      AssigneeName = "Andrew" },
    new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
      Summary = "Arrange a web meeting with the customer to get new
      requirements.", Assignee = "Janet Leverling", AssigneeName = "Janet" },
    new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
      Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
      Leverling", AssigneeName = "Janet" },
    new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
      Summary = "Fix the issues reported by the customer.", Assignee = "Steven
      walker", AssigneeName = "Steven" },
    new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
      Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
      Davloio", AssigneeName = "Nancy" },
    new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
      Summary = "Test the application in the IE browser.", Assignee = "Margaret
      hamilt", AssigneeName = "Margaret" },
    new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
      Summary = "Validate the issues reported by the customer.", Assignee =
      "Steven walker", AssigneeName = "Steven" },
    new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
      Summary = "Show the retrieved data from the server in grid control.",
      Assignee = "Margaret hamilt", AssigneeName = "Margaret" },
  }
}

```

```
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller",
AssigneeName = "Andrew" },
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt",
AssigneeName = "Margaret" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt",
AssigneeName = "Margaret" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling",
AssigneeName = "Janet" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio",
AssigneeName = "Nancy" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio",
AssigneeName = "Nancy" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller",
AssigneeName = "Andrew" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker", AssigneeName = "Steven" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling", AssigneeName = "Janet" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio",
AssigneeName = "Nancy" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling", AssigneeName = "Janet" }
};
}
```

The output will be as follows.



## Sorting in Blazor Kanban Component

The Kanban provides built-in support to arrange the cards in their columns based on the JSON data order and drop the cards in the columns based on the dropped clone.

### SortBy

Initially, users can change the arrangement of cards in the columns and position of the dropped card by using the `SortBy` property. The `SortBy` property contains three enumeration values as follows,

- `DataSourceOrder`
- `Index`
- `Custom`

### DataSource Order

The `SortBy DataSourceOrder` property does not require any `Field` mapping. In this behavior, cards are loaded based on the JSON data order, and also cards are dropped based on the JSON data order.

By default, the `SortBy` property is `DataSourceOrder`.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban;
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
</SfKanban>
```

```

@code {
public class TasksModel
{
public string Id { get; set; }
public int ListId { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Type { get; set; }
public string Priority { get; set; }
public List<string> CardTags { get; set; }
public string Tags { get; set; }
public double Estimate { get; set; }
public string Assignee { get; set; }
public int RankId { get; set; }
public string Color { get; set; }
public string Value { get; set; }
public string OrderID { get; set; }
public string Size { get; set; }
public string ImageURL { get; set; }
public string Description { get; set; }
public string Category { get; set; }
public string Price { get; set; }
public string AssigneeKey { get; set; }
public List<string> ClassName { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "Task - 29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.", Type =
"Story", Priority = "Low", CardTags = new List<string>() { "Analyze",
"Customer" }, Estimate = 3.5, Assignee = "Nancy Davloio", AssigneeKey =
"Nancy Davloio", RankId = 1, Color = "#8b447a", ClassName = new
List<string>() { "e-story", "e-low", "e-nancy" } },
new TasksModel { Id = "Task 2", Title = "Task - 29002", Status =
"InProgress", Summary = "Improve application performance", Type =
"Improvement", Priority = "Normal", CardTags = new List<string>() {
"Improvement" }, Estimate = 6, Assignee = "Andrew Fuller", AssigneeKey =
"Andrew Fuller", RankId = 1, Color = "#7d7297", ClassName = new
List<string>() { "e-improvement", "e-normal", "e-andrew" } },
new TasksModel { Id = "Task 3", Title = "Task - 29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Type = "Others", Priority = "Critical", CardTags = new
List<string>() { "Meeting" }, Estimate = 5.5, Assignee = "Janet Leverling",
AssigneeKey = "Janet Leverling", RankId = 2, Color = "#27AE60", ClassName =
new List<string>() { "e-others", "e-critical", "e-janet" } },
new TasksModel { Id = "Task 4", Title = "Task - 29004", Status =
"InProgress", Summary = "Fix the issues reported in the IE browser.", Type =
"Bug", Priority = "Release Breaker", CardTags = new List<string>() { "IE" },
Estimate = 2.5, Assignee = "Janet Leverling", AssigneeKey = "Janet
Leverling", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
"e-bug", "e-release", "e-janet" } },
new TasksModel { Id = "Task 5", Title = "Task - 29005", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Type = "Bug", Priority
= "Low", CardTags = new List<string>() { "Customer" }, Estimate = 3.5,
Assignee = "Steven walker", AssigneeKey = "Steven walker", RankId = 1, Color

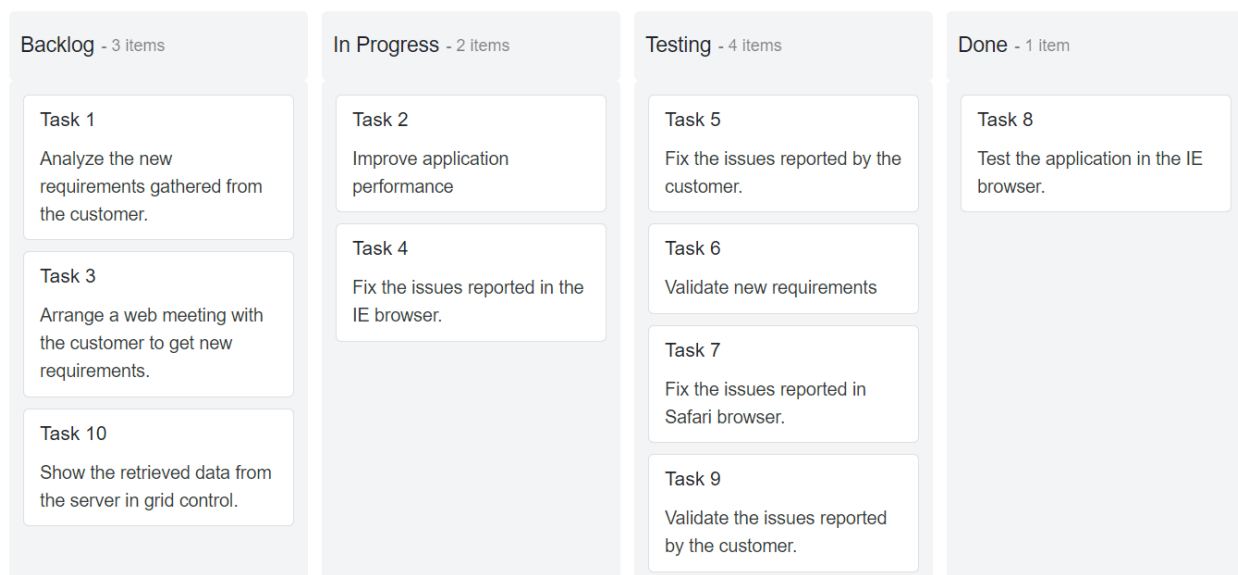
```

```

= "#cc0000", ClassName = new List<string>() { "e-bug", "e-low", "e-steven" }
},
new TasksModel { Id = "Task 6", Title = "Task - 29007", Status = "Testing",
Summary = "Validate new requirements", Type = "Improvement", Priority =
"Low", CardTags = new List<string>() { "Validation" }, Estimate = 1.5,
Assignee = "Robert King", AssigneeKey = "Robert King", RankId = 1, Color =
"#7d7297", ClassName = new List<string>() { "e-improvement", "e-low", "e-
robert" } },
new TasksModel { Id = "Task 7", Title = "Task - 29009", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Type = "Bug",
Priority = "Release Breaker", CardTags = new List<string>() { "Fix",
"Safari" }, Estimate = 1.5, Assignee = "Nancy Davloio", AssigneeKey = "Nancy
Davloio", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
"e-bug", "e-release", "e-nancy" } },
new TasksModel { Id = "Task 8", Title = "Task - 29010", Status = "Close",
Summary = "Test the application in the IE browser.", Type = "Story",
Priority = "Low", CardTags = new List<string>() { "Review", "IE" }, Estimate
= 5.5, Assignee = "Margaret hamilt", AssigneeKey = "Margaret hamilt", RankId
= 3, Color = "#8b447a", ClassName = new List<string>() { "e-story", "e-low",
"e-Margaret" } },
new TasksModel { Id = "Task 9", Title = "Task - 29011", Status = "Testing",
Summary = "Validate the issues reported by the customer.", Type = "Story",
Priority = "High", CardTags = new List<string>() { "Validation", "Fix" },
Estimate = 1, Assignee = "Steven walker", AssigneeKey = "Steven walker",
RankId = 1, Color = "#8b447a", ClassName = new List<string>() { "e-story",
"e-low", "e-nancy" } },
new TasksModel { Id = "Task 10", Title = "Task - 29015", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.", Type =
"Story", Priority = "High", CardTags = new List<string>() { "Database",
"SQL" }, Estimate = 5.5, Assignee = "Margaret hamilt", AssigneeKey =
"Margaret hamilt", RankId = 4, Color = "#8b447a", ClassName = new
List<string>() { "e-story", "e-high", "e-steven" } }
};
}

```

The output will be as follows.





### Index

SortBy **Index** property must require datasource **Field** mapping. In this behavior, cards are loaded based on mapping **Field** values, and cards are dropped based on the dropped clone.

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

---

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

---

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

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

For Example,

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

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

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban;
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
```

```

<KanbanCardSettings ContentField="Summary" HeaderField="Title">
  <Template>
    @{
      TasksModel card = (TasksModel)context;
      <div class="card-template @card.Priority">
        <div class="e-card-header">
          <div class="e-card-header-caption">
            <div class="e-card-header-title e-tooltip-text">@card.Title</div>
          </div>
        </div>
        <div class="e-card-content">
          <div class="e-text e-tooltip-text">@card.Summary</div>
        </div>
        <div class="e-card-footer" style="font-size: 12px">
          <div class="e-text">RankId: @card.RankId</div>
        </div>
      </div>
    }
  </Template>
</KanbanCardSettings>
<KanbanSortSettings SortBy="SortOrderBy.Index"
  Field="RankId"></KanbanSortSettings>
</SfKanban>
@code {
  public class TasksModel
  {
    public string Id { get; set; }
    public int ListId { get; set; }
    public string Title { get; set; }
    public string Status { get; set; }
    public string Summary { get; set; }
    public string Type { get; set; }
    public string Priority { get; set; }
    public List<string> CardTags { get; set; }
    public string Tags { get; set; }
    public double Estimate { get; set; }
    public string Assignee { get; set; }
    public int RankId { get; set; }
    public string Color { get; set; }
    public string Value { get; set; }
    public string OrderID { get; set; }
    public string Size { get; set; }
    public string ImageURL { get; set; }
    public string Description { get; set; }
    public string Category { get; set; }
    public string Price { get; set; }
    public string AssigneeKey { get; set; }
    public List<string> ClassName { get; set; }
  }
  public List<TasksModel> Tasks = new List<TasksModel>()
  {
    new TasksModel { Id = "Task 1", Title = "Task - 29001", Status = "Open",
      Summary = "Analyze the new requirements gathered from the customer.", Type =
      "Story", Priority = "Low", CardTags = new List<string>() { "Analyze",
      "Customer" }, Estimate = 3.5, Assignee = "Nancy Davloio", AssigneeKey =
      "Nancy Davloio", RankId = 1, Color = "#8b447a", ClassName = new
      List<string>() { "e-story", "e-low", "e-nancy" } },
  }
}

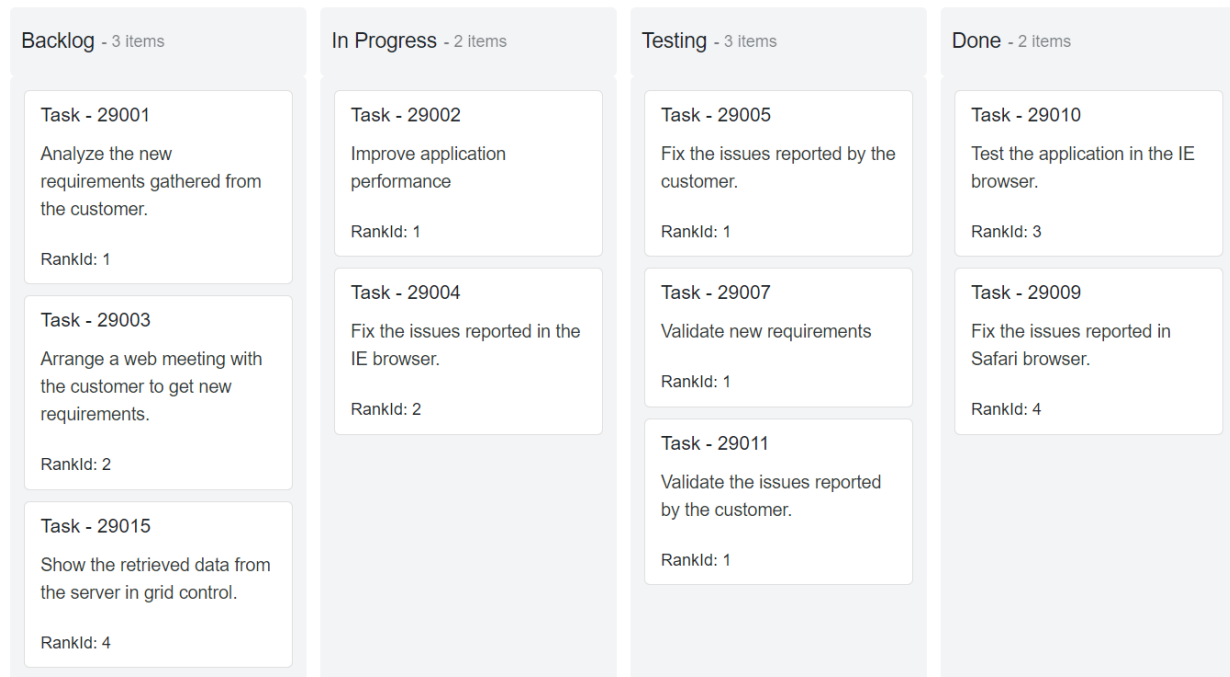
```

```

new TasksModel { Id = "Task 2", Title = "Task - 29002", Status =
"InProgress", Summary = "Improve application performance", Type =
"Improvement", Priority = "Normal", CardTags = new List<string>() {
"Improvement" }, Estimate = 6, Assignee = "Andrew Fuller", AssigneeKey =
"Andrew Fuller", RankId = 1, Color = "#7d7297", ClassName = new
List<string>() { "e-improvement", "e-normal", "e-andrew" } },
new TasksModel { Id = "Task 3", Title = "Task - 29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Type = "Others", Priority = "Critical", CardTags = new
List<string>() { "Meeting" }, Estimate = 5.5, Assignee = "Janet Leverling",
AssigneeKey = "Janet Leverling", RankId = 2, Color = "#27AE60", ClassName =
new List<string>() { "e-others", "e-critical", "e-janet" } },
new TasksModel { Id = "Task 4", Title = "Task - 29004", Status =
"InProgress", Summary = "Fix the issues reported in the IE browser.", Type =
"Bug", Priority = "Release Breaker", CardTags = new List<string>() { "IE" },
Estimate = 2.5, Assignee = "Janet Leverling", AssigneeKey = "Janet
Leverling", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
"e-bug", "e-release", "e-janet" } },
new TasksModel { Id = "Task 5", Title = "Task - 29005", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Type = "Bug", Priority
= "Low", CardTags = new List<string>() { "Customer" }, Estimate = 3.5,
Assignee = "Steven walker", AssigneeKey = "Steven walker", RankId = 1, Color
= "#cc0000", ClassName = new List<string>() { "e-bug", "e-low", "e-steven" }
},
new TasksModel { Id = "Task 6", Title = "Task - 29007", Status = "Testing",
Summary = "Validate new requirements", Type = "Improvement", Priority =
"Low", CardTags = new List<string>() { "Validation" }, Estimate = 1.5,
Assignee = "Robert King", AssigneeKey = "Robert King", RankId = 1, Color =
"#7d7297", ClassName = new List<string>() { "e-improvement", "e-low", "e-
robert" } },
new TasksModel { Id = "Task 7", Title = "Task - 29009", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Type = "Bug",
Priority = "Release Breaker", CardTags = new List<string>() { "Fix",
"Safari" }, Estimate = 1.5, Assignee = "Nancy Davloio", AssigneeKey = "Nancy
Davloio", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
"e-bug", "e-release", "e-nancy" } },
new TasksModel { Id = "Task 8", Title = "Task - 29010", Status = "Close",
Summary = "Test the application in the IE browser.", Type = "Story",
Priority = "Low", CardTags = new List<string>() { "Review", "IE" }, Estimate
= 5.5, Assignee = "Margaret hamilt", AssigneeKey = "Margaret hamilt", RankId
= 3, Color = "#8b447a", ClassName = new List<string>() { "e-story", "e-low",
"e-Margaret" } },
new TasksModel { Id = "Task 9", Title = "Task - 29011", Status = "Testing",
Summary = "Validate the issues reported by the customer.", Type = "Story",
Priority = "High", CardTags = new List<string>() { "Validation", "Fix" },
Estimate = 1, Assignee = "Steven walker", AssigneeKey = "Steven walker",
RankId = 1, Color = "#8b447a", ClassName = new List<string>() { "e-story",
"e-low", "e-nancy" } },
new TasksModel { Id = "Task 10", Title = "Task - 29015", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.", Type =
"Story", Priority = "High", CardTags = new List<string>() { "Database",
"SQL" }, Estimate = 5.5, Assignee = "Margaret hamilt", AssigneeKey =
"Margaret hamilt", RankId = 4, Color = "#8b447a", ClassName = new
List<string>() { "e-story", "e-high", "e-steven" } }
};
}

```

The output will be as follows.



### Custom

The `SortBy Custom` property must require datasource `Field` mapping. In this behavior, cards are loaded based on the `Field` mapping value and also cards are dropped based on the `Field` mapping value.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban;
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings ContentField="Summary"
  HeaderField="Title"></KanbanCardSettings>
  <KanbanSortSettings SortBy="SortOrderBy.Custom"
  Field="Summary"></KanbanSortSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public int ListId { get; set; }
public string Title { get; set; }
public string Status { get; set; }
```

```

public string Summary { get; set; }
public string Type { get; set; }
public string Priority { get; set; }
public List<string> CardTags { get; set; }
public string Tags { get; set; }
public double Estimate { get; set; }
public string Assignee { get; set; }
public int RankId { get; set; }
public string Color { get; set; }
public string Value { get; set; }
public string OrderID { get; set; }
public string Size { get; set; }
public string ImageURL { get; set; }
public string Description { get; set; }
public string Category { get; set; }
public string Price { get; set; }
public string AssigneeKey { get; set; }
public List<string> ClassName { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
    new TasksModel { Id = "Task 1", Title = "Task - 29001", Status = "Open",
        Summary = "Analyze the new requirements gathered from the customer.", Type =
        "Story", Priority = "Low", CardTags = new List<string>() { "Analyze",
        "Customer" }, Estimate = 3.5, Assignee = "Nancy Davloio", AssigneeKey =
        "Nancy Davloio", RankId = 1, Color = "#8b447a", ClassName = new
        List<string>() { "e-story", "e-low", "e-nancy" } },
    new TasksModel { Id = "Task 2", Title = "Task - 29002", Status =
        "InProgress", Summary = "Improve application performance", Type =
        "Improvement", Priority = "Normal", CardTags = new List<string>() {
        "Improvement" }, Estimate = 6, Assignee = "Andrew Fuller", AssigneeKey =
        "Andrew Fuller", RankId = 1, Color = "#7d7297", ClassName = new
        List<string>() { "e-improvement", "e-normal", "e-andrew" } },
    new TasksModel { Id = "Task 3", Title = "Task - 29003", Status = "Open",
        Summary = "Arrange a web meeting with the customer to get new
        requirements.", Type = "Others", Priority = "Critical", CardTags = new
        List<string>() { "Meeting" }, Estimate = 5.5, Assignee = "Janet Leverling",
        AssigneeKey = "Janet Leverling", RankId = 2, Color = "#27AE60", ClassName =
        new List<string>() { "e-others", "e-critical", "e-janet" } },
    new TasksModel { Id = "Task 4", Title = "Task - 29004", Status =
        "InProgress", Summary = "Fix the issues reported in the IE browser.", Type =
        "Bug", Priority = "Release Breaker", CardTags = new List<string>() { "IE" },
        Estimate = 2.5, Assignee = "Janet Leverling", AssigneeKey = "Janet
        Leverling", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
        "e-bug", "e-release", "e-janet" } },
    new TasksModel { Id = "Task 5", Title = "Task - 29005", Status = "Testing",
        Summary = "Fix the issues reported by the customer.", Type = "Bug", Priority
        = "Low", CardTags = new List<string>() { "Customer" }, Estimate = 3.5,
        Assignee = "Steven walker", AssigneeKey = "Steven walker", RankId = 1, Color
        = "#cc0000", ClassName = new List<string>() { "e-bug", "e-low", "e-steven" }
    },
    new TasksModel { Id = "Task 6", Title = "Task - 29007", Status = "Testing",
        Summary = "Validate new requirements", Type = "Improvement", Priority =
        "Low", CardTags = new List<string>() { "Validation" }, Estimate = 1.5,
        Assignee = "Robert King", AssigneeKey = "Robert King", RankId = 1, Color =
        "#7d7297", ClassName = new List<string>() { "e-improvement", "e-low", "e-
        robert" } },

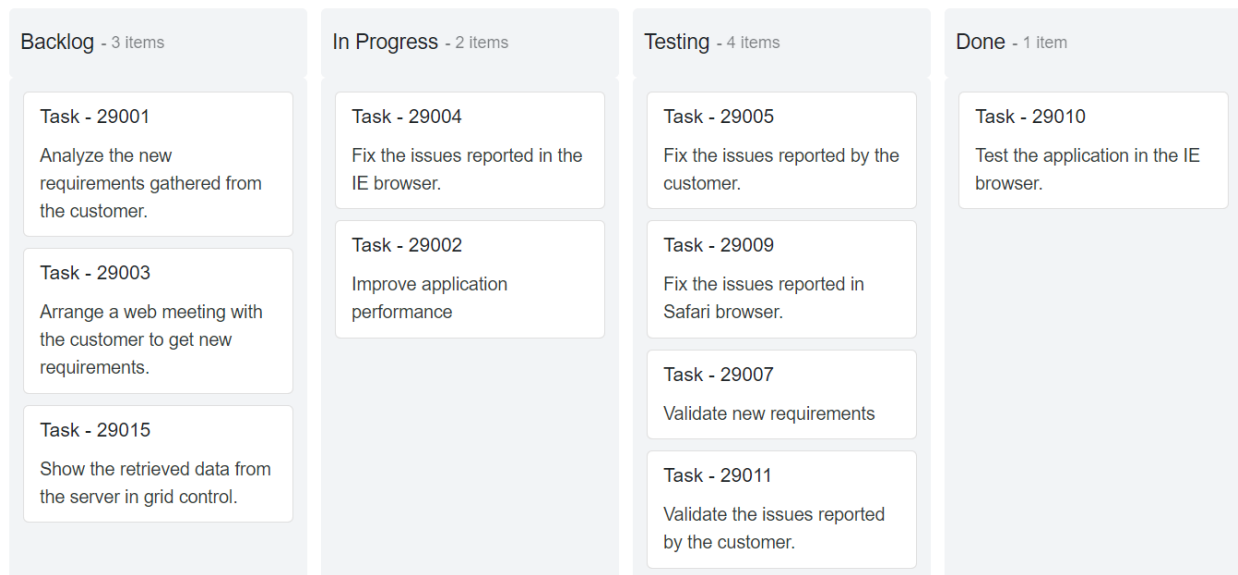
```

```

new TasksModel { Id = "Task 7", Title = "Task - 29009", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Type = "Bug",
Priority = "Release Breaker", CardTags = new List<string>() { "Fix",
"Safari" }, Estimate = 1.5, Assignee = "Nancy Davloio", AssigneeKey = "Nancy
Davloio", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
"e-bug", "e-release", "e-nancy" } },
new TasksModel { Id = "Task 8", Title = "Task - 29010", Status = "Close",
Summary = "Test the application in the IE browser.", Type = "Story",
Priority = "Low", CardTags = new List<string>() { "Review", "IE" }, Estimate
= 5.5, Assignee = "Margaret hamilt", AssigneeKey = "Margaret hamilt", RankId
= 3, Color = "#8b447a", ClassName = new List<string>() { "e-story", "e-low",
"e-Margaret" } },
new TasksModel { Id = "Task 9", Title = "Task - 29011", Status = "Testing",
Summary = "Validate the issues reported by the customer.", Type = "Story",
Priority = "High", CardTags = new List<string>() { "Validation", "Fix" },
Estimate = 1, Assignee = "Steven walker", AssigneeKey = "Steven walker",
RankId = 1, Color = "#8b447a", ClassName = new List<string>() { "e-story",
"e-low", "e-nancy" } },
new TasksModel { Id = "Task 10", Title = "Task - 29015", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.", Type =
"Story", Priority = "High", CardTags = new List<string>() { "Database",
"SQL" }, Estimate = 5.5, Assignee = "Margaret hamilt", AssigneeKey =
"Margaret hamilt", RankId = 4, Color = "#8b447a", ClassName = new
List<string>() { "e-story", "e-high", "e-steven" } }
};
}

```

The output will be as follows.



### Change the direction

Kanban board also provides support for aligning the cards in the columns using the **Direction** property inside the **KanbanSortSettings** property. Based on this, cards can be aligned in the columns either in **Ascending** or **Descending** order. Sorting direction will be performed based on **SortBy** property.

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

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

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban;
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings ContentField="Summary"
  HeaderField="Title"></KanbanCardSettings>
  <KanbanSortSettings SortBy="SortOrderBy.Custom"
  Field="Summary"></KanbanSortSettings>
</SfKanban>

@code {
public class TasksModel
{
public string Id { get; set; }
public int ListId { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Type { get; set; }
public string Priority { get; set; }
public List<string> CardTags { get; set; }
public string Tags { get; set; }
public double Estimate { get; set; }
public string Assignee { get; set; }
public int RankId { get; set; }
public string Color { get; set; }
public string Value { get; set; }
public string OrderID { get; set; }
public string Size { get; set; }
public string ImageURL { get; set; }
public string Description { get; set; }
public string Category { get; set; }
public string Price { get; set; }
public string AssigneeKey { get; set; }
public List<string> ClassName { get; set; }
}

public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "Task - 29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.", Type =
"Story", Priority = "Low", CardTags = new List<string>() { "Analyze",
"Customer" }, Estimate = 3.5, Assignee = "Nancy Davloio", AssigneeKey =
"Nancy Davloio", RankId = 1, Color = "#8b447a", ClassName = new
List<string>() { "e-story", "e-low", "e-nancy" } },
new TasksModel { Id = "Task 2", Title = "Task - 29002", Status =
"InProgress", Summary = "Improve application performance", Type =
```

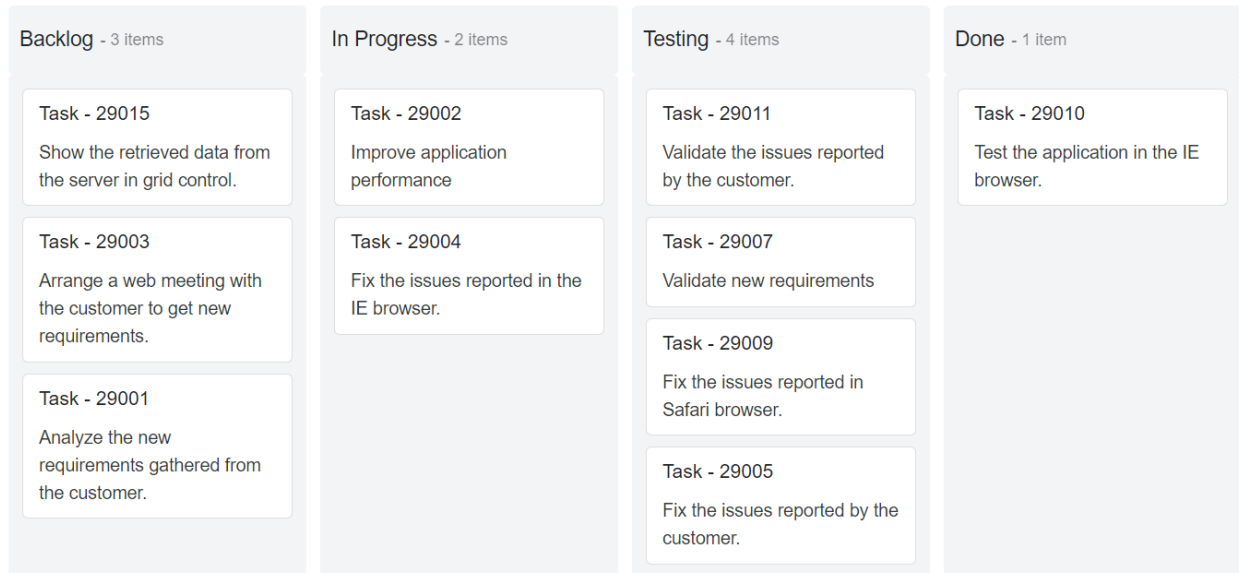
```

"Improvement", Priority = "Normal", CardTags = new List<string>() {
    "Improvement" }, Estimate = 6, Assignee = "Andrew Fuller", AssigneeKey =
    "Andrew Fuller", RankId = 1, Color = "#7d7297", ClassName = new
    List<string>() { "e-improvement", "e-normal", "e-andrew" } },
    new TasksModel { Id = "Task 3", Title = "Task - 29003", Status = "Open",
    Summary = "Arrange a web meeting with the customer to get new
    requirements.", Type = "Others", Priority = "Critical", CardTags = new
    List<string>() { "Meeting" }, Estimate = 5.5, Assignee = "Janet Leverling",
    AssigneeKey = "Janet Leverling", RankId = 2, Color = "#27AE60", ClassName =
    new List<string>() { "e-others", "e-critical", "e-janet" } },
    new TasksModel { Id = "Task 4", Title = "Task - 29004", Status =
    "InProgress", Summary = "Fix the issues reported in the IE browser.", Type =
    "Bug", Priority = "Release Breaker", CardTags = new List<string>() { "IE" },
    Estimate = 2.5, Assignee = "Janet Leverling", AssigneeKey = "Janet
    Leverling", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
    "e-bug", "e-release", "e-janet" } },
    new TasksModel { Id = "Task 5", Title = "Task - 29005", Status = "Testing",
    Summary = "Fix the issues reported by the customer.", Type = "Bug", Priority
    = "Low", CardTags = new List<string>() { "Customer" }, Estimate = 3.5,
    Assignee = "Steven walker", AssigneeKey = "Steven walker", RankId = 1, Color
    = "#cc0000", ClassName = new List<string>() { "e-bug", "e-low", "e-steven" }
    },
    new TasksModel { Id = "Task 6", Title = "Task - 29007", Status = "Testing",
    Summary = "Validate new requirements", Type = "Improvement", Priority =
    "Low", CardTags = new List<string>() { "Validation" }, Estimate = 1.5,
    Assignee = "Robert King", AssigneeKey = "Robert King", RankId = 1, Color =
    "#7d7297", ClassName = new List<string>() { "e-improvement", "e-low", "e-
    robert" } },
    new TasksModel { Id = "Task 7", Title = "Task - 29009", Status = "Testing",
    Summary = "Fix the issues reported in Safari browser.", Type = "Bug",
    Priority = "Release Breaker", CardTags = new List<string>() { "Fix",
    "Safari" }, Estimate = 1.5, Assignee = "Nancy Davloio", AssigneeKey = "Nancy
    Davloio", RankId = 2, Color = "#cc0000", ClassName = new List<string>() {
    "e-bug", "e-release", "e-nancy" } },
    new TasksModel { Id = "Task 8", Title = "Task - 29010", Status = "Close",
    Summary = "Test the application in the IE browser.", Type = "Story",
    Priority = "Low", CardTags = new List<string>() { "Review", "IE" }, Estimate
    = 5.5, Assignee = "Margaret hamilt", AssigneeKey = "Margaret hamilt", RankId
    = 3, Color = "#8b447a", ClassName = new List<string>() { "e-story", "e-low",
    "e-Margaret" } },
    new TasksModel { Id = "Task 9", Title = "Task - 29011", Status = "Testing",
    Summary = "Validate the issues reported by the customer.", Type = "Story",
    Priority = "High", CardTags = new List<string>() { "Validation", "Fix" },
    Estimate = 1, Assignee = "Steven walker", AssigneeKey = "Steven walker",
    RankId = 1, Color = "#8b447a", ClassName = new List<string>() { "e-story",
    "e-low", "e-nancy" } },
    new TasksModel { Id = "Task 10", Title = "Task - 29015", Status = "Open",
    Summary = "Show the retrieved data from the server in grid control.", Type =
    "Story", Priority = "High", CardTags = new List<string>() { "Database",
    "SQL" }, Estimate = 5.5, Assignee = "Margaret hamilt", AssigneeKey =
    "Margaret hamilt", RankId = 4, Color = "#8b447a", ClassName = new
    List<string>() { "e-story", "e-high", "e-steven" } }
    };
}

```

The output will be as follows.





### Card Editing in Blazor Kanban Component

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

- Built-in dialog module
- Dialog template

To get start quickly with Blazor Kanban Edit Dialog using Templates, you can check on this video

{% youtube

"youtube:https://www.youtube.com/watch?v=SgiECU-SZBk" %}

#### Default Dialog

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

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

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

Key | Type | Text

KanbanCardSettings.HeaderField | Input | ID

KeyField | DropDown | -

KanbanCardSettings.ContentField | TextArea | -

KanbanSwimlaneSettings.KeyField (If applicable) | DropDown | -

#### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
```

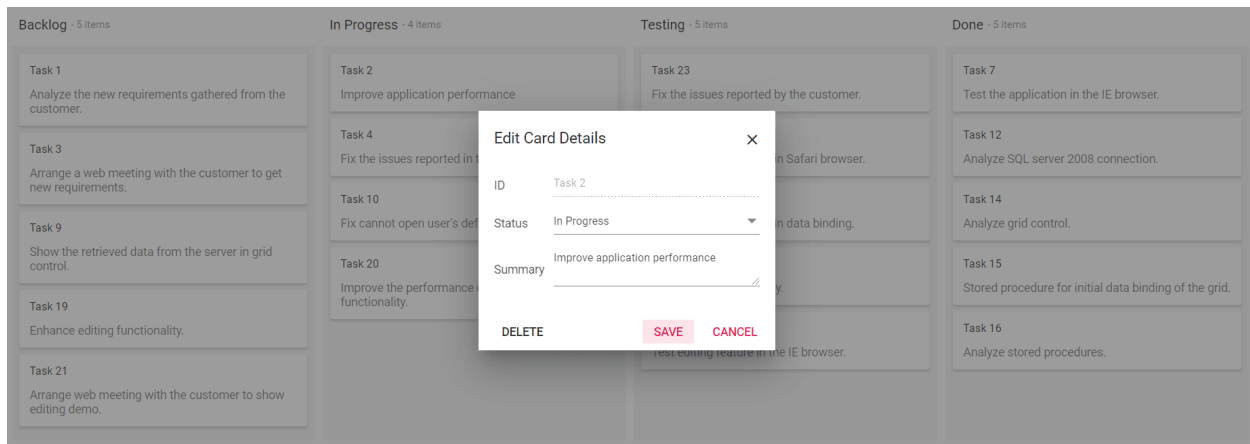
```

<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },

```

```
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}
```

Output be like the below.



### Custom Fields

You can change the default fields of dialog using the [Type](#) property inside the [KanbanDialogSettingsField](#) property. The [Key](#) property is used to map the data source value and render the corresponding component based on the specified [Type](#) property.

The following types are available in dialog fields.

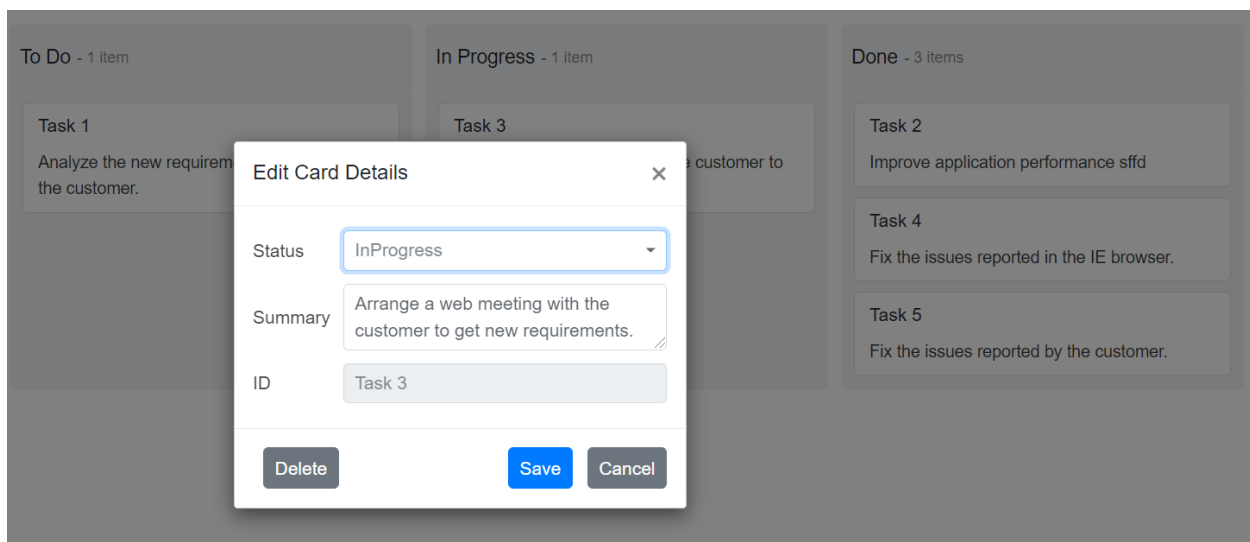
- TextBox
- DropDown
- Numeric
- TextArea

The above types can only be used once in the Custom dialog. The dialog template can be used to render many multiple drop-down lists within a dialog.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban KeyField="Status" DataSource="@Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="To Do" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
  <KanbanDialogSettings>
    <KanbanDialogSettingsFields>
      <KanbanDialogSettingsField Text="Status" Key="Status"
      Type=DialogFieldType.DropDown></KanbanDialogSettingsField>
      <KanbanDialogSettingsField Text="Summary" Key="Summary"
      Type=DialogFieldType.TextArea></KanbanDialogSettingsField>
      <KanbanDialogSettingsField Text="ID" Key="Id"
      Type=DialogFieldType.TextBox></KanbanDialogSettingsField>
    </KanbanDialogSettingsFields>
  </KanbanDialogSettings>
</SfKanban>
```

```
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
public string AssigneeName { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller",
AssigneeName = "Andrew" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "InProgress",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Nancy Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "Close",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Nancy
Davloio", AssigneeName = "Nancy" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Close",
Summary = "Fix the issues reported by the customer.", Assignee = "Andrew
Fuller", AssigneeName = "Andrew" }
};
}
```



### Dialog Template

Using the dialog template, you can render your own form fields with dialog by using the `Template`.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.DropDowns
```

```

<button class="e-btn" @onclick="@ShowAddCardDialog">Add New Card</button>
<SfKanban @ref="KanbanRef" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
  <KanbanDialogSettings>
    <Template>
      @ {
        TasksModel data = (TasksModel)context;
        <table>
          <tbody>
            <tr>
              <td class="e-label">ID</td>
              <td>
                <SfTextBox CssClass="e-field" Value="@data.Id" Enabled="false"></SfTextBox>
              </td>
            </tr>
            <tr>
              <td class="e-label">Status</td>
              <td>
                <SfDropDownList @ref="StatusRef" TValue="string" TItem="DropDownModel"
                CssClass="e-field" DataSource="@StatusData" Value="@data.Status">
                  <DropDownListFieldSettings Text="Value"
                  Value="Value"></DropDownListFieldSettings>
                </SfDropDownList>
              </td>
            </tr>
            <tr>
              <td class="e-label">Assignee</td>
              <td>
                <SfDropDownList @ref="AssigneeRef" TValue="string" TItem="DropDownModel"
                CssClass="e-field" DataSource="@AssigneeData" Value="@data.Assignee">
                  <DropDownListFieldSettings Text="Value"
                  Value="Value"></DropDownListFieldSettings>
                </SfDropDownList>
              </td>
            </tr>
            <tr>
              <td class="e-label">Summary</td>
              <td>
                <SfTextBox @ref="SummaryRef" CssClass="e-field" Multiline="true" @bind-
                Value="@data.Summary"></SfTextBox>
              </td>
            </tr>
          </tbody>
        </table>
      }
    </Template>
  </KanbanDialogSettings>
</SfKanban>

```

```

</KanbanDialogSettings>
</SfKanban>
@code {
    SfKanban<TasksModel> KanbanRef;
    SfDropDownList<string, DropDownModel> StatusRef;
    SfDropDownList<string, DropDownModel> AssigneeRef;
    SfTextBox SummaryRef;
    private List<DropDownModel> StatusData = new List<DropDownModel>()
    {
        new DropDownModel { Id = 0, Value = "Open" },
        new DropDownModel { Id = 1, Value = "InProgress" },
        new DropDownModel { Id = 2, Value = "Testing" },
        new DropDownModel { Id = 3, Value = "Close" }
    };
    private List<DropDownModel> AssigneeData = new List<DropDownModel>()
    {
        new DropDownModel { Id = 0, Value = "Nancy Davloio" },
        new DropDownModel { Id = 1, Value = "Andrew Fuller" },
        new DropDownModel { Id = 2, Value = "Janet Leverling" },
        new DropDownModel { Id = 3, Value = "Steven walker" },
        new DropDownModel { Id = 4, Value = "Robert King" },
        new DropDownModel { Id = 5, Value = "Margaret hamilt" },
        new DropDownModel { Id = 6, Value = "Michael Suyama" }
    };
    private void ShowAddCardDialog()
    {
        TasksModel data = new TasksModel()
        {
            Id = (this.Tasks.Count() + 1).ToString(),
            Title = "",
            Summary = "",
            Status = "Open",
            Assignee = "Nancy Davloio"
        };
        this.KanbanRef.OpenDialog(CurrentAction.Add, data);
    }
    private class DropDownModel
    {
        public int Id { get; set; }
        public string Value { get; set; }
    }
    public class TasksModel
    {
        public string Id { get; set; }
        public string Title { get; set; }
        public string Status { get; set; }
        public string Summary { get; set; }
        public string Assignee { get; set; }
    }
    public List<TasksModel> Tasks = new List<TasksModel>()
    {
        new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
            Summary = "Analyze the new requirements gathered from the customer.",
            Assignee = "Nancy Davloio" },
        new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
            Summary = "Improve application performance", Assignee = "Andrew Fuller" },
    }
}

```

```
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
```

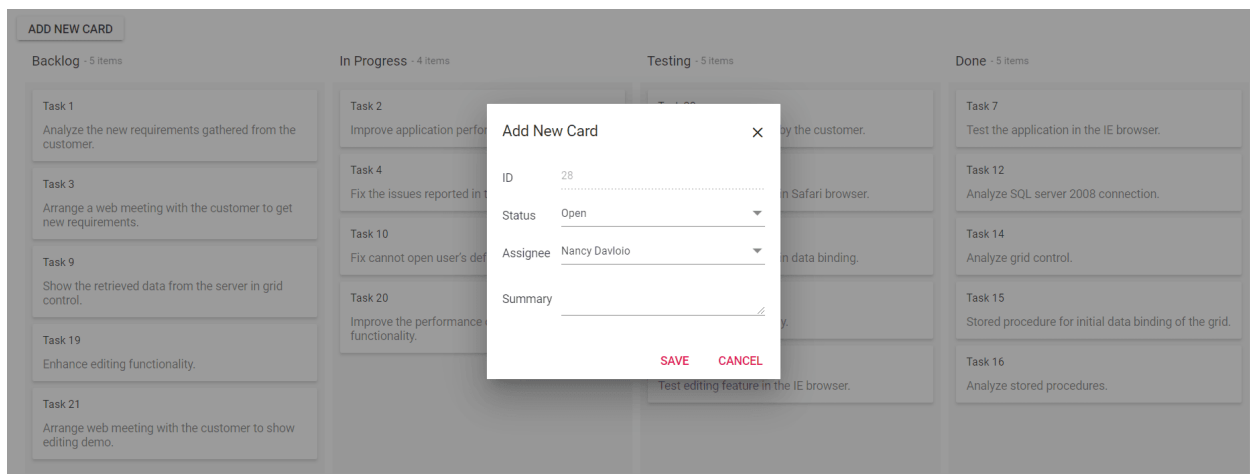


```

new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet Leverling" }
};
}

```

Output be like the below.



### Prevent Dialog

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

### ASPX-CS

```

@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
  <KanbanEvents TValue="TasksModel" DialogOpen="@OnDialogOpen"></KanbanEvents>
</SfKanban>
@code {
  public class TasksModel

```

```
{
    public string Id { get; set; }
    public string Title { get; set; }
    public string Status { get; set; }
    public string Summary { get; set; }
    public string Assignee { get; set; }
}

public List<TasksModel> Tasks = new List<TasksModel>()
{
    new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
        Summary = "Analyze the new requirements gathered from the customer.",
        Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
        Summary = "Improve application performance", Assignee = "Andrew Fuller" },
    new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
        Summary = "Arrange a web meeting with the customer to get new
        requirements.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
        Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
        Leverling" },
    new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
        Summary = "Fix the issues reported by the customer.", Assignee = "Steven
        walker" },
    new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
        Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
        Davloio" },
    new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
        Summary = "Test the application in the IE browser.", Assignee = "Margaret
        hamilt" },
    new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
        Summary = "Validate the issues reported by the customer.", Assignee =
        "Steven walker" },
    new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
        Summary = "Show the retrieved data from the server in grid control.",
        Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
        "InProgress", Summary = "Fix cannot open user's default database SQL
        error.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
        Summary = "Fix the issues reported in data binding.", Assignee = "Janet
        Leverling" },
    new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
        Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
    },
    new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
        Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
        Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
    new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
        Summary = "Stored procedure for initial data binding of the grid.", Assignee
        = "Steven walker" },
    new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
        Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
    new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
        Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
    new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
        Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
}
```

```

new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
private void OnDialogOpen(DialogOpenEventArgs<TasksModel> args)
{
    args.Cancel = true;
}
}

```

### Persisting data in server

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

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

#### URL adaptor

You can use the [UrlAdaptor](#) of SfDataManager when binding data source for remote data. During the initial load of Kanban, data are fetched from remote data and bound to the Kanban using the [Url](#) property of SfDataManager.

CRUD operations in Kanban can be mapped to server-side controller actions by using the properties [InsertUrl](#), [RemoveUrl](#), [UpdateUrl](#), and [CrudUrl](#).

- [InsertUrl](#) – You can perform a single insertion operation on the server-side.
- [UpdateUrl](#) – You can update single data on the server-side.
- [RemoveUrl](#) – You can remove single data on the server-side.
- [CrudUrl](#) – You can perform bulk data operation on the server-side.

### ASPX-CS

```

@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Kanban
<SfKanban ID="Kanban" TValue="Order" KeyField="ShipCity">
<SfDataManager Url="/api/Default" UpdateUrl="/api/Default/Update"
RemoveUrl="/api/Default/Delete" InsertUrl="/api/Default/Add"
Adaptor="Adaptors.UrlAdaptor"></SfDataManager>
<KanbanColumns>
<KanbanColumn HeaderText="Brazil" KeyField=@(new List<string>{"Brazil"})
AllowAdding=true></KanbanColumn>
<KanbanColumn HeaderText="Sweden" KeyField=@(new
List<string>{"Sweden"})></KanbanColumn>
<KanbanColumn HeaderText="India" KeyField=@(new
List<string>{"India"})></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="EmployeeID"
ContentField="ShipName"></KanbanCardSettings>
</SfKanban>

```

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

#### ASPX-CS

```

namespace Blazor_Kanban_Crud_UrlAdaptor.Controllers
{
[ApiController]
public class DefaultController : ControllerBase
{
OrderDataAccessLayer db = new OrderDataAccessLayer();
// OrderContext db = new OrderContext();
// GET: api/Default
[HttpPost]
[Route("api/[controller]")]
public object Post([FromBody] DataManagerRequest dm)
{
IEnumerable data = db.GetAllOrders(); //call the method to fetch data from
db and return to client
int count = data.Cast<Order>().Count();
return dm.RequiresCounts ? new DataResult() { Result = data, Count = count }
: (object)data;
}
[HttpPost]
[Route("api/Default/Add")]
public void Add([FromBody] CRUDModel<Order> value)
{
db.AddOrder(value.Value);
}
[HttpPost]
[Route("api/Default/Update")]
public void Update([FromBody] CRUDModel<Order> value)
{
db.UpdateOrder(value.Value);
}
[HttpPost]
[Route("api/Default/Delete")]
public void Delete([FromBody] CRUDModel<Order> value)
{

```

```
db.DeleteOrder(Convert.ToInt32(Convert.ToString(value.Key)));
}
public class CRUDModel<T> where T : class
{
    [JsonProperty("action")]
    public string Action { get; set; }
    [JsonProperty("table")]
    public string Table { get; set; }
    [JsonProperty("keyColumn")]
    public string KeyColumn { get; set; }
    [JsonProperty("key")]
    public object Key { get; set; }
    [JsonProperty("value")]
    public T Value { get; set; }
    [JsonProperty("added")]
    public List<T> Added { get; set; }
    [JsonProperty("changed")]
    public List<T> Changed { get; set; }
    [JsonProperty("deleted")]
    public List<T> Deleted { get; set; }
    [JsonProperty("params")]
    public IDictionary<string, object> Params { get; set; }
}
}
```

#### *Insert card*

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

The following code example describes the above behavior.

#### **ASPX-CS**

```
[HttpPost]
[Route("api/Default/Add")]
public void Add([FromBody] CRUDModel<Order> value)
{
    db.AddOrder(value.Value);
}
```

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

#### *Update card*

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

The following code example describes the above behavior.

#### **ASPX-CS**

```
[HttpPost]
[Route("api/Default/Update")]
public void Update([FromBody] CRUDModel<Order> value)
{
    db.UpdateOrder(value.Value);
}
```

```
}

```

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

#### Delete card

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

The following code example describes the above behavior.

#### ASPX-CS

```
[HttpPost]
[Route("api/Default/Delete")]
public void Delete([FromBody] CRUDModel<Order> value)
{
    db.DeleteOrder(Convert.ToInt32(Convert.ToString(value.Key)));
}
```

The primary key value of the card to be deleted will be bound to the `Key` parameter.

#### Bulk update

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

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

The following code example describes the above behavior.

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

#### ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Kanban
<SfKanban ID="Kanban" TValue="Order" KeyField="ShipCity">
<SfDataManager Url="/api/Default" UpdateUrl="/api/Default/Update"
RemoveUrl="/api/Default/Delete" InsertUrl="/api/Default/Add"
CrudUrl="/api/Default/Batch" Adaptor="Adaptors.UrlAdaptor"></SfDataManager>
<KanbanColumns>
<KanbanColumn HeaderText="Brazil" KeyField=@(new List<string>{"Brazil"})
AllowAdding=true></KanbanColumn>
<KanbanColumn HeaderText="Sweden" KeyField=@(new
List<string>{"Sweden"})></KanbanColumn>
<KanbanColumn HeaderText="India" KeyField=@(new
List<string>{"India"})></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="EmployeeID"
ContentField="ShipName"></KanbanCardSettings>
</SfKanban>
```

#### ASPX-CS

```
namespace Blazor_Kanban_Crud_UrlAdaptor.Controllers
{
    [ApiController]
    public class DefaultController : ControllerBase
    {
        OrderDataAccessLayer db = new OrderDataAccessLayer();
        // OrderContext db = new OrderContext();
        // GET: api/Default
        [HttpPost]
        [Route("api/[controller]")]
        public object Post([FromBody] DataManagerRequest dm)
        {
            IEnumerable data = db.GetAllOrders(); //call the method to fetch data from
            db and return to client
            int count = data.Cast<Order>().Count();
            return dm.RequiresCounts ? new DataResult() { Result = data, Count = count }
            : (object)data;
        }
        [HttpPost]
        [Route("api/Default/Add")]
        public void Add([FromBody] CRUDModel<Order> value)
        {
            db.AddOrder(value.Value);
        }
        [HttpPost]
        [Route("api/Default/Update")]
        public void Update([FromBody] CRUDModel<Order> value)
        {
            db.UpdateOrder(value.Value);
        }
        [HttpPost]
        [Route("api/Default/Delete")]
        public void Delete([FromBody] CRUDModel<Order> value)
        {
            db.DeleteOrder(Convert.ToInt32(Convert.ToString(value.Key)));
        }
        [HttpPost]
        [Route("api/Default/Batch")]
        public void Batch([FromBody] CRUDModel<Order> value)
        {
            if (value.Changed.Count > 0)
            {
                foreach (Order rec in value.Changed)
                {
                    db.UpdateOrder(rec);
                }
            }
            if (value.Added.Count > 0)
            {
                foreach (Order rec in value.Added)
                {
                    db.AddOrder(rec);
                }
            }
            if (value.Deleted.Count > 0)
            {
                foreach (Order rec in value.Deleted)
            }
        }
    }
}
```

```

{
    db.DeleteOrder(rec.EmployeeID);
}
}
}
public class CRUDModel<T> where T : class
{
    [JsonProperty("action")]
    public string Action { get; set; }
    [JsonProperty("table")]
    public string Table { get; set; }
    [JsonProperty("keyColumn")]
    public string KeyColumn { get; set; }
    [JsonProperty("key")]
    public object Key { get; set; }
    [JsonProperty("value")]
    public T Value { get; set; }
    [JsonProperty("added")]
    public List<T> Added { get; set; }
    [JsonProperty("changed")]
    public List<T> Changed { get; set; }
    [JsonProperty("deleted")]
    public List<T> Deleted { get; set; }
    [JsonProperty("params")]
    public IDictionary<string, object> Params { get; set; }
}
}
}

```

You can find the fully working sample [here](#).

### Tooltip in Blazor Kanban Component

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

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

### ASPX-CS

```

@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks"
EnableTooltip="true">
<KanbanColumns>
<KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
{"Open"}) "></KanbanColumn>
<KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{"InProgress"}) "></KanbanColumn>
<KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
{"Testing"}) "></KanbanColumn>
<KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
{"Close"}) "></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>

```



**</SfKanban>**

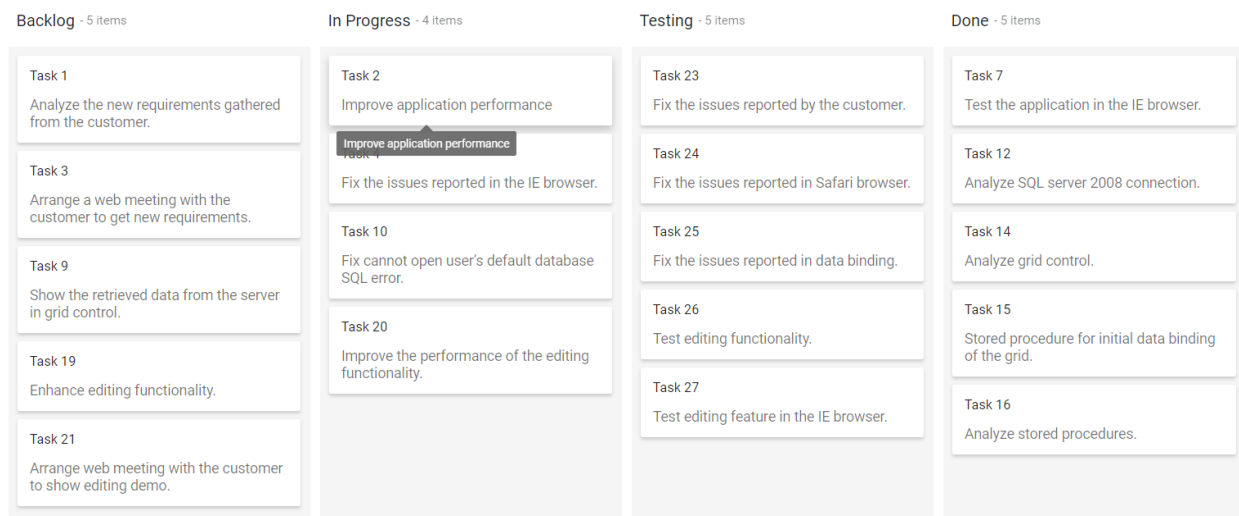
```
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
}
```

```

new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}

```

Output be like the below.



## Styling And Appearance in Blazor Kanban Component

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### WIP Validation in Blazor Kanban Component

Validate particular column using the **MinCount** or **MaxCount** properties. The corresponding columns gets different appearance when validation fails. In default layout, **ConstraintType** property accept only **Column** type. In swimlane layout, accept both **Column** and **Swimlane** constraint type.

There are two types of constraints:

1. Column
2. Swimlane

---

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

---

#### Minimum card limit

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

#### Maximum card limit

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

### CSHARP

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks">
  <KanbanColumns>
```

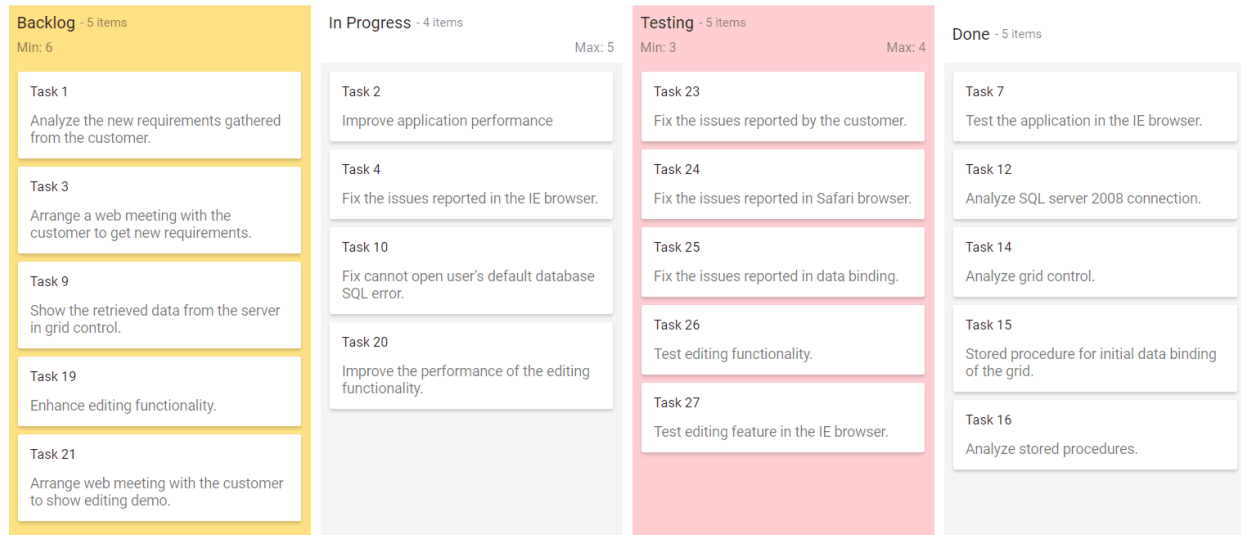
```

<KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>() {"Open"})"
ShowItemCount="true" MinCount="6"></KanbanColumn>
<KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{"InProgress"})" ShowItemCount="true" MaxCount="5"></KanbanColumn>
<KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
{"Testing"})" ShowItemCount="true" MinCount="3" MaxCount="4"></KanbanColumn>
<KanbanColumn HeaderText="Done" KeyField="@ (new List<string>() {"Close"})"
ShowItemCount="true"></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },

```

```
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}
```

Output be like the below.



## Responsive Mode in Blazor Kanban Component

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

### Layouts

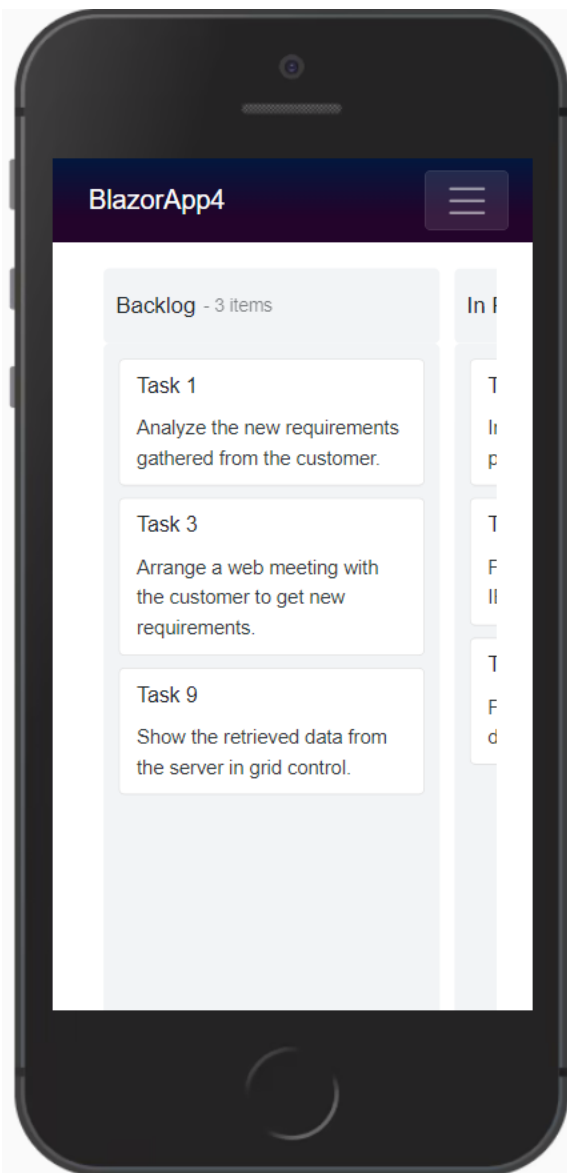
Possible layouts are:

- Default Layout
- Swimlane Layout

### Default Layout

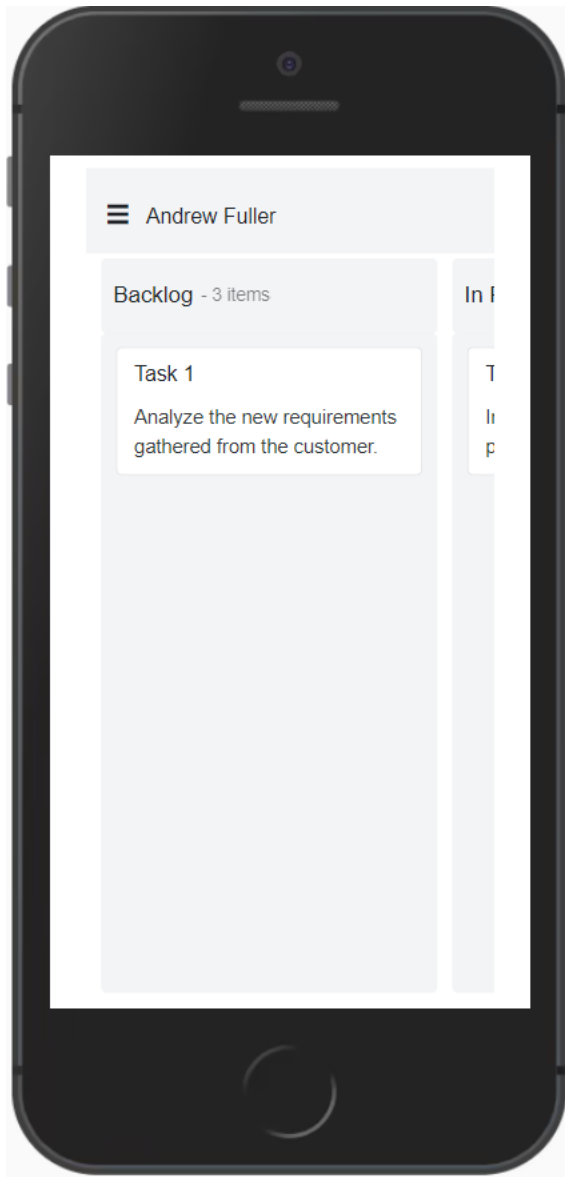
Kanban user interface is customized and redesigned for the best view on small screens. In responsive mode, the first column occupies 80% and the second column occupies 20% of the screen layout. Tap and hold the Kanban card to drag and drop it. Swipe left or right to view the columns.





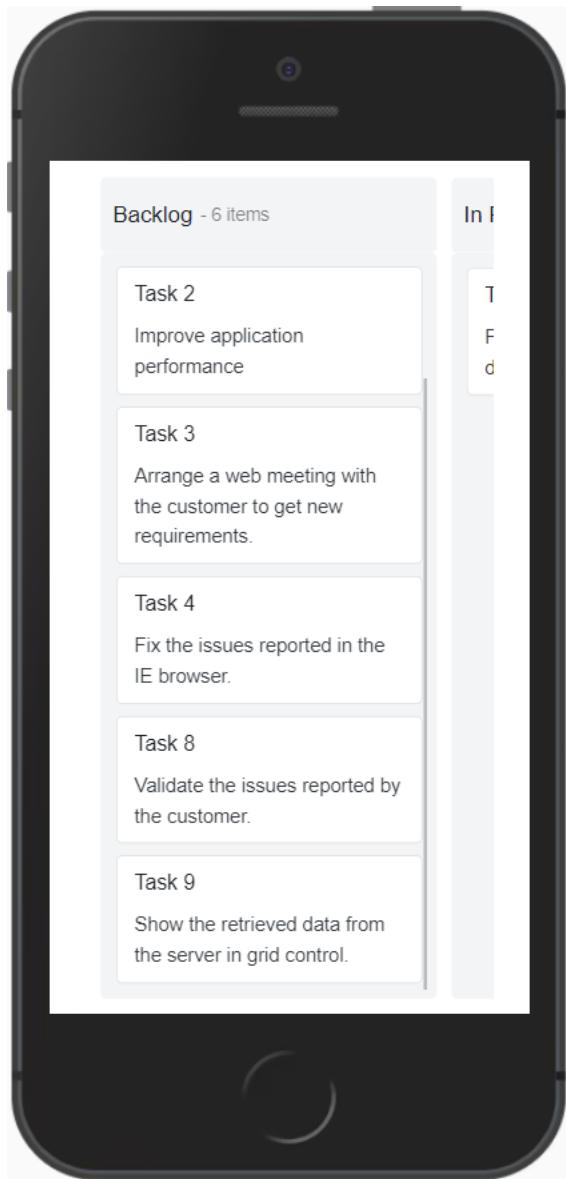
#### *Swimlane Layout*

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



### Scrolling

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

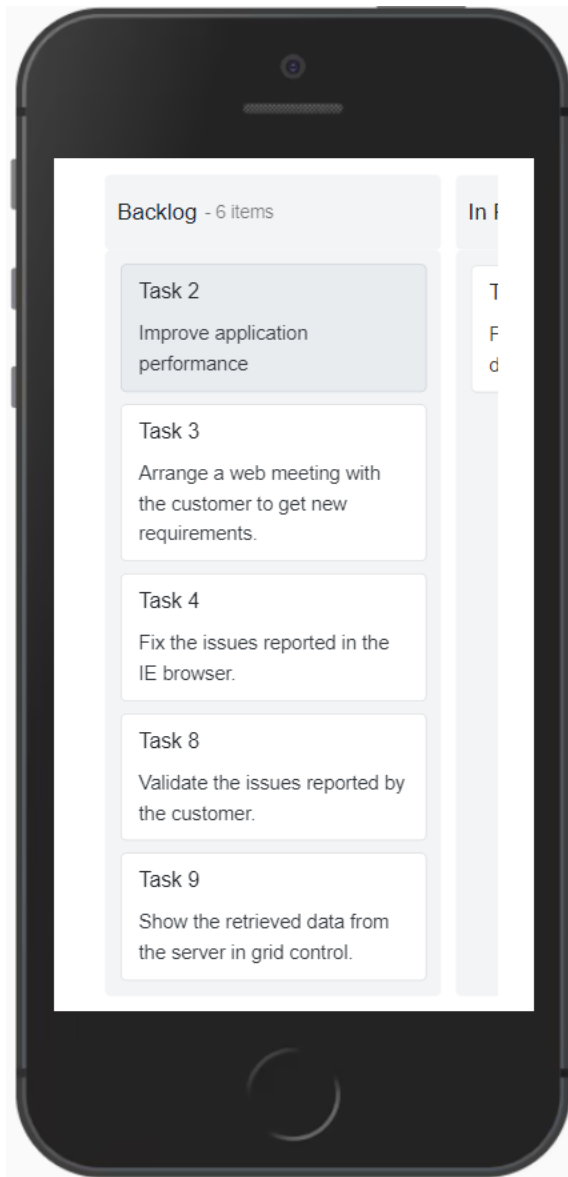


### Selection

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

#### Single Selection

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



### Localization in Blazor Kanban Component

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

Use **Resource** file to translate the static text of the kanban. 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 Blazor Server and WebAssembly project for Syncfusion Blazor components.

By using **Locale** property, you can set the culture dynamically in kanban component.

| Locale key | en-US (default) |

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

| Kanban\_Items | items |

| Kanban\_Min | Min |

| Kanban\_Max | Max |

| Kanban\_CardsSelected | Cards Selected |

| Kanban\_AddTitle | Add New Card |

| Kanban\_EditTitle | Edit Card Details |

| Kanban\_DeleteTitle | Delete Card |

| Kanban\_DeleteContent | Are you sure you want to delete this card? |

| Kanban\_Save | Save |

| Kanban\_Delete | Delete |

| Kanban\_Cancel | Cancel |

| Kanban\_Yes | Yes |

| Kanban\_No | No |

| KanbanNoCards | No cards to display |

### Localization

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

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks"
Locale="de">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>() {"Open"})"
ShowItemCount="true" MinCount="6"></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{"InProgress"})" ShowItemCount="true" MaxCount="3"></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>() {"Close"})"
ShowItemCount="true"></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
  <KanbanSwimlaneSettings KeyField="Assignee"></KanbanSwimlaneSettings>
</SfKanban>

@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
private List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
```

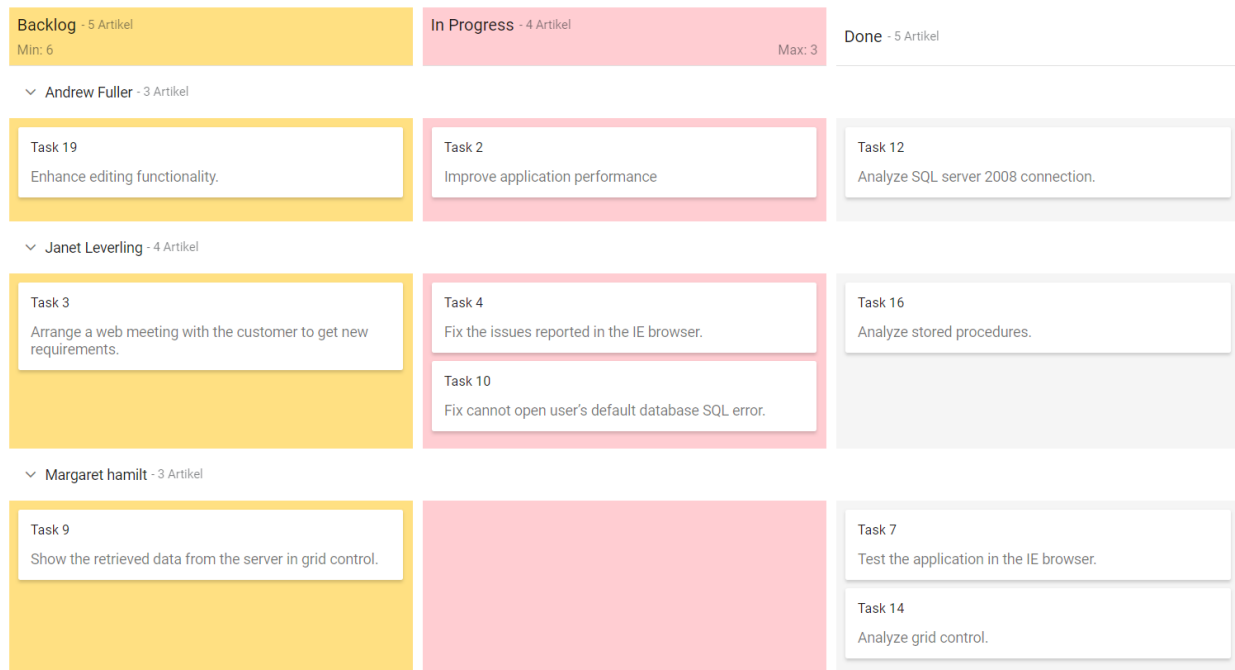
```
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
```

```

new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}

```

Output be like the below.



### Right to left (RTL)

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

### ASPX-CS

```

@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks"
Locale="ar" EnableRtl="true">
<KanbanColumns>
<KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>() { "Open" }) "
ShowItemCount="true" MinCount="6"></KanbanColumn>

```

```

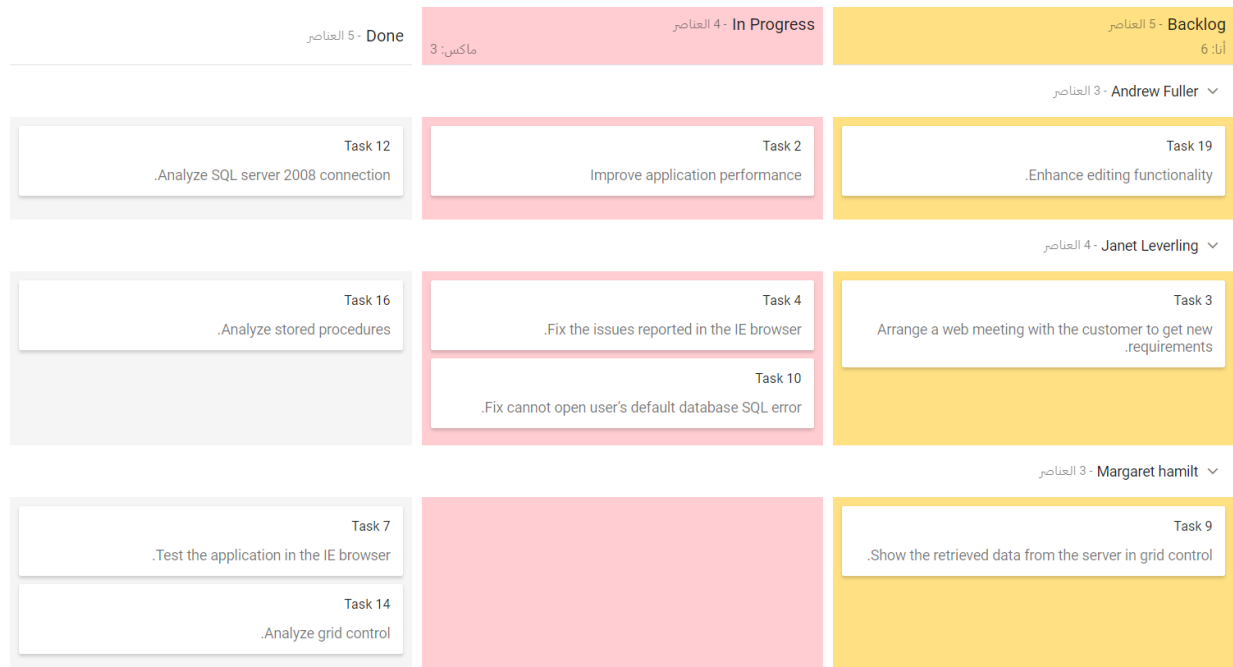
<KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{"InProgress"})" ShowItemCount="true" MaxCount="3"></KanbanColumn>
<KanbanColumn HeaderText="Done" KeyField="@ (new List<string>() {"Close"})"
ShowItemCount="true"></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
<KanbanSwimlaneSettings KeyField="Assignee"></KanbanSwimlaneSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
private List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },

```



```
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}
```

Output be like the below.



## Dimensions in Blazor Kanban Component

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

- Auto
- Pixel
- Percentage

### Auto height and width

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

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks"
Width="auto" Height="auto">
<KanbanColumns>
<KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
{"Open"}) "></KanbanColumn>
<KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{"InProgress"}) "></KanbanColumn>
<KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
{"Testing"}) "></KanbanColumn>
<KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
{"Close"}) "></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
```

```
</SfKanban>
```

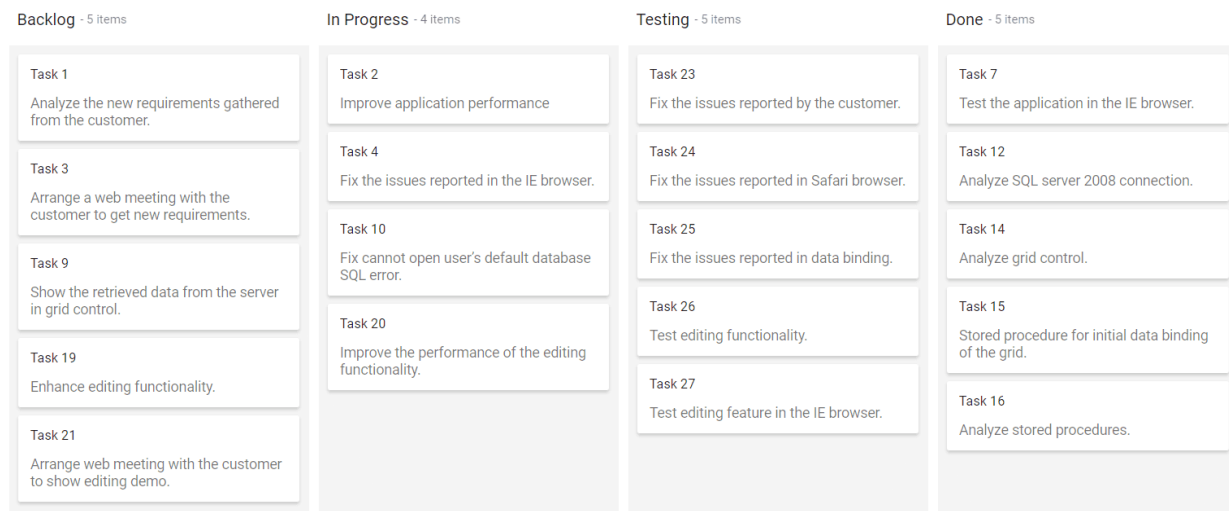
```
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
}
```

```

new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}

```

Output be like the below.



### Height and width in pixel

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

**ASPX-CS**

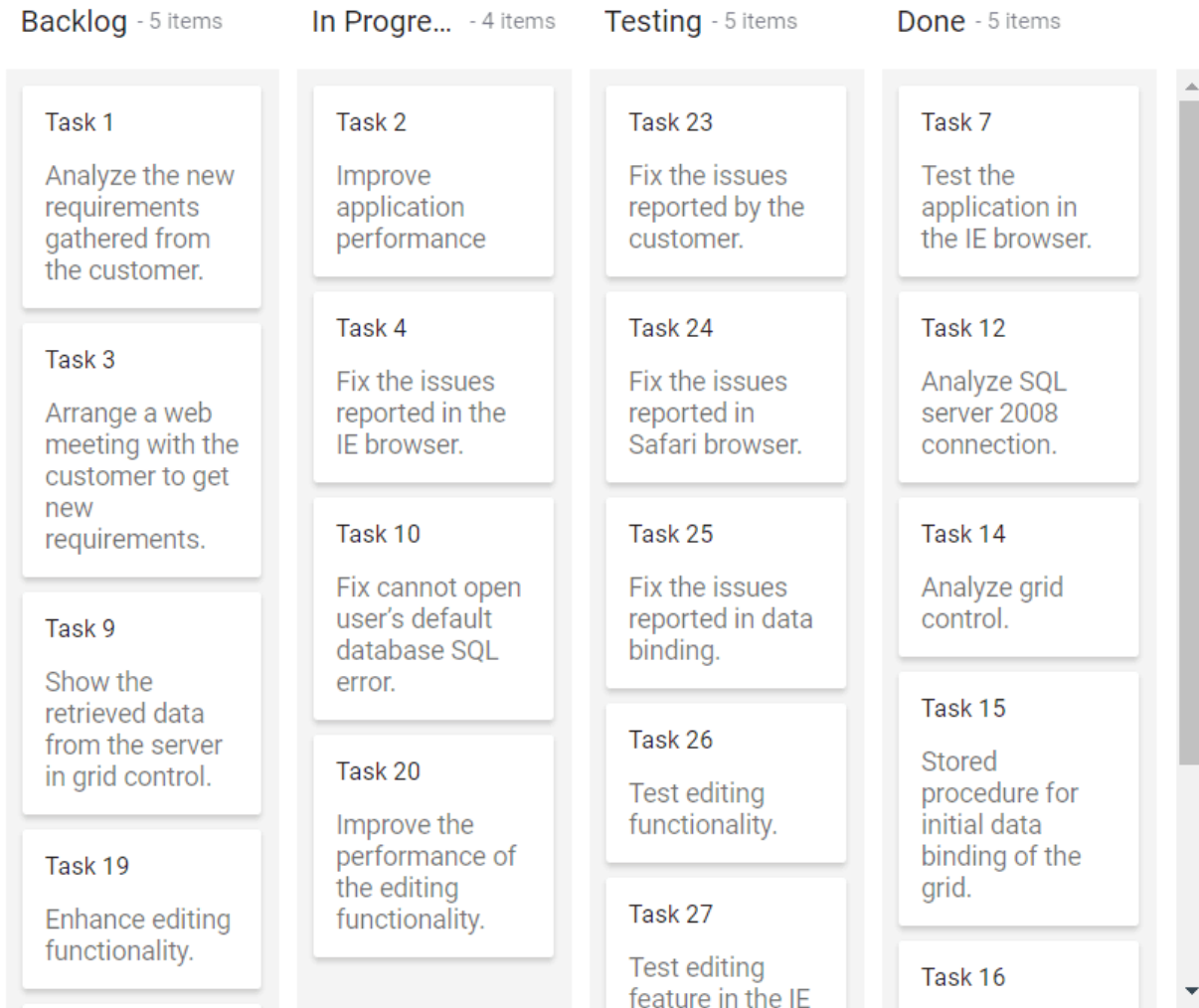
```

@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks"
Width="650px" Height="550px">
<KanbanColumns>
<KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
{"Open"})"></KanbanColumn>
<KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
{"InProgress"})"></KanbanColumn>
<KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
{"Testing"})"></KanbanColumn>
<KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
{"Close"})"></KanbanColumn>
</KanbanColumns>
<KanbanCardSettings HeaderField="Id"
ContentField="Summary"></KanbanCardSettings>
</SfKanban>
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },

```

```
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}
```

Output be like the below.



### Height and width in percentage

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

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban TValue="TasksModel" KeyField="Status" DataSource="Tasks"
Width="100%" Height="100%">
  <KanbanColumns>
    <KanbanColumn HeaderText="Backlog" KeyField="@ (new List<string>()
    { "Open" }) "></KanbanColumn>
    <KanbanColumn HeaderText="In Progress" KeyField="@ (new List<string>()
    { "InProgress" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Testing" KeyField="@ (new List<string>()
    { "Testing" }) "></KanbanColumn>
    <KanbanColumn HeaderText="Done" KeyField="@ (new List<string>()
    { "Close" }) "></KanbanColumn>
  </KanbanColumns>
  <KanbanCardSettings HeaderField="Id"
  ContentField="Summary"></KanbanCardSettings>
</SfKanban>
```

```
@code {
public class TasksModel
{
public string Id { get; set; }
public string Title { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
public List<TasksModel> Tasks = new List<TasksModel>()
{
new TasksModel { Id = "Task 1", Title = "BLAZ-29001", Status = "Open",
Summary = "Analyze the new requirements gathered from the customer.",
Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 2", Title = "BLAZ-29002", Status = "InProgress",
Summary = "Improve application performance", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 3", Title = "BLAZ-29003", Status = "Open",
Summary = "Arrange a web meeting with the customer to get new
requirements.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 4", Title = "BLAZ-29004", Status = "InProgress",
Summary = "Fix the issues reported in the IE browser.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 5", Title = "BLAZ-29005", Status = "Review",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 6", Title = "BLAZ-29006", Status = "Review",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 7", Title = "BLAZ-29007", Status = "Close",
Summary = "Test the application in the IE browser.", Assignee = "Margaret
hamilt" },
new TasksModel { Id = "Task 8", Title = "BLAZ-29008", Status = "Validate",
Summary = "Validate the issues reported by the customer.", Assignee =
"Steven walker" },
new TasksModel { Id = "Task 9", Title = "BLAZ-29009", Status = "Open",
Summary = "Show the retrieved data from the server in grid control.",
Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 10", Title = "BLAZ-29010", Status =
"InProgress", Summary = "Fix cannot open user's default database SQL
error.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 11", Title = "BLAZ-29011", Status = "Review",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 12", Title = "BLAZ-29012", Status = "Close",
Summary = "Analyze SQL server 2008 connection.", Assignee = "Andrew Fuller"
},
new TasksModel { Id = "Task 13", Title = "BLAZ-29013", Status = "Validate",
Summary = "Validate databinding issues.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 14", Title = "BLAZ-29014", Status = "Close",
Summary = "Analyze grid control.", Assignee = "Margaret hamilt" },
new TasksModel { Id = "Task 15", Title = "BLAZ-29015", Status = "Close",
Summary = "Stored procedure for initial data binding of the grid.", Assignee
= "Steven walker" },
new TasksModel { Id = "Task 16", Title = "BLAZ-29016", Status = "Close",
Summary = "Analyze stored procedures.", Assignee = "Janet Leverling" },
new TasksModel { Id = "Task 17", Title = "BLAZ-29017", Status = "Validate",
Summary = "Validate editing issues.", Assignee = "Nancy Davloio" },
}
```

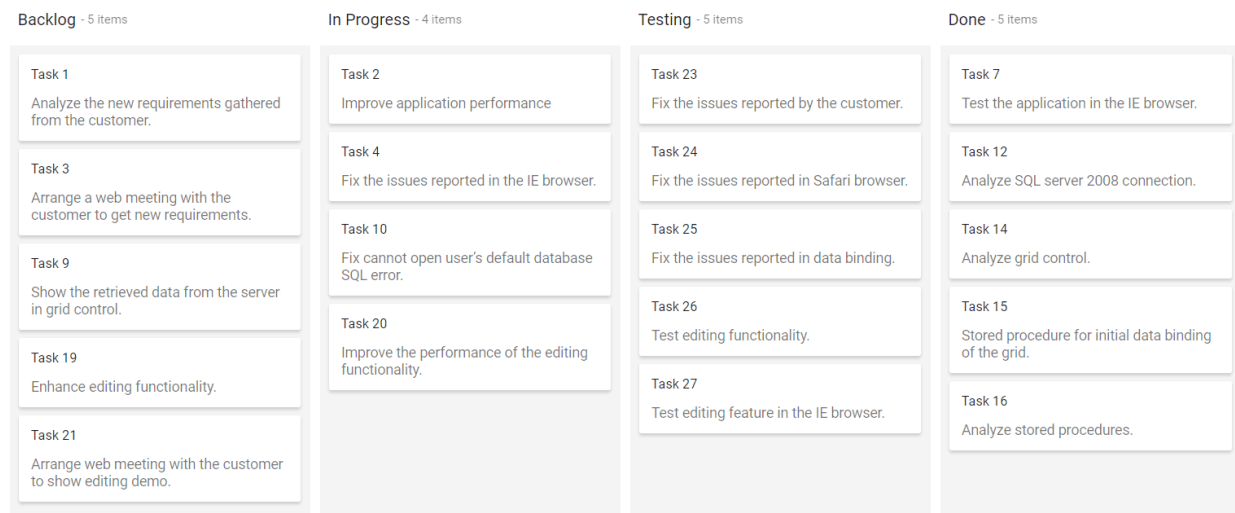


```

new TasksModel { Id = "Task 18", Title = "BLAZ-29018", Status = "Review",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 19", Title = "BLAZ-29019", Status = "Open",
Summary = "Enhance editing functionality.", Assignee = "Andrew Fuller" },
new TasksModel { Id = "Task 20", Title = "BLAZ-29020", Status =
"InProgress", Summary = "Improve the performance of the editing
functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 21", Title = "BLAZ-29021", Status = "Open",
Summary = "Arrange web meeting with the customer to show editing demo.",
Assignee = "Steven walker" },
new TasksModel { Id = "Task 22", Title = "BLAZ-29022", Status = "Review",
Summary = "Fix the editing issues reported by the customer.", Assignee =
"Janet Leverling" },
new TasksModel { Id = "Task 23", Title = "BLAZ-29023", Status = "Testing",
Summary = "Fix the issues reported by the customer.", Assignee = "Steven
walker" },
new TasksModel { Id = "Task 24", Title = "BLAZ-29024", Status = "Testing",
Summary = "Fix the issues reported in Safari browser.", Assignee = "Nancy
Davloio" },
new TasksModel { Id = "Task 25", Title = "BLAZ-29025", Status = "Testing",
Summary = "Fix the issues reported in data binding.", Assignee = "Janet
Leverling" },
new TasksModel { Id = "Task 26", Title = "BLAZ-29026", Status = "Testing",
Summary = "Test editing functionality.", Assignee = "Nancy Davloio" },
new TasksModel { Id = "Task 27", Title = "BLAZ-29027", Status = "Testing",
Summary = "Test editing feature in the IE browser.", Assignee = "Janet
Leverling" }
};
}

```

Output be like the below.



## Events in Blazor Kanban Component

This section explains the list of events of the Kanban component which will be triggered for appropriate Kanban actions.

### OnLoad

**OnLoad** event allows customization of Kanban properties before rendering.

#### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban>
  <KanbanEvents OnLoad="@OnLoadHandler" ></KanbanEvents>
</SfKanban>
@code{
public void OnLoadHandler(Object args)
{
    // Here you can customize your code
}
}
```

### ActionBegin

**ActionBegin** event triggers at the beginning of every Kanban action.

#### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban>
  <KanbanEvents ActionBegin="@ActionBeginHandler" ></KanbanEvents>
</SfKanban>
@code{
public void ActionBeginHandler(ActionEventArgs<TValue> args)
{
    // Here you can customize your code
}
}
```

### ActionComplete

**ActionComplete** event triggers on successful completion of the Kanban actions.

#### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban>
  <KanbanEvents ActionComplete="@ActionCompleteHandler" ></KanbanEvents>
</SfKanban>
@code{
public void ActionCompleteHandler(ActionEventArgs<TValue> args)
{
    // Here you can customize your code
}
}
```

### ActionFailure

**ActionFailure** event triggers when a Kanban action gets failed or interrupted and it will return an error information.

#### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
```

```
<SfKanban>
<KanbanEvents ActionFailure="@ActionFailureHandler" ></KanbanEvents>
</SfKanban>
@code{
public void ActionFailureHandler(ActionEventArgs<TValue> args)
{
// Here you can customize your code
}
}
```

### CardClick

**CardClick** event triggers on single-clicking the Kanban cards.

#### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban>
<KanbanEvents CardClick="@CardClickHandler" ></KanbanEvents>
</SfKanban>
@code{
public void CardClickHandler(CardClickEventArgs<TValue> args)
{
// Here you can customize your code
}
}
```

### CardDoubleClick

**CardDoubleClick** event triggers on double-clicking the Kanban cards.

#### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban>
<KanbanEvents CardDoubleClick="@CardDoubleClickHandler" ></KanbanEvents>
</SfKanban>
@code{
public void CardDoubleClickHandler(CardClickEventArgs<TValue> args)
{
// Here you can customize your code
}
}
```

### CardRendered

**CardRendered** event triggers before each card of the Kanban rendering on the page.

#### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban>
<KanbanEvents CardRendered="@CardRenderedHandler" ></KanbanEvents>
</SfKanban>
@code{
public void CardRenderedHandler(CardRenderedEventArgs<TValue> args)
{
// Here you can customize your code
}
```

```
}  
}
```

### DataBinding

**DataBinding** event triggers before the data binds to the Kanban.

#### ASPX-CS

```
@using Syncfusion.Blazor.Kanban  
<SfKanban>  
<KanbanEvents DataBinding="@DataBindingHandler" ></KanbanEvents>  
</SfKanban>  
@code{  
public void DataBindingHandler(DataBindingEventArgs<TValue> args)  
{  
    // Here you can customize your code  
}}
```

### DialogClose

**DialogClose** event triggers before the dialog closes.

#### ASPX-CS

```
@using Syncfusion.Blazor.Kanban  
<SfKanban>  
<KanbanEvents DialogClose="@DialogCloseHandler" ></KanbanEvents>  
</SfKanban>  
@code{  
public void DialogCloseHandler(DialogCloseEventArgs<TValue> args)  
{  
    // Here you can customize your code  
}}
```

### DialogOpen

**DialogOpen** event triggers before the dialog opens.

#### ASPX-CS

```
@using Syncfusion.Blazor.Kanban  
<SfKanban>  
<KanbanEvents DialogOpen="@DialogOpenHandler" ></KanbanEvents>  
</SfKanban>  
@code{  
public void DialogOpenHandler(DialogOpenEventArgs<TValue> args)  
{  
    // Here you can customize your code  
}}
```

### DragStart

**DragStart** event triggers when the card drag actions start.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban>
<KanbanEvents DragStart="@DragStartHandler" ></KanbanEvents>
</SfKanban>
@code{
public void DragStartHandler(DragEventArgs<TValue> args)
{
// Here you can customize your code
}
}
```

### DragStop

**DragStop** event triggers when the card drag actions stop.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban>
<KanbanEvents DragStop="@DragStopHandler" ></KanbanEvents>
</SfKanban>
@code{
public void DragStopHandler(DragEventArgs<TValue> args)
{
// Here you can customize your code
}
}
```

### QueryCellInfo

**QueryCellInfo** event triggers before each column of the Kanban rendering on the page.

### ASPX-CS

```
@using Syncfusion.Blazor.Kanban
<SfKanban>
<KanbanEvents QueryCellInfo="@QueryCellInfoHandler" ></KanbanEvents>
</SfKanban>
@code{
public void QueryCellInfoHandler(QueryCellInfoEventArgs<TValue> args)
{
// Here you can customize your code
}
}
```

## Linear Gauge

### Getting Started with Blazor Linear Gauge Component (SfLinearGauge)

The Blazor Linear Gauge is an ideal component for visualizing numeric values in a linear scale with features like multiple axes, different orientations, and more.

This section briefly explains how to include a Linear gauge in your Blazor server-Side application. Refer to this [Getting Started with Syncfusion Blazor for Serve-Side in Visual Studio](#) documentation for the introduction and configuring the common specifications.

## Importing Syncfusion Blazor Linear Gauge component in the application

1. Install **Syncfusion.Blazor.LinearGauge** NuGet package in the application 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 to 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 a component package to the application

Open the `~/_Imports.razor` file and include the **Syncfusion.Blazor.LinearGauge** namespace.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
```

## 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](#).

### Initializing Linear Gauge component in the application

The Syncfusion Linear gauge component can be initialized in any razor page inside the ~/Pages folder. For example, the Linear Gauge component is added to the ~/Pages/Index.razor page. 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.

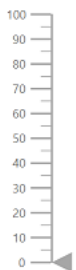
#### ASPX-CS

```

@page "/"
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer></LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>

```

After the successful compilation of your application, press F5 to run the application. The Blazor Linear gauge component will be rendered in the web browser as illustrated in the following screenshot.



### Set pointer value

Pointers are used to indicate values on an axis. You can change the pointer value using the [PointerValue](#) property in the [LinearGaugePointer](#).

In Linear Gauge, you can configure multiple axes. On each axis, you can add a pointer.

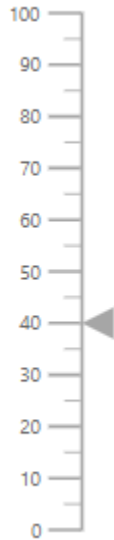
#### ASPX-CS

```

@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>

```

```
<LinearGaugePointer PointerValue="40">
</LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```



#### Add a title for Linear Gauge

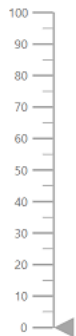
The title can be added to the linear gauge to provide a piece of quick information to the users about the context of the rendered linear gauge. You can add the title to the linear gauge using [Title](#) property in [SfLinearGauge](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge Title="Linear Gauge">
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer></LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```



Linear Gauge

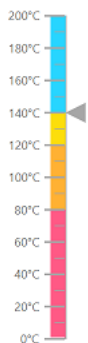


### Add ranges in the Linear gauge

The 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 [LinearGaugeRange](#). You can add any number of ranges for an axis using [LinearGaugeRange](#).

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAxes>
<LinearGaugeAxis Minimum="0" Maximum="200">
<LinearGaugeAxisLabelStyle Format="{value}°C"></LinearGaugeAxisLabelStyle>
<LinearGaugePointers>
<LinearGaugePointer PointerValue="140">
</LinearGaugePointer>
</LinearGaugePointers>
<LinearGaugeRanges>
<LinearGaugeRange Start="0" End="80" Color="#ff5985"></LinearGaugeRange>
<LinearGaugeRange Start="80" End="120" Color="#ffb133"></LinearGaugeRange>
<LinearGaugeRange Start="120" End="140" Color="#fcde0b"></LinearGaugeRange>
<LinearGaugeRange Start="140" End="200" Color="#27d5ff"></LinearGaugeRange>
</LinearGaugeRanges>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```



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 Linear Gauge Component

### Size for Linear Gauge

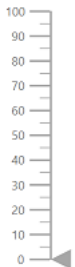
The height and width of the Linear Gauge can be set using the [Width](#) and [Height](#) properties in [SfLinearGauge](#) class.

#### *In Pixel*

The size of the Linear Gauge can be set in pixel as demonstrated below.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge Width="100px" Height="350px">
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugePointers>
        <LinearGaugePointer></LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```

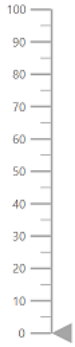


#### *In Percentage*

By setting value in percentage, Linear Gauge receives its dimension matching to its parent. For example, when the height is set as **50%**, Linear Gauge renders to half of the parent height. The Linear Gauge will be responsive when the width is set as **100%**.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge Width="100%" Height="50%">
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugePointers>
        <LinearGaugePointer></LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```



---

When the component's size is not specified, the height will be **450px** and the width will be the same as the parent element's width.

---

### Axes in Blazor Linear Gauge Component

Axis is used to indicate the numeric values in the linear scale. The Linear Gauge component can have any number of axes. The sub-elements of an axis are line, ticks, labels, ranges, and pointers.

#### Setting the start value and end value of the axis

The start value and end value for the Linear Gauge can be set using the [Minimum](#) and [Maximum](#) properties in the [LinearGaugeAxis](#) class respectively. By default, the start value of the axis is **0** and the end value of the axis is **100**.

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis Minimum="20" Maximum="200">
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```



### Line Customization

The following properties in the [LinearGaugeLine](#) class can be used to customize the axis line in the Linear Gauge.

- [Height](#) - To set the length of the axis line.
- [Width](#) - To set the thickness of the axis line.
- [Color](#) - To set the color of the axis line.
- [Offset](#) - To render the axis line with the specified distance from the Linear Gauge.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugeLine Height="150" Width="2" Color="#4286f4" Offset="2">
</LinearGaugeLine>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```



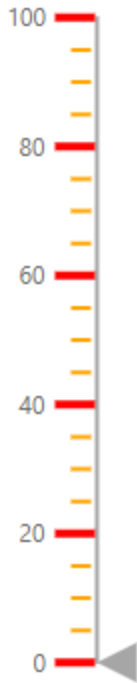
### Ticks Customization

Ticks are used to specify the interval in the axis. Ticks are of two types, major ticks and minor ticks. The following properties in the [LinearGaugeMajorTicks](#) and [LinearGaugeMinorTicks](#) classes can be used to customize the major ticks and minor ticks respectively.

- [Height](#) - To set the length of the major and minor ticks in pixel values.
- [Color](#) - To set the color of the major and minor ticks of the Linear Gauge.
- [Width](#) - To set the thickness of the major and minor ticks in pixel values.
- [Interval](#) - To set the interval for the major ticks and minor ticks in the Linear Gauge.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis Minimum=20 Maximum=140>
      <LinearGaugeMajorTicks Color="red" Interval="20" Height="20" Width="3">
      </LinearGaugeMajorTicks>
      <LinearGaugeMinorTicks Color="Orange" Interval="5" Height="10">
      </LinearGaugeMinorTicks>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
  <LinearGaugePointers>
    <LinearGaugePointer></LinearGaugePointer>
  </LinearGaugePointers>
</SfLinearGauge>
```



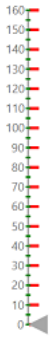
<!-- markdownlint-disable MD036 -->

#### Positioning the ticks

The minor and major ticks can be positioned by using the [Offset](#) and [Position](#) properties. The [Offset](#) is used to render the ticks with the specified distance from the axis. By default, the offset value is **0**. The possible values of the [Position](#) property are "[Inside](#)", "[Outside](#)", "[Cross](#)", and "[Auto](#)". By default, the ticks will be placed inside the axis.

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis Minimum="0" Maximum="160">
      <LinearGaugeMajorTicks Interval="10" Color="red" Height="10" Width="3"
        Position="Position.Outside">
      </LinearGaugeMajorTicks>
      <LinearGaugeMinorTicks Interval="5" Color="green" Height="5" Width="2"
        Position="Position.Cross">
      </LinearGaugeMinorTicks>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
  <LinearGaugePointers>
    <LinearGaugePointer></LinearGaugePointer>
  </LinearGaugePointers>
</SfLinearGauge>
```



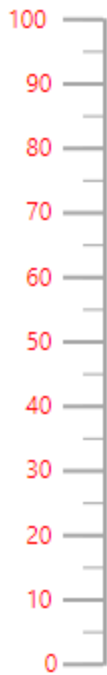
### Labels Customization

The style of the labels can be customized using the following properties in [LinearGaugeAxisLabelFont](#) class in the [LinearGaugeAxisLabelStyle](#).

- [Color](#) - To set the color of the axis label.
- [FontFamily](#) - To set the font family of the axis label.
- [FontStyle](#) - To set the font style of the axis label.
- [FontWeight](#) - To set the font weight of the axis label.
- [Opacity](#) - To set the opacity of the axis label.
- [Size](#) - To set the size of the axis label.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugeAxisLabelStyle>
        <LinearGaugeAxisLabelFont Color="red"></LinearGaugeAxisLabelFont>
      </LinearGaugeAxisLabelStyle>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```



<!-- markdownlint-disable MD036 -->

#### *Positioning the axis label*

Labels can be positioned by using [Offset](#) and [Position](#) properties in the [LinearGaugeAxisLabelStyle](#). The [Offset](#) defines the distance between the labels and ticks. By default, the offset value is **0**. The possible values of the [Position](#) property are "[Inside](#)", "[Outside](#)", "[Cross](#)", and "[Auto](#)". By default, the labels will be placed inside the axis.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugeAxisLabelStyle Offset="55" Position="Position.Cross">
      </LinearGaugeAxisLabelStyle>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```





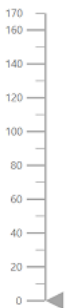
<!-- markdownlint-disable MD036 -->

#### *Customizing the display of the last label*

If the last label is not in the visible range, it will be hidden by default. The last label can be made visible by setting the [ShowLastLabel](#) property as **true** in the [LinearGaugeAxis](#) class.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis Minimum="0" Maximum="170" ShowLastLabel="true">
      <LinearGaugePointers>
        <LinearGaugePointer></LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```



<!-- markdownlint-disable MD036 -->

#### *Label Format*

Axis labels in the Linear Gauge control can be formatted using the [Format](#) property in the [LinearGaugeAxisLabelStyle](#) class. It is used to render the axis labels in a certain format or to add a user-defined unit in the label. It works with the help of placeholder like **{value}°C**, where **value** represents the axis value. For example, 20°C.

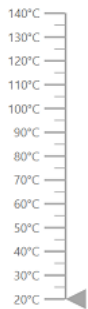
#### **ASPX-CS**

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis Minimum =20 Maximum =140>
```

```

<LinearGaugeAxisLabelStyle Format= "{value}°C">
</LinearGaugeAxisLabelStyle>
<LinearGaugePointers>
<LinearGaugePointer></LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>

```



#### Displaying numeric format in labels

The numeric formats such as currency, percentage, and so on can be displayed in the labels of the Linear Gauge using the [Format](#) property in the [SfLinearGauge](#) class. The following table describes the result of applying some commonly used label formats on numeric values.

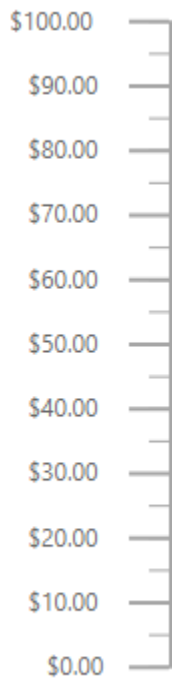
<!-- markdownlint-disable MD033 -->

Label Value	Label Format property value	Result	Description
1000	n1	1000.0	The number is rounded to 1 decimal place.
1000	n2	1000.00	The number is rounded to 2 decimal place.
1000	n3	1000.000	The number is rounded to 3 decimal place.
0.01	p1	1.0%	The number is converted to percentage with 1 decimal place.
0.01	p2	1.00%	The number is converted to percentage with 2 decimal place.
0.01	p3	1.000%	The number is converted to percentage with 3 decimal place.
1000	c1	\$1,000.0	The currency symbol is appended to number and number is rounded to 1 decimal place.
1000	c2	\$1,000.00	The currency symbol is appended to number and number is rounded to 2 decimal place.

<!-- markdownlint-disable MD036 -->

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge Format="c">
  <LinearGaugeAxes>
    <LinearGaugeAxis>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```



### Orientation

By default, the Linear Gauge is rendered vertically. To change its orientation, the [Orientation](#) property must be set to **Horizontal**.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge Orientation="Orientation.Horizontal">
  <LinearGaugeAxes>
    <LinearGaugeAxis Minimum="20" Maximum="140">
      <LinearGaugeMajorTicks Interval="10"></LinearGaugeMajorTicks>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```

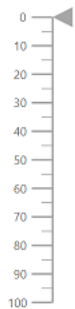


### Inverted Axis

The axis of the Linear Gauge component can be inverted by setting the [IsInversed](#) property to **true** in the [LinearGaugeAxis](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis IsInversed="true">
      <LinearGaugePointers>
        <LinearGaugePointer></LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```

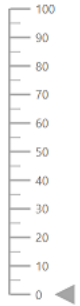


### Opposed Axis

To place an axis opposite from its original position, [OpposedPosition](#) property in the [LinearGaugeAxis](#) class must be set as **true**.

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis OpposedPosition="true">
      <LinearGaugePointers>
        <LinearGaugePointer></LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```

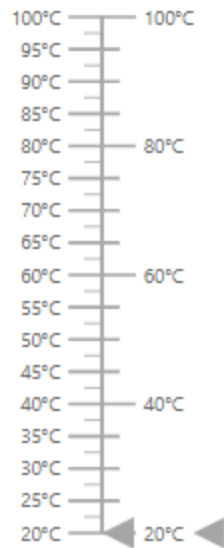


### Multiple Axes

Multiple axes can be added to the Linear Gauge by adding multiple [LinearGaugeAxis](#) classes in the [LinearGaugeAxes](#) class and customization can be done with the [LinearGaugeAxis](#) class. Each axis can be customized separately as shown in the following example.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis Minimum=20 Maximum=100>
      <LinearGaugeAxisLabelStyle Format="{value}°C">
      </LinearGaugeAxisLabelStyle>
      <LinearGaugePointers>
        <LinearGaugePointer PointerValue=20></LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
    <LinearGaugeAxis Minimum=20 Maximum=100 OpposedPosition=true>
      <LinearGaugeMajorTicks Interval="20" Height="20">
      </LinearGaugeMajorTicks>
      <LinearGaugeMinorTicks Interval="5" Height="10">
      </LinearGaugeMinorTicks>
      <LinearGaugeAxisLabelStyle Format="{value}°C">
      </LinearGaugeAxisLabelStyle>
      <LinearGaugePointers>
        <LinearGaugePointer PointerValue=20></LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```

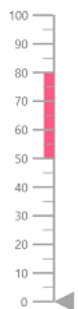


### Ranges in Blazor Linear Gauge Component

Range is the set of values in the axis. The range can be defined using the [Start](#) and [End](#) properties in the [LinearGaugeRange](#). Any number of ranges can be added to the Linear Gauge using the [LinearGaugeRanges](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis Minimum="0" Maximum="100">
      <LinearGaugeRanges>
        <LinearGaugeRange Start="50" End="80" StartWidth="20" EndWidth="20">
        </LinearGaugeRange>
      </LinearGaugeRanges>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
  <LinearGaugePointers>
    <LinearGaugePointer></LinearGaugePointer>
  </LinearGaugePointers>
</SfLinearGauge>
```



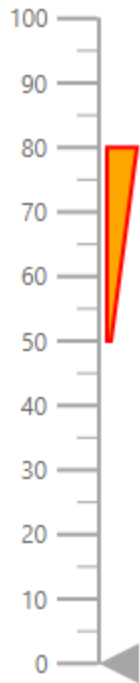
### Customizing the range

Ranges can be customized using the following properties in [LinearGaugeRange](#).

- [StartWidth](#) - Customize the range thickness at the start axis value.
- [EndWidth](#) - Customize the range thickness at the end axis value.
- [Color](#) - Customize the range color.
- [Position](#) - To place the range. By default, the range is placed outside of the axis. To change the position, this property can be set as "[Inside](#)", "[Outside](#)", "[Cross](#)", or "[Auto](#)".
- [Offset](#) - To place the range with specified distance from the axis.
- [LinearGaugeRangeBorder](#) - Customize color and width of range border.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAxes>
<LinearGaugeAxis Minimum="0" Maximum="100">
<LinearGaugeRanges>
<LinearGaugeRange Start="50" End="80" StartWidth="2" EndWidth="15"
Color="orange" Position="Position.Inside"
Offset="4">
<LinearGaugeRangeBorder Color="red" Width="2">
</LinearGaugeRangeBorder>
</LinearGaugeRange>
</LinearGaugeRanges>
<LinearGaugePointers>
<LinearGaugePointer></LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```



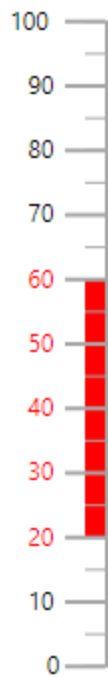
Setting the range color for the labels

To set the color of the labels like the range color, set the [UseRangeColor](#) property as **true** in the [LinearGaugeAxisLabelStyle](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugeAxisLabelStyle UseRangeColor="true">
      </LinearGaugeAxisLabelStyle>
    <LinearGaugeRanges>
      <LinearGaugeRange Start="20" End="60" Color="red">
      </LinearGaugeRange>
    </LinearGaugeRanges>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```



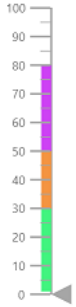


### Multiple ranges

Multiple ranges can be added to the Linear Gauge by adding collections of [LinearGaugeRange](#) in the [LinearGaugeRanges](#) and customization of ranges can be done with [LinearGaugeRange](#).

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugeRanges>
        <LinearGaugeRange Start="1" End="30" StartWidth="10" EndWidth="10"
          Color="#41f47f">
        </LinearGaugeRange>
        <LinearGaugeRange Start="30" End="50" StartWidth="10" EndWidth="10"
          Color="#f49441">
        </LinearGaugeRange>
        <LinearGaugeRange Start="50" End="80" StartWidth="10" EndWidth="10"
          Color="#cd41f4">
        </LinearGaugeRange>
      </LinearGaugeRanges>
      <LinearGaugePointers>
        <LinearGaugePointer></LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```



### Gradient Color

Gradient support allows the addition of multiple colors in the range. The following gradient types are supported in the Linear Gauge.

- Linear Gradient
- Radial Gradient

### Linear Gradient

Using linear-gradient, colors will be applied in a linear progression. The start value of the linear gradient can be set using the [StartValue](#) property. The end value of the linear gradient will be set using the [EndValue](#) property. The color stop values such as [Color](#), [Opacity](#), and [Offset](#) to be defined in [ColorStop](#).

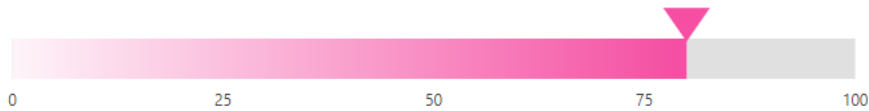
### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge Orientation="Orientation.Horizontal">
  <LinearGaugeContainer Width="30" Offset="30">
    <LinearGaugeContainerBorder Width="0" />
    <LinearGaugeAxes>
      <LinearGaugeAxis>
        <LinearGaugeAxisLabelStyle Offset="55">
          <LinearGaugeAxisLabelFont Color="#424242" />
        </LinearGaugeAxisLabelStyle>
      </LinearGaugeAxis>
    </LinearGaugeAxes>
    <LinearGaugeLine Width="0" />
    <LinearGaugeMajorTicks Height="0" Interval="25" />
    <LinearGaugeMinorTicks Height="0" />
    <LinearGaugePointers>
      <LinearGaugePointer PointerValue="80" Height="25" Width="35"
        Color="#f54ea2" Offset="-40" MarkerType="MarkerType.Triangle"
        Placement="Syncfusion.Blazor.LinearGauge.Placement.Near">
      </LinearGaugePointer>
    </LinearGaugePointers>
    <LinearGaugeRanges>
      <LinearGaugeRange Color="#f54ea2" Start="0" End="80"
        StartWidth="30" EndWidth="30" Offset="30">
      <LinearGradient StartValue="1%" EndValue="99%">
        <ColorStops>
          <ColorStop Opacity="1" Offset="0%" Color="#fef3f9">
          </ColorStop>
          <ColorStop Opacity="1" Offset="100%" Color="#f54ea2">
          </ColorStop>
        </ColorStops>
      </LinearGradient>
    </LinearGaugeRange>
  </LinearGaugeContainer>
</SfLinearGauge>
```

```

</LinearGaugeRange>
</LinearGaugeRanges>
</LinearGaugeAxis>
</LinearGaugeAxes>
</LinearGaugeContainer>
</SfLinearGauge>

```



### 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](#) class. The outer circle position of the radial gradient can be set using the [OuterPosition](#) class. The color stop values such as [Color](#), [Opacity](#), and [Offset](#) to be defined in [ColorStop](#).

### ASPX-CS

```

@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge Orientation="Orientation.Horizontal">
<LinearGaugeContainer Width="30" Offset="30">
<LinearGaugeContainerBorder Width="0" />
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugeAxisLabelStyle Offset="55">
<LinearGaugeAxisLabelFont Color="#424242" />
</LinearGaugeAxisLabelStyle>
<LinearGaugeLine Width="0" />
<LinearGaugeMajorTicks Height="0" Interval="25" />
<LinearGaugeMinorTicks Height="0" />
<LinearGaugePointers>
<LinearGaugePointer PointerValue="80" Height="25" Width="35"
Color="#f54ea2" Offset="-40" MarkerType="MarkerType.Triangle"
Placement="Syncfusion.Blazor.LinearGauge.Placement.Near">
</LinearGaugePointer>
</LinearGaugePointers>
<LinearGaugeRanges>
<LinearGaugeRange Color="#f54ea2" Start="0" End="80"
StartWidth="30" EndWidth="30" Offset="30">
<RadialGradient Radius="65%">
<InnerPosition X="50%" Y="70%"></InnerPosition>
<OuterPosition X="60%" Y="60%"></OuterPosition>
<ColorStops>
<ColorStop Opacity="0.9" Color=" #fff5f5" Offset="5%">
</ColorStop>
<ColorStop Opacity="1" Color="#f54ea2" Offset="99%">
</ColorStop>
</ColorStops>
</RadialGradient>
</LinearGaugeRange>
</LinearGaugeRanges>
</LinearGaugeAxis>

```

```

</LinearGaugeAxes>
</LinearGaugeContainer>
</SfLinearGauge>

```



If we set both gradients for the range, only the linear gradient gets rendered. If we set the [StartValue](#) and [EndValue](#) of the [LinearGradient](#) as empty strings, then the radial gradient gets rendered in the range of the Linear Gauge.

### Pointers in Blazor Linear Gauge Component

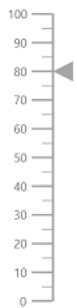
Pointers are used to indicate values on an axis. The value of the pointer can be modified using the [PointerValue](#) property in [LinearGaugePointer](#).

#### ASPX-CS

```

@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer PointerValue="80">
</LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>

```



### Types of pointer

The Linear Gauge supports the following types of pointers:

- Bar
- Marker

The type of pointer can be modified by using the [Type](#) property in [LinearGaugePointer](#).

#### Marker pointer

A marker pointer is a shape that can be used to mark the pointer value in the Linear Gauge.

#### <b>Types of marker shapes</b>

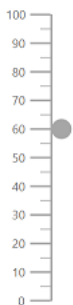
By default, the marker shape for the pointer is **InvertedTriangle**. To change the shape of the pointer, use the [MarkerType](#) property in [LinearGaugePointer](#). The following marker types are available in Linear Gauge.

- Circle
- Rectangle
- Triangle
- InvertedTriangle
- Diamond
- Image

An image can be rendered instead of rendering a shape as a pointer. It can be achieved by setting the [MarkerType](#) property to **Image** and setting the source URL of image to [ImageUrl](#) property in [LinearGaugePointer](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer PointerValue="60" MarkerType="MarkerType.Circle">
</LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```



#### <b>Marker pointer customization</b>

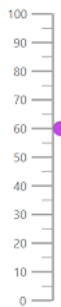
The marker pointer can be customized using the following properties and component.

- [Height](#) - To set the height of the pointer.

- [Position](#) - The position of the pointer can be changed by setting the value as "[Inside](#)", "[Outside](#)", "[Cross](#)", or "[Auto](#)".
- [Width](#) - To set the width of the pointer.
- [Color](#) - To set the color of the pointer.
- [Placement](#) - To place the pointer in the specified position. By default, the pointer is placed "[Far](#)" from the axis. To change the placement, set the [Placement](#) property as "[Near](#)", "[Center](#)", or "[None](#)".
- [Offset](#) - To place the pointer with specified distance from the axis.
- [Opacity](#) - To set the opacity of the pointer.
- [AnimationDuration](#) - To specify the duration of the animation in pointer.
- [LinearGaugePointerBorder](#) - To set the color and width for the border of the pointer.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer PointerValue="60" MarkerType="MarkerType.Circle"
Height="15" Width="15" Color="#cd41f4"
Position="Position.Outside">
</LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```



### Bar pointer

The bar pointer is used to track the axis value. The bar pointer starts from the beginning of the gauge and ends at the pointer value. To enable bar pointer, set the [Type](#) property in [LinearGaugePointer](#) as [Bar](#).

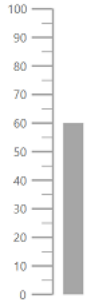
### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer PointerValue="60" Type="Point.Bar"
Color="#a6a6a6">
</LinearGaugePointer>
```

```

</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>

```



### <b>Bar pointer customization</b>

The bar pointer can be customized using the following properties and component.

- [Width](#) - To set the thickness of the bar pointer.
- [Color](#) - To set the color of the bar pointer.
- [Offset](#) - To place the bar pointer with the specified distance from it's default position.
- [Opacity](#) - To set the opacity of the bar pointer.
- [RoundedCornerRadius](#) - To set the corner radius of the bar pointer.
- [LinearGaugePointerBorder](#) - To set the color and width for the border of the pointer.
- [AnimationDuration](#) - To set the duration of the animation in bar pointer.

---

The Placement property is not applicable for the bar pointer.

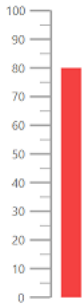
---

### ASPX-CS

```

@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer PointerValue="80" Type="Point.Bar" Width="20"
Color="#f44141">
</LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>

```



### Multiple pointers

Multiple pointers can be added to the Linear Gauge by adding multiple [LinearGaugePointer](#) in the [LinearGaugePointers](#) and customization for the pointers can be done with [LinearGaugePointer](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugePointers>
        <LinearGaugePointer PointerValue="30" MarkerType="MarkerType.Circle">
        </LinearGaugePointer>
        <LinearGaugePointer PointerValue="60" MarkerType="MarkerType.Diamond">
        </LinearGaugePointer>
        <LinearGaugePointer PointerValue="80" AnimationDuration="1000">
        </LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```



### Pointer animation

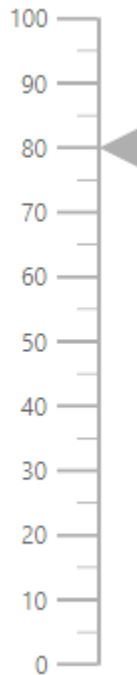
Pointer is animated on loading the gauge. This can be handled using the [AnimationDuration](#) property. The duration of the animation can be specified in milliseconds.

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugePointers>
```



```
<LinearGaugePointer PointerValue="80" AnimationDuration="1000">
</LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```



### Gradient Color

Gradient support allows the addition of multiple colors in the pointers of the Linear Gauge. The following gradient types are supported in the Linear Gauge.

- Linear Gradient
- Radial Gradient

#### Linear Gradient

Using linear gradient, colors will be applied in a linear progression. The start value of the linear gradient can be set using the [StartValue](#) property. The end value of the linear gradient will be set using the [EndValue](#) property. The color stop values such as [Color](#), [Opacity](#), and [Offset](#) are set using [ColorStop](#) property.

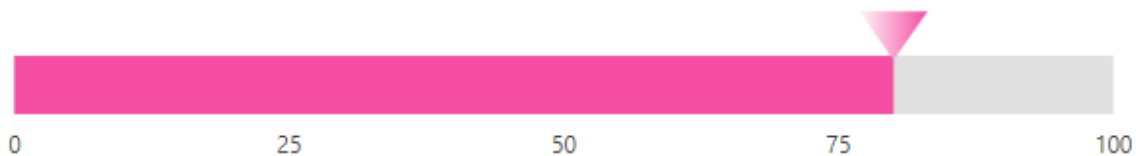
#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge Orientation="Orientation.Horizontal">
<LinearGaugeContainer Width="30" Offset="30">
<LinearGaugeContainerBorder Width="0" />
<LinearGaugeAxes>
<LinearGaugeAxis>
```

```

<LinearGaugeAxisLabelStyle Offset="55">
<LinearGaugeAxisLabelFont Color="#424242" />
</LinearGaugeAxisLabelStyle>
<LinearGaugeLine Width="0.01" />
<LinearGaugeMajorTicks Height="0.01" Interval="25" />
<LinearGaugeMinorTicks Height="0.01" />
<LinearGaugePointers>
<LinearGaugePointer PointerValue="80" Height="25" Width="35"
Offset="-40" MarkerType="MarkerType.Triangle"
Placement="Syncfusion.Blazor.LinearGauge.Placement.Near">
<LinearGradient StartValue="1%" EndValue="99%">
<ColorStops>
<ColorStop Opacity="1" Color= "#fef3f9" Offset="1%">
</ColorStop>
<ColorStop Opacity="1" Color= "#f54ea2" Offset="100%">
</ColorStop>
</ColorStops>
</LinearGradient>
</LinearGaugePointer>
</LinearGaugePointers>
<LinearGaugeRanges>
<LinearGaugeRange Color="#f54ea2" Start="0" End="80"
StartWidth="30" EndWidth="30" Offset="30">
</LinearGaugeRange>
</LinearGaugeRanges>
</LinearGaugeAxis>
</LinearGaugeAxes>
</LinearGaugeContainer>
</SfLinearGauge>

```



### 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](#). The outer circle position of the radial gradient can be set using the [OuterPosition](#). The color stop values such as [Color](#), [Opacity](#), and [Offset](#) are set using [ColorStop](#).

### ASPX-CS

```

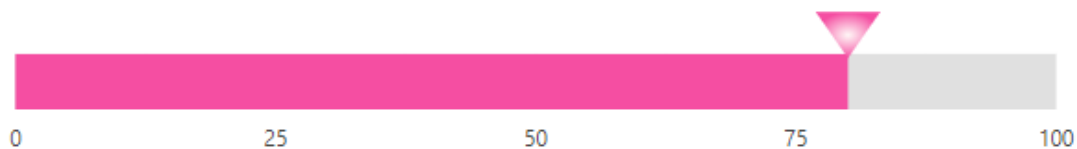
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge Orientation="Orientation.Horizontal">
<LinearGaugeContainer Width="30" Offset="30">
<LinearGaugeContainerBorder Width="0" />
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugeAxisLabelStyle Offset="55">
<LinearGaugeAxisLabelFont Color="#424242" />

```

```

</LinearGaugeAxisLabelStyle>
<LinearGaugeLine Width="0.01" />
<LinearGaugeMajorTicks Height="0.01" Interval="25" />
<LinearGaugeMinorTicks Height="0.01" />
<LinearGaugePointers>
<LinearGaugePointer PointerValue="80" Height="25" Width="35"
Offset="-40" MarkerType="MarkerType.Triangle"
Placement="Syncfusion.Blazor.LinearGauge.Placement.Near">
<RadialGradient Radius="60%">
<InnerPosition X="50%" Y="50%"></InnerPosition>
<OuterPosition X="50%" Y="50%"></OuterPosition>
<ColorStops>
<ColorStop Opacity="0.9" Color="#fff5f5" Offset="1%">
</ColorStop>
<ColorStop Opacity="1" Color="#f54ea2" Offset="99%">
</ColorStop>
</ColorStops>
</RadialGradient>
</LinearGaugePointer>
</LinearGaugePointers>
<LinearGaugeRanges>
<LinearGaugeRange Color="#f54ea2" Start="0" End="80"
StartWidth="30" EndWidth="30" Offset="30">
</LinearGaugeRange>
</LinearGaugeRanges>
</LinearGaugeAxis>
</LinearGaugeAxes>
</LinearGaugeContainer>
</SfLinearGauge>

```



If we set both gradients, only the linear gradient gets rendered. If we set the [StartValue](#) and [EndValue](#) of the [LinearGradient](#) as empty strings, then the radial gradient gets rendered in the pointer of the Linear Gauge.

## Annotations in Blazor Linear Gauge Component

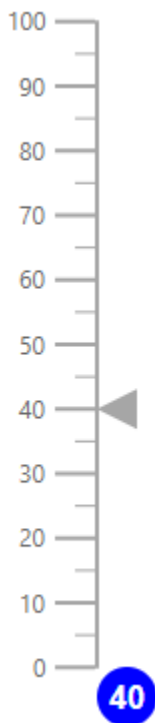
Annotations are used to mark the specific area of interest in the Linear Gauge with text, HTML elements, or images. Any number of annotations can be added to the Linear Gauge component.

### Adding annotation

To render the custom HTML elements in the Linear Gauge component, use the [ContentTemplate](#) property in the [LinearGaugeAnnotation](#). The [Content](#) property in [LinearGaugeAnnotation](#) can be used to render the element as an annotation in the Linear Gauge.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAnnotations>
    <LinearGaugeAnnotation AxisValue="0" ZIndex="1">
      <ContentTemplate>
        <div class="custom-annotation">40</div>
      </ContentTemplate>
    </LinearGaugeAnnotation>
  </LinearGaugeAnnotations>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugePointers>
        <LinearGaugePointer PointerValue="40"></LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
<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>
```



### Customization

The following properties are used to customize the annotation.

- [ZIndex](#) - Bring the annotation to the front or back, when annotation overlaps with another element.
- [AxisValue](#) - To place the annotation in the specified axis value with respect to the provided axis index.
- [AxisIndex](#) - To place the annotation in the specified axis with respect to the provided axis value.
- [HorizontalAlignment](#) - To place the annotation horizontally.
- [VerticalAlignment](#) - To place the annotation vertically.
- [X, Y](#) - To place the annotation in the specified location.

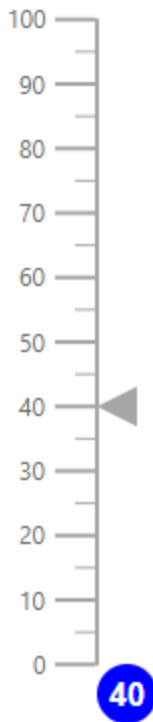
<!-- markdownlint-disable MD036 -->

#### Changing the z-index

To change the stack order of an annotation element, the [ZIndex](#) property of the [LinearGaugeAnnotation](#) can be used.

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAnnotations>
<LinearGaugeAnnotation AxisValue="0" ZIndex="1">
<ContentTemplate>
<div class="custom-annotation">40</div>
</ContentTemplate>
</LinearGaugeAnnotation>
</LinearGaugeAnnotations>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer PointerValue="40"></LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
<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>
```



<!-- markdownlint-disable MD036 -->

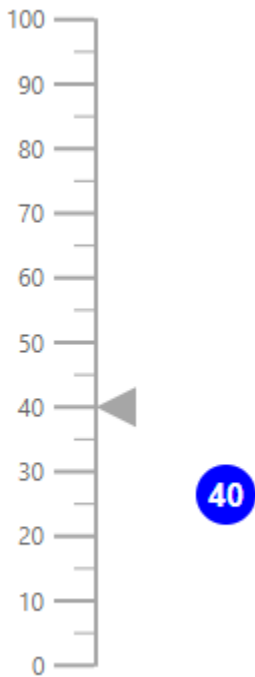
#### *Positioning an annotation*

The annotation can be placed anywhere in the Linear Gauge by setting the pixel value to the [X](#) and [Y](#) properties in the [LinearGaugeAnnotation](#).

#### **ASPX-CS**

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAnnotations>
    <LinearGaugeAnnotation AxisValue="0" ZIndex="1" X="50" Y="-100">
      <ContentTemplate>
        <div class="custom-annotation">40</div>
      </ContentTemplate>
    </LinearGaugeAnnotation>
  </LinearGaugeAnnotations>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugePointers>
        <LinearGaugePointer PointerValue="40"></LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
<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>
```



#### Alignment of annotation

The annotation can be aligned horizontally and vertically by using the [HorizontalAlignment](#) and [VerticalAlignment](#) properties respectively. The possible values can be [Center](#), [Far](#), [Near](#), and [None](#). The [HorizontalAlignment](#) and [VerticalAlignment](#) properties are not applicable when the [X](#) and [Y](#) properties are set in the [LinearGaugeAnnotation](#).

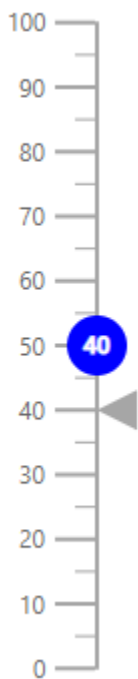
#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAnnotations>
    <LinearGaugeAnnotation ZIndex="1"
      HorizontalAlignment="Placement.Center"
      VerticalAlignment="Placement.Center">
      <ContentTemplate>
        <div class="custom-annotation">40</div>
      </ContentTemplate>
    </LinearGaugeAnnotation>
  </LinearGaugeAnnotations>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugePointers>
        <LinearGaugePointer PointerValue="40"></LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
```

```

</LinearGaugeAxes>
</SfLinearGauge>
<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

Multiple annotations can be added to the Linear Gauge component by adding the multiple [LinearGaugeAnnotation](#) in the [LinearGaugeAnnotations](#) and customization for the annotation can be done with the [LinearGaugeAnnotation](#).

### ASPX-CS

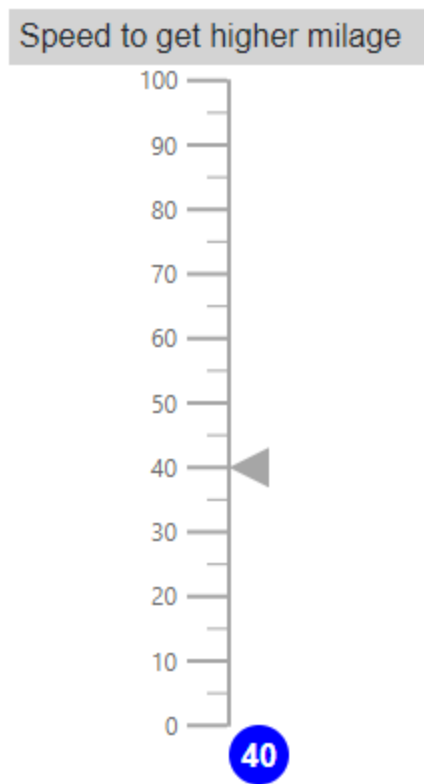
```

@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAnnotations>
<LinearGaugeAnnotation ZIndex="1" AxisValue="100"
X="-110" Y="-35">
<ContentTemplate>
<div class="custom-annotation">Speed to get higher milage</div>
</ContentTemplate>
</LinearGaugeAnnotation>
<LinearGaugeAnnotation AxisValue="0" ZIndex="1">

```



```
<ContentTemplate>
<div class="speed">40</div>
</ContentTemplate>
</LinearGaugeAnnotation>
</LinearGaugeAnnotations>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer PointerValue="40"></LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
<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: 210px;
padding: 2px 5px;
}
</style>
```



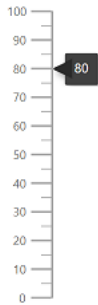
## User Interaction in Blazor Linear Gauge Component

### Tooltip

Linear Gauge displays the details about a pointer value through [LinearGaugeTooltipSettings](#), when the mouse hovers over the pointer. To enable the tooltip, set [Enable](#) property as **true**.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugeTooltipSettings Enable="true">
      </LinearGaugeTooltipSettings>
    <LinearGaugePointers>
      <LinearGaugePointer PointerValue="80">
      </LinearGaugePointer>
    </LinearGaugePointers>
  </LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```

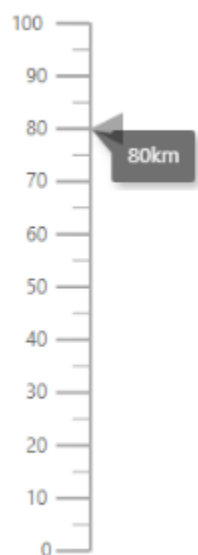


### Tooltip format

Tooltip in the Linear Gauge control can be formatted using the [Format](#) property in [LinearGaugeTooltipSettings](#). It is used to render the tooltip in certain format or to add a user-defined unit in the tooltip. By default, the tooltip shows the pointer value only. In addition to that, more information can be added in the tooltip. For example, the format **{value}km** shows pointer value with kilometer unit in the tooltip.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugeTooltipSettings Enable="true" Format="{value}km">
      </LinearGaugeTooltipSettings>
      <LinearGaugePointers>
        <LinearGaugePointer PointerValue="80">
        </LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```

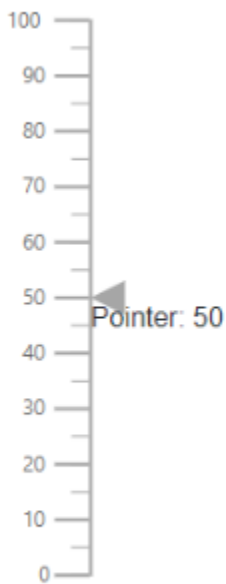


### Tooltip Template

The HTML element can be rendered in the tooltip of the Linear Gauge using the [TooltipTemplate](#) in the [LinearGaugeTooltipSettings](#).

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeTooltipSettings Enable="true">
<TooltipTemplate>
<div style="height:100px;width:100px;">Pointer: 80</div>
</TooltipTemplate>
</LinearGaugeTooltipSettings>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer PointerValue="80">
</LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```



### Customize the appearance of the tooltip

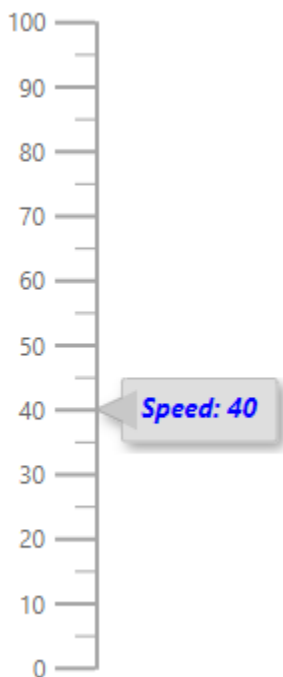
The tooltip can be customized using the following properties in [LinearGaugeTooltipSettings](#).

- [Fill](#) - To fill the color for tooltip.
- [EnableAnimation](#) - To enable or disable the tooltip animation.
- [LinearGaugeTooltipBorder](#) - To set the border color and width of the tooltip.
- [LinearGaugeTooltipTextStyle](#) - To customize the style of the text in tooltip.

- [ShowAtMousePosition](#) - To show the tooltip at the mouse position.

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugeTooltipSettings Enable="true" Format="Speed: {value}"
Fill="lightgray" EnableAnimation="true">
<LinearGaugeTooltipBorder Color="darkgray" Width="1">
</LinearGaugeTooltipBorder>
<LinearGaugeTooltipTextStyle Color="blue" FontStyle="italic"
FontWeight="bold">
</LinearGaugeTooltipTextStyle>
</LinearGaugeTooltipSettings>
<LinearGaugePointers>
<LinearGaugePointer PointerValue="40">
</LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```



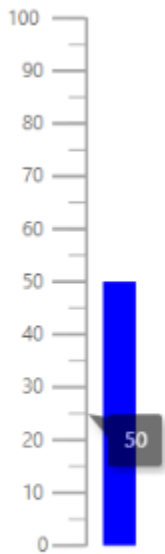
#### Positioning the tooltip

The tooltip is positioned at the [End](#) of the pointer. To change the position of the tooltip at the start, or center of the pointer, set the [Position](#) property to [Start](#) or [Center](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
```

```
<LinearGaugeTooltipSettings Enable="true" Position="TooltipPosition.Center">
</LinearGaugeTooltipSettings>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer PointerValue="50" Type="Point.Bar" Color="blue">
</LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```

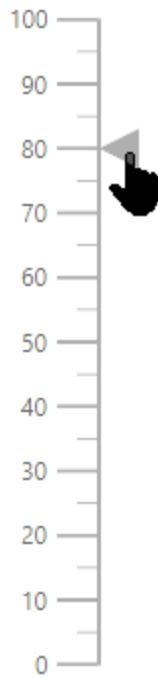


### Pointer drag

To drag either marker or bar pointer to the desired axis value, set the [EnableDrag](#) property as **true** in the [LinearGaugePointer](#).

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer PointerValue="80" EnableDrag="true">
</LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
```



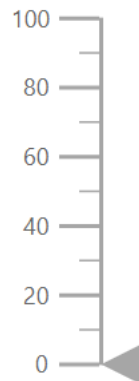
## Print And Export in Blazor Linear Gauge Component

### Print

The rendered Linear Gauge can be printed directly from the browser by calling the [PrintAsync](#) method. To use the print functionality, set the [AllowPrint](#) property as **true**.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<button @onclick="PrintGauge">Print</button>
<SfLinearGauge @ref="gauge" AllowPrint="true">
  <LinearGaugeAxes>
    <LinearGaugeAxis Minimum="0" Maximum="100">
      <LinearGaugeMajorTicks Interval="20"></LinearGaugeMajorTicks>
      <LinearGaugeMinorTicks Interval="10"></LinearGaugeMinorTicks>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
  <LinearGaugePointers>
    <LinearGaugePointer>
    </LinearGaugePointer>
  </LinearGaugePointers>
</SfLinearGauge>
@code {
  SfLinearGauge gauge;
  public async Task PrintGauge()
  {
    await this.gauge.PrintAsync();
  }
}
```



## Export

### Image Export

To use the image export functionality, set the [AllowImageExport](#) property as **true**. The rendered Linear Gauge can be exported as an image using the [ExportAsync](#) method. This method requires two parameters: export type and file name. The Linear Gauge can be exported as an image with the following formats.

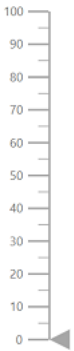
- JPEG
- PNG
- SVG

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<button @onclick="ExportGauge">Export</button>
<SfLinearGauge @ref="gauge" AllowImageExport="true">
  <LinearGaugeAxes>
    <LinearGaugeAxis Minimum="0" Maximum="100">
      <LinearGaugeMajorTicks Interval="20"></LinearGaugeMajorTicks>
      <LinearGaugeMinorTicks Interval="10"></LinearGaugeMinorTicks>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
@code {
  SfLinearGauge gauge;
  public async Task ExportGauge()
  {
    await this.gauge.ExportAsync(ExportType.PNG, "LinearGauge");
  }
}
```



Export



### PDF Export

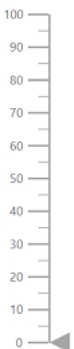
To use the PDF export functionality, set the [AllowPdfExport](#) property as **true**. The rendered Linear Gauge can be exported as PDF using the [ExportAsync](#) method. The [ExportAsync](#) method requires three parameters: export type, file name, and orientation of the PDF document. The orientation of the PDF document can be set as **Portrait** or **Landscape**.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<button @onclick="ExportGauge">Export</button>
<SfLinearGauge @ref="gauge" AllowPdfExport="true">
  <LinearGaugeAxes>
    <LinearGaugeAxis Minimum="0" Maximum="100">
      <LinearGaugeMajorTicks Interval="20"></LinearGaugeMajorTicks>
      <LinearGaugeMinorTicks Interval="10"></LinearGaugeMinorTicks>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>

@code {
  SfLinearGauge gauge;
  public async Task ExportGauge()
  {
    await this.gauge.ExportAsync(ExportType.PDF, "LinearGauge");
  }
}
```

Export



*Exporting Linear Gauge as base64 string of the file*

The Linear Gauge can be exported as base64 string for the JPEG, PNG and PDF formats. The rendered Linear Gauge can be exported as base64 string of the exported image or PDF document using the [ExportAsync](#) method. The arguments that are required for this method is export type, file name, orientation of the exported PDF document and **allowDownload** boolean value that is set as **false** to return base64 string. The value for the orientation of the exported PDF document is set as **null** for image export and **Portrait** or **Landscape** for the PDF document.

**ASPX-CS**

```
@using Syncfusion.Blazor.LinearGauge
<button @onclick="ExportGauge">Export</button>
<SfLinearGauge @ref="gauge" AllowImageExport="true">
  <LinearGaugeAxes>
    <LinearGaugeAxis Minimum="0" Maximum="100">
      <LinearGaugeMajorTicks Interval="20"></LinearGaugeMajorTicks>
      <LinearGaugeMinorTicks Interval="10"></LinearGaugeMinorTicks>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
@code {
  SfLinearGauge gauge;
  public async Task ExportGauge()
  {
    string exportString = await this.gauge.ExportAsync(ExportType.PNG,
    "LinearGauge", null, false);
    Console.WriteLine(exportString);
  }
}
```

---

The exporting of the Linear Gauge as base64 string is not applicable for the SVG format.

---

## Appearance in Blazor Linear Gauge Component

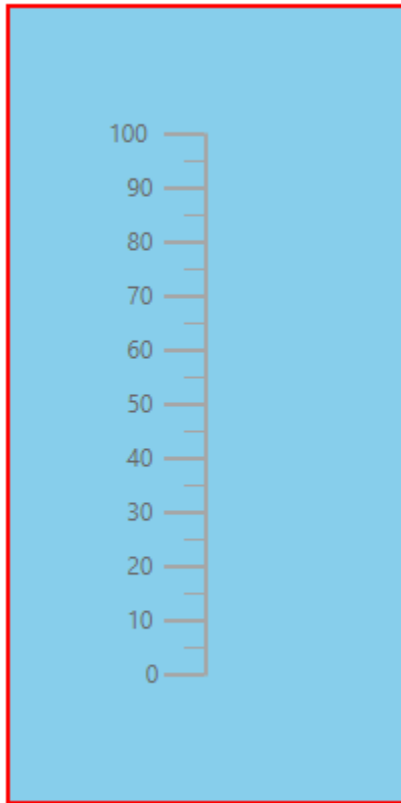
## Customizing the Linear Gauge area

The following property and components are available in the [SfLinearGauge](#) to customize the Linear Gauge area.

- [Background](#) - Applies the background color for the Linear gauge.
- [LinearGaugeBorder](#) - To customize the color and width of the border in Linear Gauge.
- [LinearGaugeMargin](#) - To customize the margins of the Linear Gauge.

**ASPX-CS**

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge Width="200px" Height="400px" Background="skyblue">
  <LinearGaugeBorder Color="#FF0000" Width="2"></LinearGaugeBorder>
  <LinearGaugeMargin Left="20" Right="20" Top="20"
  Bottom="20"></LinearGaugeMargin>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      </LinearGaugeAxis>
    </LinearGaugeAxes>
  </SfLinearGauge>
```



#### Setting up the Linear Gauge title

The title for the Linear Gauge can be set using [Title](#) property in [SfLinearGauge](#). Its appearance can be customized using the [LinearGaugeTitleStyle](#) with the below properties.

- [Color](#) - Specifies the text color of the title.
- [FontStyle](#) - Specifies the font style of the title.
- [FontWeight](#) - Specifies the font weight of the title.
- [Size](#) - Specifies the font size of the title.
- [Opacity](#) - Specifies the opacity of the title.
- [FontFamily](#) - Specifies the font family of the title.

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge Title="Linear Gauge">
  <LinearGaugeTitleStyle FontFamily="Arial" FontWeight="regular"
  FontStyle="italic" Color="#E27F2D" Size="23px">
  </LinearGaugeTitleStyle>
  <LinearGaugeAxes>
  <LinearGaugeAxis>
  </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```

## Linear Gauge



### Customizing the Linear Gauge container

The area used to render the ranges and pointers at the center position of the gauge is called container. The following types of container to be applicable for Linear Gauge.

- Normal
- Rounded Rectangle
- Thermometer

The type of the container can be modified by using the [Type](#) property in [LinearGaugeContainer](#). The container can be customized by using the following properties and component in [LinearGaugeContainer](#),

- [Offset](#) - To place the container with the specified distance from the axis of the Linear Gauge.
- [Width](#) - To set the thickness of the container.
- [Height](#) - To set the length of the container.
- [BackgroundColor](#) - To set the background color of the container.
- [LinearGaugeContainerBorder](#) - To set the color and width for the border of the container.

### Normal

The [Normal](#) type will render the container as a rectangle and this is the default container type.

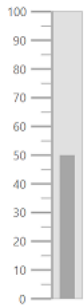
### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge  
<SfLinearGauge>
```

```

<LinearGaugeContainer Width="30">
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugePointers>
        <LinearGaugePointer PointerValue="50" Width="15" Type="Point.Bar"
          Color="#a6a6a6">
        </LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</LinearGaugeContainer>
</SfLinearGauge>

```



#### *Rounded Rectangle*

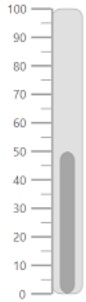
The [RoundedRectangle](#) type will render the container as a rectangle with rounded corner radius. The rounded corner radius of the container can be customized using the [RoundedCornerRadius](#) property in [LinearGaugeContainer](#).

#### **ASPX-CS**

```

@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeContainer Width="30" Type="ContainerType.RoundedRectangle">
    <LinearGaugeAxes>
      <LinearGaugeAxis>
        <LinearGaugePointers>
          <LinearGaugePointer PointerValue="50" Width="15" Type="Point.Bar"
            Color="#a6a6a6">
          </LinearGaugePointer>
        </LinearGaugePointers>
      </LinearGaugeAxis>
    </LinearGaugeAxes>
  </LinearGaugeContainer>
</SfLinearGauge>

```

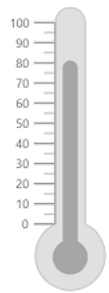


### Thermometer

The [Thermometer](#) type will render the container similar to the appearance of thermometer.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
  <LinearGaugeContainer Width="30" Type="ContainerType.Thermometer">
    <LinearGaugeAxes>
      <LinearGaugeAxis>
        <LinearGaugePointers>
          <LinearGaugePointer PointerValue="80" Width="15" Type="Point.Bar"
            Color="#a6a6a6">
        </LinearGaugePointer>
        </LinearGaugePointers>
      </LinearGaugeAxis>
    </LinearGaugeAxes>
  </LinearGaugeContainer>
</SfLinearGauge>
```



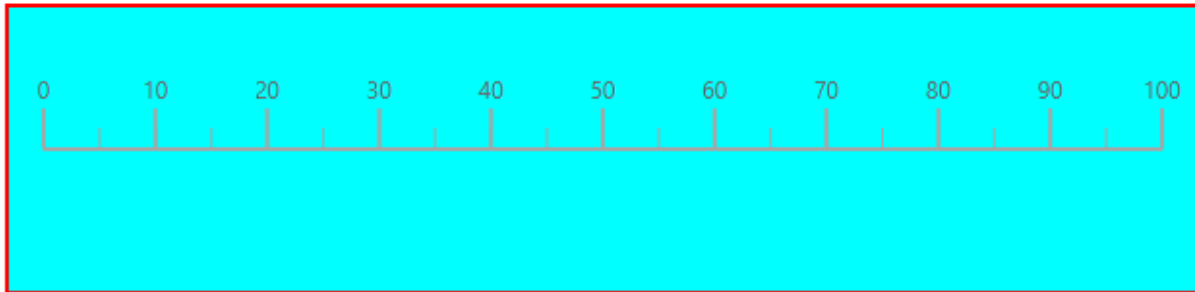
### Fitting the Linear Gauge to the control

The Linear Gauge component is rendered with margin by default. To remove the margin around the Linear Gauge, the [AllowMargin](#) property in [SfLinearGauge](#) is set as **false**.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge AllowMargin="false" Width="100%" Height="100%"
  Orientation="Orientation.Horizontal" Background="#04fbfb">
  <LinearGaugeBorder Color="#FF0000" Width="2"></LinearGaugeBorder>
  <LinearGaugeMargin Left="0" Right="0" Top="0"
    Bottom="0"></LinearGaugeMargin>
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      </LinearGaugeAxis>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
```

```
</LinearGaugeAxes>  
</SfLinearGauge>
```



To use this feature, set the [AllowMargin](#) property to **false**, the [Width](#) property to **100%** and the properties of [LinearGaugeMargin](#) to **0**.

### Accessibility in Blazor Linear Gauge Component

Linear Gauge provides built-in compliance with the [WAI-ARIA](#) specifications. The WAI-ARIA accessibility support is achieved through the attribute like `aria-label` in the SVG element. It helps to provide information about elements in a document for assistive technology. This attribute sets the text label with some default descriptions for the following elements in the Linear Gauge.

<!-- markdownlint-disable MD033 -->

Element	Default description
Gauge title	Specifies the title of the Linear gauge.
Pointer value	Specifies the value of the pointer in the Linear gauge.

To change this default description, use the [Description](#) property available in [LinearGaugePointer](#) and the [SfLinearGauge](#). It helps the screen reader to read for an assistive purpose.

### Globalization in Blazor Linear Gauge Component

The localization allows to localize the default text content in the Blazor component. For more information about localization, refer [here](#).

#### Globalization

Globalization is the process of designing and developing a component that works in different cultures. Internationalization is used to globalize the number content in Linear Gauge component using [Format](#) property in [SfLinearGauge](#). It has static text on some features such as

- Axis label
- Tooltip

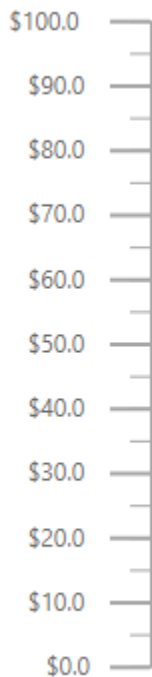
The static text on above features can be changed to any culture such as Arabic, Deutsch and French.

### Numeric Format

The text in axis labels and tooltip can be displayed in the numeric format such as currency, percentage and so on. To know more about the numeric formats in axis labels, refer [here](#). In the below example, the axis label is displayed in the currency format.

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge Format="c">
  <LinearGaugeAxes>
    <LinearGaugeAxis>
      <LinearGaugePointers>
        <LinearGaugePointer></LinearGaugePointer>
      </LinearGaugePointers>
    </LinearGaugeAxis>
  </LinearGaugeAxes>
</SfLinearGauge>
```



### Events in Blazor Linear Gauge Component

This section describes the Linear Gauge component's event that gets triggered when corresponding operations are performed. The events should be provided to the Linear Gauge by using the [LinearGaugeEvents](#).

#### AnnotationRendering

Before the annotation is rendered in the Linear Gauge, the [AnnotationRendering](#) event will be triggered. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS



```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeEvents
AnnotationRendering="AnnotationRender"></LinearGaugeEvents>
<LinearGaugeAnnotations>
<LinearGaugeAnnotation AxisValue="0" ZIndex="1" Content="40">
</LinearGaugeAnnotation>
</LinearGaugeAnnotations>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer Value="40"></LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
@code {
public void AnnotationRender(AnnotationRenderEventArgs args)
{
// Code here
}
}
```

### AxisLabelRendering

Before each axis label is rendered in the Linear Gauge, the [AxisLabelRendering](#) event is fired. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeEvents AxisLabelRendering="LabelRender"></LinearGaugeEvents>
<LinearGaugeAxes>
<LinearGaugeAxis>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
@code {
public void LabelRender(AxisLabelRenderEventArgs args)
{
// Code here
}
}
```

### Loaded

After the Linear Gauge has been loaded, the [Loaded](#) event will be triggered. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeEvents Loaded="Loaded"></LinearGaugeEvents>
<LinearGaugeAxes>
<LinearGaugeAxis>
```

```
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
@code {
public void Loaded(LoadedEventArgs args)
{
    // Code here
}
}
```

### OnDragEnd

The [OnDragEnd](#) event will be fired before the pointer drag is completed. To know more about the argument of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeEvents OnDragStart="DragEnd"></LinearGaugeEvents>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer Value="40" EnableDrag="true"></LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
@code {
public void DragEnd(PointerDragEventArgs args)
{
    // Code here
}
}
```

### OnDragStart

When the pointer drag begins, the [OnDragStart](#) event is triggered. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeEvents OnDragStart="DragStart"></LinearGaugeEvents>
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer Value="40" EnableDrag="true"></LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
@code {
public void DragStart(PointerDragEventArgs args)
{
    // Code here
}
```

```
}  
}
```

### OnGaugeMouseDown

When mouse is pressed down on the gauge, the [OnGaugeMouseDown](#) event is triggered. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge  
<SfLinearGauge>  
<LinearGaugeEvents OnGaugeMouseDown="MouseDown"></LinearGaugeEvents>  
<LinearGaugeAxes>  
<LinearGaugeAxis>  
</LinearGaugeAxis>  
</LinearGaugeAxes>  
</SfLinearGauge>  
@code {  
public void MouseDown(Syncfusion.Blazor.LinearGauge.MouseEventArgs args)  
{  
    //Code here  
}  
}
```

### OnGaugeMouseLeave

When mouse pointer leaves the gauge, the [OnGaugeMouseLeave](#) event is triggered. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge  
<SfLinearGauge>  
<LinearGaugeEvents OnGaugeMouseLeave="MouseLeave"></LinearGaugeEvents>  
<LinearGaugeAxes>  
<LinearGaugeAxis>  
</LinearGaugeAxis>  
</LinearGaugeAxes>  
</SfLinearGauge>  
@code {  
public void MouseLeave(Syncfusion.Blazor.LinearGauge.MouseEventArgs args)  
{  
    //Code here  
}  
}
```

### OnGaugeMouseUp

When the mouse pointer is released over the Linear Gauge, the [OnGaugeMouseUp](#) event is triggered. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge  
<SfLinearGauge>  
<LinearGaugeEvents OnGaugeMouseUp="MouseUp"></LinearGaugeEvents>
```

```
<LinearGaugeAxes>
<LinearGaugeAxis>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
@code {
public void MouseUp(Syncfusion.Blazor.LinearGauge.MouseEventArgs args)
{
//Code here
}
}
```

### OnLoad

Before the Linear Gauge is loaded, the [OnLoad](#) event is fired. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<SfLinearGauge>
<LinearGaugeEvents OnLoad="Load"></LinearGaugeEvents>
<LinearGaugeAxes>
<LinearGaugeAxis>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
@code {
public void Load(LoadEventArgs args)
{
// Code here
}
}
```

### OnPrint

The [OnPrint](#) event is fired before the print begins. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<button @onclick="PrintGauge">Print</button>
<SfLinearGauge @ref="gauge" AllowPrint="true">
<LinearGaugeEvents OnPrint="Print"></LinearGaugeEvents>
<LinearGaugeAxes>
<LinearGaugeAxis Minimum="0" Maximum="100">
<LinearGaugeMajorTicks Interval="20"></LinearGaugeMajorTicks>
<LinearGaugeMinorTicks Interval="10"></LinearGaugeMinorTicks>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
@code {
SfLinearGauge gauge;
public void PrintGauge()
{
this.gauge.Print();
}
```

```
}  
public void Print(PrintEventArgs args)  
{  
    // Code here  
}
```

### Resizing

Prior to the window resizing, the [Resizing](#) event is triggered. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge  
<SfLinearGauge Width="100%">  
    <LinearGaugeEvents Resizing="Resize"></LinearGaugeEvents>  
    <LinearGaugeAxes>  
        <LinearGaugeAxis>  
        </LinearGaugeAxis>  
    </LinearGaugeAxes>  
</SfLinearGauge>  
@code {  
    public void Resize(ResizeEventArgs args)  
    {  
        // Code here  
    }  
}
```

### TooltipRendering

The [TooltipRendering](#) event is fired before the tooltip is rendered. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge  
<SfLinearGauge Width="100%">  
    <LinearGaugeEvents TooltipRendering="TooltipRender"></LinearGaugeEvents>  
    <LinearGaugeTooltipSettings Enable="true"></LinearGaugeTooltipSettings>  
    <LinearGaugeAxes>  
        <LinearGaugeAxis>  
            <LinearGaugePointers>  
                <LinearGaugePointer Value="50"></LinearGaugePointer>  
            </LinearGaugePointers>  
        </LinearGaugeAxis>  
    </LinearGaugeAxes>  
</SfLinearGauge>  
@code {  
    public void TooltipRender(TooltipRenderEventArgs args)  
    {  
        // Code here  
    }  
}
```

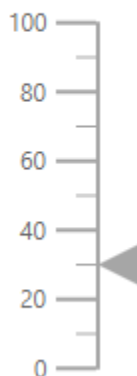
### ValueChanged

The [ValueChanged](#) event is triggered when the pointer is dragged from one value to another. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<div style="width:250px">
  <SfLinearGauge Height="250px">
    <LinearGaugeEvents ValueChange="@UpdatePointerValue"></LinearGaugeEvents>
    <LinearGaugeAxes>
      <LinearGaugeAxis>
        <LinearGaugePointers>
          <LinearGaugePointer EnableDrag="true" PointerValue="10">
        </LinearGaugePointer>
        </LinearGaugePointers>
      </LinearGaugeAxis>
    </LinearGaugeAxes>
  </SfLinearGauge>
</div>
@code {
  private double pointerValue = 10;
  public void UpdatePointerValue(ValueChangeEventArgs args)
  {
    pointerValue = args.Value;
  }
}
```

Update pointer value



### Methods in Blazor Linear Gauge Component

The following methods are available in the Linear Gauge component.

#### SetPointerValue

To change the pointer value dynamically, use the [SetPointerValue](#) method in the Linear Gauge component. The following are the arguments for this method.

Argument name	Description
---------------	-------------

-----	-----
axisIndex	Specifies the index of the axis in which the pointer value is to be updated.
pointerIndex	Specifies the index of the pointer to be updated.
pointerValue	Specifies the value of the pointer to be updated.

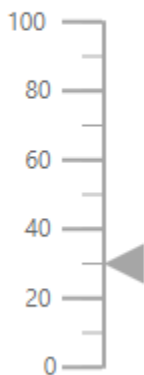
**ASPX-CS**

```

@using Syncfusion.Blazor.LinearGauge
<button style="margin-left:34px" @onclick="ChangePoinerValue">Update pointer
value
</button>
<SfLinearGauge @ref="lineargauge" Width="250px" Height="250px">
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer Value="10"></LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
@code {
SfLinearGauge lineargauge;
public void ChangePoinerValue()
{
lineargauge.SetPointerValue(0, 0, 30);
}
}

```

Update pointer value

**SetAnnotationValue**

To change the annotation content dynamically, use the [SetAnnotationValue](#) method in the Linear Gauge component. The following are the arguments for this method.

Argument name	Description	
-----	-----	

	annotationIndex		Specifies the index number of the annotation to be updated.	
	content		Specifies the text for the annotation to be updated.	
	axisValue		Specifies the value of the axis where the annotation is to be placed.	

This method will not be applicable for the [ContentTemplate](#) class in [LinearGaugeAnnotation](#).

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<button style="margin-left:34px" @onclick="ChangeAnnotationValue">Update
annotation value</button>
<SfLinearGauge @ref="lineargauge" Width="250px" Height="250px">
<LinearGaugeAnnotations>
<LinearGaugeAnnotation AxisValue="0" ZIndex="1" Content="10">
</LinearGaugeAnnotation>
</LinearGaugeAnnotations>
<LinearGaugeAxes>
<LinearGaugeAxis>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
@code {
SfLinearGauge lineargauge;
public void ChangeAnnotationValue()
{
lineargauge.SetAnnotationValue(0, "50", 50);
}
}
```

### RefreshAsync

The [RefreshAsync](#) method can be used to change the state of the component and render it again.

### ASPX-CS

```
@using Syncfusion.Blazor.LinearGauge
<button style="margin-left:34px" @onclick="RefreshAsync">Refresh
Gauge</button>
<SfLinearGauge @ref="lineargauge" Width="250px" Height="250px">
<LinearGaugeAxes>
<LinearGaugeAxis>
<LinearGaugePointers>
<LinearGaugePointer Value="10"></LinearGaugePointer>
</LinearGaugePointers>
</LinearGaugeAxis>
</LinearGaugeAxes>
</SfLinearGauge>
@code {
SfLinearGauge lineargauge;
public async Task RefreshAsync()
{
await lineargauge.RefreshAsync();
}
}
```



## ListBox

### Getting Started with Blazor ListBox Component

This section briefly explains about how to include ListBox Component in your Blazor server-side application.

To get start quickly with ListBox Component using Blazor, you can check on this video:

{% youtube

"youtube:https://www.youtube.com/watch?v=-nZ1n8zFIPI"%}

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.

---

Please ensure to check the **Include prerelease** option.

---

#### ASPX-CS

```
<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.

#### 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.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. 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 ListBox component to the application

To initialize the ListBox component add the below code to your **Index.razor** view page which is present under **~/Pages** folder. For example, the ListBox component is added in the **~/Pages/Index.razor** page.

#### ASPX-CS

```
@using Syncfusion.Blazor
<SfListBox TValue="string[]"></SfListBox>
```

Binding data source

After initializing, populate the ListBox with data using the **DataSource** property. Here, an array of object values is passed to the ListBox component.

The following example illustrates the output in your browser.

#### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@Vehicles" TItem="VehicleData">
<ListBoxFieldSettings Text="Text" Value="Id" />
</SfListBox>
```

```
@code {
public List<VehicleData> Vehicles = new List<VehicleData> {
new VehicleData { Text = "Hennessey Venom", Id = "Vehicle-01" },
new VehicleData { Text = "Bugatti Chiron", Id = "Vehicle-02" },
new VehicleData { Text = "Bugatti Veyron Super Sport", Id = "Vehicle-03" },
new VehicleData { Text = "SSC Ultimate Aero", Id = "Vehicle-04" },
new VehicleData { Text = "Koenigsegg CCR", Id = "Vehicle-05" },
new VehicleData { Text = "McLaren F1", Id = "Vehicle-06" },
new VehicleData { Text = "Aston Martin One- 77", Id = "Vehicle-07" },
new VehicleData { Text = "Jaguar XJ220", Id = "Vehicle-08" }
};
public class VehicleData {
public string Text { get; set; }
public string Id { get; set; }
}
}
```

**TValue** is type of value in the datasource to resolve type inference. It is generic type and can be given as `string[], int[]`.

### Run the application

After successful compilation of your application, simply press F5 to run the application. The Blazor ListBox 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 ListBox 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. ListBox provides built-in compliance with WAI-ARIA specifications. WAI-ARIA support is achieved through the attributes like `aria-multiselectable` and `aria-selected` applied for ListBox element and selected elements in the ListBox. It helps the people with disabilities by providing information about the widget for assistive technology in the screen readers. ListBox component contains the `listbox` role and `option` role.

| Properties | Functionality |

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

| [listbox](#) | This role will be specified for root element. |

| [aria-multiselectable](#) | Applied to the element with the ListBox role, tells assistive technologies that the list supports multiple selection. The default value is true. |

| [option](#) | Identifies each selectable element containing the name of an option. |

| [aria-selected](#) | Applied to elements with role option that are visually styled as selected to inform assistive technologies that the options are selected. |

### Keyboard interaction

<!-- markdownlint-disable MD033 -->

Keyboard shortcuts	Actions
Up arrow	Moves focus to the previous option.
Down arrow	Moves focus to the next option.
Home	Moves focus to first option.
End	Moves focus to last option.
Space	Changes the selection state of the focused option.
Ctrl + A	Selects all options in the list.
Ctrl + Shift + Home	Selects the focused option and all options up to the first option.
Ctrl + Shift + End	Selects the focused option and all options down to the last option.
Ctrl + (Up or Down)	Press Ctrl key with up / down arrow or mouse to select multiple items.

### Data Binding in Blazor ListBox Component

The ListBox loads the data from local data sources using the [DataSource](#) property.

| Fields | Type | Description |

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

| [Text](#) | `string` | Specifies the display text of each list item. |

| [Value](#) | `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 iconCss class that needs to be mapped. |

| [HtmlAttributes](#) | `string` | Allows additional attributes to configure the elements in various ways to meet the criteria. |

When binding complex data to the ListBox, fields should be mapped correctly. Otherwise, the selected item remains undefined.

### Local Data

Local data can be represented by the following ways as described below.

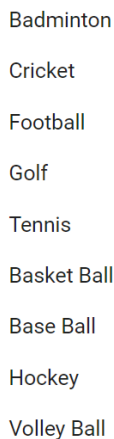
#### Array of string

The ListBox has support to load array of primitive data such as strings or numbers. Here, both value and text field acts as same.

#### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@Games" TItem="string"></SfListBox>
@code{
public string[] Games = new string[] { "Badminton", "Cricket", "Football",
"Golf", "Tennis", "Basket Ball", "Base Ball", "Hockey", "Volley Ball" };
}
```

Output will be shown as



#### Array of object

The ListBox can generate its list items through an array of object data. For this, the appropriate columns should be mapped to the [Fields](#) property.

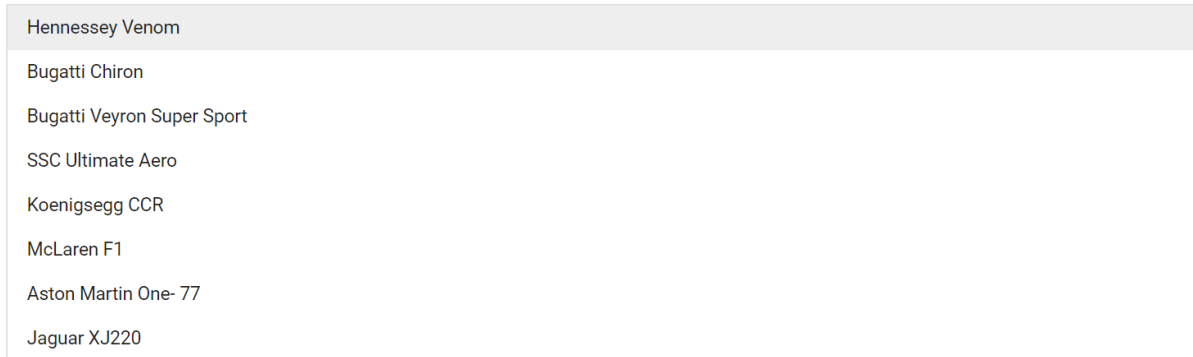
In the following example, `id` and `sports` column from complex data have been mapped to the `Value` field and `Text` field, respectively.

#### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@Vehicles" TItem="VehicleData">
<ListBoxFieldSettings Text="Text" Value="Id" />
</SfListBox>
@code {
public List<VehicleData> Vehicles = new List<VehicleData> {
new VehicleData { Text = "Hennessey Venom", Id = "Vehicle-01" },
new VehicleData { Text = "Bugatti Chiron", Id = "Vehicle-02" },
new VehicleData { Text = "Bugatti Veyron Super Sport", Id = "Vehicle-03" },
new VehicleData { Text = "SSC Ultimate Aero", Id = "Vehicle-04" },
new VehicleData { Text = "Koenigsegg CCR", Id = "Vehicle-05" },
new VehicleData { Text = "McLaren F1", Id = "Vehicle-06" },
}
```

```
new VehicleData { Text = "Aston Martin One- 77", Id = "Vehicle-07" },
new VehicleData { Text = "Jaguar XJ220", Id = "Vehicle-08" }
};
public class VehicleData {
public string Text { get; set; }
public string Id { get; set; }
}
}
```

Output will be shown as



### *Array of complex object*

The ListBox can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [Fields](#) property.

In the following example, `sports.Name` column from complex data have been mapped to the `Text` field.

### **ASPX-CS**

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@SprotsDetails" TItem="SportsData">
<ListBoxFieldSettings Text="Sports.Name" Value="Id" />
</SfListBox>
@code {
public List<SportsData> SprotsDetails = new List<SportsData>
{
new SportsData{ Id = "game0", Sports = new GameData{ Name = "Badminton" } },
new SportsData{ Id = "game1", Sports = new GameData{ Name = "Cricket" } },
new SportsData{ Id = "game2", Sports = new GameData{ Name = "Football" } },
new SportsData{ Id = "game3", Sports = new GameData{ Name = "Golf" } },
new SportsData{ Id = "game4", Sports = new GameData{ Name = "Tennis" } },
new SportsData{ Id = "game5", Sports = new GameData{ Name = "Basket Ball" } },
},
new SportsData{ Id = "game6", Sports = new GameData{ Name = "Base Ball" } },
new SportsData{ Id = "game7", Sports = new GameData{ Name = "Hockey" } }
};
public class GameData {
public string Name { get; set; }
}
public class SportsData {
public string Id { get; set; }
public GameData Sports { get; set; }
}
}
```

```
}
```

Output will be shown as

```
Badminton
Cricket
Football
Golf
Tennis
Basket Ball
Base Ball
Hockey
```

### Remote Data

The ListBox supports retrieval of data from remote data services with the help of [DataManager](#).

The following sample displays the employee names from **Employee** table.

### CSHARP

```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Data
<SfListBox TValue="string[]" TItem="OrderDetails" Query="@RemoteDataQuery">
<SfDataManager
Url="https://js.syncfusion.com/demos/Sfervices/Wcf/Northwind.svc/Orders"
CrossDomain="true"
Adaptor="Syncfusion.Blazor.Adaptors.ODataAdaptor"></SfDataManager>
<ListBoxFieldSettings Text="CustomerID" Value="CustomerID" />
</SfListBox>
@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; }
}
}
```

Output will be shown as

VINET  
TOMSP  
HANAR  
VICTE  
SUPRD  
HANAR

## Drag And Drop in Blazor ListBox Component

The ListBox has support to drag an item or a group of selected items and drop it within the same listbox or into another listbox.

The elements can be customized on drag and drop by using the following events.

Events	Description
<a href="#">DragStart</a>	Triggers when the selected element's drag starts.
<a href="#">Dragging</a>	Triggers when the selected element is being dragged.
<a href="#">OnDrop</a>	Triggers before the selected element is dropped.
<a href="#">Dropped</a>	Triggers when the selected element is dropped.

### Single ListBox

To drag and drop an item or group of item within the listbox can achieved by setting [AllowDragAndDrop](#) property to `true`.

The following sample illustrates how to drag and drop an item within the same listbox.

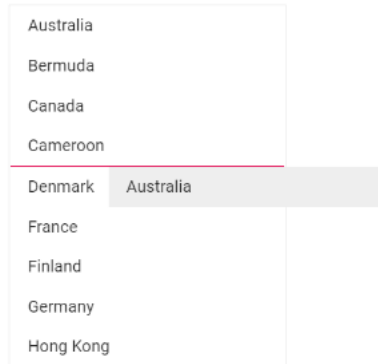
### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@GroupA" TItem="CountryCode"
AllowDragAndDrop="true">
<ListBoxFieldSettings Text="Name" Value="Code" />
</SfListBox>
@code {
public List<CountryCode> GroupA = new List<CountryCode>
{
new CountryCode{ Name = "Australia", Code = "AU" },
new CountryCode{ Name = "Bermuda", Code = "BM" },
new CountryCode{ Name = "Canada", Code = "CA" },
new CountryCode{ Name = "Cameroon", Code = "CM" },
new CountryCode{ Name = "Denmark", Code = "DK" },
new CountryCode{ Name = "France", Code = "FR" },
new CountryCode{ Name = "Finland", Code = "FI" },
new CountryCode{ Name = "Germany", Code = "DE" },
new CountryCode{ Name = "Hong Kong", Code = "HK" }
};
public class CountryCode {
public string Name { get; set; }
public string Code { get; set; }
}
```



```
}
}
```

Output will be shown as



### Multiple ListBox

To drag and drop an item or group of item between two listbox can achieved by setting [AllowDragAndDrop](#) property to `true` and [Scope](#) should be set as `combined-list` in both the listbox.

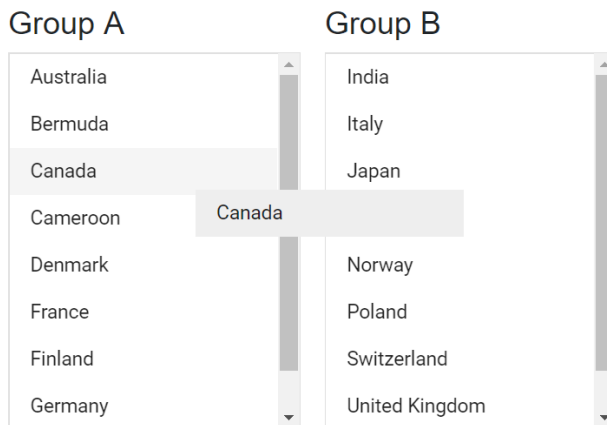
The following sample illustrates how to drag and drop an item between two listbox.

### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<div id="listbox1">
<h4>Group A</h4>
<SfListBox TValue="string[]" DataSource="@GroupA" AllowDragAndDrop="true"
Scope="combined-list" Height="290px" TItem="CountryCode">
<ListBoxFieldSettings Text="Name" Value="Code" />
</SfListBox>
</div>
<div id="listbox2">
<h4>Group B</h4>
<SfListBox TValue="string[]" DataSource="@GroupB" Scope="combined-list"
AllowDragAndDrop="true" Height="290px" TItem="CountryCode">
<ListBoxFieldSettings Text="Name" Value="Code" />
</SfListBox>
</div>
@code {
public List<CountryCode> GroupA = new List<CountryCode>
{
new CountryCode{ Name = "Australia", Code = "AU" },
new CountryCode{ Name = "Bermuda", Code = "BM" },
new CountryCode{ Name = "Canada", Code = "CA" },
new CountryCode{ Name = "Cameroon", Code = "CM" },
new CountryCode{ Name = "Denmark", Code = "DK" },
new CountryCode{ Name = "France", Code = "FR" },
new CountryCode{ Name = "Finland", Code = "FI" },
new CountryCode{ Name = "Germany", Code = "DE" },
new CountryCode{ Name = "Hong Kong", Code = "HK" }
};
public List<CountryCode> GroupB = new List<CountryCode>
{
```

```
new CountryCode{ Name = "India", Code = "IN" },
new CountryCode{ Name = "Italy", Code = "IT" },
new CountryCode{ Name = "Japan", Code = "JP" },
new CountryCode{ Name = "Mexico", Code = "MX" },
new CountryCode{ Name = "Norway", Code = "NO" },
new CountryCode{ Name = "Poland", Code = "PL" },
new CountryCode{ Name = "Switzerland", Code = "CH" },
new CountryCode{ Name = "United Kingdom", Code = "GB" },
new CountryCode{ Name = "United States", Code = "US" }
};
public class CountryCode
{
public string Name { get; set; }
public string Code { get; set; }
}
}
<style>
#listbox1 {
width: 48%;
float: left;
}
#listbox2 {
width: 48%;
float: right;
}
</style>
```

Output will be shown as,



### Dual ListBox in Blazor ListBox Component

The dual ListBox allows the user to move items between two listbox by clicking the toolbar buttons. Dual ListBox can be created by listing items in the [ToolbarSettings](#) along with the [Scope](#) property.

The following operations can be performed in dual ListBox,

| Options | Description |

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

- | MoveUp | Move the selected item in the upward direction within the listbox. |
- | MoveDown | Move the selected item in the downward direction within the listbox. |
- | MoveTo | Move the selected item to the another listbox. |
- | MoveFrom | Move the selected item from one listbox to the another listbox. |
- | MoveAllTo | Move all the items to the another listbox. |
- | MoveAllFrom | Move all the items from one listbox to the another listbox. |

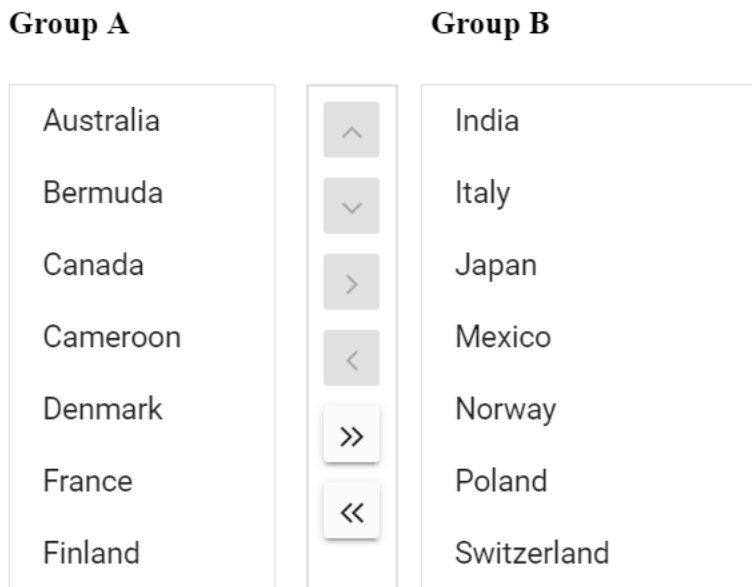
The following example illustrates how to move items from **Group A** to **Group B** listbox.

#### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<div id="listbox1">
<h4>Group A</h4>
<SfListBox TValue="string[]" DataSource="@GroupA" Scope="scope2"
TItem="CountryCode" @attributes="listbox1Attr">
<ListBoxFieldSettings Text="Name"></ListBoxFieldSettings>
<ListBoxToolBarSettings Items="@Items"></ListBoxToolBarSettings>
</SfListBox>
</div>
<div id="listbox2">
<h4>Group B</h4>
<SfListBox TValue="string[]" Scope="scope1" DataSource="@GroupB"
TItem="CountryCode" @attributes="listbox2Attr">
<ListBoxFieldSettings Text="Name"></ListBoxFieldSettings>
</SfListBox>
</div>
@code {
private readonly Dictionary<string, object> listbox1Attr = new
Dictionary<string, object>
{
{ "id", "scope1" }
};
private readonly Dictionary<string, object> listbox2Attr = new
Dictionary<string, object>
{
{ "id", "scope2" }
};
public string[] Items = new string[] { "MoveUp", "MoveDown", "MoveTo",
"MoveFrom", "MoveAllTo", "MoveAllFrom" };
public List<CountryCode> GroupA = new List<CountryCode>
{
new CountryCode{ Name = "Australia", Code = "AU" },
new CountryCode{ Name = "Bermuda", Code = "BM" },
new CountryCode{ Name = "Canada", Code = "CA" },
new CountryCode{ Name = "Cameroon", Code = "CM" },
new CountryCode{ Name = "Denmark", Code = "DK" },
new CountryCode{ Name = "France", Code = "FR" },
new CountryCode{ Name = "Finland", Code = "FI" }
};
public List<CountryCode> GroupB = new List<CountryCode>
{
new CountryCode{ Name = "India", Code = "IN" },
new CountryCode{ Name = "Italy", Code = "IT" },
```

```
new CountryCode{ Name = "Japan", Code = "JP" },
new CountryCode{ Name = "Mexico", Code = "MX" },
new CountryCode{ Name = "Norway", Code = "NO" },
new CountryCode{ Name = "Poland", Code = "PL" },
new CountryCode{ Name = "Switzerland", Code = "CH" }
};
public class CountryCode
{
    public string Name { get; set; }
    public string Code { get; set; }
}
<style>
#listbox1 {
width: 48%;
float: left;
}
#listbox2 {
width: 48%;
float: right;
}
</style>
```

Output will be shown as



## Icons and Templates in Blazor ListBox Component

### Icons

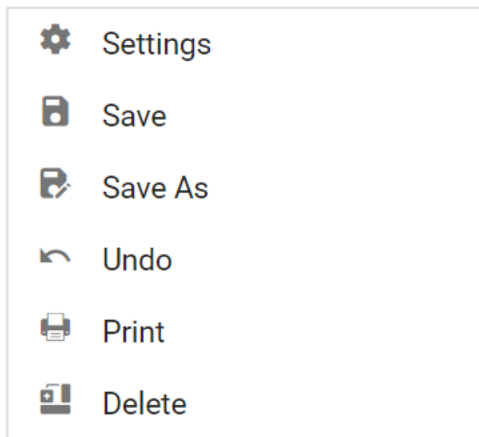
To place the icon on a listbox, set the [IconCss](#) property to `e-icons` with the required icon CSS. By default, the icon is positioned to the left side of the list.

In the following sample, icon classes are mapped with `IconCss` field.

### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@SettingsData"
TItem="SettingItems">
<ListBoxFieldSettings Text="Text" IconCss="IconCss" />
</SfListBox>
@code {
public List<SettingItems> SettingsData = new List<SettingItems> {
new SettingItems{ Text = "Settings", IconCss = "e-icons e-list-settings" },
new SettingItems{ Text = "Save", IconCss = "e-icons e-list-save" },
new SettingItems{ Text = "Save As", IconCss = "e-icons e-list-saveas" },
new SettingItems{ Text = "Undo", IconCss = "e-icons e-list-undo" },
new SettingItems{ Text = "Print", IconCss = "e-icons e-list-print" },
new SettingItems{ Text = "Delete", IconCss = "e-icons e-list-delete" }
};
public class SettingItems {
public string Text { get; set; }
public string IconCss { get; set; }
}
}
<style>
.e-list-settings:before {
content: "\e679";
}
.e-list-print:before {
content: "\e743";
}
.e-list-save:before {
content: "\e74d";
}
.e-list-saveas:before {
content: "\e72b";
}
.e-list-delete:before {
content: "\e773";
}
.e-list-undo:before {
content: "\e752";
}
}
</style>
```

Output will be shown as,



## Templates

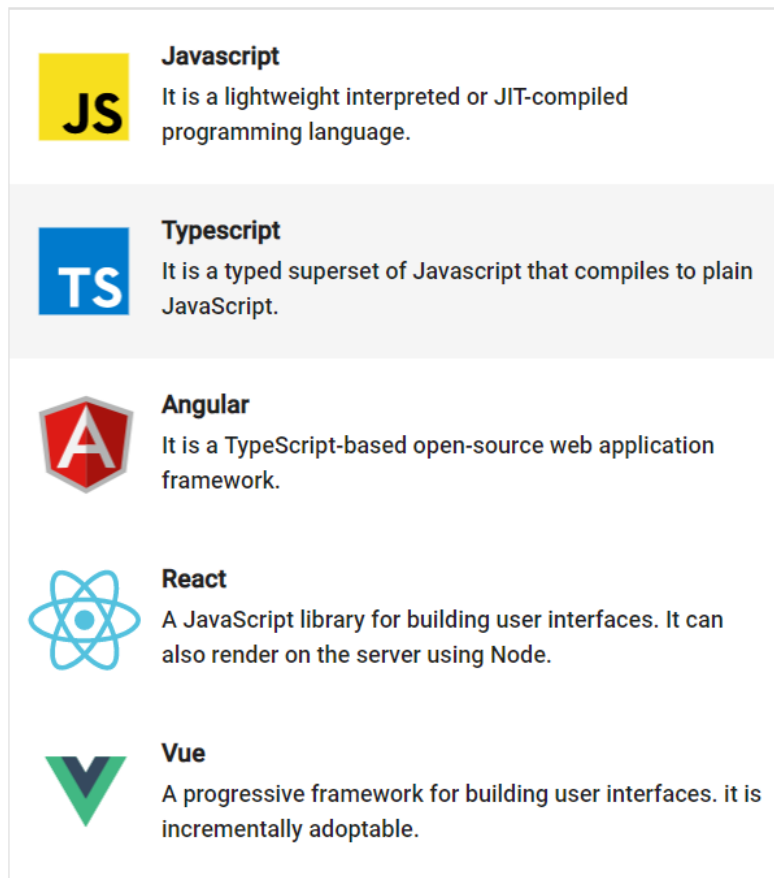
ListBox items can be customized according to the requirement using [ItemTemplate](#) property.

### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@Data" TItem="ListData">
  <ListBoxFieldSettings Text="Text"></ListBoxFieldSettings>
  <ListBoxTemplates TItem="ListData">
    <ItemTemplate>
      <div class="list-wrapper">
        <span class="@((context as ListData).Pic) e-avatar e-avatar-xlarge e-avatar-circle"></span>
        <span class="text">@((context as ListData).Text)</span><span
          class="description">@((context as ListData).Description)</span>
      </div>
    </ItemTemplate>
  </ListBoxTemplates>
</SfListBox>
@code {
  public ListData Model = new ListData();
  public List<ListData> Data = new List<ListData>
  {
    new ListData { Text = "Javascript", Pic = "javascript", Description = "It is a lightweight interpreted or JIT-compiled programming language." },
    new ListData { Text = "Typescript", Pic = "typescript", Description = "It is a typed superset of Javascript that compiles to plain JavaScript." },
    new ListData { Text = "Angular", Pic = "angular", Description = "It is a TypeScript-based open-source web application framework." },
    new ListData { Text = "React", Pic = "react", Description = "A JavaScript library for building user interfaces. It can also render on the server using Node." },
    new ListData { Text = "Vue", Pic = "vue", Description = "A progressive framework for building user interfaces. it is incrementally adoptable." }
  };
  public class ListData
  {
    public string Text { get; set; }
    public string Pic { get; set; }
  }
}
```

```
public string Description { get; set; }
}
}
<style>
.e-listbox-wrapper {
margin: auto;
max-width: 400px;
box-sizing: border-box;
}
.list-wrapper {
height: inherit;
position: relative;
padding: 14px 12px 14px 78px;
}
.list-wrapper .text,
.list-wrapper .description {
display: block;
margin: 0;
padding-bottom: 3px;
white-space: normal;
}
.list-wrapper .description {
font-size: 12px;
font-weight: 500;
}
.e-listbox-wrapper .list-wrapper .text {
font-weight: bold;
font-size: 13px;
}
.list-wrapper .e-avatar {
position: absolute;
left: 5px;
background-color: transparent;
font-size: 22px;
top: calc(50% - 33px);
}
.e-listbox-container .e-list-item {
height: auto !important;
}
.javascript {
background-image: url('./images/javascript.svg');
}
.typescript {
background-image: url('./images/typescript.svg')
}
.angular {
background-image: url('./images/angular.svg');
}
.vue {
background-image: url('./images/vue.svg');
}
.react {
background-image: url('./images/react.svg');
}
</style>
```

Output will be shown as,



### Selection in Blazor ListBox Component

The ListBox provides support to select an item or a group of item by mouse or keyboard action. There are two selection modes available in ListBox,

- Single - To select single item in the ListBox.
- Multiple - To select multiple items in the ListBox.

#### Single selection

To enable single selection in the ListBox, [Mode](#) should be set as **Single** in [SelectionSettings](#) property.

#### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@Vehicles" TItem="VehicleData">
  <ListBoxFieldSettings Text="Text" Value="Id" />
  <ListBoxSelectionSettings
    Mode="Syncfusion.Blazor.DropDowns.SelectionMode.Single"></ListBoxSelectionSettings>
</SfListBox>
@code {
  public List<VehicleData> Vehicles = new List<VehicleData> {
    new VehicleData { Text = "Hennessey Venom", Id = "Vehicle-01" },
```



```
new VehicleData { Text = "Bugatti Chiron", Id = "Vehicle-02" },
new VehicleData { Text = "Bugatti Veyron Super Sport", Id = "Vehicle-03" },
new VehicleData { Text = "SSC Ultimate Aero", Id = "Vehicle-04" },
new VehicleData { Text = "Koenigsegg CCR", Id = "Vehicle-05" },
new VehicleData { Text = "McLaren F1", Id = "Vehicle-06" },
new VehicleData { Text = "Aston Martin One- 77", Id = "Vehicle-07" },
new VehicleData { Text = "Jaguar XJ220", Id = "Vehicle-08" }
};
public class VehicleData {
public string Text { get; set; }
public string Id { get; set; }
}
}
```

Output will be shown as,

```
Hennessey Venom
Bugatti Chiron
Bugatti Veyron Super Sport
SSC Ultimate Aero
Koenigsegg CCR
McLaren F1
Aston Martin One- 77
Jaguar XJ220
```

### Multiple selection

To enable multiple selection in the ListBox, **Mode** should be set as **Multiple** in **SelectionSettings** property. To select multiple items, use the SHIFT, CTRL, and arrow keys to make selections.

By default, the selection mode is set as **Multiple**.

### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@Vehicles" TItem="VehicleData">
<ListBoxSelectionSettings
Mode="Syncfusion.Blazor.DropDowns.SelectionMode.Multiple"></ListBoxSelection
Settings>
<ListBoxFieldSettings Text="Text" Value="Id" />
</SfListBox>
@code {
public List<VehicleData> Vehicles = new List<VehicleData> {
new VehicleData { Text = "Hennessey Venom", Id = "Vehicle-01" },
new VehicleData { Text = "Bugatti Chiron", Id = "Vehicle-02" },
new VehicleData { Text = "Bugatti Veyron Super Sport", Id = "Vehicle-03" },
new VehicleData { Text = "SSC Ultimate Aero", Id = "Vehicle-04" },
new VehicleData { Text = "Koenigsegg CCR", Id = "Vehicle-05" },
new VehicleData { Text = "McLaren F1", Id = "Vehicle-06" },
new VehicleData { Text = "Aston Martin One- 77", Id = "Vehicle-07" },
new VehicleData { Text = "Jaguar XJ220", Id = "Vehicle-08" }
};
public class VehicleData {
```

```
public string Text { get; set; }
public string Id { get; set; }
}
```

Output will be shown as,



### CheckBox Selection

ListBox supports checkbox selection which is used to select multiple items. To enable the checkbox selection, set the [ShowCheckbox](#) property to `true`.

### Select All

To select all the items in the ListBox, enable the [ShowSelectAll](#) property to `true`.

### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@Vehicles" TItem="VehicleData">
  <ListBoxFieldSettings Text="Text" Value="Id" />
  <ListBoxSelectionSettings ShowCheckbox="true"
    ShowSelectAll="true"></ListBoxSelectionSettings>
</SfListBox>
@code {
public List<VehicleData> Vehicles = new List<VehicleData>
{
    new VehicleData { Text = "Hennessey Venom", Id = "Vehicle-01" },
    new VehicleData { Text = "Bugatti Chiron", Id = "Vehicle-02" },
    new VehicleData { Text = "Bugatti Veyron Super Sport", Id = "Vehicle-03" },
    new VehicleData { Text = "SSC Ultimate Aero", Id = "Vehicle-04" },
    new VehicleData { Text = "Koenigsegg CCR", Id = "Vehicle-05" },
    new VehicleData { Text = "McLaren F1", Id = "Vehicle-06" },
    new VehicleData { Text = "Aston Martin One- 77", Id = "Vehicle-07" },
    new VehicleData { Text = "Jaguar XJ220", Id = "Vehicle-08" }
};
public class VehicleData {
    public string Text { get; set; }
    public string Id { get; set; }
}
```

Output will be shown as,

☐ Select All

☒ Hennessey Venom

☐ Bugatti Chiron

☒ Bugatti Veyron Super Sport

☐ SSC Ultimate Aero

☒ Koenigsegg CCR

☐ McLaren F1

☒ Aston Martin One- 77

☐ Jaguar XJ220

## Sorting and Grouping in Blazor ListBox Component

### Sorting

The ListBox supports sorting of available items in the alphabetical order that can be either ascending or descending. This can be achieved using [SortOrder](#) property. Sort order can be **None**, **Ascending** or **Descending**.

In the following example, the **SortOrder** is set as **Descending**.

### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@CountryData"
SortOrder="Syncfusion.Blazor.DropDowns.SortOrder.Descending"
TItem="CountryCode">
<ListBoxFieldSettings Text="Name" Value="Code" />
</SfListBox>
@code {
public List<CountryCode> CountryData = new List<CountryCode> {
new CountryCode{ Name = "Australia", Code = "AU" },
new CountryCode{ Name = "Bermuda", Code = "BM" },
new CountryCode{ Name = "Canada", Code = "CA" },
new CountryCode{ Name = "Cameroon", Code = "CM" },
new CountryCode{ Name = "Denmark", Code = "DK" },
new CountryCode{ Name = "France", Code = "FR" },
new CountryCode{ Name = "Finland", Code = "FI" },
new CountryCode{ Name = "Germany", Code = "DE" },
new CountryCode{ Name = "Hong Kong", Code = "HK" }
};
public class CountryCode {
public string Name { get; set; }
public string Code { get; set; }
}
}
```

Output will be shown as,



Hong Kong  
Germany  
France  
Finland  
Denmark  
Canada  
Cameroon  
Bermuda  
Australia

### Grouping

The ListBox supports to wrap the nested element into a group based on its category. The category of each list item can be mapped with [GroupBy](#) field in the data table.

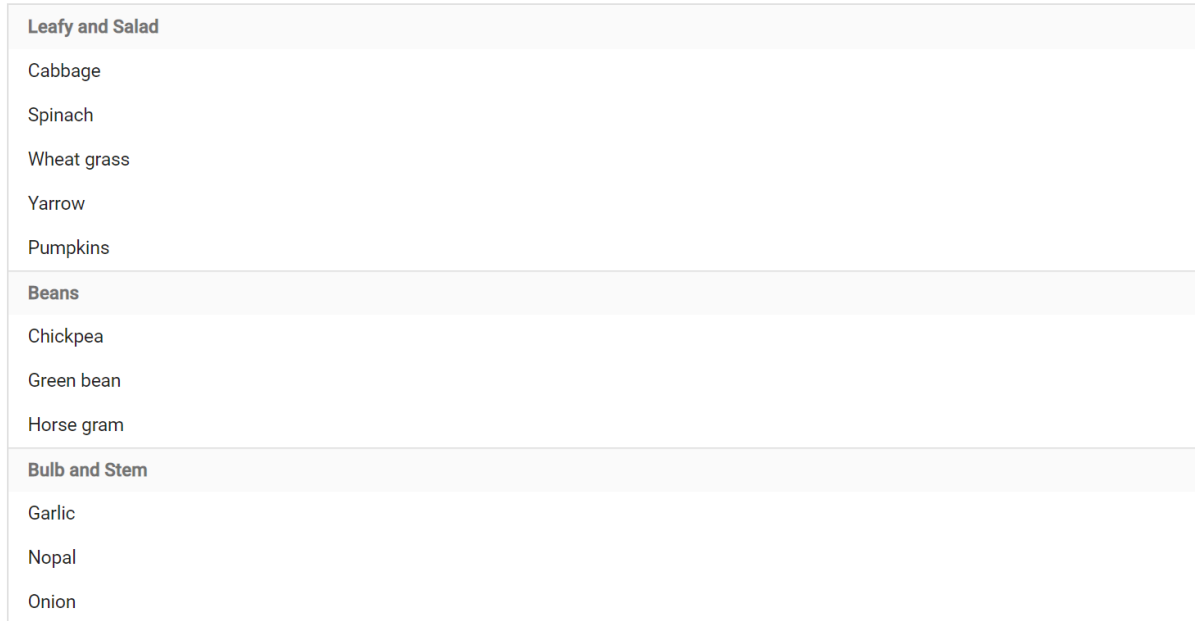
In the following example, vegetables are grouped based on its category.

### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@VegetableData"
TItem="VegetableDetail">
<ListBoxFieldSettings GroupBy = "Category" Text="Vegetable" Value="Id" />
</SfListBox>
@code {
public List<VegetableDetail> VegetableData = new List<VegetableDetail> {
new VegetableDetail{ Vegetable = "Cabbage", Category = "Leafy and Salad", Id = "item1" },
new VegetableDetail{ Vegetable = "Spinach", Category = "Leafy and Salad", Id = "item2" },
new VegetableDetail{ Vegetable = "Wheat grass", Category = "Leafy and Salad", Id = "item3" },
new VegetableDetail{ Vegetable = "Yarrow", Category = "Leafy and Salad", Id = "item4" },
new VegetableDetail{ Vegetable = "Pumpkins", Category = "Leafy and Salad", Id = "item5" },
new VegetableDetail{ Vegetable = "Chickpea", Category = "Beans", Id = "item6" },
new VegetableDetail{ Vegetable = "Green bean", Category = "Beans", Id = "item7" },
new VegetableDetail{ Vegetable = "Horse gram", Category = "Beans", Id = "item8" },
new VegetableDetail{ Vegetable = "Garlic", Category = "Bulb and Stem", Id = "item9" },
new VegetableDetail{ Vegetable = "Nopal", Category = "Bulb and Stem", Id = "item10" },
new VegetableDetail{ Vegetable = "Onion", Category = "Bulb and Stem", Id = "item11" }
};
public class VegetableDetail {
public string Vegetable { get; set; }
public string Category { get; set; }
public string Id { get; set; }
}
```

```
}
}
```

Output will be shown as,



## Styles and Appearances in Blazor ListBox Component

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

| CSS Class | Purpose of Class |

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

| .e-listbox-container | To customize the listbox container |

| .e-listbox-container .e-list-item | To customize the listbox list items |

| .e-listbox-container .e-list-item:hover:not(.e-selected):not(.e-disabled) | To customize the listbox list items on hover |

| .e-listbox-container .e-list-item.e-selected | To customize the listbox selected list item |

| .e-listboxtool-container .e-listbox-tool | To customize the listbox toolbar |

| .e-listboxtool-container .e-listbox-tool .e-btn | To customize the listbox toolbar button |

| .e-listboxtool-container .e-listbox-tool .e-btn .e-btn-icon.e-icons::before | To customize the listbox toolbar icon |

## How To

### Add/Remove Items in Blazor ListBox Component

To add an item or multiple items, **AddItem** method can be used. In the following example, the **Ferrari LaFerrari** and **McLaren P1** items will be added while clicking **Add Items** button.

## ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Buttons
<SfListBox TValue="string[]" TItem="VehicleData" DataSource="@Vehicles"
@ref="ListBoxObj">
<ListBoxFieldSettings Text="Text" Value="Id" />
</SfListBox>
<SfButton @onclick="addData">ADD ITEMS</SfButton>
@code {
SfListBox<string[], VehicleData> ListBoxObj;
public List<VehicleData> Vehicles = new List<VehicleData> {
new VehicleData { Text = "Hennessey Venom", Id = "Vehicle-01" },
new VehicleData { Text = "Bugatti Chiron", Id = "Vehicle-02" },
new VehicleData { Text = "Bugatti Veyron Super Sport", Id = "Vehicle-03" },
new VehicleData { Text = "SSC Ultimate Aero", Id = "Vehicle-04" },
new VehicleData { Text = "Koenigsegg CCR", Id = "Vehicle-05" },
new VehicleData { Text = "McLaren F1", Id = "Vehicle-06" },
new VehicleData { Text = "Aston Martin One- 77", Id = "Vehicle-07" },
new VehicleData { Text = "Jaguar XJ220", Id = "Vehicle-08" }
};
public class VehicleData {
public string Text { get; set; }
public string Id { get; set; }
}
public List<VehicleData> Item = new List<VehicleData>{
new VehicleData{ Text = "Ferrari LaFerrari", Id = "Vehicle-09"},
new VehicleData{ Text = "McLaren P1", Id = "Vehicle-10"}
};
private async Task addData() {
await ListBoxObj.AddItems(Item);
}
}
```

Output will be shown as

Hennessey Venom

Bugatti Chiron

Bugatti Veyron Super Sport

SSC Ultimate Aero

Koenigsegg CCR

McLaren F1

Aston Martin One- 77

Jaguar XJ220

ADD ITEMS

### *Remove items from the listbox*

To remove an item or multiple items, [RemoveItem](#) method can be used. In the following example, the **Ferrari LaFerrari** and **McLaren P1** items will be removed while clicking **Remove Items** button.

## ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Buttons
<SfListBox TValue="string[]" TItem="VehicleData" DataSource="@Vehicles"
@ref="ListBoxObj">
<ListBoxFieldSettings Text="Text" Value="Id" />
</SfListBox>
<SfButton @onclick="removeData">REMOVE ITEMS</SfButton>
@code {
SfListBox<string[],VehicleData> ListBoxObj;
public List<VehicleData> Vehicles = new List<VehicleData> {
new VehicleData { Text = "Hennessey Venom", Id = "Vehicle-01" },
new VehicleData { Text = "Bugatti Chiron", Id = "Vehicle-02" },
new VehicleData { Text = "Bugatti Veyron Super Sport", Id = "Vehicle-03" },
new VehicleData { Text = "SSC Ultimate Aero", Id = "Vehicle-04" },
new VehicleData { Text = "Koenigsegg CCR", Id = "Vehicle-05" },
new VehicleData { Text = "McLaren F1", Id = "Vehicle-06" },
new VehicleData { Text = "Aston Martin One- 77", Id = "Vehicle-07" },
new VehicleData { Text = "Jaguar XJ220", Id = "Vehicle-08" },
new VehicleData{ Text = "Ferrari LaFerrari", Id = "Vehicle-09"},
new VehicleData{ Text = "McLaren P1", Id = "Vehicle-10"}
};
public class VehicleData {
public string Text { get; set; }
public string Id { get; set; }
}
public List<VehicleData> Item = new List<VehicleData>{
new VehicleData{ Text = "Ferrari LaFerrari", Id = "Vehicle-09"},
new VehicleData{ Text = "McLaren P1", Id = "Vehicle-10"}
};
private async Task removeData() {
await ListBoxObj.RemoveItem(Item);
}
}
```

Output will be shown as

Hennessey Venom  
Bugatti Chiron  
Bugatti Veyron Super Sport  
SSC Ultimate Aero  
Koenigsegg CCR  
McLaren F1  
Aston Martin One- 77  
Jaguar XJ220

REMOVE ITEMS

### Bind Change Events in Blazor ListBox Component

To bind the change event in the listbox [ValueChange](#) event is used and the event is triggered when the value in the listbox changes.

## ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" TItem="VehicleData" DataSource="@Vehicles">
<ListBoxEvents TValue="string[]" ValueChange="change"
TItem="VehicleData"></ListBoxEvents>
<ListBoxFieldSettings Text="Text" Value="Id" />
</SfListBox>
@code {
public List<VehicleData> Vehicles = new List<VehicleData>
{
new VehicleData { Text = "Hennessey Venom", Id = "Vehicle-01" },
new VehicleData { Text = "Bugatti Chiron", Id = "Vehicle-02" },
new VehicleData { Text = "Bugatti Veyron Super Sport", Id = "Vehicle-03" },
new VehicleData { Text = "SSC Ultimate Aero", Id = "Vehicle-04" },
new VehicleData { Text = "Koenigsegg CCR", Id = "Vehicle-05" },
new VehicleData { Text = "McLaren F1", Id = "Vehicle-06" },
new VehicleData { Text = "Aston Martin One- 77", Id = "Vehicle-07" },
new VehicleData { Text = "Jaguar XJ220", Id = "Vehicle-08" }
};
public class VehicleData {
public string Text { get; set; }
public string Id { get; set; }
}
private void change(ListBoxChangeEventArgs<string[], VehicleData> args)
{
//Triggers when value changed
}
}
```

Output will be shown as



Hennessey Venom  
Bugatti Chiron  
Bugatti Veyron Super Sport  
SSC Ultimate Aero  
Koenigsegg CCR  
McLaren F1  
Aston Martin One- 77  
Jaguar XJ220

### Enable Scroller in Blazor ListBox Component

The ListBox supports scrolling and it can be achieved by restricting the height of the listbox using [Height](#) property.

In the following sample, **Height** of the listbox is restricted to **290px**.

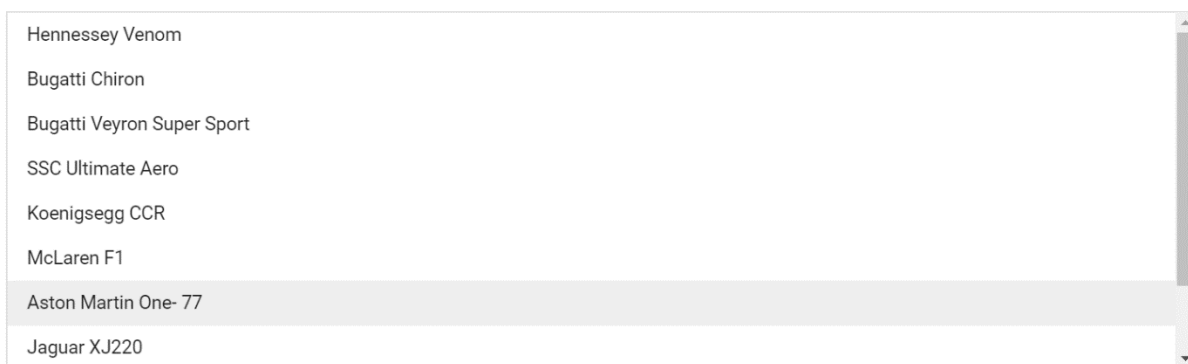
## ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" DataSource="@Vehicles" Height="290px"
TItem="VehicleData">
```



```
<ListBoxFieldSettings Text="Text" Value="Id" />
</SfListBox>
@code {
public List<VehicleData> Vehicles = new List<VehicleData> {
new VehicleData { Text = "Hennessey Venom", Id = "Vehicle-01" },
new VehicleData { Text = "Bugatti Chiron", Id = "Vehicle-02" },
new VehicleData { Text = "Bugatti Veyron Super Sport", Id = "Vehicle-03" },
new VehicleData { Text = "SSC Ultimate Aero", Id = "Vehicle-04" },
new VehicleData { Text = "Koenigsegg CCR", Id = "Vehicle-05" },
new VehicleData { Text = "McLaren F1", Id = "Vehicle-06" },
new VehicleData { Text = "Aston Martin One- 77", Id = "Vehicle-07" },
new VehicleData { Text = "Jaguar XJ220", Id = "Vehicle-08" }
};
public class VehicleData {
public string Text { get; set; }
public string Id { get; set; }
}
}
```

Output will be shown as



### Enable/Disable ListBox in Blazor ListBox Component

To enable or disable items in the listbox, [EnableItems](#) method can be used. In the following example, the **Bugatti Veyron Super Sport** and **SSC Ultimate Aero** items are disabled by default and by clicking **Enable Items** buttons, the disabled items will be enabled.

#### ASPX-CS

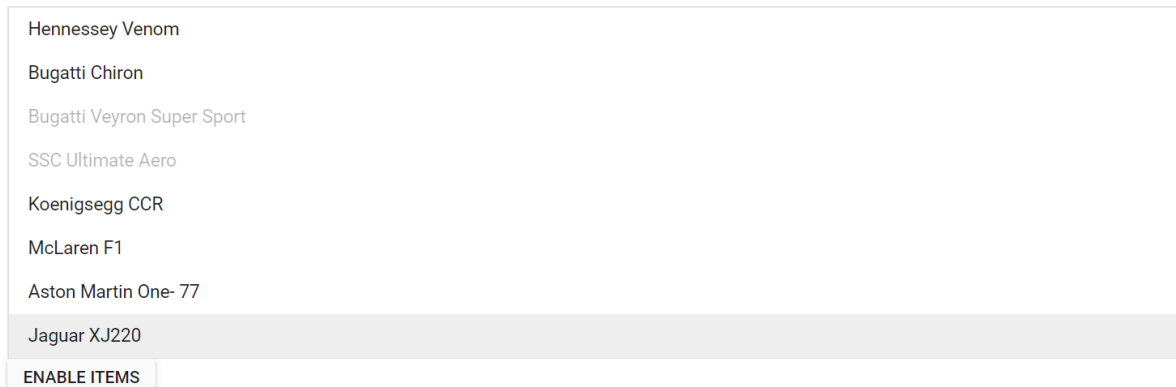
```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Buttons
<SfListBox TValue="string[]" TItem="VehicleData" DataSource="@Vehicles"
@ref="ListBoxObj">
<ListBoxFieldSettings Text="Text" Value="Text" />
<ListBoxEvents TValue="string[]" Created="created"
TItem="VehicleData"></ListBoxEvents>
</SfListBox>
<SfButton @onclick="enableData" Content="Enable Items"></SfButton>
@code {
SfListBox<string[], VehicleData> ListBoxObj;
```

```
public List<VehicleData> Vehicles = new List<VehicleData> {
    new VehicleData { Text = "Hennessey Venom", Id = "Vehicle-01" },
    new VehicleData { Text = "Bugatti Chiron", Id = "Vehicle-02" },
    new VehicleData { Text = "Bugatti Veyron Super Sport", Id = "Vehicle-03" },
    new VehicleData { Text = "SSC Ultimate Aero", Id = "Vehicle-04" },
    new VehicleData { Text = "Koenigsegg CCR", Id = "Vehicle-05" },
    new VehicleData { Text = "McLaren F1", Id = "Vehicle-06" },
    new VehicleData { Text = "Aston Martin One- 77", Id = "Vehicle-07" },
    new VehicleData { Text = "Jaguar XJ220", Id = "Vehicle-08" }
};

public class VehicleData {
    public string Text { get; set; }
    public string Id { get; set; }
}

public string[] Value = new string[] { "Bugatti Veyron Super Sport", "SSC Ultimate Aero" };
private void created(object args)
{
    ListBoxObj.EnableItems(this.Value, false);
}
private void enableData()
{
    ListBoxObj.EnableItems(this.Value, true);
}
}
```

Output will be shown as



### Select Items in Blazor ListBox Component

In the following example, **Bugatti Chiron** is selected using [SelectItems](#) method.

#### ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfListBox TValue="string[]" TItem="VehicleData" DataSource="@Vehicles"
@ref="ListBoxObj">
```

```
<ListBoxEvents TValue="string[]" Created="created"
TItem="VehicleData"></ListBoxEvents>
<ListBoxFieldSettings Text="Text" Value="Text" />
</SfListBox>
@code {
SfListBox<string[],VehicleData> ListBoxObj;
public List<VehicleData> Vehicles = new List<VehicleData> {
new VehicleData { Text = "Hennessey Venom", Id = "Vehicle-01" },
new VehicleData { Text = "Bugatti Chiron", Id = "Vehicle-02" },
new VehicleData { Text = "Bugatti Veyron Super Sport", Id = "Vehicle-03" },
new VehicleData { Text = "SSC Ultimate Aero", Id = "Vehicle-04" },
new VehicleData { Text = "Koenigsegg CCR", Id = "Vehicle-05" },
new VehicleData { Text = "McLaren F1", Id = "Vehicle-06" },
new VehicleData { Text = "Aston Martin One- 77", Id = "Vehicle-07" },
new VehicleData { Text = "Jaguar XJ220", Id = "Vehicle-08" }
};
public class VehicleData {
public string Text { get; set; }
public string Id { get; set; }
}
public string[] Value = new string[] { "Bugatti Chiron" };
private async Task created(object args)
{
await ListBoxObj.SelectItems(this.Value, true);
}
}
```

Output will be shown as



Hennessey Venom
Bugatti Chiron
Bugatti Veyron Super Sport
SSC Ultimate Aero
Koenigsegg CCR
McLaren F1
Aston Martin One- 77
Jaguar XJ220

## ListView

<!-- markdownlint-disable MD024 -->

### Getting Started with Blazor ListView Component

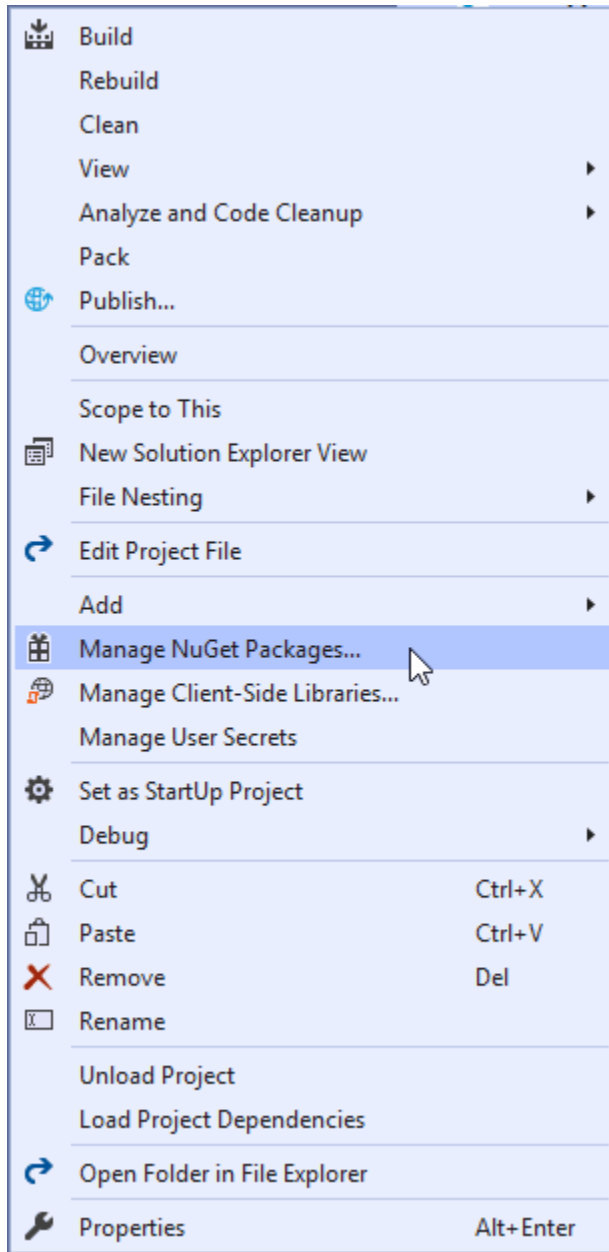
This section briefly explains about how to include a `ListView` in your Blazor server-side application. You can refer

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

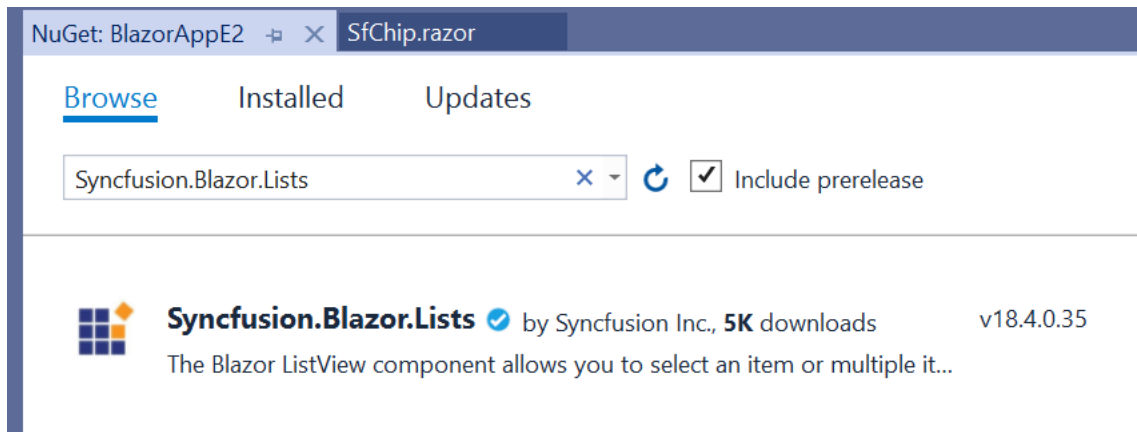
Importing Syncfusion Blazor component in the application

*Using Syncfusion.Blazor NuGet Package [New standard]*

1. Install **Syncfusion.Blazor.Lists** 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.Lists keyword in the Browse tab and install Syncfusion.Blazor.Lists NuGet package in the application.



- Once the installation process is completed, the Syncfusion Blazor Lists 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>
```

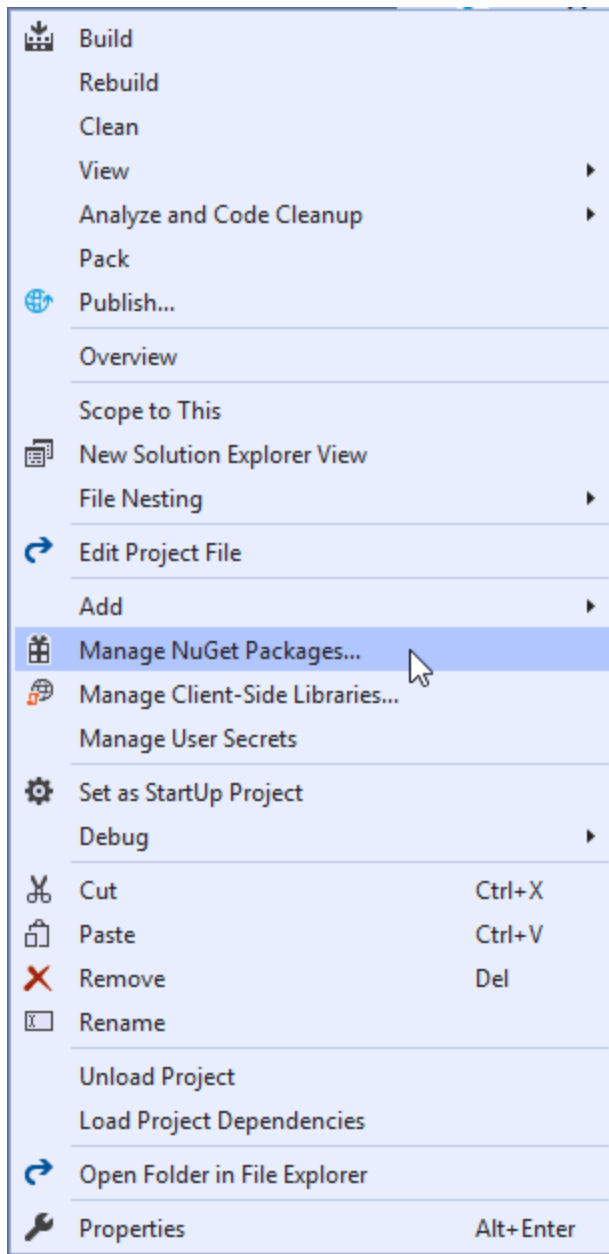
For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

#### HTML

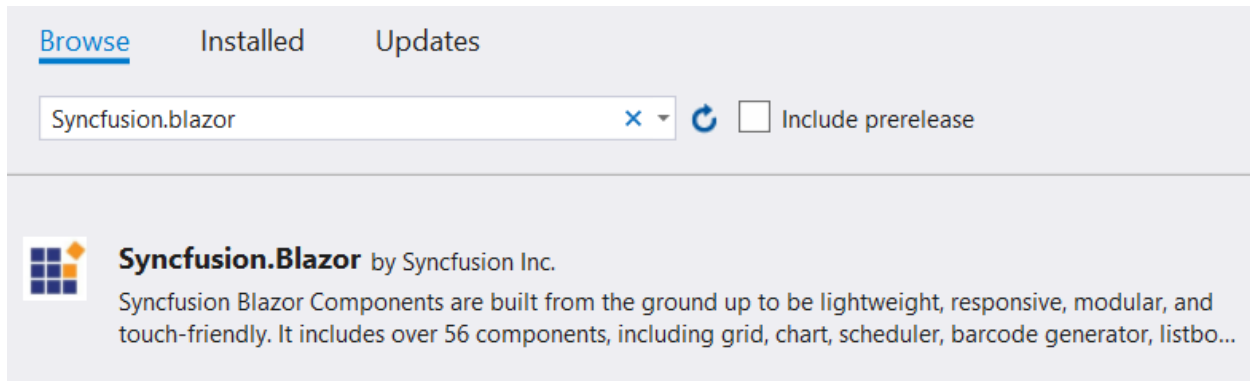
```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

#### Using Syncfusion.Blazor NuGet Package [Old standard]

- 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.



3. Once the installation process is completed, the Syncfusion Blazor 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.

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. You can generate scripts for required components alone using CRG. Refer [here](#) for more details on CRG.

#### HTML

```
<head>  
<script src="https://cdn.syncfusion.com/blazor/{ { site.blazorversion  
}}/syncfusion-blazor.min.js"></script>  
</head>
```

### Adding ListView component namespace to the application

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

#### ASPX-CS

```
@using Syncfusion.Blazor.Lists
```

### Adding ListView component to the application

Add the Syncfusion Blazor ListView component in any web page (razor) in the `Pages` folder. For example, the ListView component is added in the `~/Pages/Index.razor` page.

#### ASPX-CS

```
@using Syncfusion.Blazor.Lists
```

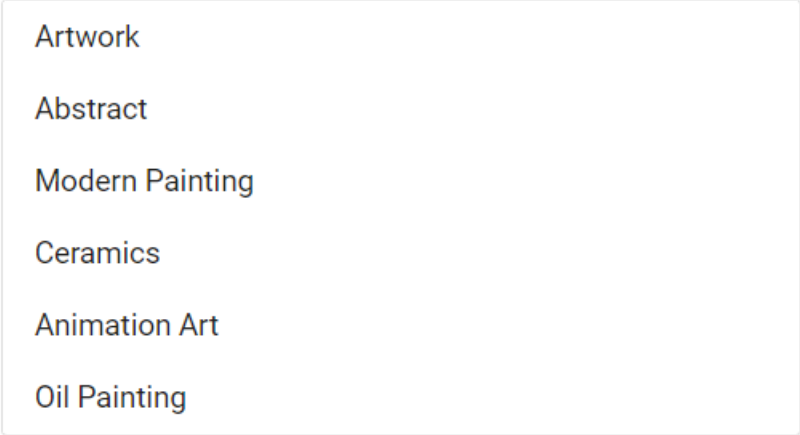


```
<SfListView DataSource="@Data">
<ListViewFieldSettings TValue="DataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
</SfListView>
@code
{
private DataModel[] Data =
{
new DataModel { Text = "ArtWork", Id = "list-01" },
new DataModel { Text = "Abstract", Id = "list-02" },
new DataModel { Text = "Modern Painting", Id = "list-03" },
new DataModel { Text = "Ceramics", Id = "list-04" },
new DataModel { Text = "Animation Art", Id = "list-05" },
new DataModel { Text = "Oil Painting", Id = "list-06" }
};
public class DataModel
{
public string Text { get; set; }
public string Id { get; set; }
}
}
```

Run the application

After successful compilation of your application, simply press **F5** to run the application.

Output be like the below.



Artwork

Abstract

Modern Painting

Ceramics

Animation Art

Oil Painting

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](#)

### Data Binding in Blazor ListView Component

ListView provides an option to load the data either from local dataSource or remote data services. This can be done through the dataSource property that supports the data type of array or DataManager.

ListView supports different kind of data services such as OData, OData V4, and Web API, and data formats like XML, JSON, and, JSONP with the help of DataManager Adaptors.

Fields	Type	Description
Id	string	Specifies ID attribute of list item, mapped in dataSource.
Text	string	Specifies list item display text field.
IsChecked	string	Specifies checked status of list item.
IsVisible	string	Specifies visibility state of list item.
Enabled	string	Specifies enabled state of list item.
IconCss	string	Specifies the icon class of each list item that will be added before to the list item text.
Child	string	Specifies child dataSource fields.
Tooltip	string	Specifies tooltip title text field.
GroupBy	string	Specifies category of each list item.
SortBy	string	Specifies sorting field, that is used to sort the listview data.
HtmlAttributes	string	Specifies list item html attributes field.

When complex data bind to ListView, you should map the ListViewFieldSettings properly. Otherwise, the ListView properties remain as undefined or null.

#### Bind to local data

Local data can be represented in Array of JSON data:

#### Array of JSON data

ListView can generate its list items through an array of complex data. To get it work properly, you should map the appropriate columns to the field property.

#### ASPX-CS

```
@using Syncfusion.Blazor.Lists
<SfListView DataSource="@Data">
  <ListViewFieldSettings TValue="DataModel" Id="Id"
  Text="Text"></ListViewFieldSettings>
</SfListView>
@code {
public string HeaderTitle = "Listview";
List<DataModel> Data = new List<DataModel>();
protected override void OnInitialized()
{
base.OnInitialized();
Data.Add(new DataModel { Text = "Hennessey Venom", Id = "list-01" });
Data.Add(new DataModel { Text = "Bugatti Chiron", Id = "list-02" });
Data.Add(new DataModel { Text = "Bugatti Veyron Super Sport", Id = "list-03" });
Data.Add(new DataModel { Text = "SSC Ultimate Aero", Id = "list-04" });
Data.Add(new DataModel { Text = "Koenigsegg CCR", Id = "list-05" });
Data.Add(new DataModel { Text = "McLaren F1", Id = "list-06" });
}
```

```
Data.Add(new DataModel { Text = "Aston Martin One- 77", Id = "list-07" });
Data.Add(new DataModel { Text = "Jaguar XJ220", Id = "list-08" });
Data.Add(new DataModel { Text = "McLaren P1", Id = "list-09" });
Data.Add(new DataModel { Text = "Ferrari LaFerrari", Id = "list-10" });
}
public class DataModel
{
    public string Id { get; set; }
    public string Text { get; set; }
}
}
```

Output be like the below.

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

#### Bind to remote data

The ListView supports to retrieve the data from remote data services with the help of DataManager control. The Query property allows to fetch data and return it to the ListView from the database.

In the following sample, first 6 products from the Product table of NorthWind data service are displayed.

#### ASPX-CS

```
@using Syncfusion.Blazor.Lists
@using Syncfusion.Blazor.Data
<SfListView HeaderTitle="Products" ShowHeader="true" TValue="Data"
Query="@query">
<ListViewFieldSettings TValue="Data" Id="ProductID"
Text="ProductName"></ListViewFieldSettings>
<SfDataManager Url="https://services.odata.org/V4/Northwind/Northwind.svc/"
Adaptor="Adaptors.ODataV4Adaptor"></SfDataManager>
</SfListView>
```

```
@code {
public static List<string> column = new List<string>()
{
    "ProductID", "ProductName"
};
Query query = new Query().From("Products").Select(column).Take(6);
public class Data
{
    public string ProductID { get; set; }
    public string ProductName { get; set; }
}
}
```

Output be like the below.

## Products

---

Chai

Chang

Aniseed Syrup

Chef Anton's Cajun Seasoning

Chef Anton's Gumbo Mix

Grandma's Boysenberry Spread

## Entity Framework

You need to follow the below steps to consume data from the **Entity Framework** in the ListView component.

*Handle CRUD in data access layer class*

Now add methods **AddProduct**, **DeleteProduct** in the “**DataAccessLayer.cs**” to handle the insert and remove operations respectively. The **CRUD** list items are bound to the **Products** parameter. Please refer the following code.

### C#

```
using System.Collections.Generic;
using System.Linq;
using EFListView.Shared.Models;
using EFListView.Shared.DataAccess;
using Microsoft.EntityFrameworkCore;
namespace EFListView.Shared.DataAccess
{
    public class DataAccessLayer
    {
        DataContext db = new DataContext();
        public DbSet<Products> GetAllProducts()
```

```
{
    try
    {
        return db.Products;
    }
    catch
    {
        throw;
    }
}

public void AddProduct(Products products)
{
    try
    {
        db.Products.Add(products);
        db.SaveChanges();
    }
    catch
    {
        throw;
    }
}

public void DeleteProduct(Products products)
{
    try
    {
        db.Products.Remove(products);
        db.SaveChanges();
    }
    catch
    {
        throw;
    }
}
}
```

#### *Enable CRUD in Web API*

Now you have to create a new **Post** and **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.

#### **CSHARP**

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using EFListView.Shared.DataAccess;
using EFListView.Shared.Models;
namespace EFListView.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ProductsController : ControllerBase
    {

```

```

DataAccessLayer db = new DataAccessLayer();
[HttpGet]
public object Get()
{
    return db.GetAllProducts().ToList();
}
[HttpPost]
public object Post([FromBody]Products product)
{
    db.AddProduct(product);
    return product;
}
[HttpDelete]
public void Delete([FromBody]Products product)
{
    db.DeleteProduct(product);
}
}
}

```

#### *Configure the ListView to perform CRUD operations*

You can perform CRUD operations like Add and Delete by using the **AddItem**, **RemoveItem** methods.

- **AddItem** - Add a new list item into the ListView.
- **RemoveItem** - Delete a selected list item in the ListView.

#### **ASPX-CS**

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Lists
@using Syncfusion.Blazor.Buttons
@using EFListView.Shared.Models
<div class="btn-cls">
    <SfButton @onclick="Add"> Add </SfButton>
    <SfButton @onclick="Delete"> Delete the selected item </SfButton>
</div>
<div class="row">
    <div class="col-md-4">
        <SfListView CssClass="listview" TValue="Products" Height="400px"
        @ref="List">
            <ListViewFieldSettings TValue="Products" Text="ProductName"
            Id="ProductID"></ListViewFieldSettings>
            <SfDataManager Url="api/Products" Adaptor="Adaptors.WebApiAdaptor"
            CrossDomain="true"></SfDataManager>
        </SfListView>
    </div>
</div>
@code{
    SfListView<Products> List;
    List<Products> selectedItems = new List<Products>();
    List<Products> product = new List<Products>()
    {
        new Products{ ProductID = 100, ProductName = "Alice"}
    };
}

```

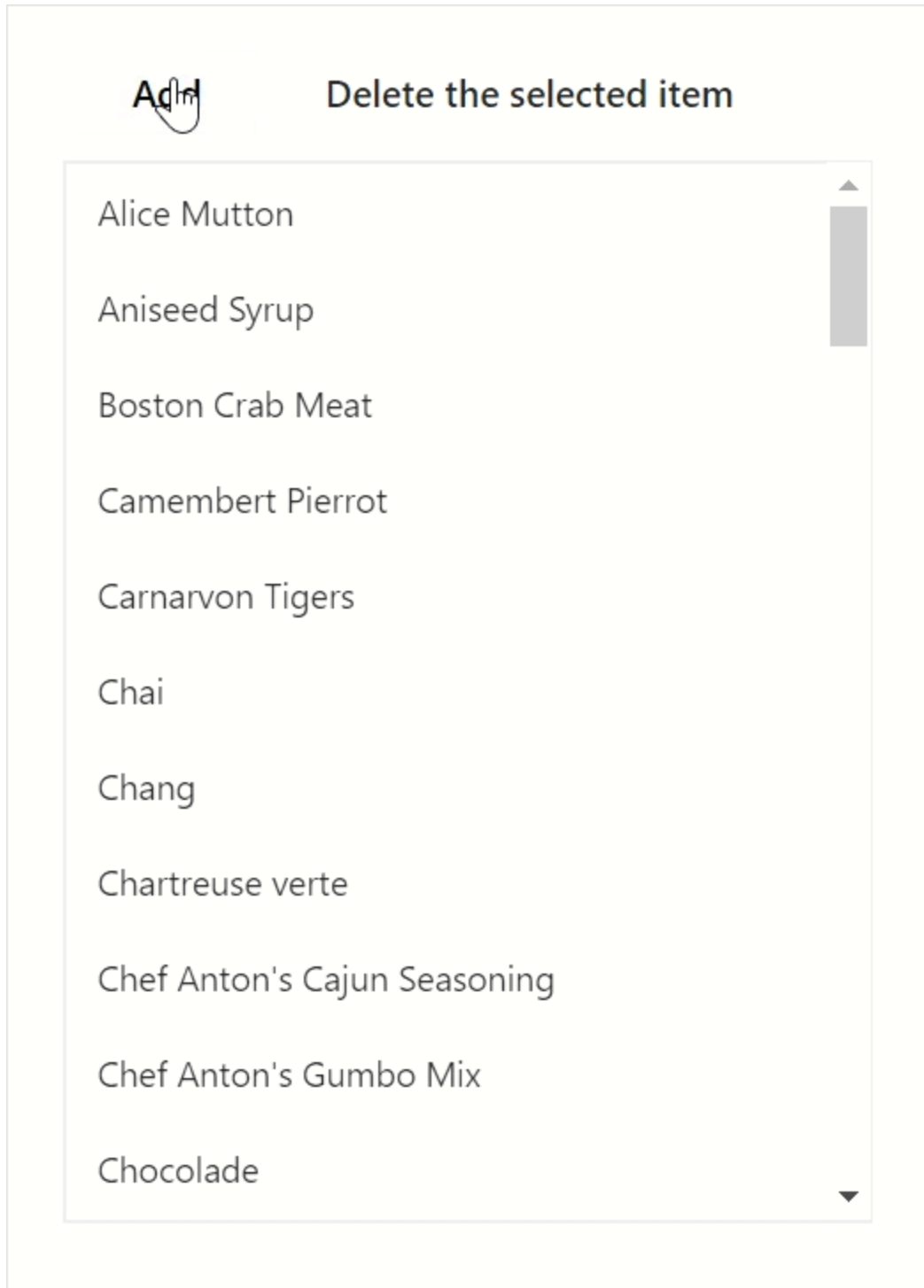
```
public void Add()
{
    this.List.AddItem(product, null);
}
async void Delete()
{
    var items = await this.List.GetSelectedItems();
    if (items.Data != null)
    {
        selectedItems = items.Data;
        Products list = new Products() { ProductID = selectedItems[0].ProductID,
        ProductName = selectedItems[0].ProductName };
        await this.List.RemoveItem(list);
    }
}
}
<style>
.listView {
border: 1px solid #dddddd;
}
.btn-cls {
margin: 0px 5px 10px 5px;
}
</style>
```

---

You can find the fully working sample [here](#).

---

The following GIF represents the ListView with Add, Delete the list items,



### Grouping in Blazor ListView Component

The ListView supports to wrap the nested element into a group based on the category. The category of each list item can be mapped with GroupBy field in the data table, that also supports single-level navigation.

In the following sample, The cars are grouped based on its category by using the GroupBy field in ListViewFieldSettings.



**ASPX-CS**

```
@using Syncfusion.Blazor.Lists
<SfListView DataSource="@DataSource">
<ListViewFieldSettings TValue="DataModel" Id="Id" Text="Text"
GroupBy="Type"></ListViewFieldSettings>
</SfListView>
@code {
public string HeaderTitle = "Listview";
List<DataModel> DataSource = new List<DataModel>()
{
new DataModel { Id = "1", Text = "1", Type = "Odd"},
new DataModel { Id = "2", Text = "2", Type = "Even"},
new DataModel { Id = "3", Text = "3", Type = "Odd"},
new DataModel { Id = "4", Text = "4", Type = "Even"},
};
public class DataModel
{
public string Id { get; set; }
public string Text { get; set; }
public string Type { get; set; }
}
}
```

**Listview****Odd**

1

3

**Even**

2

4

[Check list in Blazor ListView Component](#)

The ListView supports checkbox in default and group-lists which is used to select multiple items. The checkbox can be enabled by the **ShowCheckBox** property.

The Checkbox will be useful in the scenario where we need to select multiple options. For Example, In Shipping cart we can be able to select or unselect the desired items before checkout and also it will be useful in selecting multiple items that belongs to same category using the group list.

**ASPX-CS**

```
@using Syncfusion.Blazor.Lists
<SfListView DataSource="@Data" ShowCheckBox="true">
```

```
<ListViewFieldSettings TValue="DataModel" Id="Id" Text="Text"
IsChecked="IsChecked"></ListViewFieldSettings>
</SfListView>
@code {
private DataModel[] Data = {
new DataModel { Text = "Hennessey Venom", Id = "list-01" },
new DataModel { Text = "Bugatti Chiron", Id = "list-02" },
new DataModel { Text = "Bugatti Veyron Super Sport", Id = "list-03",
IsChecked = true },
new DataModel { Text = "SSC Ultimate Aero", Id = "list-04" },
new DataModel { Text = "Koenigsegg CCR", Id = "list-05" },
new DataModel { Text = "McLaren F1", Id = "list-06" },
new DataModel { Text = "Aston Martin One- 77", Id = "list-07" },
new DataModel { Text = "Jaguar XJ220", Id = "list-08" },
new DataModel { Text = "McLaren P1", Id = "list-09" },
new DataModel { Text = "Ferrari LaFerrari", Id = "list-10" }
};
public class DataModel
{
public string Text { get; set; }
public string Id { get; set; }
public bool IsChecked { get; set; }
}
}
```

<input type="checkbox"/> Hennessey Venom
<input type="checkbox"/> Bugatti Chiron
<input checked="" type="checkbox"/> Bugatti Veyron Super Sport
<input type="checkbox"/> SSC Ultimate Aero
<input type="checkbox"/> Koenigsegg CCR
<input type="checkbox"/> McLaren F1
<input type="checkbox"/> Aston Martin One- 77
<input type="checkbox"/> Jaguar XJ220
<input type="checkbox"/> McLaren P1
<input type="checkbox"/> Ferrari LaFerrari

### Checkbox Position

In ListView the checkbox can be positioned into either **Left** or **Right** side of the list-item text. This can be achieved by **CheckBoxPosition** property. By default, checkbox will be positioned to **Left** of list-item text.

### ASPX-CS

```
@using Syncfusion.Blazor.Lists
<SfListView DataSource="@Data" ShowCheckBox="true"
CheckBoxPosition="CheckBoxPosition.Right">
<ListViewFieldSettings TValue="DataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
</SfListView>
@code {
private DataModel[] Data = {
new DataModel { Text = "Hennessey Venom", Id = "list-01" },
new DataModel { Text = "Bugatti Chiron", Id = "list-02" },
new DataModel { Text = "Bugatti Veyron Super Sport", Id = "list-03"},
new DataModel { Text = "SSC Ultimate Aero", Id = "list-04" },
new DataModel { Text = "Koenigsegg CCR", Id = "list-05" },
new DataModel { Text = "McLaren F1", Id = "list-06" },
new DataModel { Text = "Aston Martin One- 77", Id = "list-07" },
new DataModel { Text = "Jaguar XJ220", Id = "list-08" },
new DataModel { Text = "McLaren P1", Id = "list-09" },
new DataModel { Text = "Ferrari LaFerrari", Id = "list-10" }
};
public class DataModel
{
public string Text { get; set; }
public string Id { get; set; }
}
}
```

## Listview

Hennessey Venom	<input type="checkbox"/>
Bugatti Chiron	<input type="checkbox"/>
Bugatti Veyron Super Sport	<input type="checkbox"/>
SSC Ultimate Aero	<input type="checkbox"/>
Koenigsegg CCR	<input type="checkbox"/>
McLaren F1	<input type="checkbox"/>
Aston Martin One- 77	<input type="checkbox"/>
Jaguar XJ220	<input type="checkbox"/>
McLaren P1	<input type="checkbox"/>
Ferrari LaFerrari	<input type="checkbox"/>

### Nested list in Blazor ListView Component

The ListView component supports Nested list. For that, the child property should be defined for the nested list in the array of JSON.

#### ASPX-CS

```
@using Syncfusion.Blazor.Lists
<SfListView DataSource="@ListData" ShowHeader="true"
HeaderTitle="Continent">
<ListViewFieldSettings TValue="DataModel" Id="Id" Text="Text"
Child="Child"></ListViewFieldSettings>
</SfListView>
@code{
List<DataModel> ListData = new List<DataModel>();
protected override void OnInitialized()
{
base.OnInitialized();
ListData.Add(new DataModel
{
Text = "Asia",
Id = "01",
Category = "Continent",
Child = new List<DataModel>() {
new DataModel {
Text = "India",
Id = "1",
Category = "Asia",
```

```
Child = new List<DataModel> () {
    new DataModel {
        Id = "1001",
        Text = "Delhi",
        Category = "India"
    },
    new DataModel {
        Text = "Kashmir",
        Id = "1002",
        Category = "India"
    },
    new DataModel {
        Text = "Goa",
        Id = "1003",
        Category = "India"
    }
},
new DataModel {
    Text = "China",
    Id = "2",
    Category = "Asia",
    Child = new List<DataModel> () {
        new DataModel {
            Text = "Zhejiang",
            Id = "2001",
            Category = "China"
        },
        new DataModel {
            Text = "Hunan",
            Id = "2002",
            Category = "China"
        },
        new DataModel {
            Text = "Shandong",
            Id = "2003",
            Category = "China"
        }
    }
}
});
ListData.Add(new DataModel
{
    Text = "North America",
    Id = "02",
    Category = "Continent",
    Child = new List<DataModel>() {
        new DataModel {
            Text = "USA",
            Id = "3",
            Category = "North America",
            Child = new List<DataModel> () {
                new DataModel {
                    Text = "California",
                    Id = "3001",
                    Category = "USA"
```

```
    },
    new DataModel {
        Text = "New York",
        Id = "3002",
        Category = "USA"
    },
    new DataModel {
        Text = "Florida",
        Id = "3003",
        Category = "USA"
    }
},
new DataModel {
    Text = "Canada",
    Id = "4",
    Category = "North America",
    Child = new List<DataModel> () {
        new DataModel {
            Text = "Ontario",
            Id = "4001",
            Category = "Canada"
        },
        new DataModel {
            Text = "Alberta",
            Id = "4002",
            Category = "Canada"
        },
        new DataModel {
            Text = "Manitoba",
            Id = "4003",
            Category = "Canada"
        }
    }
},
});
ListData.Add(new DataModel
{
    Text = "Europe",
    Id = "03",
    Category = "Continent",
    Child = new List<DataModel>() {
        new DataModel {
            Text = "Germany",
            Id = "5",
            Category = "Europe",
            Child = new List<DataModel> () {
                new DataModel {
                    Text = "Berlin",
                    Id = "5001",
                    Category = "Germany"
                },
                new DataModel {
                    Text = "Bavaria",
                    Id = "5002",
                    Category = "Germany"
                }
            }
        }
    }
});
```

```
    },
    new DataModel {
        Text = "Hesse",
        Id = "5003",
        Category = "Germany"
    }
},
new DataModel {
    Text = "France",
    Id = "6",
    Category = "Europe",
    Child = new List<DataModel> () {
        new DataModel {
            Text = "Paris",
            Id = "6001",
            Category = "France"
        },
        new DataModel {
            Text = "Lyon",
            Id = "6002",
            Category = "France"
        },
        new DataModel {
            Text = "Marseille",
            Id = "6003",
            Category = "France"
        }
    }
});
}

public class DataModel
{
    public string Id { get; set; }
    public string Text { get; set; }
    public string Category { get; set; }
    public List<DataModel> Child { get; set; }
}
```

## Continent

Asia



North America



Europe



## Customizing Templates in Blazor ListView Component

The ListView component is designed to customize each list items and group title. It uses Blazor **Template engine** to render the elements.

### Header Template

ListView header can be customized with the help of the **HeaderTemplate** property. To customize header template in your application, set your customized template string to **HeaderTemplate** property along with **ShowHeader** property as **true** to display the ListView header.

In the following example, we have rendered ListView with customized header which contains search, add and sort buttons.

### ASPX-CS

```
@using Syncfusion.Blazor.Lists
<SfListView DataSource="@FruitsData" ShowHeader="true">
<ListViewFieldSettings TValue="DataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
<ListViewTemplates TValue="DataModel">
<HeaderTemplate>
<div class="headerContainer">
<span class="fruitHeader">Fruits</span>
</div>
</HeaderTemplate>
</ListViewTemplates>
</SfListView>
@code{
List<DataModel> FruitsData = new List<DataModel>();
protected override void OnInitialized()
{
base.OnInitialized();
FruitsData.Add(new DataModel { Text = "Date", Id = "1" });
FruitsData.Add(new DataModel { Text = "Fig", Id = "2" });
FruitsData.Add(new DataModel { Text = "Apple", Id = "3" });
FruitsData.Add(new DataModel { Text = "Apricot", Id = "4" });
FruitsData.Add(new DataModel { Text = "Grape", Id = "5" });
FruitsData.Add(new DataModel { Text = "Strawberry", Id = "6" });
FruitsData.Add(new DataModel { Text = "Pineapple", Id = "7" });
FruitsData.Add(new DataModel { Text = "Melon", Id = "8" });
FruitsData.Add(new DataModel { Text = "Lemon", Id = "9" });
FruitsData.Add(new DataModel { Text = "Cherry", Id = "10" });
}
public class DataModel
{
public string Text { get; set; }
public string Id { get; set; }
}
}
```



## Fruits

Date  
Fig  
Apple  
Apricot  
Grape  
Strawberry  
Pineapple  
Melon  
Lemon

### Template

ListView items can be customized with the help of the `Template` property. To customize list items in your application, set your customized template string to `Template` property.

The following built-in CSS classes can be used to customize the list-items.

CSS class	Description
<code>e-list-template</code> , <code>e-list-wrapper</code>	These classes are used to differentiate normal and template rendering, which are mandatory for template rendering. The <code>e-list-template</code> class should be added to the root of the ListView element and <code>e-list-wrapper</code> class should be added to the template element wrapper.
<code>e-list-content</code>	This class is used to align list content and it should be added to the content element <pre>&lt;br&gt;&lt;br&gt; &lt;div class="e-list-wrapper"&gt;&lt;br&gt;&lt;b&gt;&lt;span class="e-list-content"&gt;ListItem&lt;/span&gt;&lt;/b&gt; &lt;br&gt;&lt;/div&gt;</pre>
<code>e-list-avatar</code>	This class is used for avatar customization. It should be added to the template element wrapper. After adding it, we can customize our element with <b>Avatar</b> classes <pre>&lt;br&gt;&lt;br&gt; &lt;div class="e-list-wrapper"&gt;&lt;b&gt;e-list-avatar&lt;/b&gt;&lt;/div&gt; &lt;br&gt; &lt;b&gt;&lt;span class="e-avatar e-avatar-circle"&gt;MR&lt;/span&gt;&lt;/b&gt;&lt;br&gt;&lt;span class="e-list-content"&gt;ListItem&lt;/span&gt;&lt;br&gt;&lt;/div&gt;</pre>
<code>e-list-avatar-right</code>	This class is used to align avatar to right side of the list item. It should be added to the template element wrapper. After adding it, we can customize our element with <b>Avatar</b> classes <pre>&lt;br&gt;&lt;br&gt; &lt;div class="e-list-wrapper"&gt;&lt;b&gt;e-list-avatar-right&lt;/b&gt;&lt;/div&gt; &lt;br&gt; &lt;span class="e-list-</pre>

```
content">ListItem</span><br/><b><span class="e-avatar e-avatar-circle">MR</span></b><br/>
</div>|
```

| e-list-badge | This class is used for badge customization .It should be added to the template element wrapper. After adding it, we can customize our element with **Badge** classes <br/><br/> <div class="e-list-wrapper<b>e-list-badge</b>"> <br/> <span class="e-list-content">ListItem</span><br/><b><span class="e-badge e-badge-primary">MR</span></b><br/> </div>|

| e-list-multi-line | This class is used for multi-line customization. It should be added to the template element wrapper. After adding it, we can customize List item's header and description <br/><br/><div class="e-list-wrapper<b>e-list-multi-line</b>"> <br/> <span class="e-list-content">ListItem</span><br/></div>|

| e-list-item-header | This class is used to align a list header and it should be added to the header element along with the multi-line class <br/><br/> <div class="e-list-wrapper<b>e-list-multi-line</b>"><br/> <b><span class="e-list-item-header">ListItem Header</span></b><br/> <span class="e-list-content">ListItem</span><br/></div>|

In the following example, we have customized list items with built-in CSS classes.

#### ASPX-CS

```
@using Syncfusion.Blazor.Lists
<SfListView Id="List"
DataSource="@ListData"
HeaderTitle="Contacts"
ShowHeader="true"
CssClass="e-list-template"
Width="350"
SortOrder="Syncfusion.Blazor.Lists.SortOrder.Ascending">
<ListViewFieldSettings TValue="DataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
<ListViewTemplates TValue="DataModel">
<Template>
<div class="e-list-wrapper e-list-multi-line e-list-avatar">
@if (((context as DataModel).Avatar) != "")
{
<span class="e-avatar e-avatar-circle">@((context as
DataModel).Avatar)</span>
}
else
{
<span class="@((context as DataModel).Pic) e-avatar e-avatar-circle">
</span>
}
<span class="e-list-item-header">@((context as DataModel).Text)</span>
<span class="e-list-content">@((context as DataModel).Contact)</span>
</div>
</Template>
</ListViewTemplates>
</SfListView>
@code{
List<DataModel> ListData = new List<DataModel>();
protected override void OnInitialized()
```

```
{
base.OnInitialized();
ListData.Add(new DataModel
{
Text = "Jenifer",
Contact = "(206) 555-985774",
Id = "1",
Avatar = "J",
Pic = "pic01"
});
ListData.Add(new DataModel
{
Text = "Amenda",
Contact = "(206) 555-3412",
Id = "2",
Avatar = "A",
Pic = ""
});
ListData.Add(new DataModel
{
Text = "Isabella",
Contact = "(206) 555-8122",
Id = "4",
Avatar = "",
Pic = "pic02"
});
ListData.Add(new DataModel
{
Text = "William ",
Contact = "(206) 555-9482",
Id = "5",
Avatar = "W",
Pic = ""
});
ListData.Add(new DataModel
{
Text = "Jacob",
Contact = "(71) 555-4848",
Id = "6",
Avatar = "",
Pic = "pic04"
});
ListData.Add(new DataModel
{
Text = "Matthew",
Contact = "(71) 555-7773",
Id = "7",
Avatar = "M",
Pic = ""
});
ListData.Add(new DataModel
{
Text = "Oliver",
Contact = "(71) 555-5598",
Id = "8",
Avatar = "",
Pic = "pic03"
});
}
```

```
});  
ListData.Add(new DataModel  
{  
    Text = "Charlotte",  
    Contact = "(206) 555-1189",  
    Id = "9",  
    Avatar = "C",  
    Pic = ""  
});  
}  
public class DataModel  
{  
    public string Id { get; set; }  
    public string Text { get; set; }  
    public string Contact { get; set; }  
    public string Avatar { get; set; }  
    public string Pic { get; set; }  
}  
}  
  
<style>  
.e-listview.e-control {  
    width: 400px;  
}  
.pic01 {  
    background-image:  
    url("https://ej2.syncfusion.com/demos/src/grid/images/1.png");  
}  
.pic02 {  
    background-image:  
    url("https://ej2.syncfusion.com/demos/src/grid/images/3.png");  
}  
.pic03 {  
    background-image:  
    url("https://ej2.syncfusion.com/demos/src/grid/images/5.png");  
}  
.pic04 {  
    background-image:  
    url("https://ej2.syncfusion.com/demos/src/grid/images/2.png");  
}  
#List .e-list-item:nth-child(1) .e-avatar {  
    background-color: #039be5;  
}  
#List .e-list-item:nth-child(2) .e-avatar {  
    background-color: #e91e63;  
}  
#List .e-list-item:nth-child(6) .e-avatar {  
    background-color: #009688;  
}  
#List .e-list-item:nth-child(8) .e-avatar {  
    background-color: #000088;  
}  
</style>
```

## Contacts



**Jenifer**

(206) 555-985774



**Amenda**

(206) 555-3412



**Isabella**

(206) 555-8122



**William**

(206) 555-9482



**Jacob**

(71) 555-4848



**Matthew**

(71) 555-7773

### Group template

ListView group header can be customized with the help of the `GroupTemplate` property. To customize the group template in your application, set your customized template string to `GroupTemplate` property.

In the following example, we have grouped ListView based on the category. The category of each list item should be mapped with `GroupBy` field of the data. We have also displayed grouped list items count in the group list header.

### ASPX-CS

```
@using Syncfusion.Blazor.Lists
<SfListView ID="list" DataSource="@ListData" CssClass="e-list-template">
<ListViewFieldSettings Id="Id" Text="Name" TValue="DataModel"
GroupBy="Category"></ListViewFieldSettings>
<ListViewTemplates TValue="DataModel">
<Template>
<div class="e-list-wrapper e-list-multi-line e-list-avatar">
<img class="e-avatar e-avatar-circle" src=@context.Image
style="background:#BCBCBC" />
<span class="e-list-item-header">@context.Name</span>
<span class="e-list-content">@context.Contact</span>
</div>
</Template>
<GroupTemplate>
<div>
<span class="category">Type: @context.Text</span>
</div>
</GroupTemplate>
</ListViewTemplates>
</SfListView>
@code{
List<DataModel> ListData = new List<DataModel>();
protected override void OnInitialized()
{
base.OnInitialized();
ListData.Add(new DataModel { Name = "Nancy", Contact = "(206) 555-985774",
Id = "1", Image = "https://ej2.syncfusion.com/demos/src/grid/images/1.png",
Category = "Experience" });
ListData.Add(new DataModel { Name = "Janet", Contact = "(206) 555-3412", Id
= "2", Image = "https://ej2.syncfusion.com/demos/src/grid/images/3.png",
Category = "Fresher" });
ListData.Add(new DataModel { Name = "Margaret", Contact = "(206) 555-8122",
Id = "4", Image = "https://ej2.syncfusion.com/demos/src/grid/images/4.png",
Category = "Experience" });
ListData.Add(new DataModel { Name = "Andrew ", Contact = "(206) 555-9482",
Id = "5", Image = "https://ej2.syncfusion.com/demos/src/grid/images/2.png",
Category = "Experience" });
ListData.Add(new DataModel { Name = "Steven", Contact = "(71) 555-4848", Id
= "6", Image = "https://ej2.syncfusion.com/demos/src/grid/images/5.png",
Category = "Fresher" });
ListData.Add(new DataModel { Name = "Michael", Contact = "(71) 555-7773", Id
= "7", Image = "https://ej2.syncfusion.com/demos/src/grid/images/6.png",
Category = "Experience" });
}
```

```
ListData.Add(new DataModel { Name = "Robert", Contact = "(71) 555-5598", Id = "8", Image = "https://ej2.syncfusion.com/demos/src/grid/images/7.png", Category = "Fresher" });
ListData.Add(new DataModel { Name = "Laura", Contact = "(206) 555-1189", Id = "9", Image = "https://ej2.syncfusion.com/demos/src/grid/images/8.png", Category = "Experience" });
}

public class DataModel
{
    public string Name { get; set; }
    public string Contact { get; set; }
    public string Id { get; set; }
    public string Image { get; set; }
    public string Category { get; set; }
}

<style>
.e-listview.e-control {
width: 400px;
}
#List .e-list-group-item {
height: 56px;
line-height: 56px;
}
#List .count {
float: right;
}
</style>
```

**Type: Experience****Nancy**

(206) 555-985774

**Margaret**

(206) 555-8122

**Andrew**

(206) 555-9482

**Michael**

(71) 555-7773

**Laura**

(206) 555-1189

**Type: Fresher****Janet**

(206) 555-3412

**Steven**

(71) 555-4848

### Virtualization in Blazor ListView Component

UI virtualization loads only viewable list items in a view port, which will improve the ListView performance while loading a large number of data.

#### Getting started

UI virtualization can be enabled in the ListView by setting the `EnableVirtualization` property to true. It has two types of scrollers as follows:

**Window scroll:** This scroller is used in the ListView by default.

**Container scroll:** This scroller is used, when the height property of the ListView is set.

#### ASPX-CS



```
@using Syncfusion.Blazor.Lists
<SfListView DataSource="@ListData" EnableVirtualization="true">
<ListViewFieldSettings TValue="DataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
</SfListView>
@code{
List<DataModel> ListData = new List<DataModel>();
protected override void OnInitialized()
{
base.OnInitialized();
ListData.Add(new DataModel
{
Text = "Nancy",
Id = "0"
});
ListData.Add(new DataModel
{
Text = "Andrew",
Id = "1"
});
ListData.Add(new DataModel
{
Text = "Janet",
Id = "2"
});
ListData.Add(new DataModel
{
Text = "Margaret",
Id = "3"
});
ListData.Add(new DataModel
{
Text = "Steven",
Id = "4"
});
ListData.Add(new DataModel
{
Text = "Laura",
Id = "5"
});
ListData.Add(new DataModel
{
Text = "Robert",
Id = "6"
});
ListData.Add(new DataModel
{
Text = "Michael",
Id = "7"
});
ListData.Add(new DataModel
{
Text = "Albert",
Id = "8"
});
ListData.Add(new DataModel
{
```

```
Text = "Nolan",  
Id = "9"  
});  
for (int i = 10; i < 1000; i++)  
{  
    int index = new Random().Next(0, 10);  
    ListData.Add(new DataModel  
    {  
        Text =  
ListData[index].GetType().GetProperty("Text").GetValue(ListData[index],  
null).ToString(),  
        Id = i.ToString()  
    });  
}  
}  
  
public class DataModel  
{  
    public string Id { get; set; }  
    public string Text { get; set; }  
}
```

Nancy  
Andrew  
Janet  
Margaret  
Steven  
Laura  
Robert  
Michael  
Albert  
Nolan  
Margaret  
Andrew  
Nancy  
Michael  
Laura  
Janet  
Janet  
Albert

### CSS Structure in Blazor ListView Component

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

#### Customizing ListView

Use the following CSS to customize the ListView.

#### **CSS**

```
.e-listview {  
border: 5px solid rgb(173, 255, 47);  
}
```

### Customizing the list items

Use the following CSS to customize the items of ListView.

#### **CSS**

```
.e-listview .e-list-item {  
    text-align: center;  
    color: pink;  
    background-color: #2fa1ff;  
}
```

### Customizing ListView's header

Use the following CSS to customize the header of ListView control.

#### **CSS**

```
.e-listview .e-list-header{  
    color: #2fa1ff;  
    justify-content: center;  
}
```

### Customizing group header of ListView

Use the following CSS to customize the category of the group items.

#### **CSS**

```
.e-listview .e-list-group-item {  
    color: rgb(173, 255, 47);  
    background-color: maroon;  
    text-align: end;  
}
```

### Customizing the hover state of ListView control

Use the following CSS to customize the list item when hovering.

#### *Customizing ListView hover state with the checkbox checked*

#### **CSS**

```
.e-listview .e-list-item.e-hover.e-active.e-checklist {  
    color: rgb(83, 5, 79);  
    background-color: rgb(173, 255, 47);  
}
```

#### *Customizing ListView hover state*

#### **CSS**

```
.e-listview .e-list-item.e-hover {  
    color:red;  
    background-color: rgb(173, 255, 47);  
}
```

### Customizing selected item of ListView control

Use the following CSS to customize the selected list item.

*Customizing ListView's selected item with the checkbox checked***CSS**

```
.e-listview .e-list-item.e-checklist.e-focused.e-active {
color: rgb(83, 5, 79);
background-color: rgb(0, 15, 100);
}
```

*Customizing ListView's selected item***CSS**

```
.e-listview .e-list-item.e-focused {
color: #2fa1ff;
background-color: rgb(0, 15, 100);
}
```

## Accessibility in Blazor ListView Component

## Keyboard interaction

The following key shortcuts are used to access the ListView control without any interruption.

| Keyboard shortcuts | Actions |

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

| Arrow Up | Move to the previous list item. |

| Arrow Down | Move to the next list item. |

| Select | Select the targeted list from the whole list. |

| Back | Get back to the previous lists if it is in nested list. |

**ASPX-CS**

```
@using Syncfusion.Blazor.Lists
@using Syncfusion.Blazor.Data
<SfListView DataSource="@ListData" ShowHeader="true"
HeaderTitle="Continent">
<ListViewFieldSettings TValue="DataModel" Id="Id" Text="Text"
Child="Child"></ListViewFieldSettings>
</SfListView>
@code {
List<DataModel> ListData = new List<DataModel>();
protected override void OnInitialized()
{
base.OnInitialized();
ListData.Add(new DataModel
{
Text = "Asia",
Id = "01",
Category = "Continent",
Child = new List<DataModel>() {
new DataModel {
Text = "India",
Id = "1",
Category = "Asia",
```

```
Child = new List<DataModel> () {
    new DataModel {
        Id = "1001",
        Text = "Delhi",
        Category = "India"
    },
    new DataModel {
        Text = "Kashmir",
        Id = "1002",
        Category = "India"
    },
    new DataModel {
        Text = "Goa",
        Id = "1003",
        Category = "India"
    }
},
new DataModel {
    Text = "China",
    Id = "2",
    Category = "Asia",
    Child = new List<DataModel> () {
        new DataModel {
            Text = "Zhejiang",
            Id = "2001",
            Category = "China"
        },
        new DataModel {
            Text = "Hunan",
            Id = "2002",
            Category = "China"
        },
        new DataModel {
            Text = "Shandong",
            Id = "2003",
            Category = "China"
        }
    }
}
});
ListData.Add(new DataModel
{
    Text = "North America",
    Id = "02",
    Category = "Continent",
    Child = new List<DataModel>() {
        new DataModel {
            Text = "USA",
            Id = "3",
            Category = "North America",
            Child = new List<DataModel> () {
                new DataModel {
                    Text = "California",
                    Id = "3001",
                    Category = "USA"
```

```
    },
    new DataModel {
        Text = "New York",
        Id = "3002",
        Category = "USA"
    },
    new DataModel {
        Text = "Florida",
        Id = "3003",
        Category = "USA"
    }
},
new DataModel {
    Text = "Canada",
    Id = "4",
    Category = "North America",
    Child = new List<DataModel> () {
        new DataModel {
            Text = "Ontario",
            Id = "4001",
            Category = "Canada"
        },
        new DataModel {
            Text = "Alberta",
            Id = "4002",
            Category = "Canada"
        },
        new DataModel {
            Text = "Manitoba",
            Id = "4003",
            Category = "Canada"
        }
    }
}
});
ListData.Add(new DataModel
{
    Text = "Europe",
    Id = "03",
    Category = "Continent",
    Child = new List<DataModel>() {
        new DataModel {
            Text = "Germany",
            Id = "5",
            Category = "Europe",
            Child = new List<DataModel> () {
                new DataModel {
                    Text = "Berlin",
                    Id = "5001",
                    Category = "Germany"
                },
                new DataModel {
                    Text = "Bavaria",
                    Id = "5002",
                    Category = "Germany"
                }
            }
        }
    }
});
```

```

    },
    new DataModel {
        Text = "Hesse",
        Id = "5003",
        Category = "Germany"
    }
    },
    new DataModel {
        Text = "France",
        Id = "6",
        Category = "Europe",
        Child = new List<DataModel> () {
            new DataModel {
                Text = "Paris",
                Id = "6001",
                Category = "France"
            },
            new DataModel {
                Text = "Lyon",
                Id = "6002",
                Category = "France"
            },
            new DataModel {
                Text = "Marseille",
                Id = "6003",
                Category = "France"
            }
        }
    }
    });
}

public class DataModel
{
    public string Id { get; set; }
    public string Text { get; set; }
    public string Category { get; set; }
    public List<DataModel> Child { get; set; }
}

```

### ARIA attributes

The following ARIA attributes are applicable for ListView control based on its state.

| Properties | Functionality |

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

| aria-selected | It indicates the selected list from the whole list. |

| aria-level | It defines the hierarchical structure of a list item. |



## How To

### Get selected items from Blazor ListView Component

Single or many items can be selected by users in the ListView control. An API is used to get selected items from the list items. This is called as the `GetCheckedItemsAsync` method.

#### GetCheckedItemsAsync method

Return type   Purpose
----- -----
Data   Returns the collections of list items data
Index   Returns the index of the selected item (applicable only in Virtualization)
ParentId   Returns the currently selected item's Parent Id (applicable only in Nested List)
Text   Returns array of text of selected item lists

#### ASPX-CS

```
@using Syncfusion.Blazor.Lists
<div style="display: flex">
<div class="margin">
<SfListView @ref="@SfList"
DataSource="@DataSource"
ShowCheckBox="true">
<ListViewFieldSettings TValue="ListDataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
</SfListView>
</div>
<div class="margin">
<div class="padding">
<button class="e-btn" @onclick="@OnSelect">Get Selected Items</button>
</div>
<div class="padding">
<table>
<tr>
<th>Text</th>
<th>Id</th>
</tr>
@foreach (var item in SelectedItems)
{
<tr>
<td>@item.Text</td>
<td>@item.Id</td>
</tr>
}
</table>
</div>
</div>
</div>
@code
{
SfListView<ListDataModel> SfList;
List<ListDataModel> SelectedItems = new List<ListDataModel>();
List<ListDataModel> DataSource = new List<ListDataModel>()
{
```

```
new ListDataModel{ Id = "1", Text = "Artwork"},
new ListDataModel{ Id = "2", Text = "Abstract"},
new ListDataModel{ Id = "3", Text = "Modern Painting"},
new ListDataModel{ Id = "4", Text = "Ceramics"},
new ListDataModel{ Id = "5", Text = "Animation Art"},
new ListDataModel{ Id = "6", Text = "Oil Painting"},
};
async void OnSelect()
{
    var items = await SfList.GetCheckedItemsAsync();
    if (items.Data != null)
    {
        SelectedItems = items.Data;
        this.StateHasChanged();
    }
}
public class ListDataModel
{
    public string Id { get; set; }
    public string Text { get; set; }
}
<style>
.margin {
margin: 10px;
width: 300px;
}
.padding {
padding: 10px 0;
}
table {
width: 100%;
}
</style>
```

☐ Artwork☐ Abstract☒ Modern Painting☐ Ceramics☒ Animation Art☐ Oil PaintingGET SELECTED ITEMS

Text	Id
Modern Painting	3
Animation Art	5

Add and remove list items in Blazor ListView Component

You can add or remove list items from the ListView control using the `ObservableCollection`.

Refer to the following steps to add or remove a list item.

- Bind the `onclick` handler to the delete icon created in step 1. Within the click event, remove the list item by passing the

delete icon list item to `OnDelete` method.

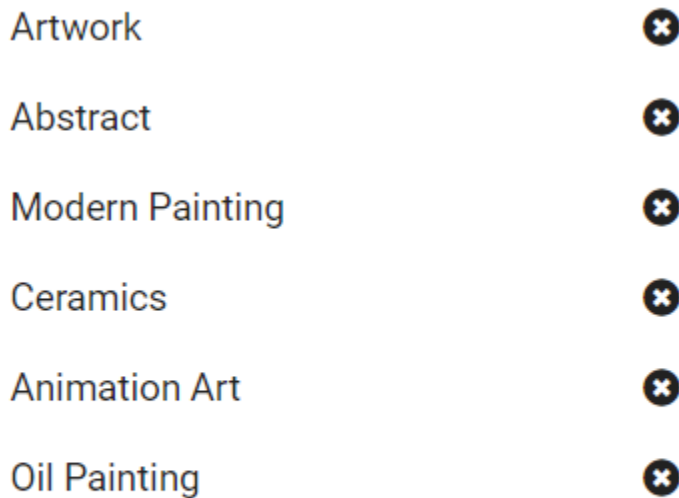
### ASPX-CS

```
@using Syncfusion.Blazor.Lists
@using System.Collections.ObjectModel
<div class="flex">
<div class="margin">
<SfListView ID="sample-list-flat" DataSource="@DataSource">
<ListViewFieldSettings TValue="ListDataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
<ListViewTemplates TValue="ListDataModel">
<Template>
@{
ListDataModel item = context as ListDataModel;
<div class="text-content">
@item.Text
<span class="delete-icon" @onclick="@(() => { OnDelete(item); })"></span>
</div>
}
</Template>
</ListViewTemplates>
</SfListView>
</div>
</div>
<div class="flex">
<button class="e-btn" @onclick="@AddItem">Add item</button>
</div>
@code
{
ObservableCollection<ListDataModel> DataSource = new
ObservableCollection<ListDataModel>()
{
new ListDataModel{ Id = "1", Text = "Artwork"},
new ListDataModel{ Id = "2", Text = "Abstract"},
new ListDataModel{ Id = "3", Text = "Modern Painting"},
new ListDataModel{ Id = "4", Text = "Ceramics"},
new ListDataModel{ Id = "5", Text = "Animation Art"},
new ListDataModel{ Id = "6", Text = "Oil Painting"},
};
void OnDelete(ListDataModel listDataModel)
{
DataSource.RemoveAt(DataSource.ToList<ListDataModel>().FindIndex(e => e.Id
== listDataModel.Id));
}
void AddItem()
{
var random = new Random();
DataSource.Add(new ListDataModel
{
Id = random.Next(100, 300).ToString(),
```

```

Text = "Item " + random.Next(100, 300).ToString(),
});
}
public class ListDataModel
{
public string Id { get; set; }
public string Text { get; set; }
}
}
<style>
.flex {
display: flex;
justify-content: center;
}
.margin {
margin: 10px;
width: 300px;
}
#sample-list-flat.e-listview .e-content .delete-icon::after {
font-family: "e-icon";
content: "\e700";
float: right;
cursor: pointer;
}
@@font-face {
font-family: "e-icon";
src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMjltSfIAAAEoAAAAVmNtYXNlEODVAAABiAAAADZnbHlmXOn
iGAAAACgAAAFaAGVhZBC1AhkAAADQAAAAANmhoZWEIUQQDAAArAAAACRobXR4CAAAAAAAYAAAAA
IbG9jYQCgAAAAAAHAAAAABmlheHABDgCYAAABCAAAACBuYw1lv4Bt4QAAAwgAAAIzcG9zdJx8QW4
AAAUkAAAAOwABAAAEAAAAAFwEAAAAAAD9AABAAAAAAGABAAAAQAAPwCDV18
PPPUACwQAAAAANbRXpQAAAAA1tFelAAAAAD9AP0AAAAACAACAAAAAEEAAACAIwAAgAAAAA
AAgAAAAoACgAAAP8AAAAAQAQAAZAABQAAaokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAACAAAAAAwAAABQAAwABAAAFAAEACIAAAEAQAQAQA5wD//wAA5wD//wAAAAEABAAAAEAAAA
AAAAAoAAAAIAAAAA/QD9AALAIsAAAEHFwcnByc3JzcXNwUfHz8fLx8PHgLuhIRrg4NrhIRrg4P
9iQECaWQGBwcJCwsMDQ4PDxEREhMUFBUWFhcXFxkYGRkaGhkZGBkXFxcWFhUUFBMSEREPdW4NDAs
LCQcHBgQDAgEBAgMEBgCHCQsLDA0ODw8RERITFBQVFhYXFxcZGBkZGhoZGRgZFxcXFhYVFBQTEhE
RDw8ODQwLcwkHBwYEAwICg4OGa4SEa4ODaoCE7hoZGRgZFxcXFhYVFBQTEhERDw8ODQwLcwkHBwY
EAwIBAQIDBAYHBwLcwwNDg8PERESExQUFRYWFxcXGRgZGRoaGRkYGRcXFxYWFQRUEXIREQ8PDg0
MCwsJBwcGBAMCAQECAWQGBwcJCwsMDQ4PDxEREhMUFBUWFhcXFxkYGRkAAAAASAN4AAQAAAAA
BAAAAQAAAAAQAAGAAEAQAAAAAAGAAHAAcAAQAAAAAAwAGAA4AAQAAAAAABAAGABQAAQAAAAA
ABQALABoAAQAAAAAABgAGACUAAQAAAAAACgAsACsAAQAAAAACwASAFcAAwABBAkAAAAACAGkAAwA
BBakAAQAMAGsAAwABBAkAAgAOAHcAAwABBAkAAwAMAIUAAwABBAkABAAMAJEAAwABBAkABQAWAJ0
AAwABBAkABgAMALMAAwABBAkACgBYAL8AAwABBAkACwAkARcgZGVsZXRLUmVndWxhcmRlbGV0ZWR
lbGV0ZVZlc3luY2Zlc2lubi5jb20AIABkAGUAbABLAHQAZQBSAGUAZwB1AGwAYQByAGQ
AZQBsAGUAdABLAGQAZQBsAGUAdABLAfYAZQByAHMAaQBvAG4AIAAxAAC4AMABkAGUAbABLAHQAZQ
BAG8ABgB0ACAAZwB1AG4AZQByAGEAdABLAGQAIAAB1AHMAaQBvAGcAIABTAKAbgBjAGYAdQBzAGk
AbwBuACAATQB1AHQAcgBvACAAUwB0AHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwB
uAC4AYwBvAG0AAAAAGAAAAAAKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAQIBAwARY2lyY2x
lLWNsb3NlLS0tMDIAAAA=) format("trueType");
font-weight: normal;
font-style: normal;
}
</style>

```



ADD ITEM

#### Use dynamic templates in Blazor ListView based on device

The Syncfusion Essential Blazor controls are desktop and mobile-friendly. So, you can use Syncfusion controls in both modes. The control templates are not always fixed. Applications may need to load various templates depending upon the device.

#### Integration

In the ListView control, template support is being used. In some cases, the control wrapper is always responsive across all devices, but the template contents are dynamically changed with unspecified (sample side) dimensions. CSS customization is also needed in sample-side to align template content responsively in both mobile and desktop modes. Here, two templates have been loaded for mobile and desktop modes. To check the device mode, we can use the Microsoft.AspNetCore.Http package and check for the UserAgent to detect mobile or desktop.

#### ASPX-CS

```
@using Syncfusion.Blazor.Lists
@using Microsoft.AspNetCore.Http
@inject IHttpContextAccessor httpContextAccessor
<div class="flex flex__center">
<div class="margin">
<SfListView DataSource="@DataSource"
ShowHeader="true"
HeaderTitle="Syncfusion Blog"
CssClass="e-list-template">
<ListViewFieldSettings TValue="ListDataModel" Id="Id"
Text="Name"></ListViewFieldSettings>
<ListViewTemplates TValue="ListDataModel">
<Template>
@{
ListDataModel item = context as ListDataModel;
```

```

<div class="post flex e-list-wrapper e-list-multi-line">
<div class="flex image vertical__center flex__1">

</div>
<div class="flex vertical flex__center flex__4">
<div class="flex">
<div class="bold flex__1">@item.Name</div>
@if (IsMobile)
{
<div id="list-logo">
<span class="bookmark" title="We can customize this element to perform our
own action"></span>
<span class="comments" title="We can customize this element to perform our
own action"></span>
<span class="share" title="We can customize this element to perform our own
action"></span>
</div>
}
</div>
<div class="small__font">@item.Content</div>
<div class="flex">
<div class="timeStamp flex__1">@item.TimeStamp</div>
@if (!IsMobile)
{
<div id="list-logo flex__2">
<span class="bookmark" title="We can customize this element to perform our
own action"></span>
<span class="comments" title="We can customize this element to perform our
own action"></span>
<span class="share" title="We can customize this element to perform our own
action"></span>
</div>
}
</div>
</div>
</div>
}
</Template>
</ListViewTemplates>
</SfListView>
</div>
</div>
@code
{
bool IsMobile;
List<ListDataModel> DataSource = new List<ListDataModel>()
{
new ListDataModel{ Name = "IBM Open-Sources Web Sphere Liberty Code",
Content = "In September, IBM announced that it would be open-sourcing the
code for WebSphere...", Id = "1", Image =
"https://ej2.syncfusion.com/demos/src/listview/images/1.png", TimeStamp =
"Syncfusion Blog - October 19, 2017" },
new ListDataModel{ Name = "Must Reads: 5 Big Data E-books to upend your
development", Content = "Our first e-book was published in May 2012-jQuery
Succinctly was the start of over...", Id = "2", Image =
"https://ej2.syncfusion.com/demos/src/listview/images/2.png", TimeStamp =
"Syncfusion Blog - October 18, 2017" },

```

```

new ListDataModel{ Name = "The Syncfusion Global License: Your Questions,
Answered ", Content = "Syncfusion recently hosted a webinar to cover the
ins and outs of the Syncfusion global...", Id = "4", Image =
"https://ej2.syncfusion.com/demos/src/listview/images/3.png", Timestamp =
"Syncfusion Blog - October 18, 2017" },
new ListDataModel{ Name = "Know = What is Coming from Microsoft this Fall
", Content = "On October 17, Microsoft will release its Fall Creators
Update for the Windows 10 platform...", Id = "5", Image =
"https://ej2.syncfusion.com/demos/src/listview/images/6.png", Timestamp =
"Syncfusion Blog - October 17, 2017"},
};
protected override void OnInitialized()
{
    var userAgent = httpContextAccessor.HttpContext.Request.Headers["User-
Agent"];
    IsMobile = (userAgent[0] as string).ToLower().Contains("mobile");
}
public class ListDataModel
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Timestamp { get; set; }
    public string Image { get; set; }
    public string Content { get; set; }
}
}
<style>
.flex {
display: flex;
}
.flex__center {
justify-content: center;
}
.vertical__center {
align-items: center;
}
.vertical {
flex-direction: column;
}
.flex__1 {
flex: 1;
}
.flex__2 {
flex: 2;
}
.flex__3 {
flex: 3;
}
.flex__4 {
flex: 4;
}
.bold {
font-weight: 500;
}
.margin {
margin: 10px;
width: 350px;
}

```

```
.timeStamp {
font-size: 10px;
padding: 4px 0;
}
.small_font {
font-size: 13px;
margin: 2px 0;
}
.e-listview .bookmark::before {
content: "\e700";
font-size: 14px;
}
.e-listview .share::before {
content: "\e701";
font-size: 14px;
}
.e-listview .comments::before {
content: "\e703";
font-size: 14px;
}
.e-listview .bookmark::before,
.e-listview .share::before,
.e-listview .comments::before {
color: grey;
font-family: 'Bookmarks';
margin-left: 3px;
cursor: pointer;
}
@@font-face {
font-family: 'Bookmarks';
src: url(data:application/x-font-ttf;charset=utf-8;base64,AAEAAAAKAIAAAwAgTlMvMj0gSRgAAAEoAAAAMnTYXDO185qAAABkAAAAAJnbHlmRXCI8wAAAEAAAAFkaGVhZA8SahsAADQAAAAANmhoZWEHMqNTAAAArAAAACRobXR4D7gAAAAAAAAAAAAAQBg9jYQDwaIAAAAAHUAAAAcmlheHABEQAYAAABCAAAACBuYW1lFuNPLWAAA0QAAAI9cG9zdLaVZAwAAWEAAAAAXQABAAADUv9qAFoEAAAA//4D6gABAAAAAAAAAAAAAAAAAABAABAAAAQAAGHTc9V8PPPUACwPoAAAAANYFEqYAAAAAlgUSpgAAAAAD6gPqAAAAACAACAAAAAAAAAAAAEAACAAAwAAAAAAGAAAAOACGAAP8AAAAAAAAAAQPuAZAABQAAAnoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMMAAA
AAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnAwNS/2oAWgPqAJYAAAAABAAAAAAAAABAAAAAPoAAA
D6AAAA+gAAAAAAAAIAAADAAAAFAADAEEAAAAUAQAALgAAAAAYABAABAAALnAecD//8AAOcA5wP//wA
AAAAAQAGAAGAAAAABAAIAAAwAAAAAAAAA+AIAAsgAAAAAMAAAAAAxwD6gANABkAJQAAExE3FxEHLgEnNDcjDgElMxUzFSMVIzUjNTMHhgEXPgE3LgEnDgHQ190MWXcCCWU0RAGWKFBQKFBQLJdRkZdAQFdRkZdAwn8+fn5AnMBAndZHx0BRWhQKFBQKA5GXQICXUZGXQEBOXAAAAABAAAAAAPqA+oAJAACQEuASMOAQceARcyNjCBHgEXPgE3LgIHCQEWmz4BNy4BJw4Barn+Qxm1HD5WAgtJTQRwyEWHGC1I5PlUBAVOCKf5YabUmND5WAQFWFPkfUA2T+7hesAKo3OUwBEQ7+6zJAAGJLOTpLASUBBgEMHAFLOTpLAQFLAAACAAAAAAPqA4EADwAcAAABHGEXmjcxJz4BNS4BJw4BBTMVNzMnJjU+ATclIQIOA4ZlFROGLzM8AoZmZYb98YWBvgIRBLOG/QYBVGWHAgrmhyBpQGWWagOW0SLCBza2h7MDiAAAAAASAN4AAQAAAA
AAAAABAAAAAQAAAAAAQAJAEEAAQAAAAAAAgAHAAoAAQAAAAAAAwAJABEAAQAAAAAABAAJABoAAQA
AAAAABQALACMAAQAAAAAABgAJAC4AAQAAAAAACGsAdCAAQAAAAAACWASAGMAAwABBakAAAAACAHU
AAwABBakAAQASAhCAAwABBakAAgAOAiKAawABBakAAwASAjCAAwABBakABAASAKKaAwABBakABQA
WALSAAwABBakABgASANEAAwABBakACgBYAOMAawABBakACwAkATsgQm9va2lhcmmtzUmVndWxhcKJvb2tYXJrc0Jvb2tYXJrc1Zlcnpbn24gMS4wQm9va2lhcmmtzRm9udCBnZW5lcmF0ZWQgdXNpbmcgU3luY2Zlc2lwbibNZXRYbyBTdHVkaW93d3cuc3luY2Zlc2lwbibi5jb20AIBACAG8AbwBrAG0AYQB
yAGsAcwBSAGUAzwBlAGwAYQByAEIABwBvAGsAbQBhAHIAawBzAEIABwBvAGsAbQBhAHIAawBzAfY
AZQByAHMAAQbvAG4AIAAxAC4AMABCAG8AbwBrAG0AYQByAGsAcwBGAG8AbgB0ACAAZwBlAG4AZQB
yAGEAdABlAGQAiABlAHMAAQbuAGcAIABTAHkAbgBjAGYAdQBzAGkAbwBuACAATQB1AHQAcgBvACA
AUwB0AHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYwBvAG0AAAAAAgAAAAA
```



```
AAAAKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEAQIBAwEEAQUADGJvb2ttYXJrLWFkZApzaGFyZS0  
tLTaxF21lc3NhZ2VzLWluZm9ybWFOaW9uLTaxAAAAAA=) format('truetype');  
font-weight: normal;  
font-style: normal;  
}  
</style>
```

## Syncfusion Blog



### IBM Open-Sources Web Sphere Liberty Code

In September, IBM announced that it would be open-sourcing the code for WebSphere...

Syncfusion Blog - October 19, 2017



### Must Reads: 5 Big Data E-books to upend your development

Our first e-book was published in May 2012- jQuery Succinctly was the start of over...

Syncfusion Blog - October 18, 2017



### The Syncfusion Global License: Your Questions, Answered

Syncfusion recently hosted a webinar to cover the ins and outs of the Syncfusion global...

Syncfusion Blog - October 18, 2017



### Know = What is Coming from Microsoft this Fall

On October 17, Microsoft will release its Fall Creators Update for the Windows 10 platform...

Syncfusion Blog - October 17, 2017



### Create mobile contact layout using Blazor ListView

You can customize the ListView using the `Template` property. Refer to the following steps to customize ListView as mobile contact view with our `avatar`.

- Render the ListView with `DataSource` that has avatar data. You can set avatar data as either text or class names. Refer to the following codes.

#### C#

```
new ListDataModel {  
    Text = "Amenda",  
    Contact = "(206) 555-3412",  
    Id = "2",  
    Avatar = "A",  
    Pic = ""  
},
```

- Set `avatar` classes in ListView template to customize contact icon. In the following codes, medium size avatar has been set using the class name `e-avatar e-avatar-circle` from data source.

#### ASPX-CS

```
ListDataModel item = context as ListDataModel;  
<div class="e-list-wrapper e-list-multi-line e-list-avatar">  
    @if (item.Avatar != "")  
    {  
        <span class="e-avatar e-avatar-circle">@item.Avatar</span>  
    }  
    else  
    {  
        <span class="@item.Pic e-avatar e-avatar-circle"> </span>  
    }  
    <span class="e-list-item-header">@item.Text</span>  
    <span class="e-list-content">@item.Contact</span>  
</div>
```

- Sort the contact names using the `SortOrder` property of ListView.
- Enable the `ShowHeader` property, and set the `HeaderTitle` as `Contacts`.

#### ASPX-CS

```
@using Syncfusion.Blazor.Lists  
<div class="flex flex__center">  
    <div class="margin">  
        <SfListView DataSource="@DataSource"  
            ShowHeader="true"  
            HeaderTitle="Contacts"  
            CssClass="e-list-template"  
            SortOrder="Syncfusion.Blazor.Lists.SortOrder.Ascending">
```

```

<ListViewFieldSettings TValue="ListDataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
<ListViewTemplates TValue="ListDataModel">
<Template>
@{
ListDataModel item = context as ListDataModel;
<div class="e-list-wrapper e-list-multi-line e-list-avatar">
@if (item.Avatar != "")
{
<span class="e-avatar e-avatar-circle">@item.Avatar</span>
}
else
{
<span class="@item.Pic e-avatar e-avatar-circle"> </span>
}
<span class="e-list-item-header">@item.Text</span>
<span class="e-list-content">@item.Contact</span>
</div>
}
</Template>
</ListViewTemplates>
</SfListView>
</div>
</div>
@code
{
List<ListDataModel> DataSource = new List<ListDataModel>() {
new ListDataModel {
Text = "Jenifer",
Contact = "(206) 555-985774",
Id = "1",
Avatar = "JE",
},
new ListDataModel {
Text = "Amenda",
Contact = "(206) 555-3412",
Id = "2",
Avatar = "AM",
},
new ListDataModel {
Text = "Isabella",
Contact = "(206) 555-8122",
Id = "4",
Avatar = "IS",
},
new ListDataModel {
Text = "William ",
Contact = "(206) 555-9482",
Id = "5",
Avatar = "WI",
},
new ListDataModel {
Text = "Jacob",
Contact = "(71) 555-4848",
Id = "6",
Avatar = "JA",
},
},

```

```
new ListDataModel {
    Text = "Matthew",
    Contact = "(71) 555-7773",
    Id = "7",
    Avatar = "MA",
},
new ListDataModel {
    Text = "Oliver",
    Contact = "(71) 555-5598",
    Id = "8",
    Avatar = "OL",
},
new ListDataModel {
    Text = "Charlotte",
    Contact = "(206) 555-1189",
    Id = "9",
    Avatar = "CH",
}
};
public class ListDataModel
{
    public string Id
    {
        get;
        set;
    }
    public string Text
    {
        get;
        set;
    }
    public string Avatar
    {
        get;
        set;
    }
    public string Pic
    {
        get;
        set;
    }
    public string Contact
    {
        get;
        set;
    }
}
<style>
.flex {
display: flex;
}
.flex__center {
justify-content: center;
}
.vertical__center {
align-items: center;
```

```
}  
.vertical {  
flex-direction: column;  
}  
.flex__1 {  
flex: 1;  
}  
.flex__2 {  
flex: 2;  
}  
.flex__3 {  
flex: 3;  
}  
.flex__4 {  
flex: 4;  
}  
.bold {  
font-weight: 500;  
}  
.margin {  
margin: 10px;  
width: 350px;  
}  
.small__font {  
font-size: 13px;  
margin: 2px 0;  
}  
</style>
```

## Contacts

---

**JE** **Jenifer**  
(206) 555-985774

**AM** **Amenda**  
(206) 555-3412

**IS** **Isabella**  
(206) 555-8122

**WI** **William**  
(206) 555-9482

**JA** **Jacob**  
(71) 555-4848

**MA** **Matthew**  
(71) 555-7773

**OL** **Oliver**  
(71) 555-5598

[Filter and search list items using Blazor ListView Component](#)

The filtered data can be displayed in the ListView control depending upon on user inputs. Refer to the following steps to render the ListView with filtered data.

- Render a textbox to get input for filtering data.
- Render ListView with `DataSource`, and set the `SortOrder` property.
- Bind the `Input` event for textbox to perform filtering operation. To filter list data, pass the text value to `OnInput` to manipulate the data, and then update filtered data as ListView `dataSource`.

### CSHARP

```
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.Lists
<div id="container">
<div id="sample">
<SfTextBox Placeholder="Filter" Input="@OnInput"></SfTextBox>
<SfListView ID="list" DataSource="@ListData">
<ListViewFieldSettings TValue="ListDataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
</SfListView>
</div>
</div>
@code
{
List<ListDataModel> ListData = new List<ListDataModel>();
List<ListDataModel> DataSource = new List<ListDataModel>() {
new ListDataModel {
Text = "Hennessey Venom",
Id = "list-01"
},
new ListDataModel {
Text = "Bugatti Chiron",
Id = "list-02"
},
new ListDataModel {
Text = "Bugatti Veyron Super Sport",
Id = "list-03"
},
new ListDataModel {
Text = "SSC Ultimate Aero",
Id = "list-04"
},
new ListDataModel {
Text = "Koenigsegg CCR",
Id = "list-05"
},
new ListDataModel {
Text = "McLaren F1",
Id = "list-06"
}
};
protected override void OnInitialized()
{
ListData = DataSource;
}
void OnInput(InputEventArgs eventArgs)
{
ListData = DataSource.FindAll(e =>
e.Text.ToLower().StartsWith(eventArgs.Value));
}
```



```
public class ListDataModel
{
    public string Id
    {
        get;
        set;
    }
    public string Text
    {
        get;
        set;
    }
}
<style>
#list {
box-shadow: 0 1px 4px #ddd;
border-bottom: 1px solid #ddd;
}
#sample {
height: 220px;
margin: 0 auto;
display: block;
max-width: 350px;
}
</style>
```

Filter

Hennessey Venom

Bugatti Chiron

Bugatti Veyron Super Sport

SSC Ultimate Aero

Koenigsegg CCR

McLaren F1

Blazor ListView Component with hyper-link navigation

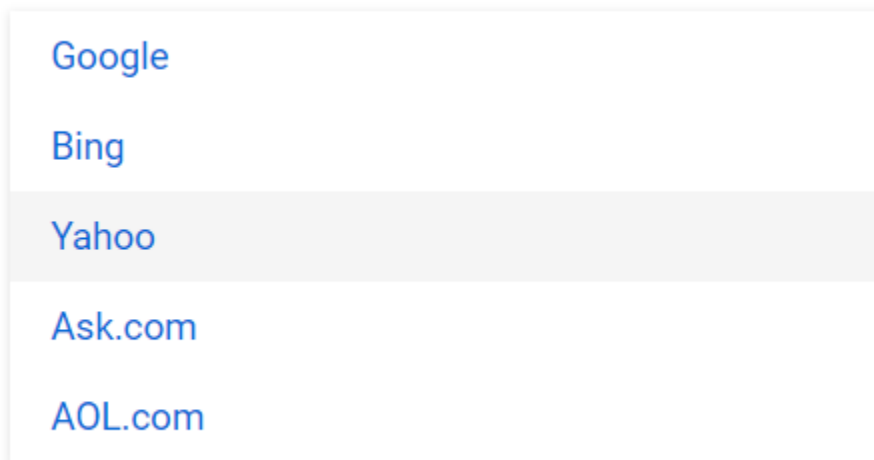
We can use `anchor` tag along with `href` attribute in our ListView `Template` property for navigation.

In the below sample, we have rendered **ListView** with search engines URL.

### ASPX-CS

```
@using Syncfusion.Blazor.Lists
<div id="container">
<div id="sample">
<SfListView ID="list" DataSource="@DataSource">
<ListViewFieldSettings TValue="ListDataModel" Id="Id"
Text="Name"></ListViewFieldSettings>
<ListViewTemplates TValue="ListDataModel">
<Template>
<a target='_blank' href="@((context as ListDataModel).Url)">
@((context as ListDataModel).Name)
</a>
</Template>
</ListViewTemplates>
</SfListView>
</div>
</div>
@code
{
List<ListDataModel> DataSource = new List<ListDataModel>() {
new ListDataModel {
Id = "1",
Name = "Google",
Url = "https://www.google.com"
},
new ListDataModel {
Id = "2",
Name = "Bing",
Url = "https://www.bing.com"
},
new ListDataModel {
Id = "3",
Name = "Yahoo",
Url = "https://www.yahoo.com"
},
new ListDataModel {
Id = "4",
Name = "Ask.com",
Url = "https://www.ask.com"
},
new ListDataModel {
Id = "5",
Name = "AOL.com",
Url = "https://www.aol.com"
}
};
public class ListDataModel
{
public string Id
{
get;
set;
}
public string Name
```

```
{
  get;
  set;
}
public string Url
{
  get;
  set;
}
}
}
<style>
#list {
box-shadow: 0 1px 4px #ddd;
border-bottom: 1px solid #ddd;
}
#sample {
height: 220px;
margin: 0 auto;
display: block;
max-width: 350px;
}
</style>
```



### Chat window user interface using Blazor ListView Component

ListView can be customized as chat window. To achieve that, use the **ListView Template**.

- The Listview template is used to showcase the ListView as chat window.
- Avatar control is used to design the image of contact person.

Refer the below template code snippet for Template of chat window.

#### **ASPX-CS**

```
<div class="flex item_container">
```

```

<div class="flex flex__1 vertical__center flex__center @(currentData.Chat ==
"sender" ? "flex_order__2" : "")">
@if (currentData.Avatar != "")
{
<span class="e-avatar e-avatar-circle">@currentData.Avatar</span>
}
else
{
<span class="@currentData.Pic e-avatar e-avatar-circle"></span>
}
</div>
<div class="flex content__container flex__8 vertical padding
@(currentData.Chat == "sender" ? "right__align" : "left__align")">
<div class="bold">@currentData.Text</div>
<div class="small__font">@currentData.Contact</div>
</div>
</div>

```

### Chat order in template

In ListView template, we have rendered the list items based on receiver and sender information from dataSource of listview.

### Adding messages to chat window

- Use textbox to get message from user.
- Add the textbox message to ListView dataSource using ObservableCollection.

## C#

```

void OnSend()
{
    if (SfTextBox.Value != "")
    {
        DataSource.Add(new ListDataModel
        {
            Text = "Amenda",
            Contact = SfTextBox.Value,
            Id = new Random().Next(300, 900).ToString(),
            Avatar = "A",
            Pic = "",
            Chat = "receiver"
        });
    }
}

```

## ASPX-CS

```

@using Syncfusion.Blazor.Inputs
@using System.Collections.ObjectModel
<div id="container">
<div id="sample">
<SfListView ID="list"
DataSource="@DataSource"
ShowHeader="true"

```

```

Height="420px"
HeaderTitle="Chat">
<ListViewFieldSettings TValue="ListDataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
<ListViewTemplates TValue="ListDataModel">
<Template>
@{
ListDataModel currentData = context as ListDataModel;
<div class="flex item__container">
<div class="flex flex__1 vertical__center flex__center @(currentData.Chat ==
"sender" ? "flex__order__2" : "")">
@if (currentData.Avatar != "")
{
<span class="e-avatar e-avatar-circle">@currentData.Avatar</span>
}
else
{
<span class="@currentData.Pic e-avatar e-avatar-circle"></span>
}
</div>
<div class="flex content__container flex__8 vertical padding
@(currentData.Chat == "sender" ? "right__align" : "left__align")">
<div class="bold">@currentData.Text</div>
<div class="small__font">@currentData.Contact</div>
</div>
</div>
}
</Template>
</ListViewTemplates>
</SfListView>
<div class="flex">
<div class="flex__8 padding">
<SfTextBox Placeholder="Type your message"
@ref="@SfTextBox"
></SfTextBox>
</div>
<div class="flex__1">
<button class="e-btn" @onclick="@OnSend">Send</button>
</div>
</div>
</div>
</div>
@code
{
SfTextBox SfTextBox;
ObservableCollection<ListDataModel> DataSource = new
ObservableCollection<ListDataModel>() {
new ListDataModel {
Text = "Jenifer",
Contact = "Hi",
Id = "1",
Avatar = "",
Pic = "pic01",
Chat = "sender"
},
new ListDataModel {
Text = "Amenda",

```

```
        Contact = "Hello",
        Id = "2",
        Avatar = "A",
        Pic = "",
        Chat = "receiver"
    },
    new ListDataModel {
        Text = "Jenifer",
        Contact = "What Knid of application going to launch",
        Id = "4",
        Avatar = "",
        Pic = "pic02",
        Chat = "sender"
    },
    new ListDataModel {
        Text = "Amenda ",
        Contact = "A knid of Emergency broadcast App",
        Id = "5",
        Avatar = "A",
        Pic = "",
        Chat = "receiver"
    },
    new ListDataModel {
        Text = "Jacob",
        Contact = "Can you please elaborate",
        Id = "6",
        Avatar = "",
        Pic = "pic04",
        Chat = "sender"
    },
    };

    void OnSend()
    {
        if (SfTextBox.Value != "")
        {
            DataSource.Add(new ListDataModel
            {
                Text = "Amenda",
                Contact = SfTextBox.Value,
                Id = new Random().Next(300, 900).ToString(),
                Avatar = "A",
                Pic = "",
                Chat = "receiver"
            });
        }
    }

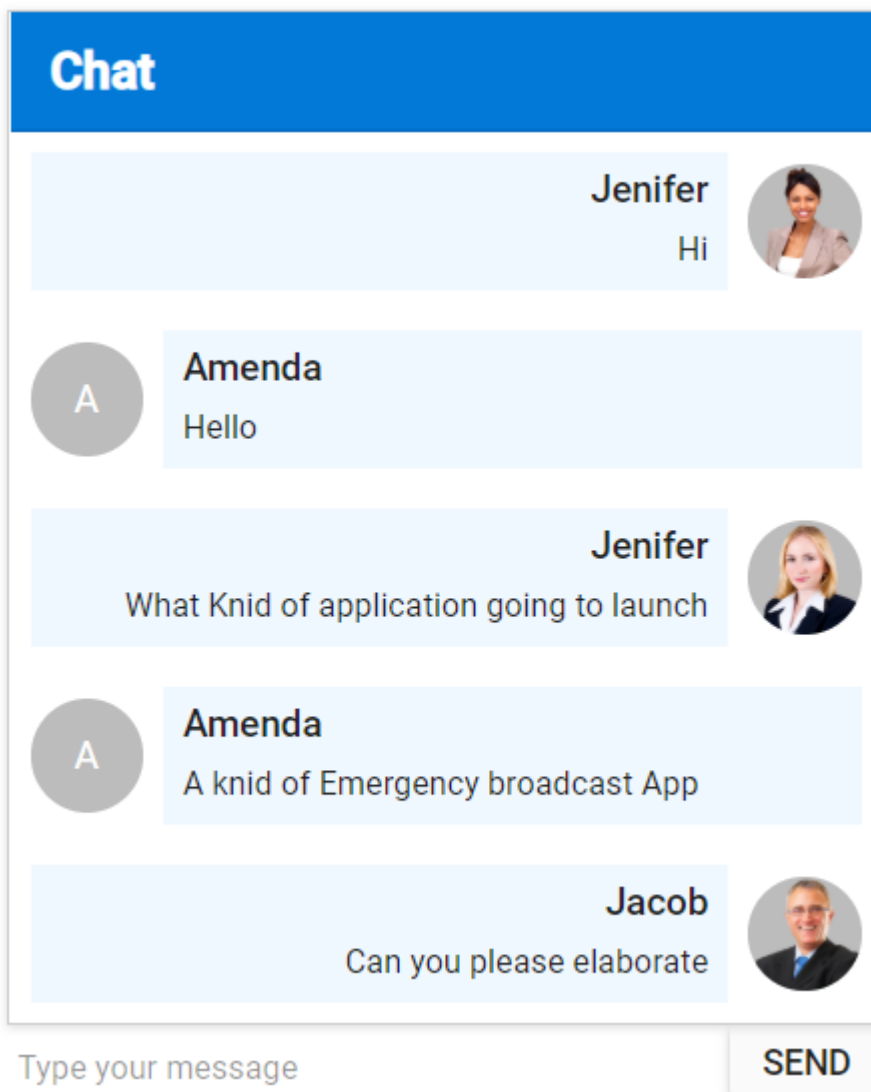
    public class ListDataModel
    {
        public string Id
        {
            get;
            set;
        }
        public string Chat
        {
            get;
            set;
        }
    }
}
```

```
}
public string Pic
{
    get;
    set;
}
public string Avatar
{
    get;
    set;
}
public string Text
{
    get;
    set;
}
public string Contact
{
    get;
    set;
}
}
}
}
<style>
#list {
    box-shadow: 0 1px 4px #ddd;
    border-bottom: 1px solid #ddd;
}
#sample {
    height: 220px;
    margin: 0 auto;
    display: block;
    max-width: 350px;
}
#list {
    margin: 0 auto;
    border: 1px solid #ccc;
}
#list .e-list-item {
    height: auto;
    cursor: pointer;
    line-height: 22px;
    padding: 8px;
}
#list.e-listview .e-list-header {
    background-color: #0278d7;
    color: white;
}
#list .e-list-item.e-active {
    background-color: transparent;
}
#list .e-list-item.e-hover {
    background-color: transparent;
}
.padding {
    padding: 4px;
}
```

```
.right__align {
text-align: right;
margin-right: 8px;
padding-right: 8px;
}
.left__align {
margin-left: 8px;
padding-left: 8px;
}
.content__container {
background-color: aliceblue;
}
.flex {
display: flex;
}
.flex__center {
justify-content: center;
}
.vertical__center {
align-items: center;
}
.vertical {
flex-direction: column;
}
.flex__order__1 {
order: 1;
}
.flex__order__2 {
order: 2;
}
.flex__1 {
flex: 1;
}
.flex__2 {
flex: 2;
}
.flex__3 {
flex: 3;
}
.flex__5 {
flex: 5;
}
.flex__8 {
flex: 8;
}
.bold {
font-weight: 500;
}
.margin {
margin: 10px;
width: 350px;
}
.small__font {
font-size: 13px;
margin: 2px 0;
}
.pic01 {
```



```
background-image:
url("https://ej2.syncfusion.com/demos/src/grid/images/1.png");
}
.pic02 {
background-image:
url("https://ej2.syncfusion.com/demos/src/grid/images/3.png");
}
.pic03 {
background-image:
url("https://ej2.syncfusion.com/demos/src/grid/images/5.png");
}
.pic04 {
background-image:
url("https://ej2.syncfusion.com/demos/src/grid/images/2.png");
}
</style>
```



## Create dual list using Blazor ListView Component

The dual list contains two ListView. This allows you to move list items from one list to another using the client-side events. This section explains how to integrate the ListView control to achieve dual list.

### Use cases

- Stock exchanges of two different countries
- Job applications (skill sets)

### Integration of Dual List

Here, two ListView controls have been used to display the list items. An Blazor Button is used to transfer data between the ListView, and a textbox is used to achieve the UI of filtering support.

The dual list supports:

- Moving whole data from one list to another.
- Moving selected data from one list to another.
- Filtering the list by using a client-side typed character.

In the ListView control, sorting is enabled using the `SortOrder` property, and the `Clicked` event is triggered while selecting an item. Here, the `Clicked` event is triggered to enable and disable button states.

### Manipulating data

#### Moving whole data from the first list to the second list(>>)

Here, the whole data can be moved from the first ListView to the second by clicking the first button. When clicking the button, the whole list items are sliced, and `concat` with the second ListView. This button is enabled only when the data source of the first ListView is not empty.

#### Moving whole data from the second list to the first list(<<)\*\*

The functionality of the second button is the same as above, and data is transferred from the second list to the first list. This button is enabled only when the data source of the second ListView is not empty.

#### Moving selected item from one list to another list (>) and (<)\*\*

The `Clicked` event is triggered when clicking a list item in the ListView. The selected items can be transferred between two lists. These buttons will be enabled when selecting an item in lists.

### ASPX-CS

```
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.Lists
<div id="container">
<div class="sample flex">
<div class="flex">
<div class="padding">
<SfTextBox Placeholder="Filter" Input="@ (e => OnInput(e, 1)) "></SfTextBox>
<SfListView DataSource="@FirstData">
<ListViewFieldSettings TValue="ListDataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
<ListViewEvents TValue="ListDataModel" Clicked="@ (e => OnSelected(e,
1)) "></ListViewEvents>
</SfListView>
</div>
```

```

<div class="flex vertical vertical__center flex__center padding">
<div class="padding">
<button disabled="@(!FirstListData.Any())" class="e-btn" @onclick="@e =>
OnButtonClick(1)" ">@(">")</button>
</div>
<div class="padding">
<button disabled="@ (FirstSelected == null)" class="e-btn" @onclick="@e =>
OnButtonClick(2)" ">@(">")</button>
</div>
<div class="padding">
<button disabled="@ (SecondSelected == null)" class="e-btn" @onclick="@e =>
OnButtonClick(3)" ">@("<")</button>
</div>
<div class="padding">
<button disabled="@(!SecondListData.Any())" class="e-btn" @onclick="@e =>
OnButtonClick(4)" ">@("<")</button>
</div>
</div>
<div class="padding">
<SfTextBox Placeholder="Filter" Input="@e => OnInput(e, 2)"></SfTextBox>
<SfListView DataSource="@SecondData">
<ListViewFieldSettings Id="Id" Text="Text"
TValue="ListDataModel"></ListViewFieldSettings>
<ListViewEvents TValue="ListDataModel" Clicked="@e => OnSelected(e,
2)"></ListViewEvents>
</SfListView>
</div>
</div>
</div>
</div>
@code
{
List<ListDataModel> FirstData;
List<ListDataModel> SecondData;
ListDataModel FirstSelected;
ListDataModel SecondSelected;
protected override void OnInitialized()
{
FirstData = new List<ListDataModel>(FirstListData);
SecondData = new List<ListDataModel>(SecondListData);
}
void OnButtonClick(int buttonIndex)
{
switch (buttonIndex)
{
case 1:
FirstListData.ForEach(e => SecondListData.Add(e));
FirstListData.Clear();
FirstData.Clear();
FirstData = new List<ListDataModel>(FirstListData);
SecondData = new List<ListDataModel>(SecondListData);
break;
case 2:
if (FirstSelected != null)
{
SecondListData.Add(FirstSelected);

```

```

FirstListData.RemoveAt(FirstListData.FindIndex(e => e.Id ==
FirstSelected.Id));
FirstData = new List<ListDataModel>(FirstListData);
SecondData = new List<ListDataModel>(SecondListData);
FirstSelected = null;
}
break;
case 3:
if (SecondSelected != null)
{
FirstListData.Add(SecondSelected);
SecondListData.RemoveAt(SecondListData.FindIndex(e => e.Id ==
SecondSelected.Id));
FirstData = new List<ListDataModel>(FirstListData);
SecondData = new List<ListDataModel>(SecondListData);
SecondSelected = null;
}
break;
case 4:
SecondListData.ForEach(e => FirstListData.Add(e));
SecondData.Clear();
SecondListData.Clear();
SecondData = new List<ListDataModel>(SecondListData);
FirstData = new List<ListDataModel>(FirstListData);
break;
default:
break;
}
}
void OnSelected(ClickEventArgs<ListDataModel> eventArgs, int listViewIndex)
{
if (listViewIndex == 1)
{
FirstSelected = eventArgs.ItemData;
}
else
{
SecondSelected = eventArgs.ItemData;
}
}
void OnInput(InputEventArgs eventArgs, int listViewIndex)
{
if (listViewIndex == 1)
{
FirstData = FirstListData.FindAll(e =>
e.Text.ToLower().Contains(eventArgs.Value.ToLower()));
}
else
{
SecondData = SecondListData.FindAll(e =>
e.Text.ToLower().Contains(eventArgs.Value.ToLower()));
}
}
List<ListDataModel> SecondListData = new List<ListDataModel>() {
new ListDataModel {
Text = "Aston Martin One- 77",
Id = "07"

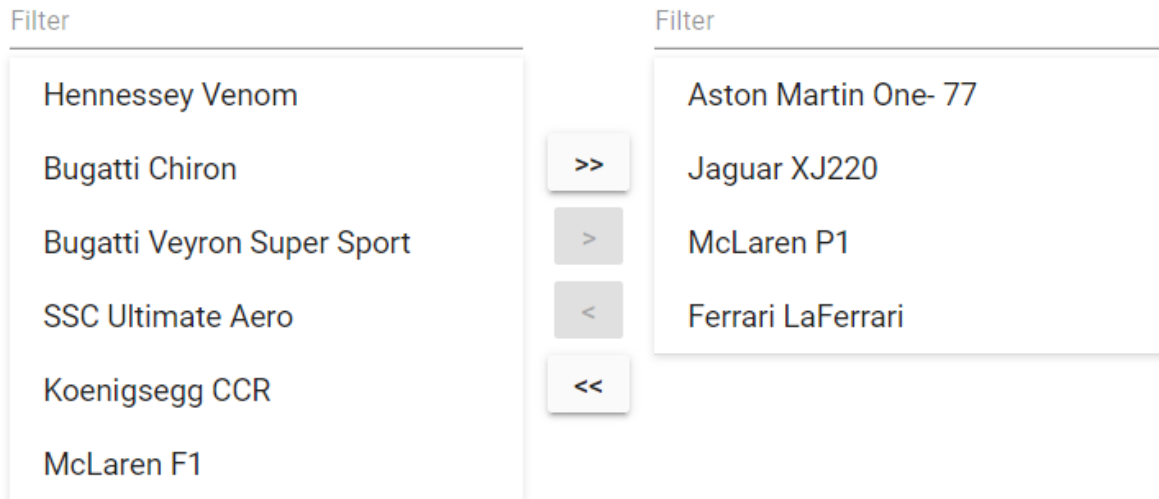
```

```
    },
    new ListDataModel {
        Text = "Jaguar XJ220",
        Id = "08"
    },
    new ListDataModel {
        Text = "McLaren P1",
        Id = "09"
    },
    new ListDataModel {
        Text = "Ferrari LaFerrari",
        Id = "14"
    },
    };
    List<ListDataModel> FirstListData = new List<ListDataModel>() {
        new ListDataModel {
            Text = "Hennessey Venom",
            Id = "01"
        },
        new ListDataModel {
            Text = "Bugatti Chiron",
            Id = "02"
        },
        new ListDataModel {
            Text = "Bugatti Veyron Super Sport",
            Id = "03"
        },
        new ListDataModel {
            Text = "SSC Ultimate Aero",
            Id = "04"
        },
        new ListDataModel {
            Text = "Koenigsegg CCR",
            Id = "05"
        },
        new ListDataModel {
            Text = "McLaren F1",
            Id = "06"
        }
    };
    public class ListDataModel
    {
        public string Id
        {
            get;
            set;
        }
        public string Text
        {
            get;
            set;
        }
    }
}

<style>
.e-listview.e-lib {
box-shadow: 0 1px 4px #ddd;
```

```
border-bottom: 1px solid #ddd;
width: 250px;
}
.sample {
justify-content: center;
min-height: 280px;
}
.padding {
padding: 4px;
}
.right__align {
text-align: right;
margin-right: 8px;
padding-right: 8px;
}
.left__align {
margin-left: 8px;
padding-left: 8px;
}
.content__container {
background-color: aliceblue;
}
.flex {
display: flex;
}
.flex__center {
justify-content: center;
}
.vertical__center {
align-items: center;
}
.vertical {
flex-direction: column;
}
.flex__order__1 {
order: 1;
}
.flex__order__2 {
order: 2;
}
.flex__1 {
flex: 1;
}
.flex__2 {
flex: 2;
}
.flex__3 {
flex: 3;
}
.flex__5 {
flex: 5;
}
.flex__8 {
flex: 8;
}
.bold {
font-weight: 500;
}
```

```
}  
.margin {  
margin: 10px;  
}  
.small__font {  
font-size: 13px;  
margin: 2px 0;  
}  
</style>
```



### Customize Blazor ListView Component to Grid Layout

In Listview, list items can be rendered in grid layout with following data manipulations.

- Add Item
- Remove Item
- Sort Items
- Filter Items

#### Grid Layout

In this section, we will discuss about rendering of list items in grid layout.

- Initialize and render ListView with dataSource which will render list items in list layout.
- Now, add the below CSS to list item. This will make list items to render in grid layout

#### CSS

```
#container .e-listview .e-list-item {  
height: 100px;  
width: 100px;  
float: left;  
}
```

In the below sample, we have rendered List items in grid layout.

### C#

```
<div id="container">
<div class="sample flex">
<div class="flex">
<div class="padding">
<SfListView DataSource="@ListItems"></SfListView>
</div>
</div>
</div>
</div>
@code
{
List<int> ListItems = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12 };
}
<style>
#container .e-listview {
box-shadow: 0 1px 4px #ddd;
border-bottom: 1px solid #ddd;
width: 400px;
}
#container .e-listview .e-list-item {
height: 100px;
width: 100px;
float: left;
}
#container .e-listview .e-list-item .e-text-content {
display: flex;
align-items: center;
justify-content: center;
cursor: pointer;
}
#container .e-listview .e-list-text {
width: unset;
}
.sample {
justify-content: center;
min-height: 280px;
}
.padding {
padding: 4px;
}
.flex {
display: flex;
}
.flex__center {
justify-content: center;
}
.margin {
margin: 10px;
}
</style>
```



*Data manipulation*

In this section, we will discuss about ListView data manipulations.

*Add Item*

In the below sample, you can add new item by clicking add button which will open dialog box with name text box. After entering the item details, click the add button. This will add your new item.

*Remove item*

In the below sample, you can remove by hovering the item which will show delete button and click that delete button to delete that from your list.

*Sort Items*

ListView can be sorted either in Ascending or Descending order. To enable sorting in your ListView, set **SortOrder** as **Ascending** or **Descending**. You can also set sorting after control initialization.

In the below sample, we have sorted in **Ascending** order. To sort it in descending, click on sort order icon and vice versa.

**ASPX-CS**

```
@using Syncfusion.Blazor.Popups
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Lists
@using ListviewSort = Syncfusion.Blazor.Lists.SortOrder
<div id="container">
<div class="sample flex">
<div class="flex">
<div>
<div class="headerContainer">
<div class="e-input-group">
<input id="search" class="e-input" placeholder="Search" @bind-
value="@SearchValue" @oninput="@ (e => Search(e.Value.ToString()))" />
</div>
<button id="sort"
class="e-btn e-small e-round e-primary e-icon-btn"
title="Sort"
@onclick="@ (e => ListSortOrder = (ListSortOrder == ListviewSort.Ascending) ?
ListviewSort.Descending : ListviewSort.Ascending)">
<span class="e-btn-icon e-icons e-sort-icon-ascending"></span>
</button>
<button id="add"
class="e-btn e-small e-round e-primary e-icon-btn"
title="Add"
@onclick="@ (e => DialogObj.Show())">
<span class="e-btn-icon e-icons e-add-icon"></span>
</button>
<SfDialog @ref="DialogObj"
Target="#container"
ShowCloseIcon="true"
Header="@ ("Add item")"
@bind-Visible="@Visible"
Width="300px"
Height="230px">
<DialogTemplates>
<Content>
<div id="listDialog">
<div class="input_name">
```

```

<label for="name">Item text: </label>
<input id="name" class="e-input" type="text" placeholder="Enter text" @bind-
value="@Value" />
</div>
</div>
</Content>
</DialogTemplates>
<DialogButtons>
<DialogButton OnClick="@ (e => Add()) "
ButtonModel="@DialogBtn"></DialogButton>
</DialogButtons>
</SfDialog>
</div>
</div>
<div>
<div class="listview-container">
<SfListView DataSource="@DataSource" SortOrder="@ListSortOrder">
<ListViewFieldSettings TValue="ListDataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
<ListViewTemplates TValue="ListDataModel">
<Template>
@{
ListDataModel currentData = (ListDataModel)context;
<div>
@currentData.Text
<span class="e-badge e-badge-notification e-badge-overlap e-badge-danger e-
badge-circle"
@onclick="@ (e => Remove(currentData)) ">
<span class="delete-icon"></span>
</span>
</div>
</div>
</Template>
</ListViewTemplates>
</SfListView>
</div>
</div>
</div>
</div>
</div>
@code
{
SfDialog DialogObj;
ListviewSort ListSortOrder = ListviewSort.Ascending;
DialogButtonModel DialogBtn = new DialogButtonModel { Content = "Add",
IsPrimary = true, CssClass = "e-flat" };
string Value = "";
string SearchValue = "";
bool Visible = false;
List<ListDataModel> DataSourceOG = new List<ListDataModel>(
Enumerable.Range(10, 22)
.Select(
index => new ListDataModel
{
Id = index.ToString(),
Text = "Item " + index.ToString(),
}
)
)

```

```

).ToList()
);
List<ListDataModel> DataSource;
protected override void OnInitialized()
{
    DataSource = new List<ListDataModel>(DataSourceOG);
}
void Search(string value)
{
    if (value != "")
    {
        DataSource = new List<ListDataModel>(DataSourceOG.Where(e =>
e.Text.ToLower().Contains(value.ToLower())));
    } else
    {
        DataSource = new List<ListDataModel>(DataSourceOG);
    }
}
void Remove(ListDataModel data)
{
    DataSourceOG.RemoveAt(DataSourceOG.FindIndex(e => e.Id == data.Id));
    DataSource = new List<ListDataModel>(DataSourceOG);
}
void Add()
{
    DialogObj.Hide();
    DataSourceOG.Add(new ListDataModel { Id = Guid.NewGuid().ToString(), Text =
Value });
    DataSource = new List<ListDataModel>(DataSourceOG);
    Value = "";
}
public class ListDataModel
{
    public string Id
    {
        get;
        set;
    }
    public string Text
    {
        get;
        set;
    }
}
<style>
.headerContainer {
height: 48px;
line-height: 48px;
background: rgb(2, 120, 215);
color: white;
margin-bottom: 3px;
}
#container .e-listview .e-content {
overflow: visible;
}
#container .e-listview {

```

```
overflow: visible;
}
#container .listview-container {
display: inline-block;
height: 300px;
}
.headerContainer .e-input-group {
margin-left: 20px;
width: 200px;
background: white;
height: 31px;
}
.headerContainer #search {
height: 21px;
margin-left: 10px;
}
#listDialog .input_name {
margin-bottom: 20px;
}
.headerContainer #add,
.headerContainer #sort {
float: right;
margin-right: 15px;
margin-top: 7px;
background: white;
color: black
}
.headerContainer .e-input-search::before {
font-family: 'e-icons';
content: '\e961';
margin-top: 3px;
}
.headerContainer .e-input-group .e-input-search {
padding: 0 10px 0 10px;
}
.headerContainer .e-sort-icon-ascending::before {
content: '\e840';
}
.headerContainer .e-sort-icon-descending::before {
content: '\e83f';
}
.headerContainer .e-add-icon::before {
content: '\e823';
}
</style>
<style>
#container .e-listview {
box-shadow: 0 1px 4px #ddd;
border-bottom: 1px solid #ddd;
}
#container .flex {
display: flex;
flex-direction: column;
width: 400px;
margin: auto;
}
#container .e-listview .e-list-item {
```

```
height: 100px;
width: 100px;
float: left;
padding: 0;
}
#container .e-listview .e-list-item .e-blazor-template {
display: flex;
align-items: center;
justify-content: center;
cursor: pointer;
height: 100%;
}
#container .e-listview .e-list-item .delete-icon {
font-size: 9px;
font-family: 'e-icons';
}
#container .e-listview .e-badge {
z-index: 10;
display: none;
}
#container .e-listview .e-hover .e-badge {
display: unset;
}
#container .e-listview .e-active .e-badge {
display: unset;
}
#container .e-listview .e-list-item .delete-icon::before {
content: '\e7fc';
color: white;
}
.sample {
justify-content: center;
min-height: 280px;
}
.padding {
padding: 4px;
}
.flex {
display: flex;
}
.flex__center {
justify-content: center;
}
.margin {
margin: 10px;
}
</style>
```

+

↑

Item 10

Item 11

Item 12

Item 13

Item 14

Item 15

Item 16

Item 17

Item 18

Item 19

Item 20

Item 21

Item 22

Item 23

Item 24

Item 25

Item 26

Item 27

Item 28

Item 29

Item 30

Item 31

### Get selected items from listview template in Blazor ListView Component

Single or multiple items can be selected by users in the ListView control. By default, `dataSource` `Id` and `Text` is mapped in default rendering of listview, since it returns the selected item data properly. But in the custom template, `dataSource` and the corresponding mapping (text, id, elements rendered inside li element) will vary as per the application requirement.

So, we need to map id attribute to listview items using `ListViewFieldSettings` of `DataSource` to get the selected item data properly while working with custom templates. Refer to the below code snippet for template sample.

#### ASPX-CS

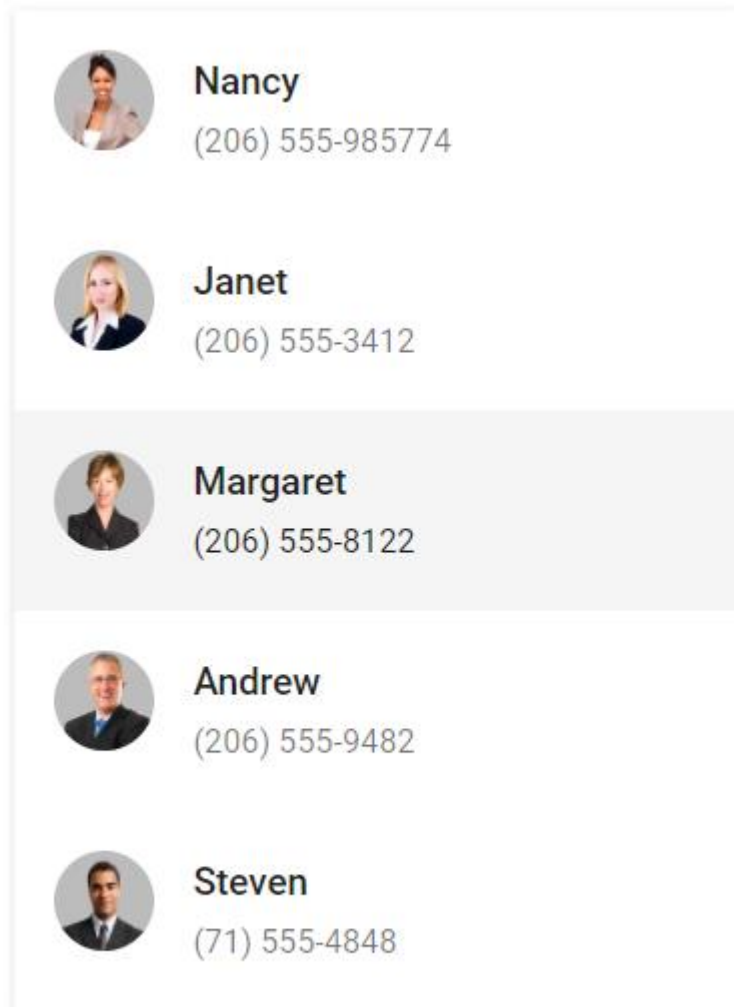
```
@using Syncfusion.Blazor.Lists
<div id="container">
<div class="sample flex vertical-center">
<div class="padding">
<SfListView DataSource="@DataSource" CssClass="e-list-template">
<ListViewFieldSettings TValue="ListDataModel" Id="Id"
Text="Name"></ListViewFieldSettings>
<ListViewTemplates TValue="ListDataModel">
<Template>
@{
ListDataModel currentData = (ListDataModel)context;
<div class="e-list-wrapper e-list-multi-line e-list-avatar" @onclick="(e =>
OnSelect(currentData))">

<span class="e-list-item-header">@currentData.Name</span>
<span class="e-list-content">@currentData.Contact</span>
</div>
}
</Template>
</ListViewTemplates>
</SfListView>
</div>
<div class="padding">
<h3>@ (Selected?.Name) </h3>
</div>
</div>
</div>
@code
{
ListDataModel Selected;
List<ListDataModel> DataSource = new List<ListDataModel>() {
new ListDataModel { Name = "Nancy", Contact = "(206) 555-985774", Id = "1",
Image = "https://ej2.syncfusion.com/demos/src/grid/images/1.png", Category =
"Experience" },
new ListDataModel { Name = "Janet", Contact = "(206) 555-3412", Id = "2",
Image = "https://ej2.syncfusion.com/demos/src/grid/images/3.png", Category =
"Fresher" },
new ListDataModel { Name = "Margaret", Contact = "(206) 555-8122", Id = "4",
Image = "https://ej2.syncfusion.com/demos/src/grid/images/4.png", Category =
"Experience" },
new ListDataModel { Name = "Andrew ", Contact = "(206) 555-9482", Id = "5",
Image = "https://ej2.syncfusion.com/demos/src/grid/images/2.png", Category =
"Experience" },
}
```

```
new ListDataModel { Name = "Steven", Contact = "(71) 555-4848", Id = "6",  
Image = "https://ej2.syncfusion.com/demos/src/grid/images/5.png", Category =  
"Fresher" },  
};  
void OnSelect(ListDataModel listData)  
{  
    Selected = listData;  
}  
public class ListDataModel  
{  
    public string Id  
    {  
        get;  
        set;  
    }  
    public string Name  
    {  
        get;  
        set;  
    }  
    public string Image  
    {  
        get;  
        set;  
    }  
    public string Category  
    {  
        get;  
        set;  
    }  
    public string Contact  
    {  
        get;  
        set;  
    }  
}  
<style>  
#container .e-listview {  
    box-shadow: 0 1px 4px #ddd;  
    border-bottom: 1px solid #ddd;  
}  
.sample {  
    justify-content: center;  
    min-height: 280px;  
}  
.vertical-center {  
    align-items: center;  
}  
.padding {  
    padding: 4px;  
}  
.flex {  
    display: flex;  
}  
.flex__center {  
    justify-content: center;
```



```
}  
.margin {  
margin: 10px;  
}  
</style>
```



Margaret

#### Trace events of listview in Blazor ListView Component

The ListView control triggers events based on its actions. The events can be used as extension points to perform custom operations. Refer to the following steps to trace the ListView events:

1. Render the ListView with `DataSource`, and bind the `OnActionBegin`, `OnActionComplete`, and `Selected` events.
2. Perform custom operations in `OnActionBegin`, `OnActionComplete`, and `Selected` events.
3. Provide event log details for `OnActionBegin` and `OnActionComplete` events, and they will be displayed in the event trace panel when the ListView action starts and the dataSource bound successfully.

4. Get the selected item details from the `SelectEventArgs` in the select event, and display the selected list item text in the event trace panel while selecting list items.
5. Use clear button to remove event trace information.

### ASPX-CS

```
@using Syncfusion.Blazor.Lists
<div id="container">
<div class="flex vertical-center">
<div class="sample padding">
<SfListView DataSource="@DataSource">
<ListViewFieldSettings TValue="ListDataModel" Id="Id"
Text="Text"></ListViewFieldSettings>
<ListViewEvents TValue="ListDataModel"
Clicked="@ (e => Events.Add(e.Text + " is selected"))"
OnActionBegin="@ (e => Events.Add("OnActionBegin is triggered"))"
OnActionComplete="@ (e => Events.Add("OnActionComplete is triggered"))">
</ListViewEvents>
</SfListView>
</div>
<div class="sample padding tracker">
<ul style="list-style: none">
@for (var i = Events.Count - 1; i >= 0; i--)
{
<li>@Events[i]</li>
}
</ul>
</div>
</div>
</div>
@code
{
List<string> Events = new List<string>();
List<ListDataModel> DataSource = new List<ListDataModel>() {
new ListDataModel { Id = "1", Text = "Text 1" },
new ListDataModel { Id = "2", Text = "Text 2" },
new ListDataModel { Id = "3", Text = "Text 3" },
new ListDataModel { Id = "4", Text = "Text 4" },
new ListDataModel { Id = "5", Text = "Text 5" },
new ListDataModel { Id = "6", Text = "Text 6" },
new ListDataModel { Id = "7", Text = "Text 7" },
};
public class ListDataModel
{
public string Id
{
get;
set;
}
public string Text
{
get;
set;
}
}
}
```

```
<style>
#container .e-listview {
box-shadow: 0 1px 4px #ddd;
border-bottom: 1px solid #ddd;
}
.tracker {
max-height: 250px;
overflow: auto;
}
.sample {
justify-content: center;
min-height: 280px;
width: 350px;
}
.vertical-center {
align-items: center;
}
.padding {
padding: 4px;
}
.flex {
display: flex;
}
.flex__center {
justify-content: center;
}
.margin {
margin: 10px;
}
</style>
```

Text 1

Text 2

Text 3

Text 4

Text 5

Text 6

Text 7

Text 5 is selected

Text 2 is selected

Text 3 is selected

Text 4 is selected

Text 3 is selected

Text 1 is selected

Text 3 is selected

Text 5 is selected

OnActionComplete is triggered

OnActionBegin is triggered

## Maps

### Getting Started with Blazor Maps Component

This section briefly explains how to include a Maps 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 Maps component in the application

1. Install **Syncfusion.Blazor.Maps** NuGet package in the application 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/{:version:}/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/{:version:}/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.Maps** namespace.

#### CSHARP

```
@using Syncfusion.Blazor.Maps
```

#### 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 Maps component

The Syncfusion Maps component can be initialized in any razor page inside the **~/Pages** folder. For example, the Maps 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 following code can be added.

#### ASPX-CS

```
@page "/"
<SfMaps>
</SfMaps>
```

The Maps will not show any content on the web page while running the application because the properties related to the layer are not initialized in the above code.

#### Adding GeoJSON data in Maps layer

Bind GeoJSON data to the Maps to render any geometric shape in SVG (Scalable Vector Graphics) for powerful data visualization of shapes. For example, you can render the World map and make desired customizations on it. You can also add any number of layers in the Maps.

You can use the [ShapeData](#) property in [MapsLayer](#) to load the GeoJSON shape data into the Maps component.

#### ASPX-CS

```
<SfMaps>
<MapsLayers>
@* To load shape data *@
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
```



---

The "world-map.json" file contains the World map GeoJSON data.

---

#### Bind data source

The [DataSource](#) property is used to represent statistical data in the Maps component. We can define a list of objects as a data source to the Maps component. This data source will be further used to color the map, display data labels, display tooltips, and more. Assign the below list **SecurityCouncilDetails** to the [DataSource](#) property in [MapsLayer](#).

#### ASPX-CS

```
@code {
public List<UNCouncilCountry> SecurityCouncilDetails = new
List<UNCouncilCountry>{
new UNCouncilCountry { Name= "China", Membership= "Permanent"},
new UNCouncilCountry { Name= "France", Membership= "Permanent" },
new UNCouncilCountry { Name= "Russia", Membership= "Permanent"},
new UNCouncilCountry { Name= "Kazakhstan", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "Poland", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "Sweden", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "United Kingdom", Membership= "Permanent"},
new UNCouncilCountry { Name= "United States", Membership= "Permanent"},
new UNCouncilCountry { Name= "Bolivia", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "Eq. Guinea", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "Ethiopia", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "Côte d Ivoire", Membership= "Permanent"},
new UNCouncilCountry { Name= "Kuwait", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "Netherlands", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "Peru", Membership= "Non-Permanent"}
```

```
};
public class UNCouncilCountry
{
    public string Name { get; set; }
    public string Membership { get; set; }
};
}
```

The United Nations Security Council data is referred from [source](#).

You should also specify the field names in the shape data and data source to the [ShapePropertyPath](#) and [ShapeDataPath](#) properties, respectively. These are used to identify the appropriate shapes and match the specific data source values to them.

Please [refer to the section](#) for more information on data binding.

### ASPX-CS

```
<SfMaps>
<MapsLayers>
@*To map shape data name field and data source field*@
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapePropertyPath='new string[] {"name"}'
DataSource="SecurityCouncilDetails"
ShapeDataPath="Name" TValue="UNCouncilCountry">
</MapsLayer>
</MapsLayers>
</SfMaps>
```

For example, consider field names specified in [ShapePropertyPath](#) and [ShapeDataPath](#) have the same value: **“United States”**. So corresponding color, data label and tooltip related settings will be applied to the **United States** shape.

### Apply color mapping

The color mapping supports customization of shape colors based on the underlying value of shape received from the bound data source. The values from the field name specified in the [ShapeDataPath](#) property will be compared for the shapes with the values in the field name specified in the [ColorValuePath](#) property in [MapsShapeSettings](#). Also, specify color and value in [MapsShapeColorMapping](#). Here, in this example, **“#EDB46F”** is specified for **“Permanent”** and **“#F1931B”** is specified for **“Non-Permanent”**.

### ASPX-CS

```
<SfMaps>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapePropertyPath='new string[] {"name"}'
DataSource="SecurityCouncilDetails"
ShapeDataPath="Name" TValue="UNCouncilCountry">
@* Color mapping related configuration *@
<MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Membership">
<MapsShapeColorMappings>
```

```

<MapsShapeColorMapping Value="Permanent" Color='new string[]
{ "#EDB46F" }'></MapsShapeColorMapping>
<MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
{ "#F1931B" }'></MapsShapeColorMapping>
</MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>

```




---

Refer [code block](#) to know the property value of **SecurityCouncilDetails**.

---

### Adding data labels

Label provides information to users about the shapes, and you can enable label text to the shapes in the Maps component by setting the [Visible](#) property as **true** and field name from data source in the [LabelPath](#) property in [MapsDataLabelSettings](#).

### ASPX-CS

```

<SfMaps>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapePropertyPath='new string[] { "name" }'
DataSource="SecurityCouncilDetails"
ShapeDataPath="Name" TValue="UNCouncilCountry">
@* To add data labels *@

```



```

<MapsDataLabelSettings Visible="true" LabelPath="Name"
IntersectionAction="IntersectionAction.Hide"></MapsDataLabelSettings>
<MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Membership">
<MapsShapeColorMappings>
<MapsShapeColorMapping Value="Permanent" Color='new string[]
{"#EDB46F"}'></MapsShapeColorMapping>
<MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
{"#F1931B"}'></MapsShapeColorMapping>
</MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>

```

Refer [code block](#) to know the property value of **SecurityCouncilDetails**.



### Adding title for Maps

Title can be added to the Maps to provide quick information to the users about the shapes rendered in the component. You can add a title using [Text](#) property in [MapsTitleSettings](#).

### ASPX-CS

```

<SfMaps>
@* To add title *@
<MapsTitleSettings Text="Members of the UN Security
Council"></MapsTitleSettings>
</MapsLayers>

```

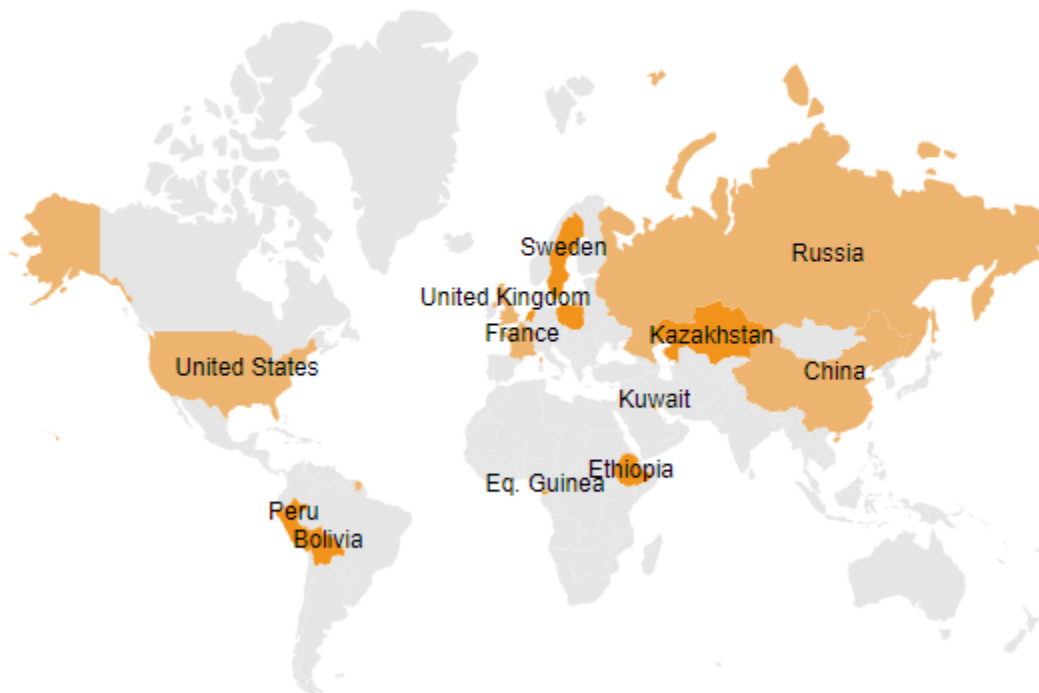
```

<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapePropertyPath='new string[] {"name"}'
DataSource="SecurityCouncilDetails"
ShapeDataPath="Name" TValue="UNCouncilCountry">
<MapsDataLabelSettings Visible="true" LabelPath="Name"
IntersectionAction="IntersectAction.Hide"></MapsDataLabelSettings>
<MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Membership">
<MapsShapeColorMappings>
<MapsShapeColorMapping Value="Permanent" Color='new string[]
{"#EDB46F"}'></MapsShapeColorMapping>
<MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
{"#F1931B"}'></MapsShapeColorMapping>
</MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>

```

Refer [code block](#) to know the property value of **SecurityCouncilDetails**.

Members of the UN Security Council



### Enable legend

The legend items are used to denote color mapping categories, and you can show legend for the Maps by setting the [Visible](#) property to **true** in [MapsLegendSettings](#).

### ASPX-CS

```

<SfMaps>

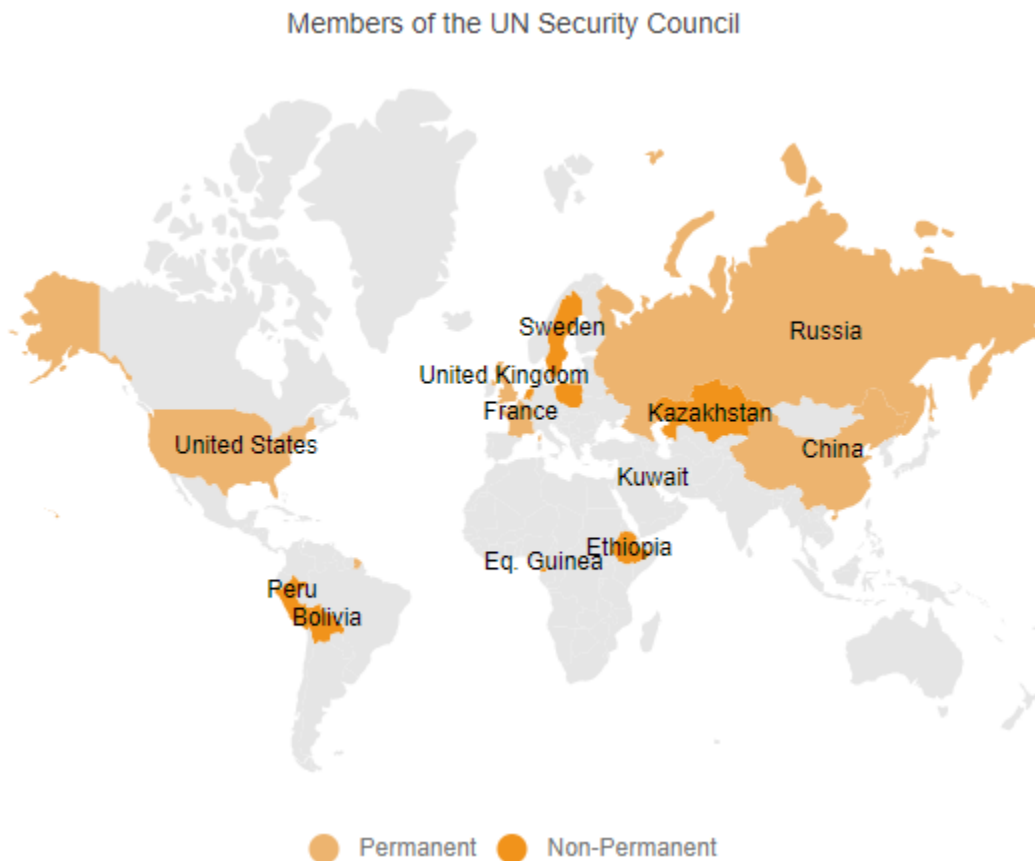
```

```

<MapTitleSettings Text="Members of the UN Security
Council"></MapTitleSettings>
@* To add legend *@
<MapsLegendSettings Visible="true"></MapsLegendSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapePropertyPath='new string[] {"name"}'
DataSource="SecurityCouncilDetails"
ShapeDataPath="Name" TValue="UNCouncilCountry">
<MapsDataLabelSettings Visible="true" LabelPath="Name"
IntersectionAction="IntersectAction.Hide"></MapsDataLabelSettings>
<MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Membership">
<MapsShapeColorMappings>
<MapsShapeColorMapping Value="Permanent" Color='new string[]
{"#EDB46F"}'></MapsShapeColorMapping>
<MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
{"#F1931B"}'></MapsShapeColorMapping>
</MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>

```

Refer [code block](#) to know the property value of **SecurityCouncilDetails**.



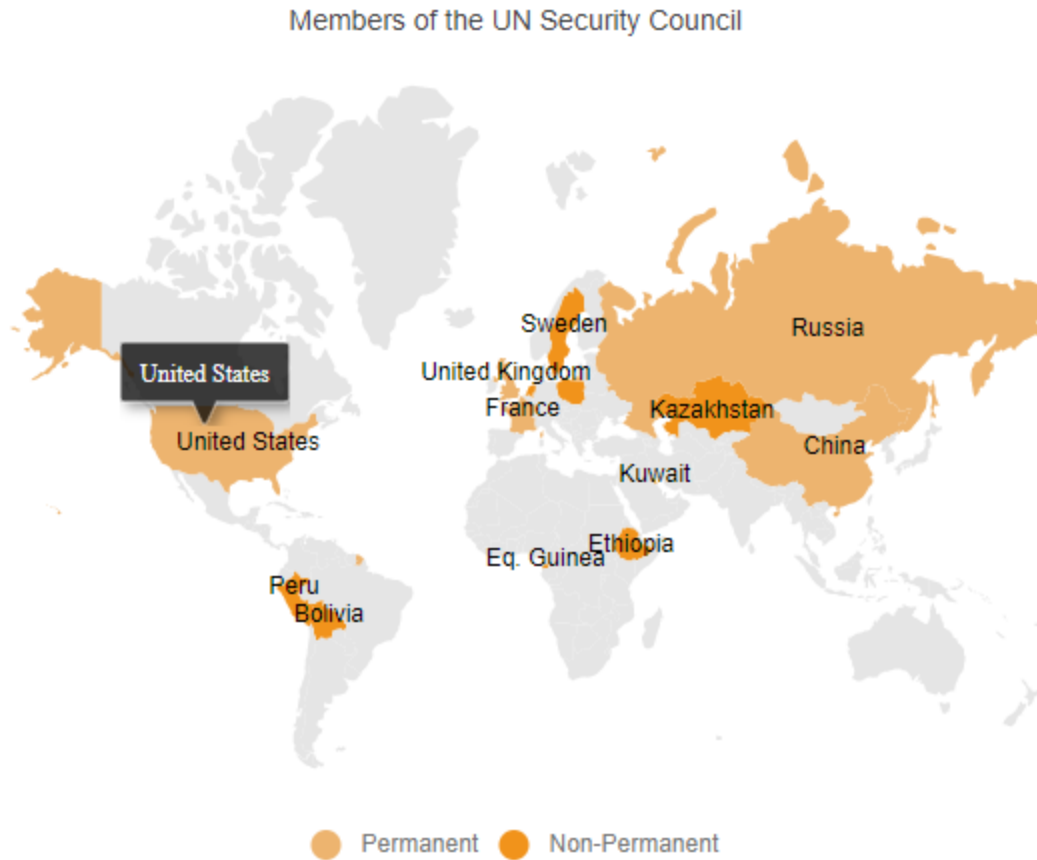
### Enable tooltip

The tooltip can be used when you cannot display information using the data labels due to space constraints. You can enable tooltip by setting the [Visible](#) property to **true** in [MapsLayerTooltipSettings](#).

### ASPX-CS

```
<SfMaps>
  <MapsTitleSettings Text="Members of the UN Security
  Council"></MapsTitleSettings>
  <MapsLegendSettings Visible="true"></MapsLegendSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
    "https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
    ShapePropertyPath='new string[] {"name"}'
    DataSource="SecurityCouncilDetails"
    ShapeDataPath="Name" TValue="UNCouncilCountry">
      <MapsDataLabelSettings Visible="true" LabelPath="Name"
      IntersectionAction="IntersectAction.Hide"></MapsDataLabelSettings>
      <MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Membership">
        <MapsShapeColorMappings>
          <MapsShapeColorMapping Value="Permanent" Color='new string[]
          {"#EDB46F"}'></MapsShapeColorMapping>
          <MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
          {"#F1931B"}'></MapsShapeColorMapping>
        </MapsShapeColorMappings>
      </MapsShapeSettings>
      @* To add tooltip for the shape *@
      <MapsLayerTooltipSettings Visible='true'
      ValuePath="Name"></MapsLayerTooltipSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```

Refer [code block](#) to know the property value of **SecurityCouncilDetails**.



See also

- [Getting Started with Syncfusion Blazor for WebAssembly application in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for server-side application in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for server-side application in .NET Core CLI](#)

## Populate Data in Blazor Maps Component

This section explains how to populate data inputs and provide it to the Maps component.

### Shape data

The shape data collection describes geographical shape information that is available in GeoJSON format. The Map shapes are rendered with this data. The custom shapes such as seat selection in bus, seat selection in a cricket stadium and more useful information can be also added as [ShapeData](#) in the [MapsLayer](#).

### Data source

The [DataSource](#) property is used to represent statistical data in the Maps component, and it accepts a collection of values as input. For example, a list of objects as input can be provided to the data source. This data source will be used to color the map, display data labels, and display tooltip, among other things.

The data source is populated with list of objects relative to shape data. In the below example, **PopulationDetails** can be used as data source in Maps.

### ASPX-CS

```
@code{
public class PopulationDetail
{
    public string Code { get; set; }
    public double Value { get; set; }
    public string Name { get; set; }
    public double Population { get; set; }
    public double Density { get; set; }
};
private List<PopulationDetail> PopulationDetails = new
List<PopulationDetail> {
    new PopulationDetail {
        Code = "AF",
        Value= 53,
        Name= "Afghanistan",
        Population= 29863010,
        Density= 119
    },
    new PopulationDetail {
        Code= "AL",
        Value= 117,
        Name= "Albania",
        Population= 3195000,
        Density= 111
    },
    new PopulationDetail {
        Code= "DZ",
        Value= 15,
        Name= "Algeria",
        Population= 34895000,
        Density= 15
    }
};
}
```

### Data binding

The following properties in the [MapsLayer](#) are used for binding data in the Maps component. Both the properties are related to each other.

- ShapePropertyPath
- ShapeDataPath

### ShapePropertyPath

The [ShapePropertyPath](#) property is used to refer the field name in the [ShapeData](#) property of shape layers to identify the shape. When the values of [ShapeDataPath](#) property from the [DataSource](#) property and [ShapePropertyPath](#) property from the [ShapeData](#) property match, then the associated object from the data source is bound to the corresponding shape.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLayers>
```

```
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapePropertyPath='new string[] { "name"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
```

`world-map.json` file contains following data and its field **name** value is used to map the corresponding shape with the provided data source.

### JSON

```
[
{
  "type": "Feature",
  "properties": {
    "admin": "Afghanistan",
    "name": "Afghanistan",
    "continent": "Asia"
  },
  "geometry": { "type": "Polygon", "coordinates": [[[ 61.21081709172573, ...
  ],
  ...
]
```

#### *ShapeDataPath*

The [ShapeDataPath](#) property is similar to the [ShapePropertyPath](#) property, but it refers to the field name in the [DataSource](#) property. For example, following population data contains field **Name**, **Population** and **Density**. Here the **Name** field is set to the [ShapeDataPath](#) to map the corresponding value of field name in shape data.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
DataSource="PopulationDetails" ShapePropertyPath='new string[] { "name"}'
ShapeDataPath="Name" TValue="PopulationDetail">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class PopulationDetail
{
public string Name { get; set; }
public double Population { get; set; }
public double Density { get; set; }
};
private List<PopulationDetail> PopulationDetails = new
List<PopulationDetail> {
new PopulationDetail {
Name= "Afghanistan",
Population= 29863010,
```

```
Density= 119
},
...
};
}
```

In the above example, both **name** fields contain the same value as **Afghanistan**, this value is matched in both shape data and data source, so that the details associated with **Afghanistan** will be mapped to the corresponding shape and used to color the corresponding shape, display data labels, display tooltips, and more.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
DataSource="PopulationDetails"
ShapeDataPath="Name"
ShapePropertyPath='new string[] { "name" }' TValue="PopulationDetail">
@* It display data label for bounded items *@
<MapsDataLabelSettings Visible="true"
LabelPath="Name"></MapsDataLabelSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class PopulationDetail
{
public string Code { get; set; }
public double Value { get; set; }
public string Name { get; set; }
public double Population { get; set; }
public double Density { get; set; }
};
private List<PopulationDetail> PopulationDetails = new
List<PopulationDetail> {
new PopulationDetail {
Code = "AF",
Value= 53,
Name= "Afghanistan",
Population= 29863010,
Density= 119
},
new PopulationDetail {
Code= "AL",
Value= 117,
Name= "Albania",
Population= 3195000,
Density= 111
},
new PopulationDetail {
Code= "DZ",
Value= 15,
Name= "Algeria",
```



```
Population= 34895000,  
Density= 15  
}  
};  
}
```



#### Fetching data from JSON file

To read the JSON file data, convert it to the C# object, and assign it to the [DataSource](#) property.

The **Http.GetJsonAsync** is used in the **OnInitAsync** lifecycle method to load JSON file data. As this will be executed asynchronously, check whether **populationDensity** is available, render the Maps component, or display the loading statement.

#### ASPX-CS

```
@inject HttpClient Http;  
@using Syncfusion.Blazor.Maps  
@if (PopulationDensity == null)  
{  
    <p><em>Loading Maps component...</em></p>  
}  
else  
{  
    <SfMaps>  
        <MapsLayers>  
            <MapsLayer ShapeData='new {dataOptions  
                ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'  
                DataSource="PopulationDensity">
```

```
ShapeDataPath="Name"
ShapePropertyPath='new string[] { "name" }' TValue="PopulationData">
<MapsDataLabelSettings Visible="true"
LabelPath="Name"></MapsDataLabelSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
}
@code{
PopulationData[] PopulationDensity;
protected override async Task OnInitAsync()
{
PopulationDensity = await Http.GetJsonAsync<PopulationDensity[]>("sample-
data/PopulationDensity.json");
}
public class PopulationData
{
public string Code { get; set; }
public double Value { get; set; }
public string Name { get; set; }
public double Population { get; set; }
public float Density { get; set; }
}
}
```

Here, the `PopulationDensity.json` file contains following data.

### JSON

```
[
{
"code": "AF",
"value": 53,
"name": "Afghanistan",
"population": 29863010,
"density": 119
},
{
"code": "AL",
"value": 117,
"name": "Albania",
"population": 3195000,
"density": 111
},
{
"code": "DZ",
"value": 15,
"name": "Algeria",
"population": 34895000,
"density": 15
}
]
```



### Layers in Blazor Maps Component

The Maps component is rendered through [MapsLayers](#) and any number of layers can be added to the Maps.

#### Multilayer

The Multilayer support allows loading multiple shape files and map providers in a single container, enabling Maps to display more information. The shape layer or map providers are the main layers of the Maps. Multiple layers can be added as **SubLayer** over the main layers using the [Type](#) property in [MapsLayer](#).

#### Sublayer

Sublayer is a type of shape file layer. It allows loading multiple shape files in a single map view. For example, a sublayer can be added over the main layer to view geographic features such as rivers, valleys and cities in a map of a country. Similar to the main layer, elements in the Maps such as markers, bubbles, color mapping and legends can be added to the sub-layer.

In this example, the United States map shape is used as shape data by utilizing **usa.ts** file, and **texas.ts** and **california.ts** files are used as sub-layers in the United States map.

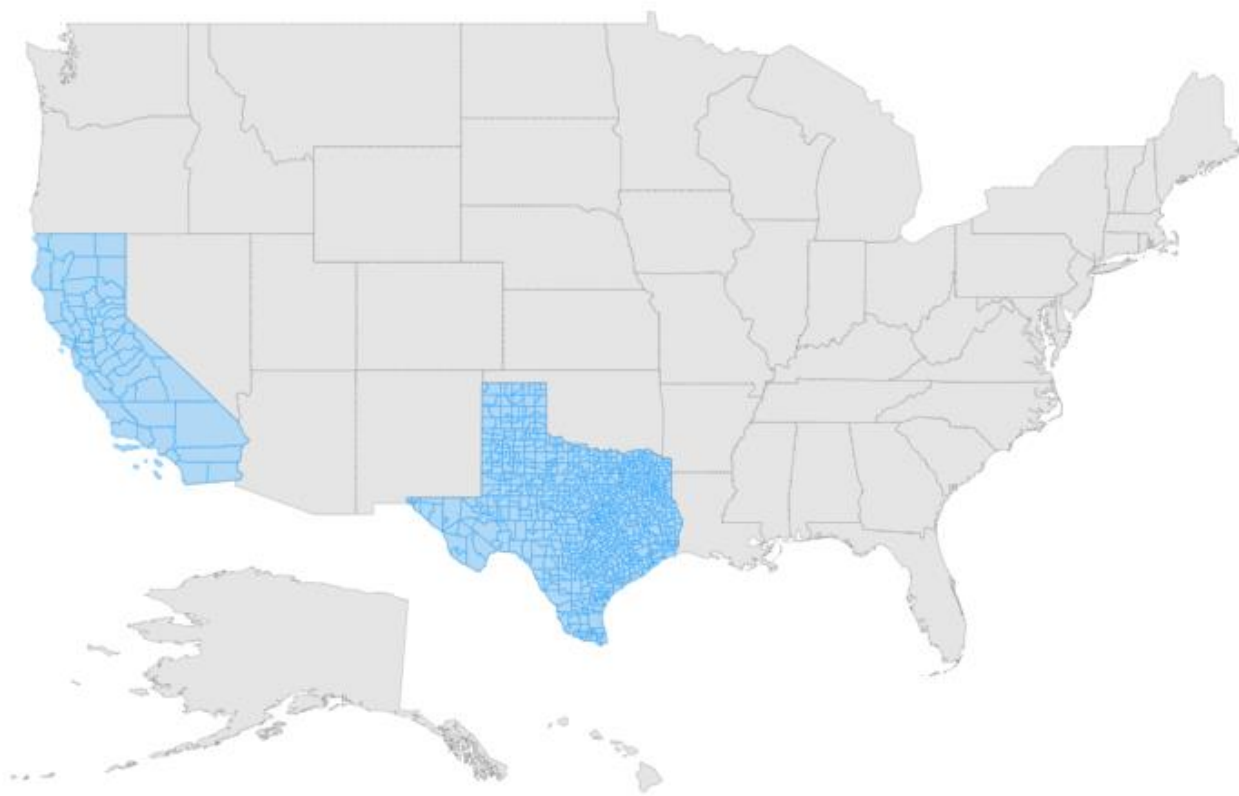
#### **CSHARP**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
    <MapsShapeSettings Fill="#E5E5E5">
```

```

<MapsShapeBorder Color="black" Width="0.1"></MapsShapeBorder>
</MapsShapeSettings>
</MapsLayer>
<MapsLayer ShapeData='new {dataOptions =
  "https://cdn.syncfusion.com/maps/map-data/texas.json"}'
  Type="Syncfusion.Blazor.Maps.Type.SubLayer" TValue="string">
  <MapsShapeSettings Fill="rgba(141, 206, 255, 0.6)">
  <MapsShapeBorder Color="#1a9cff" Width="0.25"></MapsShapeBorder>
  </MapsShapeSettings>
</MapsLayer>
<MapsLayer ShapeData='new {dataOptions=
  "https://cdn.syncfusion.com/maps/map-data/california.json"}'
  Type="Syncfusion.Blazor.Maps.Type.SubLayer" TValue="string">
  <MapsShapeSettings Fill="rgba(141, 206, 255, 0.6)">
  <MapsShapeBorder Color="#1a9cff" Width="0.25"></MapsShapeBorder>
  </MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>

```

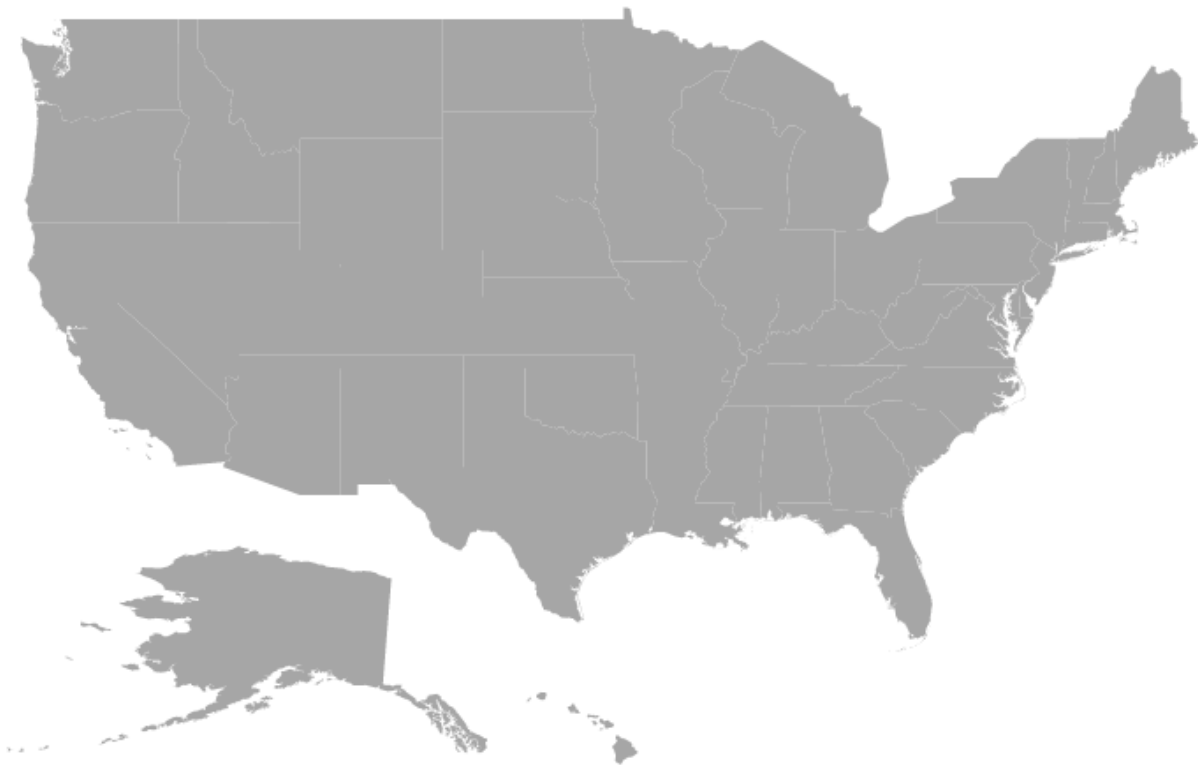


#### Displaying different layer in the view

Multiple shape files and map providers can be loaded simultaneously in Maps. The [BaseLayerIndex](#) property is used to determine which layer on the user interface should be displayed. This property is used for the Maps drill-down feature, so when the [BaseLayerIndex](#) value is changed, the corresponding shape is loaded. In this example, two layers can be loaded with the World map and the United States map. Based on the given [BaseLayerIndex](#) value the corresponding shape will be loaded in the user interface. If the [BaseLayerIndex](#) value is set to **0**, then the world map will be loaded.

## CSHARP

```
@using Syncfusion.Blazor.Maps
@* To switch the layer, set `BaseLayerIndex` *@
<SfMaps BaseLayerIndex="1">
  <MapsLayers>
    <MapsLayer ShapeData='new { dataOptions =
      "https://cdn.syncfusion.com/maps/map-data/world-map.json" }'
      TValue="string"/>
    <MapsLayer ShapeData='new { dataOptions =
      "https://cdn.syncfusion.com/maps/map-data/usa.json" }' TValue="string"/>
  </MapsLayers>
</SfMaps>
```



See also

- [Display geometry shapes in Bing maps](#)

## Providers

### OpenStreetMap in Blazor Maps Component

The OpenStreetMap (OSM) is the online Maps provider built by a community of developers. It is free to use under an open license. It allows to view geographical data in a collaborative way from anywhere on the earth. The OSM map provides small tile images based on our requests and combines those images into a single image to display the map area in the Maps component.

### Adding OpenStreetMap

The OSM Maps can be rendered using by setting the [UrlTemplate](#) property with the OSM tile server URL. For more details about the OSM tile server, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer
      UrlTemplate="https://tile.openstreetmap.org/level/tileX/tileY.png"
      TValue="string"></MapsLayer>
    </MapsLayers>
  </SfMaps>
```



### Enable zooming and panning

The OSM Maps layer can be zoomed and panned. Zooming helps to get a closer look at a particular area on a Maps for in-depth analysis. Panning helps to move a Maps around to focus the targeted area.

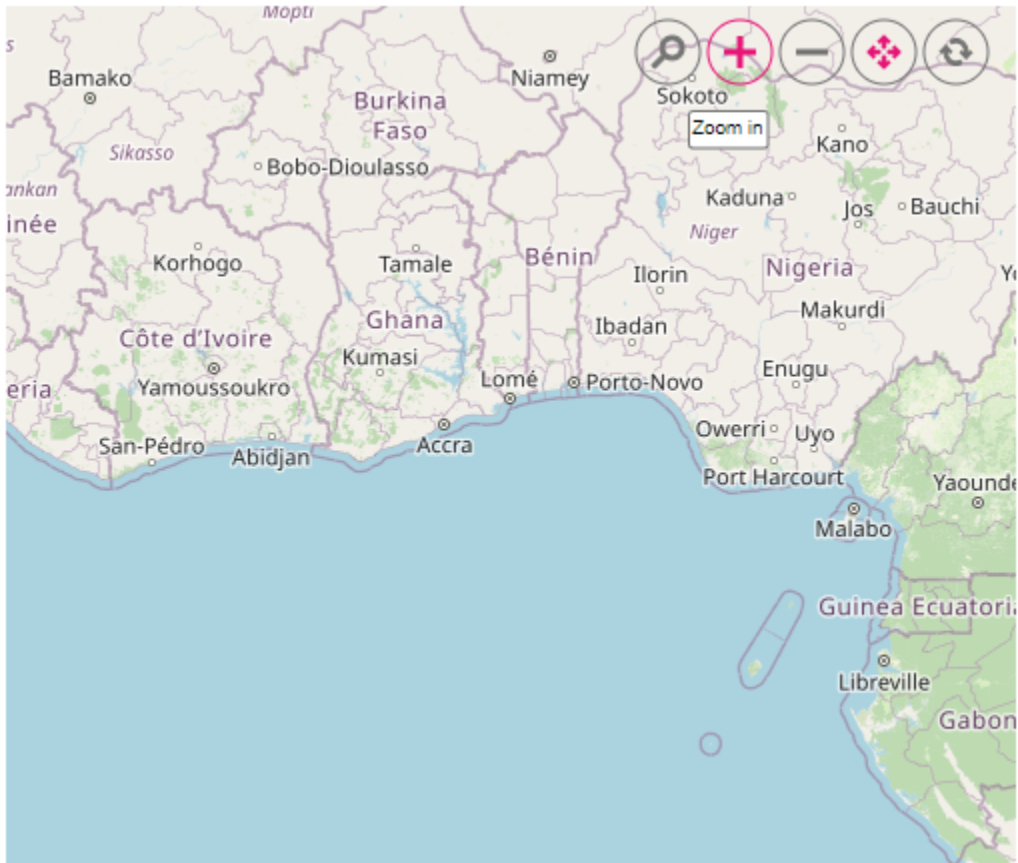
#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  @* To zoom and pan *@
  <MapsZoomSettings Enable="true">
    <MapsZoomToolbarSettings>
```

```

<MapsZoomToolBarButton ToolbarItems="new List<ToolBarItem>() {
ToolBarItem.Zoom, ToolBarItem.ZoomIn, ToolBarItem.ZoomOut,
ToolBarItem.Pan, ToolBarItem.Reset }"></MapsZoomToolBarButton>
</MapsZoomToolBarSettings>
</MapsZoomSettings>
<MapsLayers>
<MapsLayer
UrlTemplate="https://tile.openstreetmap.org/level/tileX/tileY.png"
TValue="string"></MapsLayer>
</MapsLayers>
</SfMaps>

```



#### *Adding markers and navigation line*

Markers can be added to the layers of OSM Maps by setting the corresponding location's coordinates of latitude and longitude using [MapsMarker](#). Navigation lines can be added on top of an OSM Maps layer for highlighting a path among various places by setting the corresponding location's coordinates of latitude and longitude in the [MapsNavigationLine](#).

#### **ASPX-CS**

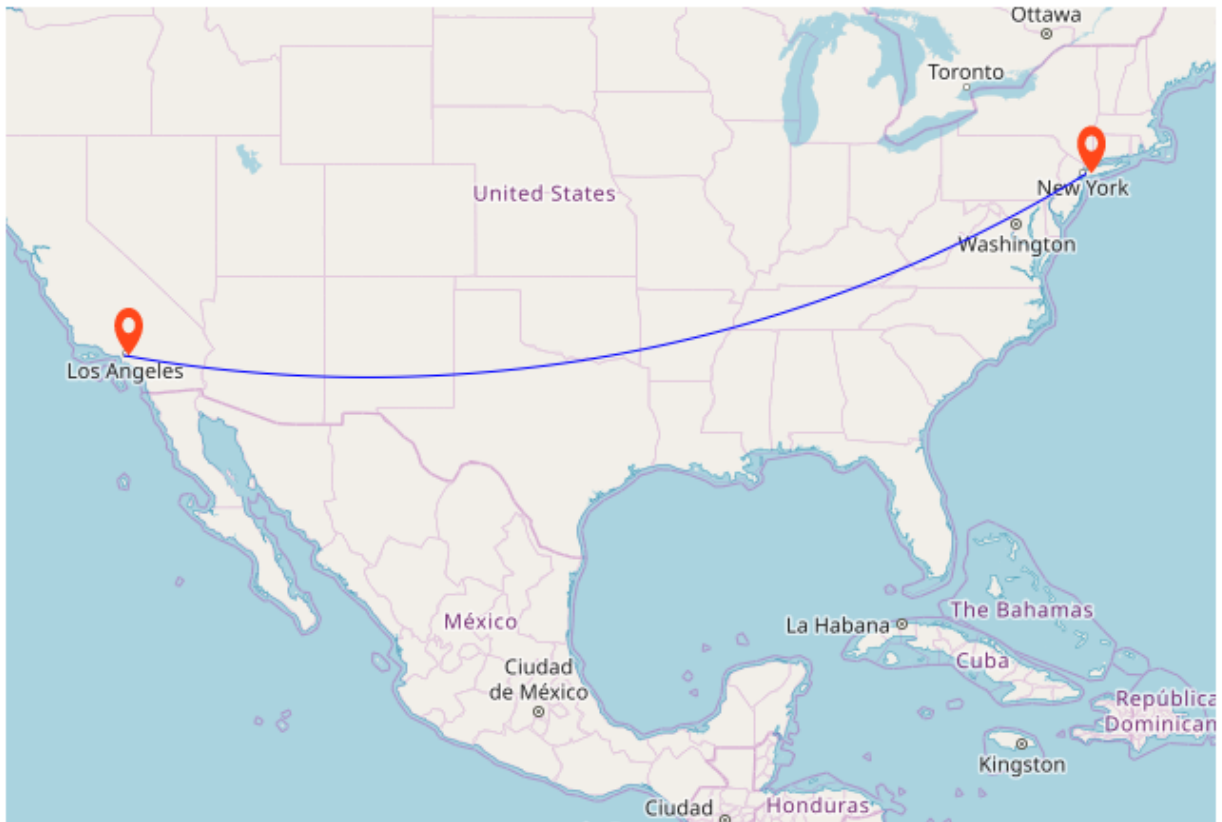
```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsZoomSettings ZoomFactor="4"></MapsZoomSettings>
<MapsCenterPosition Latitude="29.394708" Longitude="-
94.954653"></MapsCenterPosition>
<MapsLayers>

```

```
<MapsLayer
UrlTemplate="https://tile.openstreetmap.org/level/tileX/tileY.png"
TValue="string">
@* Add marker *@
<MapsMarkerSettings>
<MapsMarker Visible="true" Height="25" Width="15" DataSource="Cities"
TValue="City">
</MapsMarker>
</MapsMarkerSettings>
@* Add navigation line *@
<MapsNavigationLines>
<MapsNavigationLine Visible="true" Color="blue" Angle="0.1" Latitude="new
double[] {34.060620, 40.724546}"
Longitude="new double[] {-118.330491, -73.850344}">
</MapsNavigationLine>
</MapsNavigationLines>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
}
private List<City> Cities = new List<City> {
new City { Latitude = 34.060620, Longitude = -118.330491, Name="California"
},
new City{ Latitude = 40.724546, Longitude = -73.850344, Name="New York"}
};
}
```



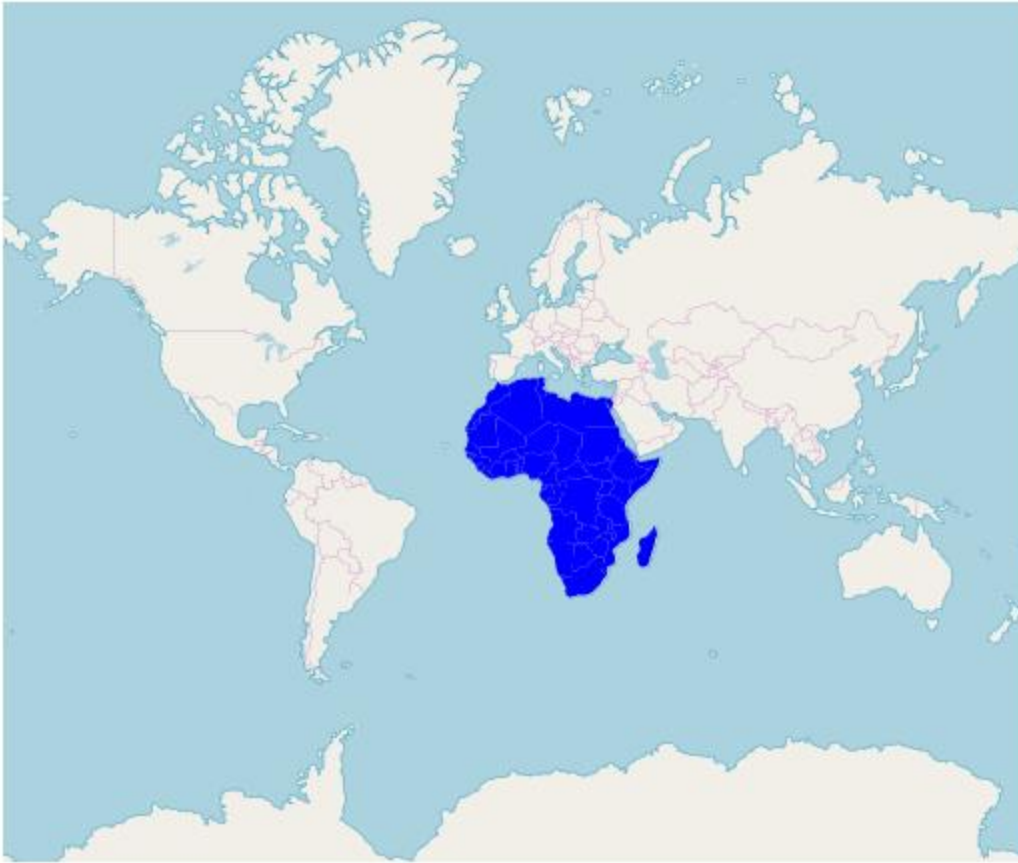


### Adding sublayer

Any GeoJSON shape can be rendered as a sublayer on top of the OSM Maps layer for highlighting a particular continent or country in the OSM map by adding another layer and specifying the [Type](#) of [MapsLayer](#) to [SubLayer](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer
      UrlTemplate="https://tile.openstreetmap.org/level/tileX/tileY.png"
      TValue="string">
    </MapsLayer>
    @* To add geometry shape as sublayer *@
    <MapsLayer ShapeData='new {dataOptions =
      "https://cdn.syncfusion.com/maps/map-data/africa.json"}'
      Type="Syncfusion.Blazor.Maps.Type.SubLayer" TValue="string">
      <MapsShapeSettings Fill="blue"></MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```



### Bing Maps in Blazor Maps Component

Bing Maps is an online Maps provider, owned by Microsoft, for accessing the external geospatial imagery services for deep-zoom satellite view which is supported in the Blazor Maps component. This provides the ability to visualize satellite, aerial, and street Maps without using any external shape files. As like OSM, it provides Maps tile images based on our requests and combines those images into a single one to display Maps area.

#### Adding Bing Maps

The Bing Maps can be rendered by setting the [UrlTemplate](#) property with the URL generated from the `GetBingUrlTemplate` method in the Maps component. The format of the required URL of Bing Maps varies from other map providers. As a result, we have included a built-in `GetBingUrlTemplate` method that returns the URL in a generic format. In the meantime, a subscription key is required for Bing Maps. Follow the steps in this [link](#) to generate an API key, then append it to the Bing Maps URL before passing it to the `GetBingUrlTemplate` method. The URL returned by this method must be passed to the `UrlTemplate` property.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer UrlTemplate="@UrlTemplate" TValue="string"></MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
```

```
public string UrlTemplate;  
protected override async Task OnInitializedAsync()  
{  
    UrlTemplate = await  
    SfMaps.GetBingUrlTemplate("https://dev.virtualearth.net/REST/V1/Imagery/Meta  
data/RoadOnDemand?output=json&uriScheme=https&key=");  
}  
}
```

In the above URL passed to the `GetBingUrlTemplate` method, specify the Bing Maps key.



#### *Types of Bing Maps*

Bing Maps provides different types of Maps and it can be viewed in the Maps component.

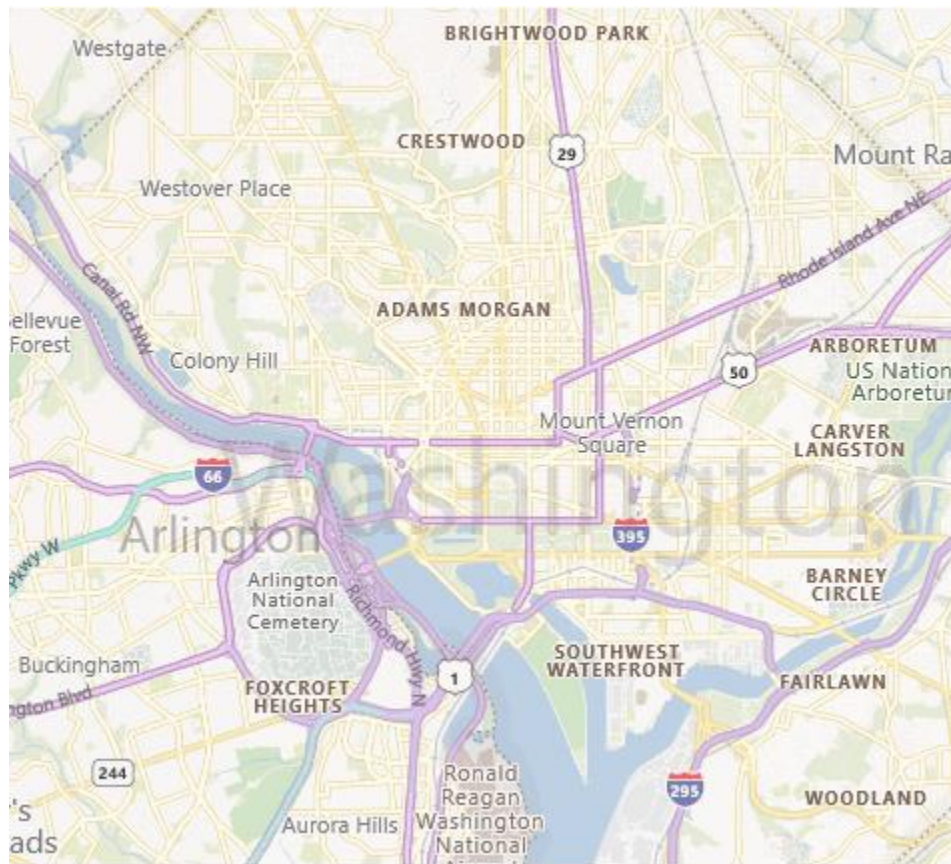
- **Aerial** - Displays satellite images to highlight roads and major landmarks for easy identification.
- **AerialWithLabelsOnDemand** - Displays aerial Maps with labels for the continent, country, ocean, etc.
- **Road** - Displays the default Maps view of roads, buildings, and geography.
- **CanvasDark** - Displays dark version of the road Maps.
- **CanvasLight** - Displays light version of the road Maps.
- **CanvasGray** - Displays grayscale version of the road Maps.

The above types can also be rendered in the Maps component by specifying their URL in the `UrlTemplate` property in the `MapsLayer` class. You can learn more about the available types and the URL for it by visiting the official websites of Bing Maps.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer UrlTemplate="@UrlTemplate" TValue="string"></MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
  public string UrlTemplate;
  protected override async Task OnInitializedAsync()
  {
    UrlTemplate = await
    SfMaps.GetBingUrlTemplate("https://dev.virtualearth.net/REST/V1/Imagery/Meta
    data/CanvasGray?output=json&uriScheme=https&key=");
  }
}
```

In the above URL passed to the `GetBingUrlTemplate` method, specify the Bing Maps key.



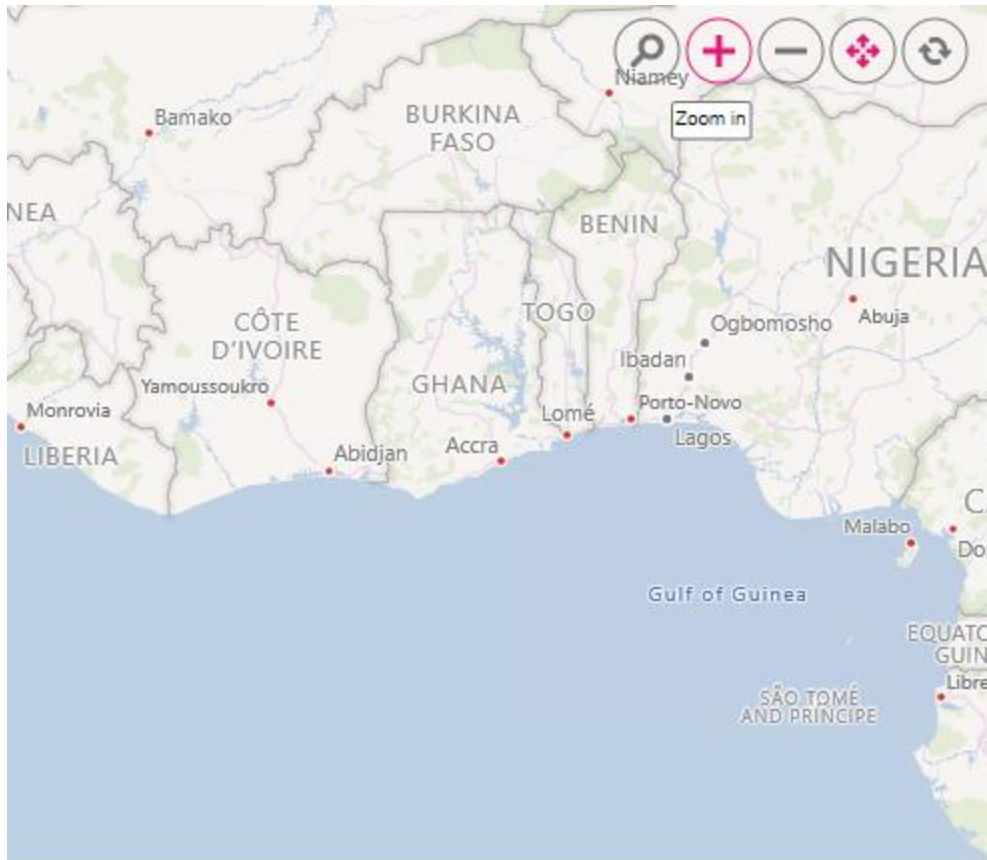
### *Enable zooming and panning*

Bing Maps layer can be zoomed and panned. Zooming helps to get a closer look at a particular area on Maps for in-depth analysis. Panning helps to move Maps around to focus the targeted area.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
@* To zoom and pan *@
<MapsZoomSettings Enable="true">
<MapsZoomToolbarSettings>
<MapsZoomToolbarButton ToolbarItems="new List<ToolbarItem>() {
ToolbarItem.Zoom, ToolbarItem.ZoomIn, ToolbarItem.ZoomOut,
ToolbarItem.Pan, ToolbarItem.Reset }"></MapsZoomToolbarButton>
</MapsZoomToolbarSettings>
</MapsZoomSettings>
<MapsLayers>
<MapsLayer UrlTemplate="@UrlTemplate" TValue="string"></MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public string UrlTemplate;
protected override async Task OnInitializedAsync()
{
UrlTemplate = await
SfMaps.GetBingUrlTemplate("https://dev.virtualearth.net/REST/V1/Imagery/Meta
data/RoadOnDemand?output=json&uriScheme=https&key=");
}
}
```

In the above URL passed to the `GetBingUrlTemplate` method, specify the Bing Maps key.



#### *Adding markers and navigation line*

Markers can be added to the layers of Bing Maps by setting the corresponding location's coordinates of latitude and longitude using [MapsMarker](#). Navigation lines can be added on top of an Bing Maps layer for highlighting a path among various places by setting the corresponding location's coordinates of latitude and longitude in the [MapsNavigationLine](#).

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings ZoomFactor="4"></MapsZoomSettings>
  <MapsCenterPosition Latitude="29.394708" Longitude="-
  94.954653"></MapsCenterPosition>
  <MapsLayers>
    <MapsLayer UrlTemplate="@UrlTemplate" TValue="string">
      @* Add marker *@
      <MapsMarkerSettings>
        <MapsMarker Visible="true" Height="25" Width="15" DataSource="Cities"
        TValue="City">
        </MapsMarker>
      </MapsMarkerSettings>
    </MapsLayer>
    @* Add navigation line *@
    <MapsNavigationLines>
      <MapsNavigationLine Visible="true" Color="blue" Angle="0.1" Latitude="new
      double[]{34.060620, 40.724546}"
      Longitude="new double[]{-118.330491,-73.850344}">
      </MapsNavigationLine>
    </MapsNavigationLines>
  </MapsLayers>
</SfMaps>
```

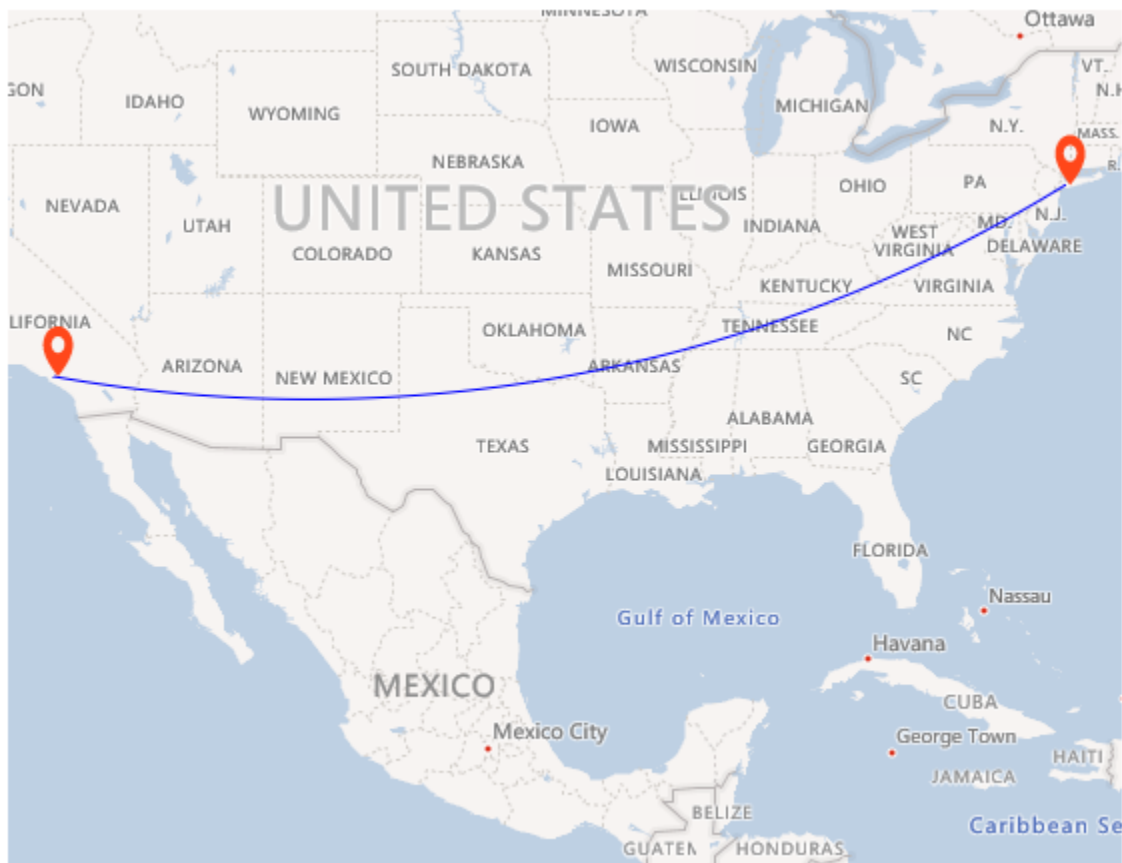


```

</MapsNavigationLines>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
}
private List<City> Cities = new List<City> {
new City { Latitude = 34.060620, Longitude = -118.330491, Name="California"
},
new City{ Latitude = 40.724546, Longitude = -73.850344, Name="New York"}
};
public string UrlTemplate;
protected override async Task OnInitializedAsync()
{
UrlTemplate = await
SfMaps.GetBingUrlTemplate("https://dev.virtualearth.net/REST/V1/Imagery/Meta
data/RoadOnDemand?output=json&uriScheme=https&key=");
}
}

```

In the above URL passed to the `GetBingUrlTemplate` method, specify the Bing Maps key.



### *Adding sublayer*

Any GeoJSON shape can be rendered as a sublayer on top of the Bing Maps layer for highlighting a particular continent or country in Bing Maps by adding another layer and specifying the [Type](#) of [MapsLayer](#) to **SubLayer**.

---

[Refer to section](#) to learn how to add a sublayer in Bing Maps

---

### Other Maps in Blazor Maps Component

Apart from OpenStreetMap and Bing Maps, you can also render Maps from other online map service providers by specifying the URL provided by those providers in the [UrlTemplate](#) property. The URL template concept has been implemented in such a way that any online map service providers using the following template can benefit from previewing their Map in the Syncfusion Blazor Maps component.

Sample Template: `https://< domain_name >/maps/basic/{z}/{x}/{y}.png`

- "\${z}" - It represents zoom factor (level).
- "\${x}" - It indicates tile image x-position (tileX).
- "\${y}" - It indicates tile image y-position (tileY).

In this case, the key generated for those online map service providers can also be appended to the URL. This allows you to create personalized Maps with your own content and imagery. In this example, Google Maps is rendered.

---

Refer to [Google Maps Licensing](#).

---

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer
      UrlTemplate="http://mt1.google.com/vt/lyrs=m@129&hl=en&x=tileX&y=tileY&z=level" TValue="string">
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```





### Enable zooming and panning

Tile Maps layer can be zoomed and panned. Zooming helps to get a closer look at a particular area on a Maps for in-depth analysis. Panning helps to move a Maps around to focus the targeted area.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
@* To zoom and pan *@
<MapsZoomSettings Enable="true">
<MapsZoomToolBarSettings>
<MapsZoomToolBarButton ToolBarItem="new List<ToolBarItem>() {
ToolBarItem.Zoom, ToolBarItem.ZoomIn, ToolBarItem.ZoomOut,
ToolBarItem.Pan, ToolBarItem.Reset }"></MapsZoomToolBarButton>
</MapsZoomToolBarSettings>
</MapsZoomSettings>
<MapsLayers>
<MapsLayer
UrlTemplate="http://mt1.google.com/vt/lyrs=m@129&hl=en&x=tileX&y=tileY&z=lev
el" TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
```



### Adding markers and navigation line

Markers can be added to the layers of tile Maps by setting the corresponding location's coordinates of latitude and longitude using [MapsMarker](#) class. Navigation lines can be added on top of the tile Maps layer for highlighting a path among various places by setting the corresponding location's coordinates of latitude and longitude in the [MapsNavigationLine](#).

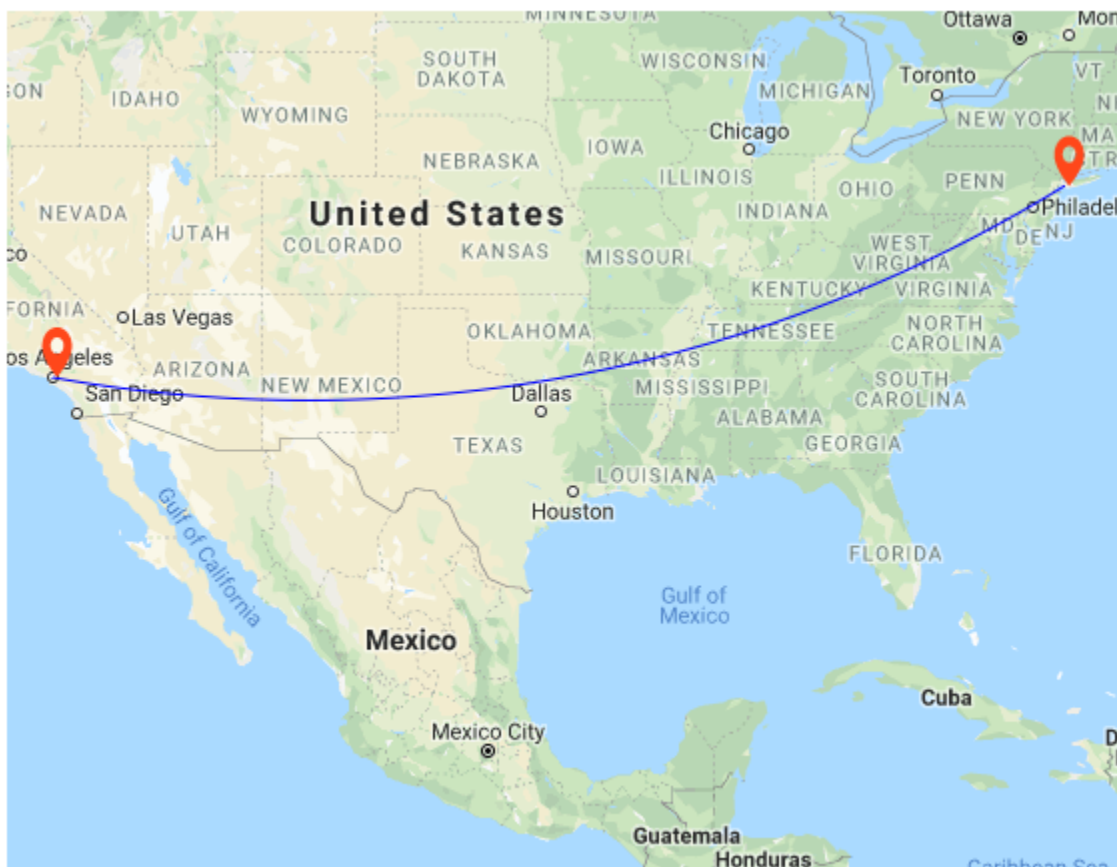
### CSHARP

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsZoomSettings ZoomFactor="4"></MapsZoomSettings>
<MapsCenterPosition Latitude="29.394708" Longitude="-
94.954653"></MapsCenterPosition>
<MapsLayers>
<MapsLayer
UrlTemplate="http://mt1.google.com/vt/lyrs=m@129&hl=en&x=tileX&y=tileY&z=lev
el" TValue="string">
@* Add markers *@
<MapsMarkerSettings>
<MapsMarker Visible="true" Height="25" Width="15" DataSource="Cities"
TValue="City">
</MapsMarker>
</MapsMarkerSettings>
@* Add navigation line *@
```

```

<MapsNavigationLines>
<MapsNavigationLine Visible="true" Color="blue" Angle="0.1" Latitude="new
double[] {34.060620, 40.724546}"
Longitude="new double[] {-118.330491, -73.850344}">
</MapsNavigationLine>
</MapsNavigationLines>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
}
private List<City> Cities = new List<City> {
new City { Latitude = 34.060620, Longitude = -118.330491, Name="California"
},
new City{ Latitude = 40.724546, Longitude = -73.850344, Name="New York"}
};
}

```



### Adding sublayer

Any GeoJSON shape can be rendered as a sublayer on top of the tile Maps layer for highlighting a particular continent or country in tile maps by adding another layer and specifying the [Type](#) property of [MapsLayer](#) to **SubLayer**.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer
      UrlTemplate="http://mt1.google.com/vt/lyrs=m@129&hl=en&x=tileX&y=tileY&z=level" TValue="string">
    </MapsLayer>
    @* To add geometry shape as sublayer *@
    <MapsLayer ShapeData='new {dataOptions =
      "https://cdn.syncfusion.com/maps/map-data/africa.json"}'
      Type="Syncfusion.Blazor.Maps.Type.SubLayer" TValue="string">
    <MapsShapeSettings Fill="blue"></MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```



#### *Other supportive online map service providers*

The Maps component can also render the following online map service providers, which are listed below.

- MapBox
- TomTom
- ESRI

## ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLayers>
@* Renders Mapbox map *@
<MapsLayer UrlTemplate="https://api.mapbox.com/styles/v1/mapbox/streets-
v11/tiles/level/tileX/tileY?access_token=" TValue="string">
</MapsLayer>
@* Renders TomTom map *@
<!--<MapsLayer
UrlTemplate="http://api.tomtom.com/map/1/tile/basic/main/level/tileX/tileY.p
ng?key=zzVjM8webeABaPadifIf9hFpmdC9XzmG" TValue="string">
</MapsLayer-->
@* Renders ESRI map *@
<!--<MapsLayer UrlTemplate="https://ibasemaps-
api.arcgis.com/arcgis/rest/services/World_Imagery/MapServer/tile/level/tileY
/tileX?apiKey=AAPK04316d918e224b339f72d107b5aef880I2MT0hI3L2xIX4DMcuEELiOcb4
DRmxGp_-hqlsFhziOvqBwel-uIA-87Dp9h3eI" TValue="string">
</MapsLayer-->
</MapsLayers>
</SfMaps>
```





## Customization in Blazor Maps Component

### Setting the size for Maps

The width and height of the Maps can be set using the [Width](#) and [Height](#) properties in the Maps component. Percentage or pixel values can be used for the height and width values.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps Height="600px" Width="300px">
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      TValue="string">
      @* To customize map shape *@
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```



### Maps title

The title for the Maps can be set using the [MapsTitleSettings](#) class. It can be customized using the following properties and classes.

- [Alignment](#) - To customize the alignment for the text in the title for the Maps. The possible values are [Center](#), [Near](#) and [Far](#).
- [Description](#) - To set the description of the title in Maps.
- [Text](#) - To set the text for the title in Maps.
- [MapsTitleTextStyle](#) - To customize the text of the title in Maps.
- [MapsSubtitleSettings](#) - To customize the subtitle for the Maps.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps Height="300px">
  <MapsTitleSettings Text="Maps Component" Description="Maps"
  Alignment="Syncfusion.Blazor.Maps.Alignment.Center">
    <MapsTitleTextStyle Color="Green" FontFamily="Times New Roman"
    FontStyle="italic" FontWeight="bold">
    </MapsTitleTextStyle>
    <MapsSubtitleSettings Text="Default sample"></MapsSubtitleSettings>
  </MapsTitleSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
    ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
    TValue="string">
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```

## *Maps Component*

### Default sample



### Setting theme

The Maps control supports following themes.

- Material
- Fabric
- Bootstrap
- HighContrast
- MaterialDark
- FabricDark
- BootstrapDark
- Bootstrap4
- HighContrastLight
- Tailwind

By default, the Maps are rendered by the **Material** theme. The theme of the Maps component is changed using the [Theme](#) property.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps Theme="Syncfusion.Blazor.Theme.HighContrastLight">
  <MapsLayers>
```



```
<MapsLayer ShapeData='new {dataOptions  
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'  
TValue="string">  
@* To customize map shape *@  
</MapsLayer>  
</MapsLayers>  
</SfMaps>
```



### Customizing Maps container

The following property and classes are available to customize the container in the Maps.

- [Background](#) - To apply the background color to the container in the Maps.
- [MapsBorder](#) - To customize the color and width of the border of the Maps.
- [MapsMargin](#) - To customize the margins of the Maps.

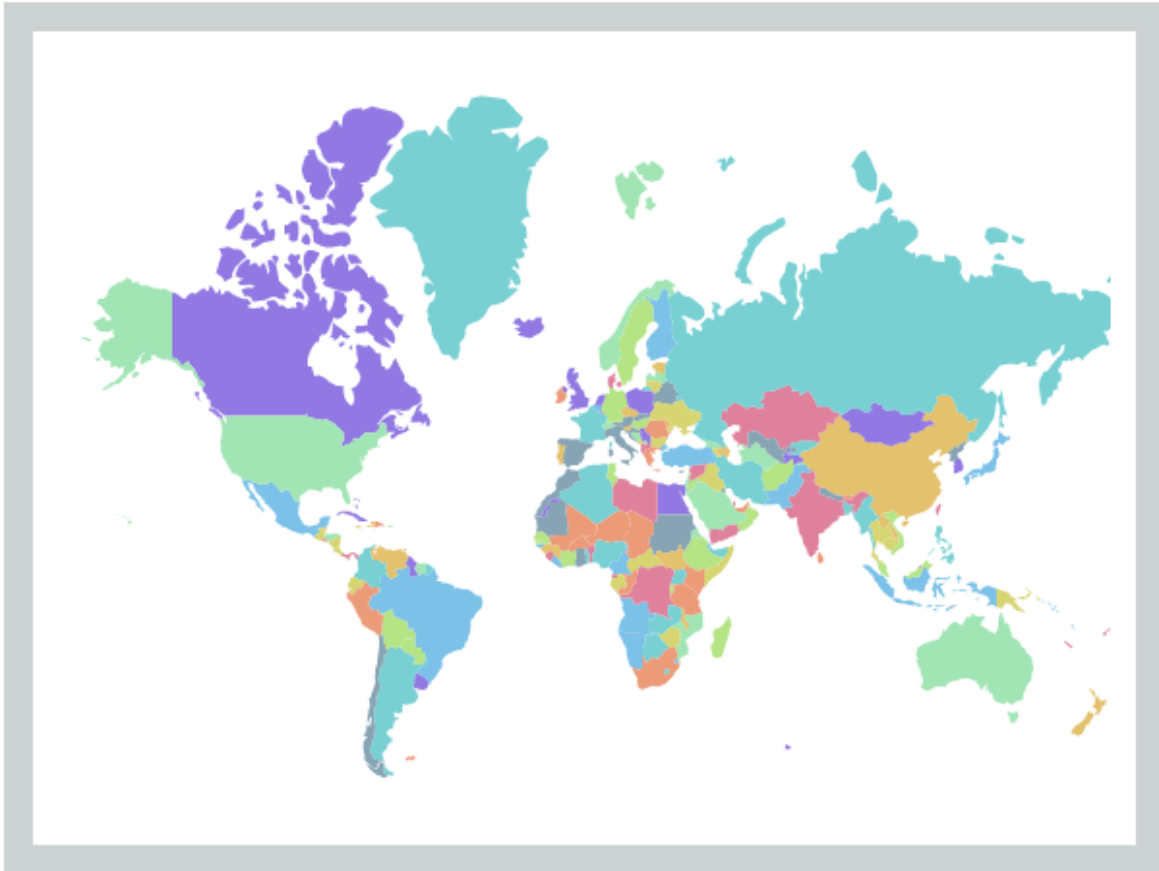
### ASPX-CS

```
@using Syncfusion.Blazor.Maps  
<SfMaps Height="300px" Width="400px" Background="#CCD1D1">  
<MapsBorder Color="green" Width="2"></MapsBorder>  
<MapsMargin Bottom="10" Left="10" Right="10" Top="10"></MapsMargin>  
<MapsLayers>
```

```

<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
TValue="string">
<MapsShapeSettings Autofill="true"></MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>

```



### Customizing Maps area

By default, the background color of the shape maps is set as **white**. To modify the background color of the Maps area, the [Background](#) property in the [MapsAreaSettings](#) is used. The border of the Maps area can be customized using the [MapsAreaBorder](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsAreaSettings Background="#e6e2d3">
<MapsBorder Color="green" Width="2"></MapsBorder>
</MapsAreaSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
TValue="string">

```

```
@* To set shape color automatically *@  
<MapsShapeSettings Autofill="true"></MapsShapeSettings>  
</MapsLayer>  
</MapsLayers>  
</SfMaps>
```



### Customizing the shapes

The following properties and class are available in [MapsShapeSettings](#) to customize the shapes of the Maps component.

- [Fill](#) - To apply the color to the shapes.
- [Autofill](#) - To apply the palette colors to the shapes if it is set as true.
- [Palette](#) - To set the custom palette for the shapes.
- [DashArray](#) - To define the pattern of dashes and gaps that is applied to the outline of the shapes.
- [Opacity](#) - To customize the transparency for the shapes.
- [MapsShapeBorder](#) - To customize the color and width of the border of the shapes.

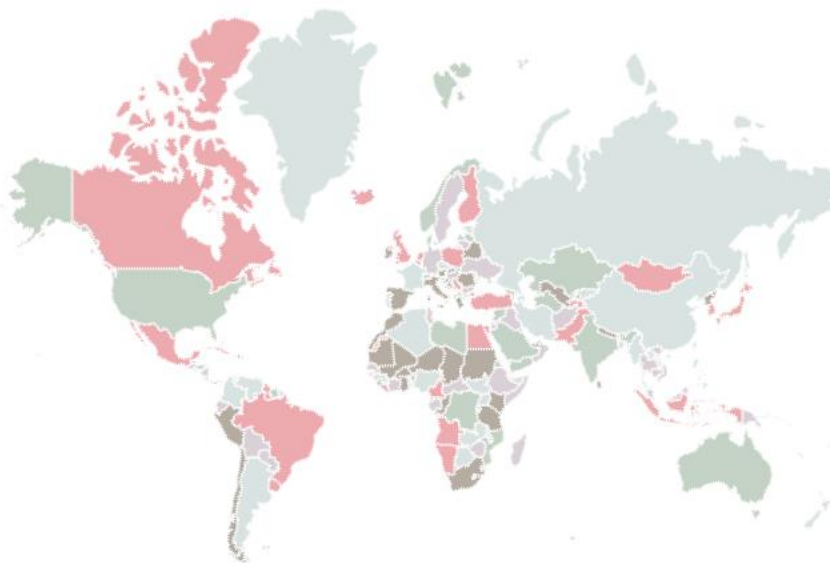
### ASPX-CS

```
@using Syncfusion.Blazor.Maps  
<SfMaps>
```

```

<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
TValue="string">
@* To customize map shape *@
<MapsShapeSettings Autofill="true" Palette='new string[] {"#d6cbd3",
"#ecala6", "#bdcebe", "#ada397", "#d5e1df"}' DashArray="1" Opacity=0.9>
<MapsShapeBorder Color="#FFFFFF" Width="2"></MapsShapeBorder>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>

```



Setting color to the shapes from the data source

The color for each shape in the Maps can be set using the [ColorValuePath](#) property of [MapsShapeSettings](#). The value for the [ColorValuePath](#) property is the field name from the data source of the [MapsShapeSettings](#) which contains the color values.

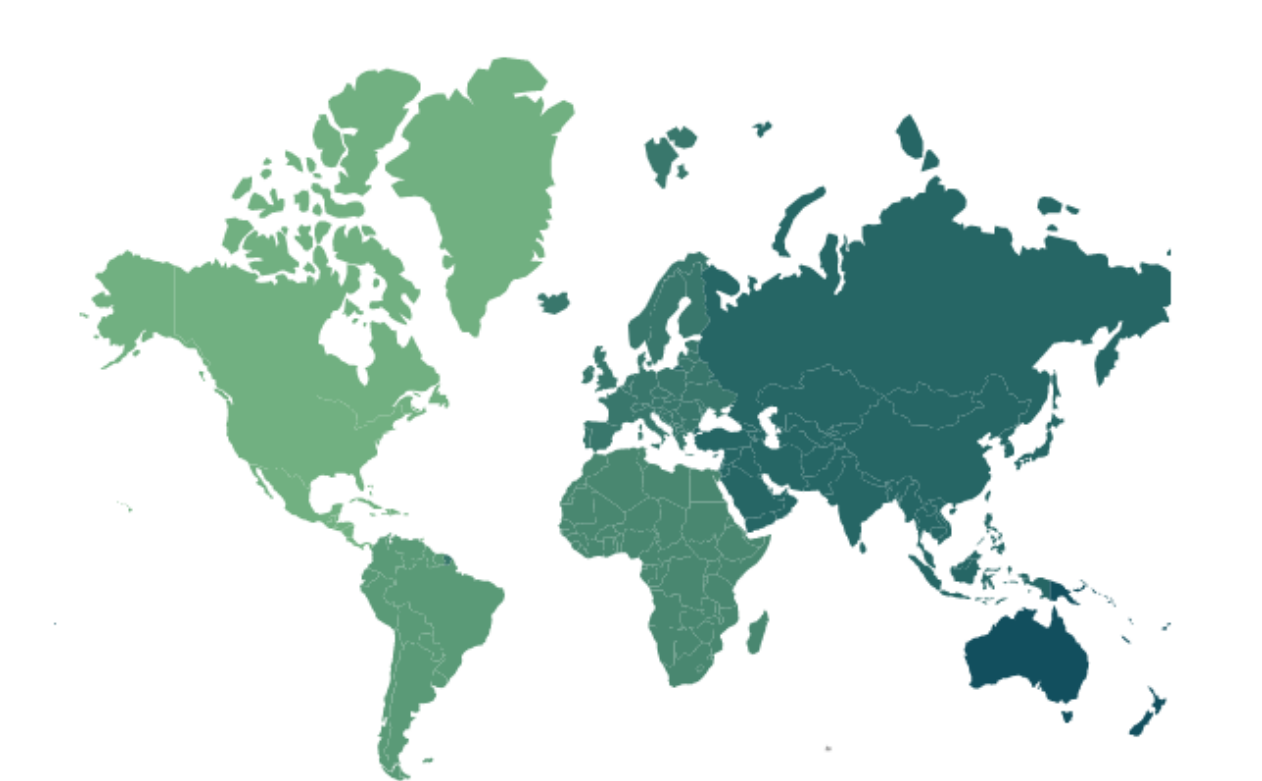
#### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapeDataPath="Continent" ShapePropertyPath='new string[] {"continent"}'
DataSource="ShapeColor" TValue="Data">
<MapsShapeSettings ColorValuePath="Color"></MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class Data
{

```

```
public string Continent { get; set; }  
public string Color { get; set; }  
};  
public List<Data> ShapeColor = new List<Data>{  
new Data { Continent= "North America", Color= "#71B081" },  
new Data { Continent= "South America", Color= "#5A9A77" },  
new Data { Continent= "Africa", Color= "#498770" },  
new Data { Continent= "Europe", Color= "#39776C" },  
new Data { Continent= "Asia", Color= "#266665" },  
new Data { Continent= "Australia", Color= "#124F5E" }  
};  
}
```



### Projection type

The Maps control supports the following projection types:

- Mercator
- Equirectangular
- Miller
- Eckert3
- Eckert5
- Eckert6
- Winkel3

- AitOff

By default, the Maps are rendered by the **Mercator** projection type in which the Maps are rendered based on the coordinates. So, the Maps is not stretched. To change the type of projection in the Maps, the [ProjectionType](#) property is used.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
@* To change Maps projection *@
<SfMaps ProjectionType="ProjectionType.Miller">
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      TValue="string">
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```



#### Color Mapping in Blazor Maps Component

Color mapping is used to customize the shape colors based on the given values. It has three types.

1. Range color mapping
2. Equal color mapping
3. Desaturation color mapping.

To add color mapping to the shapes of the Maps, bind the data source to the [DataSource](#) property of the [MapsLayer](#) and set the field name which contains the color value in the data source to the [ColorValuePath](#) property.

Types of color mapping

*Range color mapping*

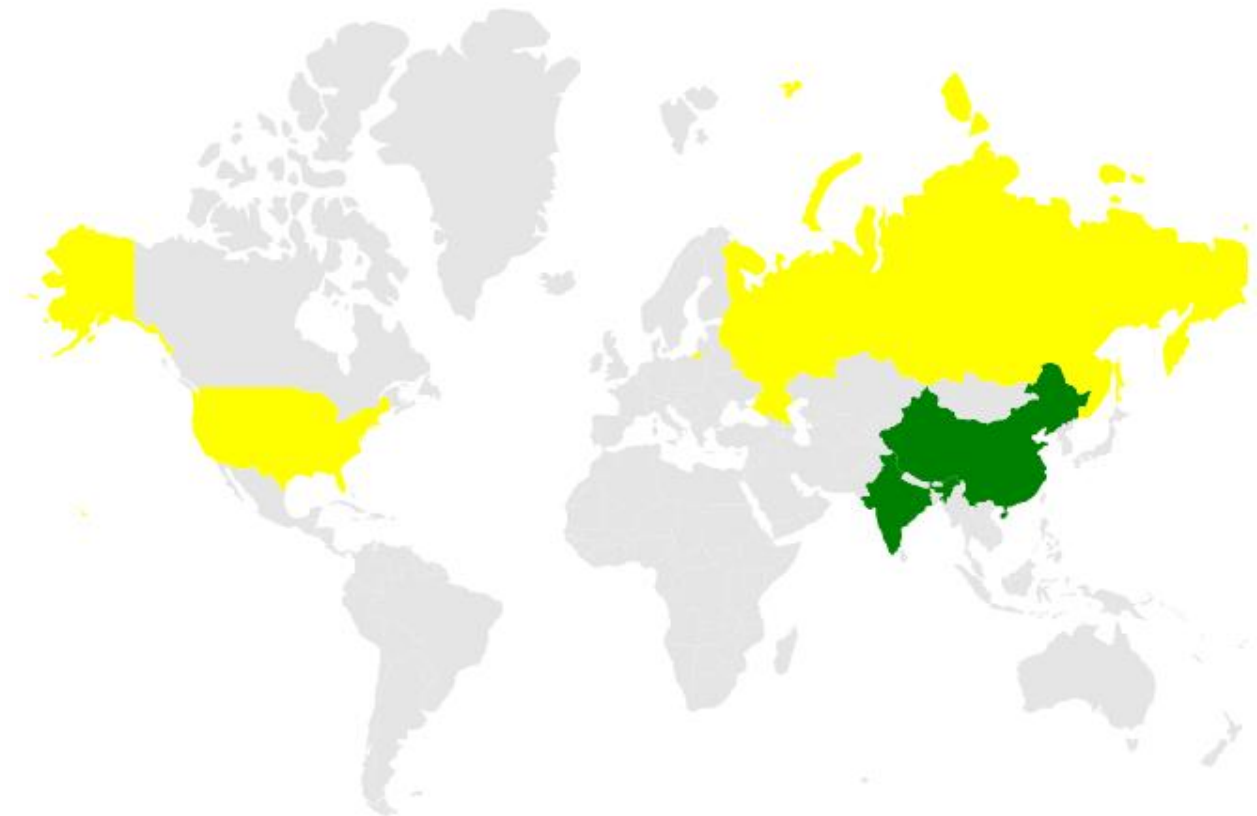
Range color mapping applies the color to the shapes of the Maps which matches the numeric values in the data source within the given color mapping ranges. The [StartRange](#) and [EndRange](#) properties in the [MapsShapeColorMapping](#) are used to specify the color mapping ranges in the Maps.

Bind the **PopulationDetails** data to the [DataSource](#) property of [MapsLayer](#) and set the [ColorValuePath](#) property of [MapsShapeSettings](#) class as **Density**. The range values can be set using the [StartRange](#) and [EndRange](#) properties in the [MapsShapeColorMapping](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
DataSource="PopulationDetails"
ShapeDataPath="Name" ShapePropertyPath='new string[] {"name"}'
TValue="PopulationDetail">
  @* To apply color based on density range *@
  <MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Density">
  <MapsShapeColorMappings>
  <MapsShapeColorMapping StartRange="0.00001" EndRange="100" Color='new
string[] {"yellow"}' />
  <MapsShapeColorMapping StartRange="100" EndRange="400" Color='new string[]
{"green"}' />
  </MapsShapeColorMappings>
  </MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class PopulationDetail
{
    public string Code { get; set; }
    public double Value { get; set; }
    public string Name { get; set; }
    public double Population { get; set; }
    public double Density { get; set; }
};
public List<PopulationDetail> PopulationDetails = new List<PopulationDetail>
{
    new PopulationDetail
    {
        Code = "US", Value = 34, Name = "United States", Population = 325020000,
        Density = 33
    },
    new PopulationDetail
    {
        Code = "RU", Value = 9, Name = "Russia", Population = 142905208, Density =
        8.3
    }
}
```

```
},
new PopulationDetail
{
    Code = "In", Value = 384, Name = "India", Population = 1198003000, Density = 364
},
new PopulationDetail
{
    Code = "CN", Value = 143, Name = "China", Population = 1389750000, Density = 144
}
};
}
```



#### *Equal color mapping*

Equal color mapping applies the color to the shapes of the Maps when the [Value](#) property of [MapsShapeColorMapping](#) matches with the values provided in the data source.

The following example demonstrates the permanent and non-permanent countries in the UN security council, in 2017. Bind the **CouncilMemberDetails** data to the [DataSource](#) property of [MapsLayer](#) class and set the [ColorValuePath](#) property of [MapsShapeSettings](#) class as **Membership**. Set the [Value](#) property in the [MapsShapeColorMapping](#) class to **Permanent** and **Non-Permanent** in the different set of shape color mapping properties. If the corresponding value of the [ColorValuePath](#) property matches with the corresponding field name in the data source, then the given color will be applied.

#### **ASPX-CS**



```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
DataSource="CouncilMemberDetails" ShapeDataPath="Country"
ShapePropertyPath='new string[] { "name" }' TValue="UNCouncil">
      @* To apply color based on membership type *@
      <MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Membership">
        <MapsShapeColorMappings>
          <MapsShapeColorMapping Value="Permanent" Color='new string[] { "#D84444" }' />
          <MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
          { "#316DB5" }' />
        </MapsShapeColorMappings>
      </MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code{
public class UNCouncil
{
public string Country { get; set; }
public string Membership { get; set; }
};
public List<UNCouncil> CouncilMemberDetails = new List<UNCouncil>{
new UNCouncil { Country= "China", Membership= "Permanent"},
new UNCouncil { Country= "France",Membership= "Permanent" },
new UNCouncil { Country= "Russia",Membership= "Permanent"},
new UNCouncil { Country= "Kazakhstan",Membership= "Non-Permanent"},
new UNCouncil { Country= "Poland",Membership= "Non-Permanent"},
new UNCouncil { Country= "Sweden",Membership= "Non-Permanent"}
};
}
```



#### *Desaturation color mapping*

Desaturation color mapping applies the color to the shapes of the Maps similar to the range color mapping. The opacity will be applied in this color mapping based on the [MinOpacity](#) and [MaxOpacity](#) properties in the [MapsShapeColorMapping](#).

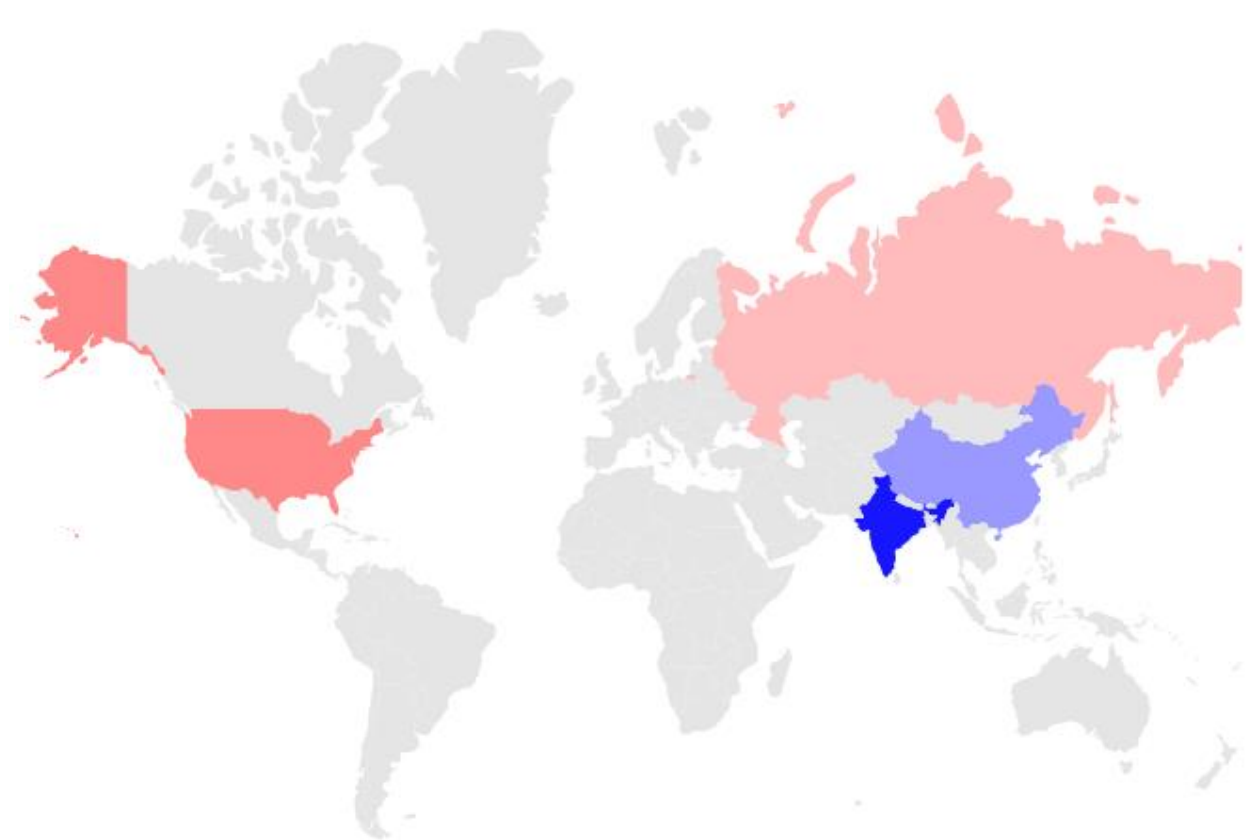
The following example shows how to apply desaturation color mapping to shapes with the data source **PopulationDetails** that is available in the [Range color mapping](#) section.

Bind the **PopulationDetails** data to the [DataSource](#) property of [MapsLayer](#) and set the [ColorValuePath](#) property of [MapsShapeSettings](#) as **Density**. The range values can be set using the [StartRange](#) and [EndRange](#) properties in the [MapsShapeColorMapping](#).

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
DataSource="populationDetails" ShapeDataPath="Name" ShapePropertyPath='new
string[] {"name"}' TValue="PopulationDetail">
      <MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Density">
        <MapsShapeColorMappings>
          <MapsShapeColorMapping StartRange="100" EndRange="400" Color='new string[]
{"blue"}' MinOpacity="0.3" MaxOpacity="1" />
        </MapsShapeColorMappings>
      </MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
```

```
</SfMaps>
```



### Multiple colors for a single shape

Multiple colors can be added to the color mapping which can be used as gradient effect to a specific shape based on the ranges in the data source. By using the [Color](#) property of [MapsShapeColorMapping](#), any number of colors can be set to the shapes as a gradient.

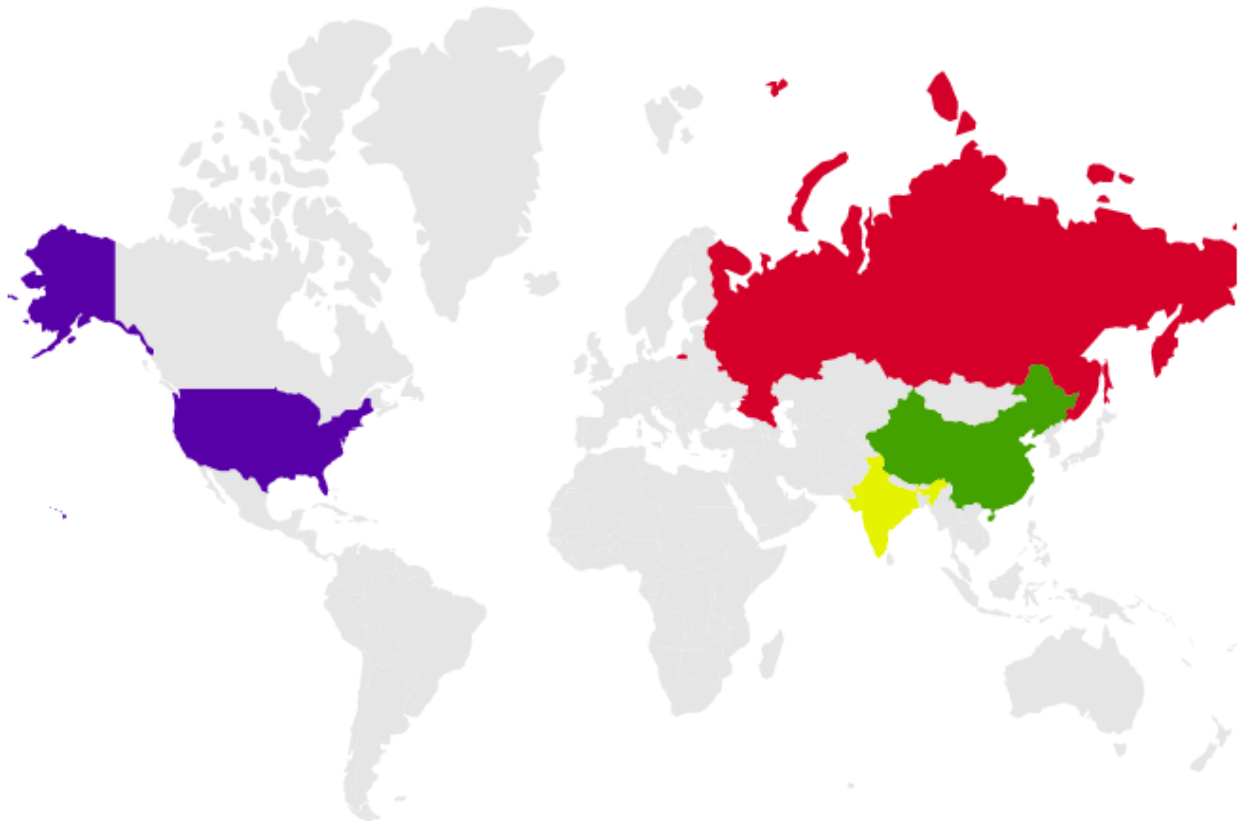
The following example demonstrates how to use multiple colors in color mapping with the data source **PopulationDetails** that is available in the [Range color mapping](#) section.

Bind the **PopulationDetails** data to the [DataSource](#) property of [MapsLayer](#) and set the [ColorValuePath](#) property of [MapsShapeSettings](#) as **Density**. The range values can be set using the [StartRange](#) and [EndRange](#) properties in the [MapsShapeColorMapping](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      DataSource="populationDetails" ShapeDataPath="Name" ShapePropertyPath='new
      string[] { "name"}' TValue="PopulationDetail">
      <MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Density">
        <MapsShapeColorMapping>
          <MapsShapeColorMapping StartRange="0.00001" EndRange="50" Color='new
          string[] { "red", "blue"}' />
        </MapsShapeColorMapping>
      </MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```

```
<MapsShapeColorMapping StartRange="50" EndRange="400" Color='new string[] {
"green", "yellow"}' />
</MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
```



#### Color for items excluded from color mapping

Color mapping can be applied to the shapes in the Maps which does not match color mapping criteria such as range or equal values using the [Color](#) property of [MapsShapeColorMapping](#).

---

The following example shows how to set the color for items excluded from the color mapping with the data source **PopulationDetails** that is available in the [Range color mapping](#) section.

---

In the following example, color mapping is added for the ranges from 0 to 300. If there are any records in the data source that are outside of this range, the color mapping will not be applied. To apply the color for these excluded items, set the [Color](#) property alone in the [MapsShapeColorMapping](#).

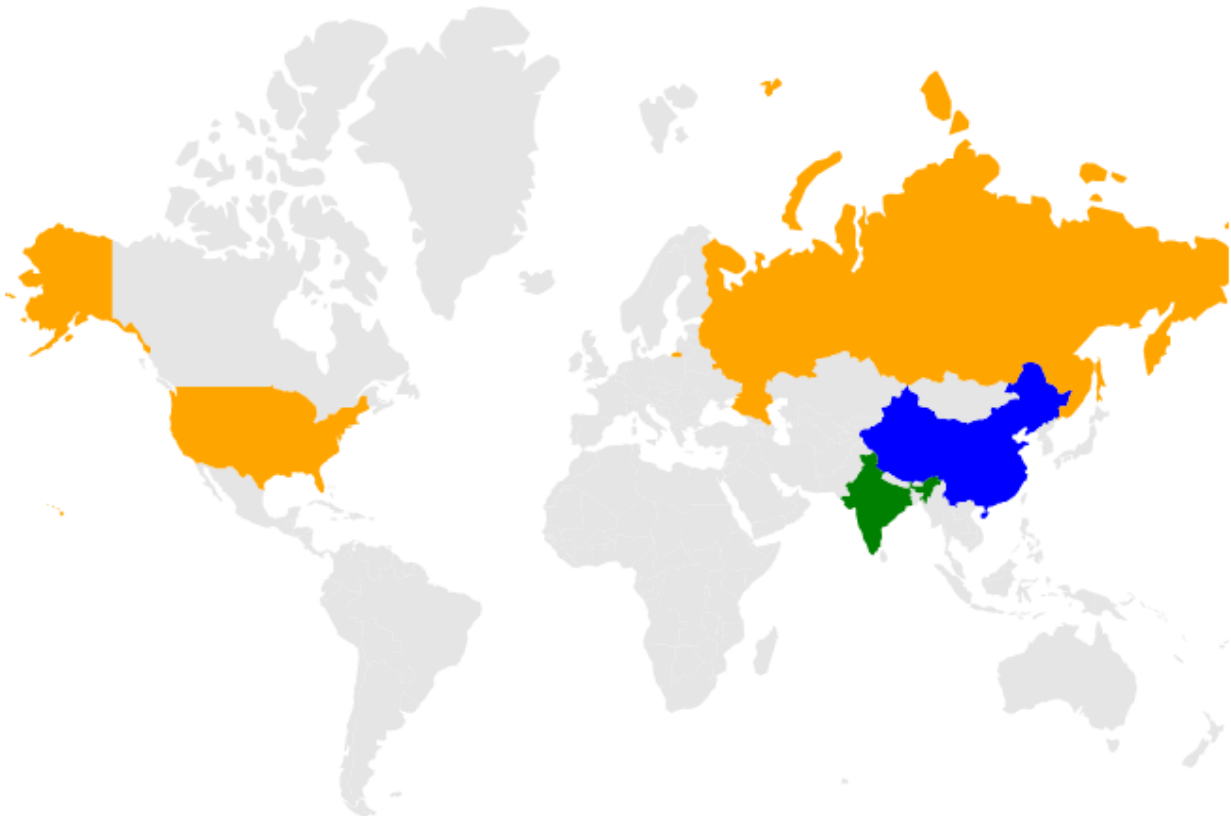
#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
DataSource="populationDetails" ShapeDataPath="Name" ShapePropertyPath='new
string[] {"name"}' TValue="PopulationDetail">
```

```

<MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Density">
  <MapsShapeColorMappings>
    <MapsShapeColorMapping StartRange="0.00001" EndRange="100" Color='new
string[] {"orange"}' />
    <MapsShapeColorMapping StartRange="100" EndRange="300" Color='new string[]
{"blue"}' />
    @* To apply color for excluded items *@
    <MapsShapeColorMapping Color='new string[] {"green"}' />
  </MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>

```



### Color mapping for bubbles

The color mapping types such as range color mapping, equal color mapping and desaturation color mapping are applicable for bubbles in the Maps. To add color mapping for bubbles of the Maps, bind the data source to the [DataSource](#) property of [MapsBubble](#) and set the field name which contains the color value in the data source to the [ColorValuePath](#) property. Multiple colors for a single set of bubbles and color for excluded items from [MapsBubbleColorMapping](#) are also applicable for bubbles.

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps Height="600" Width="400">
  <MapsLayers>

```

```
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapeDataPath="Name" ShapePropertyPath='new string[] {"name"}'
TValue="BubbleData">
<MapsShapeSettings Fill="#E5E5E5"/>
<MapsBubbleSettings>
<MapsBubble Visible="true" ValuePath="Population"
ColorValuePath="Population" MinRadius=5 DataSource="BubbleColorMapping"
TValue="BubbleData">
<MapsBubbleColorMappings>
<MapsBubbleColorMapping Value="38332521" Color='new string[] {"#D84444"}' />
<MapsBubbleColorMapping Value="19651127" Color='new string[] {"#316DB5"}' />
<MapsBubbleColorMapping Value="3090416" Color='new string[] {"blue"}' />
</MapsBubbleColorMappings>
</MapsBubble>
</MapsBubbleSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class BubbleData
{
public string Name { get; set; }
public string Population { get; set; }
};
public List<BubbleData> BubbleColorMapping = new List<BubbleData>{
new BubbleData { Name= "India", Population= "38332521"},
new BubbleData { Name= "Russia", Population= "19651127" },
new BubbleData { Name= "Pakistan", Population= "3090416"},
};
}
```



## Data labels in Blazor Maps Component

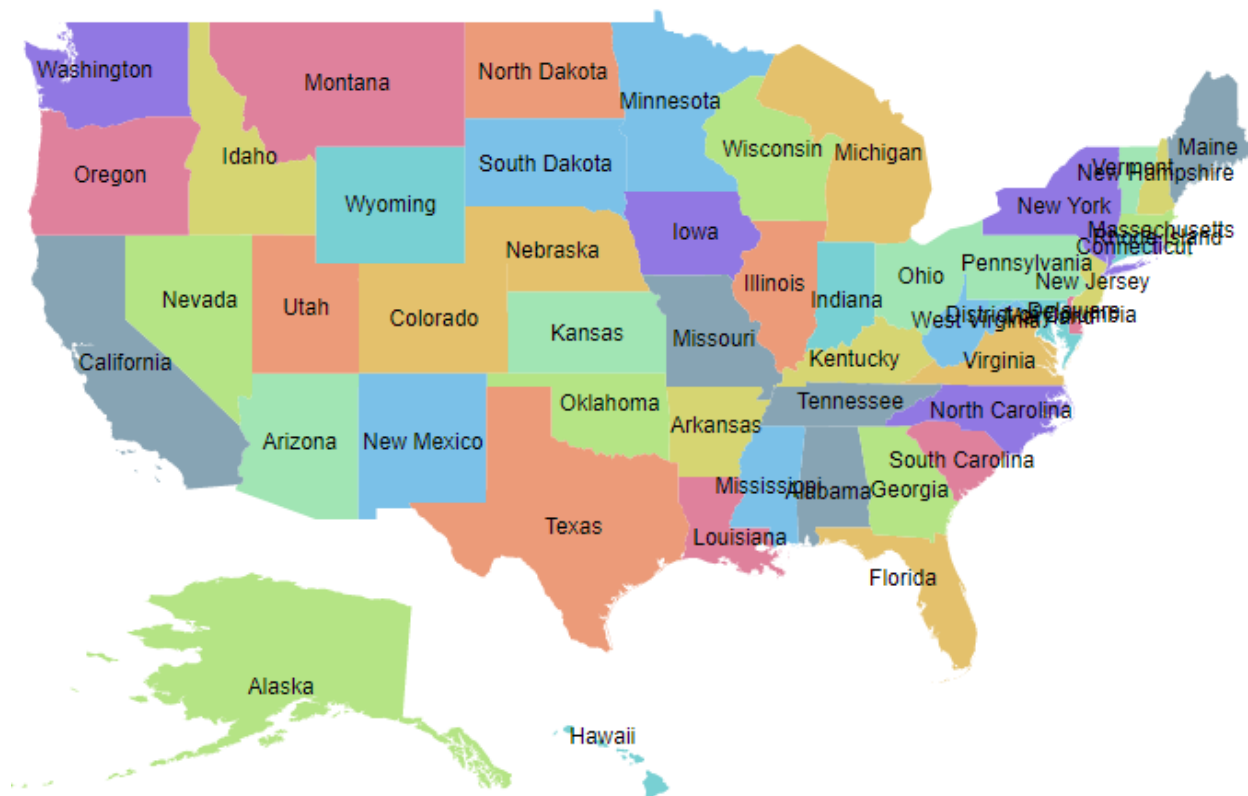
Data labels provide information to users about the shapes of the Maps component. It can be enabled by setting the [Visible](#) property of the [MapsDataLabelSettings](#) to **true**.

### Adding data labels

To display data labels in the Maps, the [LabelPath](#) property of [MapsDataLabelSettings](#) must be used. The value of the [LabelPath](#) property can be taken from the field name in the shape data or data source. In the following example, the value of the [LabelPath](#) property is the field name in the shape data of the Maps layer.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      @* To add data labels *@
      <MapsDataLabelSettings Visible="true"
        LabelPath="name"></MapsDataLabelSettings>
      <MapsShapeSettings Autofill="true"></MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```



In the following example, the value of [LabelPath](#) property is set from the field name in the data source of the layer settings.

## ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps ID="Maps">
<MapsLayers>
<MapLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
TVValue="PopulationDetail"
DataSource="PopulationDetailss" ShapeDataPath="@ShapeDataPath"
ShapePropertyPath="@ShapePropertyPath">
@* To add data labels *@
<MapsDataLabelSettings Visible="true"
LabelPath="Continent"></MapsDataLabelSettings>
<MapsShapeSettings Autofill="true"></MapsShapeSettings>
</MapLayer>
</MapsLayers>
</SfMaps>

@code{
public class PopulationDetail
{
public string Code { get; set; }
public double Value { get; set; }
public string Name { get; set; }
public double Population { get; set; }
public double Density { get; set; }
public string Color { get; set; }
public string Continent { get; set; }
}

```



```

};
public List<PopulationDetail> PopulationDetailss = new
List<PopulationDetail> {
    new PopulationDetail {
        Code = "AF", Value= 53, Name= "Afghanistan", Population= 29863010, Density=
        119, Color = "Red", Continent = "Asia"
    },
    new PopulationDetail {
        Code= "AL", Value= 117, Name= "Albania", Population= 3195000, Density= 111,
        Color = "Blue", Continent = "Europe"
    },
    new PopulationDetail {
        Code= "DZ", Value= 15, Name= "Algeria", Population= 34895000, Density= 15,
        Color = "Green", Continent = "Africa"
    }
};
public string[] ShapePropertyPath = { "name" };
public string ShapeDataPath = "Name";
}

```



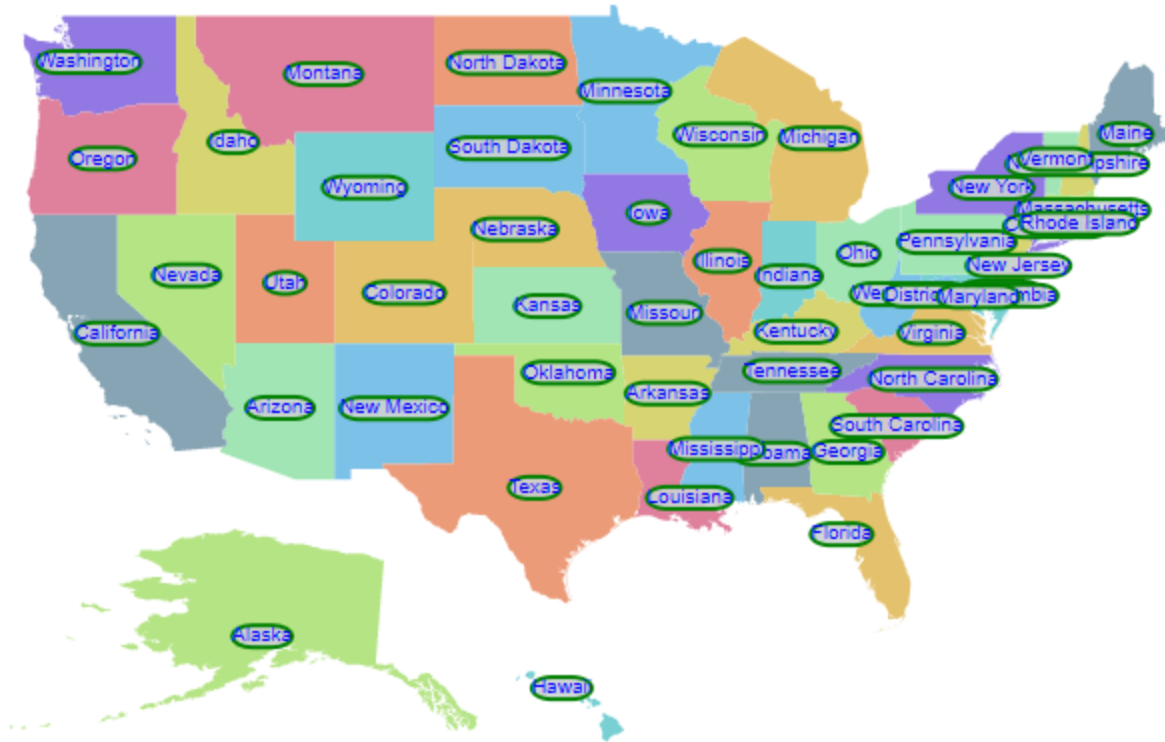
### Customization

The following properties and classes are available in the [MapsDataLabelSettings](#) to customize the data label of the Maps component.

- [MapsLayerDataLabelBorder](#) - To customize the color and width for the border of the data labels in Maps.
- [Fill](#) - To apply the color of the data labels in Maps.
- [Opacity](#) - To customize the transparency of the data labels in Maps.
- [MapsLayerDataLabelTextStyle](#) - To customize the text style of the data labels in Maps.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      @* To add data labels *@
      <MapsDataLabelSettings Visible="true" LabelPath="name" Fill="red"
        Opacity="0.9">
        <MapsLayerDataLabelBorder Color="green"
          Width="2"></MapsLayerDataLabelBorder>
        <MapsLayerDataLabelTextStyle Color="blue" Size="12px" FontStyle="Sans-serif"
          FontWeight="normal">
        </MapsLayerDataLabelTextStyle>
      </MapsDataLabelSettings>
    <MapsShapeSettings Autofill="true"></MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```



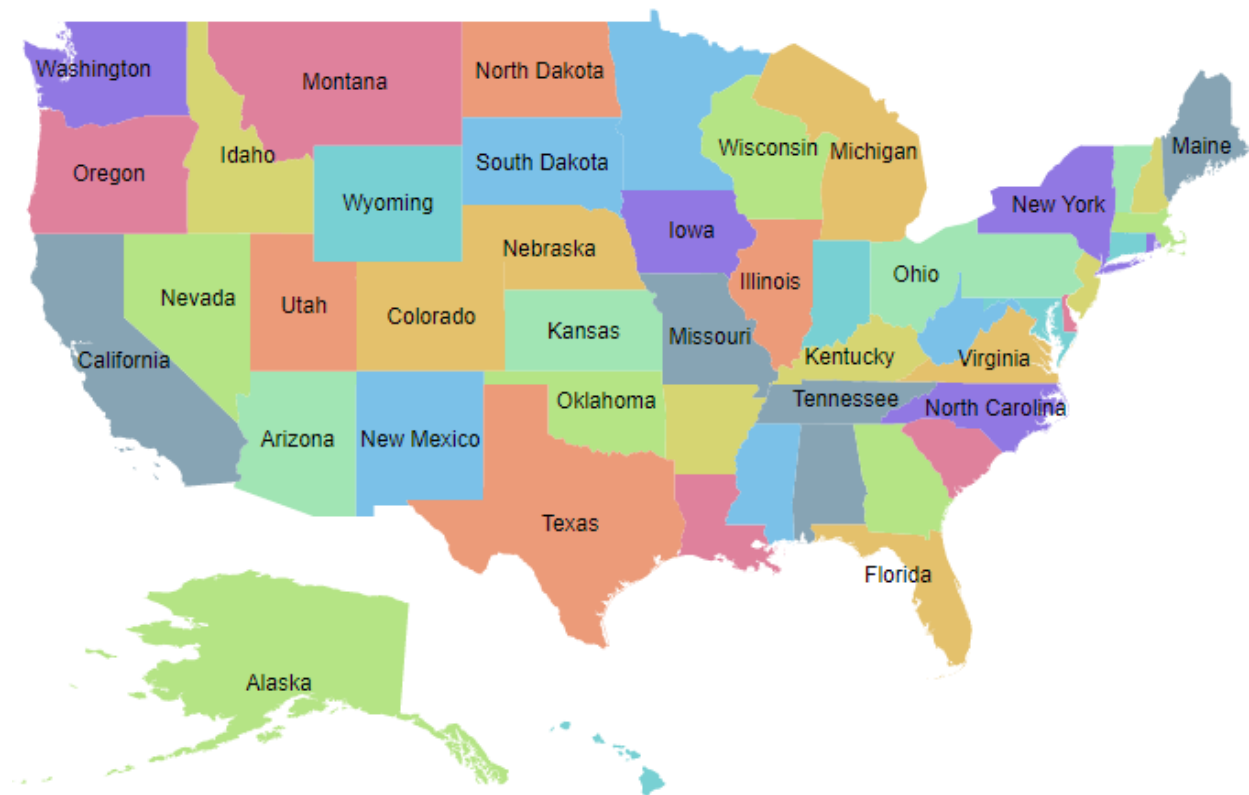
### Smart labels

The Maps component provides an option to handle the labels when they intersect with the corresponding shape borders using the [SmartLabelMode](#) property. The following options are available in the [SmartLabelMode](#) property.

- None
- Hide
- Trim

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      @* To hide intersect labels with shape border *@
      <MapsDataLabelSettings Visible="true" LabelPath="name"
        SmartLabelMode="SmartLabelMode.Hide">
      </MapsDataLabelSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```



### Intersect action

The Maps component provides an option to handle the labels when a label intersects with another label using the [IntersectionAction](#) property. The following options are available in the [IntersectionAction](#) property.

- None
- Hide
- Trim

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapLayers>
    <MapLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      @* To trim intersect labels *@
      <MapDataLabelSettings Visible="true" LabelPath="name"
        IntersectionAction="IntersectionAction.Trim">
      </MapDataLabelSettings>
      <MapShapeSettings Autofill="true"></MapShapeSettings>
    </MapLayer>
  </MapLayers>
</SfMaps>
```



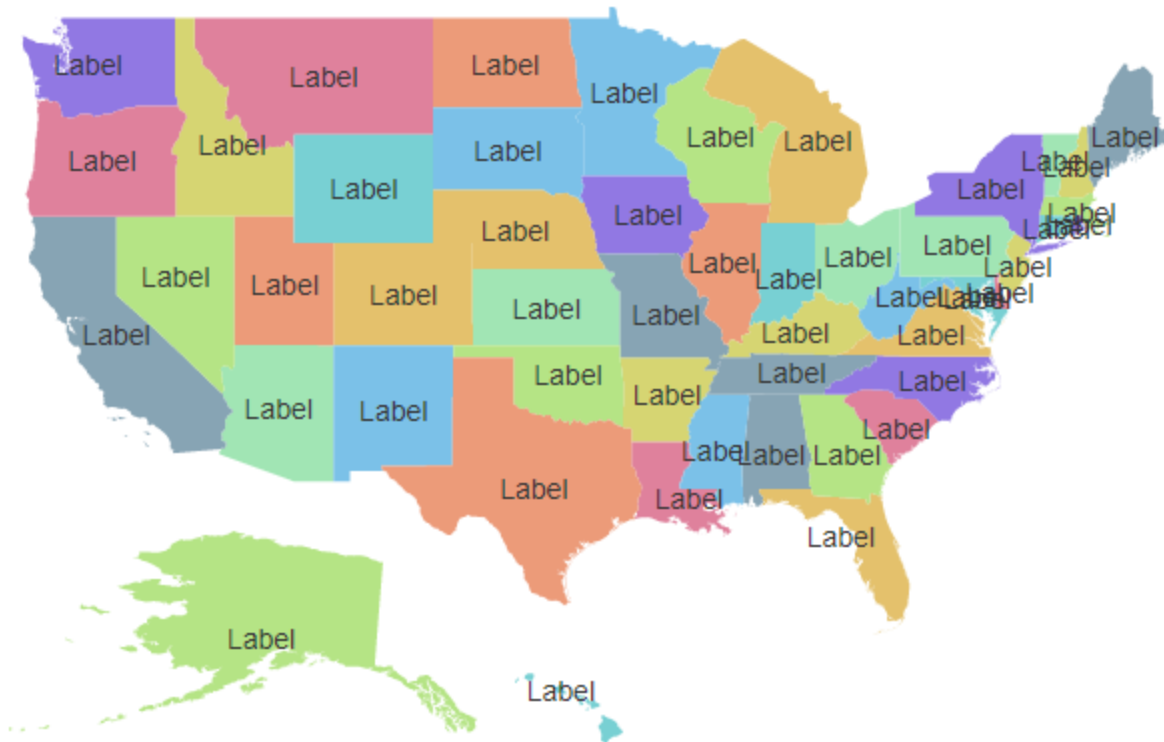
#### Adding data label as a template

The data label can be added as a template in the Maps component. The [LabelTemplate](#) property of [MapsDataLabelSettings](#) is used to set the data label as a template. Any text or HTML element can be added as the template in data labels.

The customization properties of data label, [SmartLabelMode](#) and [IntersectionAction](#) properties are not applicable to [LabelTemplate](#) property. The styles can be applied to the label template using the CSS styles of the template element.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      @* To trim intersect labels *@
      <MapsDataLabelSettings Visible="true">
        <LabelTemplate>
          @{ <p>Label</p> }
        </LabelTemplate>
      </MapsDataLabelSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```



### Markers in Blazor Maps Component

Markers are notes that are used to leave a message on the Maps. It indicates or marks a specific location with desired symbols on the Maps. It can be enabled by setting the [Visible](#) property of the [MapsMarker](#) to **true**.

#### Adding marker

To add the markers, the [DataSource](#) property of the [MapsMarker](#) has a list of objects that contains the data for markers. Using this property, any number of markers can be added to the layers of the Maps. By default, it displays the markers based on the specified latitude and longitude in the given data source. Each data source object should contain the following list of properties.

- Latitude - Specifies the position of the marker in latitude co-ordinate.
- Longitude - Specifies the position of the marker in longitude co-ordinate.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" DataSource="California" Height="25" Width="15"
TValue="City"></MapsMarker>
```

```
<MapsMarker Visible="true" DataSource="NewYork" Height="25" Width="15"  
TValue="City"></MapsMarker>  
<MapsMarker Visible="true" DataSource="Iowa" Height="25" Width="15"  
TValue="City"></MapsMarker>  
</MapsMarkerSettings>  
<MapsShapeSettings Fill="lightgray"></MapsShapeSettings>  
</MapsLayer>  
</MapsLayers>  
</SfMaps>  
@code {  
    public class City  
    {  
        public double Latitude { get; set; }  
        public double Longitude { get; set; }  
    };  
    public List<City> California = new List<City> {  
        new City {Latitude=35.145083,Longitude=-117.960260}  
    };  
    public List<City> NewYork = new List<City> {  
        new City { Latitude=40.724546, Longitude=-73.850344 }  
    };  
    public List<City> Iowa = new List<City> {  
        new City {Latitude= 41.657782, Longitude=-91.533857}  
    };  
}
```



### Adding marker template

The marker can be added as a template in the Maps component. The [MarkerTemplate](#) property of the [MapsMarker](#) is used to set HTML element as a template for the marker.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps Height="600" Width="700">
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" DataSource="Europe" TValue="City">
          <MarkerTemplate>
            @{
              <div id="marker4" class="markerTemplate">Europe</div>
            }
          </MarkerTemplate>
        </MapsMarker>
        <MapsMarker Visible="true" DataSource="NorthAmerica" TValue="City">
          <MarkerTemplate>
            @{
              <div id="marker5" class="markerTemplate" style="width:50px">North
              America</div>
            }
          </MarkerTemplate>
        </MapsMarker>
        <MapsMarker Visible="true" DataSource="SouthAmerica" TValue="City">
          <MarkerTemplate>
            @{
              <div id="marker6" class="markerTemplate" style="width:50px">South
              America</div>
            }
          </MarkerTemplate>
        </MapsMarker>
      </MapsMarkerSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
  public class City
  {
    public double Latitude { get; set; }
    public double Longitude { get; set; }
  };
  public List<City> Europe = new List<City> {
    new City { Latitude=49.95121990866204, Longitude=18.468749999999998 }
  };
  public List<City> NorthAmerica = new List<City> {
    new City { Latitude= 59.88893689676585, Longitude= -109.3359375 }
  };
  public List<City> SouthAmerica = new List<City> {
    new City { Latitude= -6.64607562172573, Longitude=-55.546874999999999 }
  };
}
```





### Customization

The following properties and class are available in [MapsMarker](#) to customize the markers of the Maps component.

- [MapsMarkerBorder](#) - To customize the color and width of the border for the markers in Maps.
- [Fill](#) - To apply the color for markers in Maps.
- [DashArray](#) - To define the pattern of dashes and gaps that is applied to the outline of the markers in Maps.
- [Height](#) - To customize the height of the markers in Maps.
- [Width](#) - To customize the width of the markers in Maps.
- [Opacity](#) - To customize the transparency of the markers in Maps.
- [AnimationDelay](#) - To change the time delay in the transition for markers.
- [AnimationDuration](#) - To change the time duration of animation for markers.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new { dataOptions =
      "https://cdn.syncfusion.com/maps/map-data/world-map.json" }'
      TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" DataSource="MarkerData" Height="25" Width="15"
          Fill="red" DashArray="1" Opacity="0.9"
          Shape="Syncfusion.Blazor.Maps.MarkerType.Balloon" TValue="City">
          <MapsMarkerBorder Color="green" Width="2"></MapsMarkerBorder>
        </MapsMarker>
      </MapsMarkerSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```

```
</MapsMarkerSettings>
<MapsShapeSettings Fill="lightgray"></MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
};
public List<City> MarkerData = new List<City> {
new City { Latitude=35.145083, Longitude=-117.960260 },
new City { Latitude=40.724546, Longitude=-73.850344 },
new City { Latitude= 41.657782, Longitude=-91.533857 }
};
}
```



### Marker shapes

The Maps component supports the following marker shapes. To set the shape of the marker, the [Shape](#) property in [MapsMarker](#) is used.

- Balloon
- Circle
- Cross
- Diamond

- Image
- Rectangle
- Start
- Triangle
- VerticalLine
- HorizontalLine

#### *Rendering marker shape as image*

To render a marker as an image in Maps, set the [Shape](#) property of [MapsMarker](#) as **Image** and specify the path of the image to [ImageUrl](#) property. There is another way to render a marker as an image using the [ImageUrlValuePath](#) property of the [MapsMarker](#). Bind the field name that contains the path of the image in the data source to the [ImageUrlValuePath](#) property.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new { dataOptions =
      "https://cdn.syncfusion.com/maps/map-data/world-map.json" }'
      TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" DataSource="MarkerData" Height="25" Width="15"
          TValue="City"
          Shape="Syncfusion.Blazor.Maps.MarkerType.Image"
          ImageUrl="src/maps/images/ballon.png">
        </MapsMarker>
      </MapsMarkerSettings>
      <MapsShapeSettings Fill="lightgray"></MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
  public class City
  {
    public double Latitude { get; set; }
    public double Longitude { get; set; }
  };
  public List<City> MarkerData = new List<City> {
    new City { Latitude=35.145083,Longitude=-117.960260 },
    new City { Latitude=40.724546, Longitude=-73.850344 },
    new City { Latitude= 41.657782, Longitude=-91.533857 }
  };
}
```



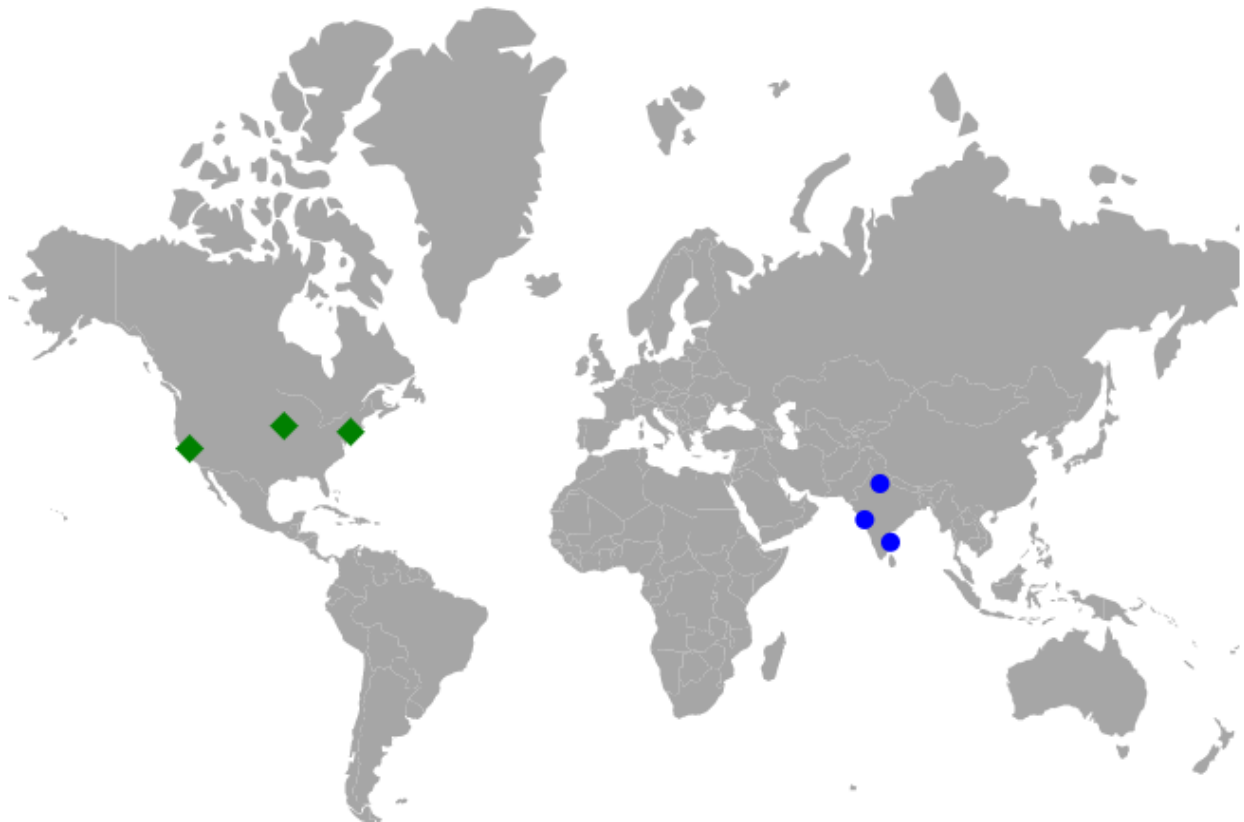
### Multiple marker groups

Multiple groups of markers can be added in the Maps by adding multiple [MapsMarker](#) in the [MapsMarkerSettings](#) and customization for the markers can be done with the [MapsMarker](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" DataSource="CitiesInUS"
Shape="MarkerType.Diamond" Height="15" Fill="green" Width="15"
TValue="City">
          <MapsMarkerTooltipSettings ValuePath="Name"
Visible="true"></MapsMarkerTooltipSettings>
        </MapsMarker>
      </MapsMarkerSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
public class City
```

```
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
};
private List<City> CitiesInUS = new List<City> {
new City { Latitude = 37.0000, Longitude = -120.0000, Name = "California" },
new City { Latitude= 40.7127, Longitude = -74.0059, Name = "New York" },
new City { Latitude = 42, Longitude = -93, Name = "Iowa" }
};
private List<City> CitiesInIndia = new List<City> {
new City { Latitude = 19.228825, Longitude = 72.854118, Name= "Mumbai" },
new City { Latitude = 28.610001, Longitude = 77.230003, Name= "Delhi" },
new City { Latitude = 13.067439, Longitude = 80.237617, Name= "Chennai" }
};
}
```



Customize marker shapes from data source

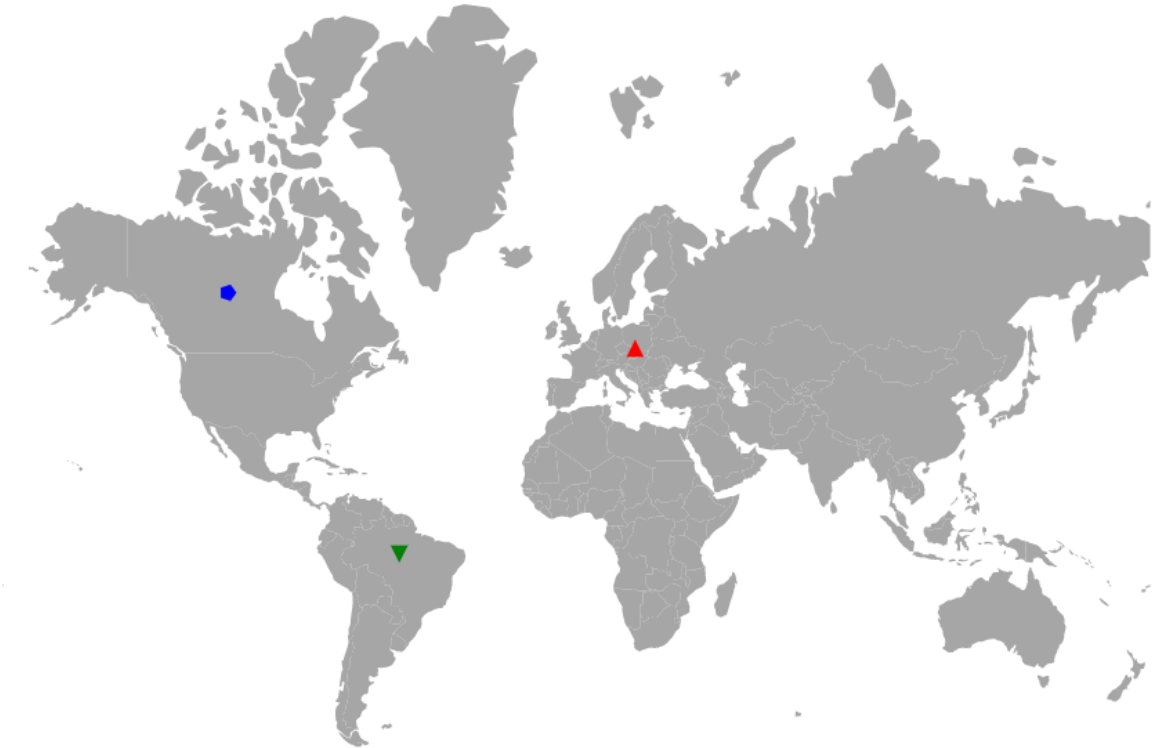
*Bind different colors and shapes to the marker from data source*

Using the [ShapeValuePath](#) and [ColorValuePath](#) properties, the color and shape of the marker can be applied from the given data source. Bind the data source to the [DataSource](#) property of the [MapsMarker](#) and set the field names that contains the shape and color values in the data source to the [ShapeValuePath](#) and [ColorValuePath](#) properties.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
```

```
<SfMaps>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
<MapsMarkerSettings>
<MapsMarker Visible='true' DataSource='MarkerDataSource'
ShapeValuePath="shape" ColorValuePath="color" TValue="MapMarkerDataSource">
</MapsMarker>
</MapsMarkerSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class MapMarkerDataSource
{
public double latitude { get; set; }
public double longitude { get; set; }
public string name { get; set; }
public string color { get; set; }
public string shape { get; set; }
};
public List<MapMarkerDataSource> MarkerDataSource = new
List<MapMarkerDataSource> {
new MapMarkerDataSource{ latitude= 49.95121990866204, longitude=
18.468749999999998, name= "Europe", color="red", shape="Triangle" },
new MapMarkerDataSource{ latitude= 59.88893689676585, longitude= -
109.3359375, name= "North America", color="blue", shape="Pentagon" },
new MapMarkerDataSource{ latitude= -6.64607562172573, longitude= -
55.546874999999999, name= "South America", color="green",
shape="InvertedTriangle" }
};
}
```



#### Setting value path from the data source

The latitude and longitude values are used to determine the location of each marker in the Maps. The [LatitudeValuePath](#) and [LongitudeValuePath](#) properties are used to specify the value path that presents in the data source of the marker. In the following example, the field name from the data source is set to the [LatitudeValuePath](#) and [LongitudeValuePath](#) properties.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new { dataOptions =
      "https://cdn.syncfusion.com/maps/map-data/world-map.json" }'
      TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" DataSource="MarkerData"
          LatitudeValuePath="Latitude" LongitudeValuePath="Longitude" TValue="City">
        </MapsMarker>
      </MapsMarkerSettings>
      <MapsShapeSettings Fill="lightgray"></MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
  public class City
  {
    public double Latitude { get; set; }
    public double Longitude { get; set; }
  };
  public List<City> MarkerData = new List<City> {
    new City { Latitude=35.145083, Longitude=-117.960260 },
```

```
new City { Latitude=40.724546, Longitude=-73.850344 },  
new City { Latitude= 41.657782, Longitude=-91.533857 }  
};  
}
```



### Marker zooming

The Maps can be initially scaled to the center value based on the marker distance. This can be achieved by setting the [ShouldZoomInitially](#) property in [MapsZoomSettings](#) as **true**.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps  
<SfMaps>  
<MapsLayers>  
<MapsLayer ShapeData='new {dataOptions=  
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">  
<MapsMarkerSettings>  
<MapsMarker Visible='true' DataSource='MarkerDataSource'  
TValue="MapMarkerDataSource">  
</MapsMarker>  
</MapsMarkerSettings>  
</MapsLayer>  
</MapsLayers>  
<MapsZoomSettings Enable='true'  
HorizontalAlignment="Syncfusion.Blazor.Maps.Alignment.Near"  
ShouldZoomInitially="true"></MapsZoomSettings>  
</SfMaps>  
@code {
```



```
public class MapMarkerDataSource
{
    public double latitude { get; set; }
    public double longitude { get; set; }
    public string name { get; set; }
};

public List<MapMarkerDataSource> MarkerDataSource = new
List<MapMarkerDataSource> {
    new MapMarkerDataSource{ latitude= 49.95121990866204, longitude=
18.468749999999998, name= "Europe" },
    new MapMarkerDataSource{ latitude= 59.88893689676585, longitude= -
109.3359375, name= "North America" },
    new MapMarkerDataSource{ latitude= -6.64607562172573, longitude= -
55.546874999999999, name= "South America" }
};
}
```



### Marker clustering

Maps provide support to cluster the markers when they overlap each other. The number on a cluster indicates how many overlapped markers it contains. If zooming is performed on any of the cluster locations in Maps, the number on the cluster will decrease, and the individual markers will be seen on

the map. When zooming out, the overlapping marker will increase. So that it can cluster again and increase the count over the cluster.

To enable clustering in markers, set the [AllowClustering](#) property of [MapsMarkerClusterSettings](#) as **true** and customization of clustering can be done with [MapsMarkerClusterSettings](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings Enable="true"></MapsZoomSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" DataSource="LargestCities" Height="25" Width="15"
TValue="City">
        </MapsMarker>
      </MapsMarkerSettings>
      <MapsMarkerClusterSettings AllowClustering="true" Shape="MarkerType.Circle"
Fill="#008CFF" Height="25" Width="25">
      <MapsLayerMarkerClusterLabelStyle
Color="white"></MapsLayerMarkerClusterLabelStyle>
    </MapsMarkerClusterSettings>
    <MapsShapeSettings Fill="lightgray">
    </MapsShapeSettings>
  </MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
public double Area { get; set; }
};
private List<City> LargestCities = new List<City> {
new City { Latitude=40.6971494, Longitude= -74.2598747, Name="New York",
Area=8683 },
new City { Latitude=40.0024137, Longitude= -75.2581194, Name="Philadelphia",
Area=4661 },
new City { Latitude=42.3142647, Longitude= -71.11037, Name="Boston",
Area=4497 },
new City { Latitude=42.3526257, Longitude= -83.239291, Name="Detroit",
Area=3267 },
new City { Latitude=47.2510905, Longitude= -123.1255834, Name="Washington",
Area=2996 },
new City { Latitude=25.7823907, Longitude= -80.2994995, Name="Miami",
Area=2891 },
new City { Latitude=19.3892246, Longitude= -70.1305136, Name="San Juan",
Area=2309 }
};
}
```



#### Customization of marker cluster

The following properties and classes are available to customize the marker clustering in the Maps component.

- [MapsLayerMarkerClusterBorder](#) - To customize the color and width of the border of cluster in Maps.
- [MapsLayerMarkerClusterConnectorLineSettings](#) - To customize the connector line in cluster separating the markers.
- [DashArray](#) - To customize the dash array for the marker cluster in Maps.
- [Fill](#) - Applies the color of the cluster in Maps.
- [Height](#) - To customize the height of the marker cluster in Maps.
- [ImageUrl](#) - To customize the URL path for the marker cluster when the cluster shape is set as image in Maps.
- [MapsLayerMarkerClusterLabelStyle](#) - To customize the text in marker cluster.
- [Offset](#) - To customize the offset position for the marker cluster in Maps.
- [Opacity](#) - To customize the opacity of the marker cluster.
- [Shape](#) - To customize the shape for the cluster of markers.
- [Width](#) - To customize the width of the marker cluster in Maps.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings Enable="true"></MapsZoomSettings>
  <MapsLayers>
```

```

<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
<MapsMarkerSettings>
<MapsMarker Visible="true" DataSource="LargestCities" Height="25" Width="15"
TValue="City">
</MapsMarker>
</MapsMarkerSettings>
<MapsMarkerClusterSettings AllowClustering="true" AllowClusterExpand="true"
Shape="MarkerType.Circle"
Fill="#008CFF" Height="25" Width="25" Offset="10" Opacity="0.9">
<MapsLayerMarkerClusterConnectorLineSettings Color="Orange" Opacity="0.8"
Width="2"></MapsLayerMarkerClusterConnectorLineSettings>
<MapsLayerMarkerClusterLabelStyle
Color="green"></MapsLayerMarkerClusterLabelStyle>
</MapsMarkerClusterSettings>
<MapsShapeSettings Fill="lightgray">
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
public double Area { get; set; }
};
private List<City> LargestCities = new List<City> {
new City { Latitude=40.6971494, Longitude= -74.2598747, Name="New York",
Area=8683 },
new City { Latitude=40.0024137, Longitude= -75.2581194, Name="Philadelphia",
Area=4661 },
new City { Latitude=42.3142647, Longitude= -71.11037, Name="Boston",
Area=4497 },
new City { Latitude=42.3526257, Longitude= -83.239291, Name="Detroit",
Area=3267 },
new City { Latitude=47.2510905, Longitude= -123.1255834, Name="Washington",
Area=2996 },
new City { Latitude=25.7823907, Longitude= -80.2994995, Name="Miami",
Area=2891 },
new City { Latitude=19.3892246, Longitude= -70.1305136, Name="San Juan",
Area=2309 }
};
}

```

![Blazor Maps Marker with Custom Cluster](./images/Marker/Cluster-customization.PNG)

### Expanding the marker cluster

The cluster is formed by grouping an identical and non-identical marker from the surrounding area. By clicking on the cluster and setting the [AllowClusterExpand](#) property in [MapsMarkerClusterSettings](#) as **true** to expand the identical markers. If zooming is performed in any of the locations of the cluster, the number on the cluster will decrease and the overlapping marker will be split into an individual marker on the map. When performing zoom out, it will increase the marker count and then cluster it again.

**ASPX-CS**

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
<MapsMarkerSettings>
<MapsMarker Visible="true" DataSource='MarkerDataSource'
TValue="MapMarkerDataSource"/>
</MapsMarkerSettings>
<MapsMarkerClusterSettings AllowClustering="true" AllowClusterExpand="true"
Shape="MarkerType.Circle" Height="40" Width="40">
<MapsLayerMarkerClusterLabelStyle
Color="white"></MapsLayerMarkerClusterLabelStyle>
</MapsMarkerClusterSettings>
</MapsLayer>
</MapsLayers>
<MapsZoomSettings Enable='true' MouseWheelZoom="true"></MapsZoomSettings>
</SfMaps>
@code {
public class MapMarkerDataSource
{
public double latitude { get; set; }
public double longitude { get; set; }
public string name { get; set; }
};
public List<MapMarkerDataSource> MarkerDataSource = new
List<MapMarkerDataSource> {
new MapMarkerDataSource{ latitude= 49.95121990866204, longitude=
18.468749999999998, name= "Europe" },
new MapMarkerDataSource{ latitude= 49.95121990866204, longitude=
18.468749999999998, name= "Europe" },
new MapMarkerDataSource{ latitude= 49.95121990866204, longitude=
18.468749999999998, name= "Europe" },
new MapMarkerDataSource{ latitude= 49.95121990866204, longitude=
18.468749999999998, name= "Europe" },
new MapMarkerDataSource{ latitude= 49.95121990866204, longitude=
18.468749999999998, name= "Europe" },
new MapMarkerDataSource{ latitude= 49.95121990866204, longitude=
18.468749999999998, name= "Europe" },
new MapMarkerDataSource{ latitude= 49.95121990866204, longitude=
18.468749999999998, name= "Europe" },
new MapMarkerDataSource{ latitude= 59.88893689676585, longitude= -
109.3359375, name= "North America" },
new MapMarkerDataSource{ latitude= -6.64607562172573, longitude= -
55.546874999999999, name= "South America" }
};
}

```



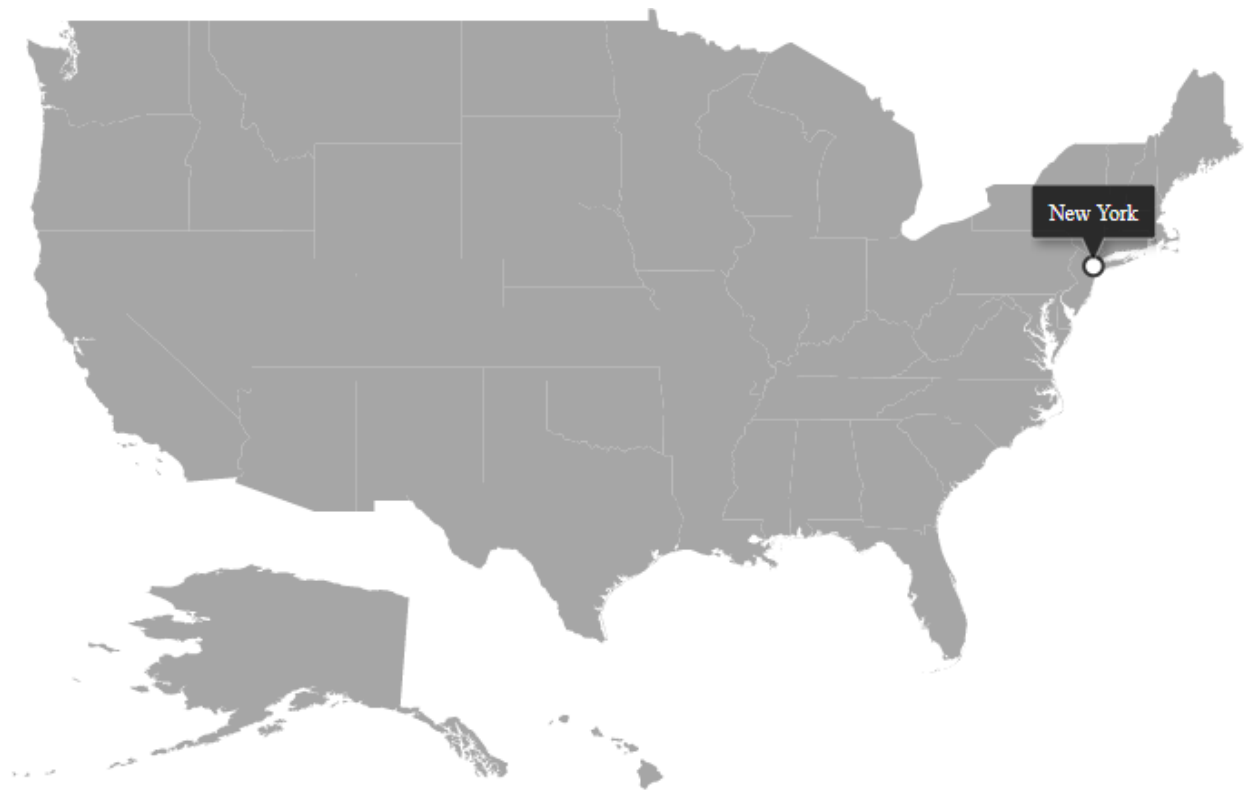
### Tooltip for marker

Tooltip is used to display more information about a marker on mouse over or touch end event. This can be enabled separately for marker by setting the [Visible](#) property of [MapsMarkerTooltipSettings](#) to **true**. The [ValuePath](#) property in the [MapsMarkerTooltipSettings](#) takes the field name that presents in data source and displays that value as tooltip text.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" Shape="MarkerType.Circle" Fill="white" Width="20"
          DataSource="HighestPopulation" TValue="City">
          <MapsMarkerBorder Width="2" Color="#333"></MapsMarkerBorder>
          <MapsMarkerTooltipSettings Visible="true"
            ValuePath="Name"></MapsMarkerTooltipSettings>
        </MapsMarker>
      </MapsMarkerSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
  public class City
  {
    public double Latitude { get; set; }
  }
}
```

```
public double Longitude { get; set; }  
public string Name { get; set; }  
};  
public List<City> HighestPopulation = new List<City> {  
    new City { Latitude = 40.7424509, Longitude = -74.0081468, Name = "New York"  
    }  
};  
}
```



See also

- [Add different types of markers](#)

### Bubble in Blazor Maps Component

Bubbles in the Maps control represent the underlying data values of the Maps. It can be scattered throughout the Maps shapes that contain values in the data source. Bubbles are enabled by setting the [Visible](#) property of [MapsBubble](#) to **true**. To add bubbles to the Maps, bind the data source to the [DataSource](#) property of the [MapsBubble](#) and set the field name, that contains the numerical data, in the data source to the [ValuePath](#) property.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps  
<SfMaps>  
<MapsLayers>  
<MapsLayer ShapeData='new {dataOptions  
    ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'>
```

```
DataSource="PopulationDetails" ShapeDataPath="Name" ShapePropertyPath='new
string[] { "name" }' TValue="Country">
@* To add bubbles based on population count *@
<MapsBubbleSettings>
<MapsBubble Visible="true" ValuePath="Population" ColorValuePath="Color"
DataSource="PopulationDetails" TValue="Country">
</MapsBubble>
</MapsBubbleSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class Country
{
public string Name { get; set; }
public double Population { get; set; }
public string Color { get; set; }
};
public List<Country> PopulationDetails = new List<Country> {
new Country
{
Name ="United States", Population = 325020000, Color = "#b5e485"
},
new Country
{
Name = "Russia", Population = 142905208, Color = "#7bc1e8"
},
new Country
{
Name="India", Population=1198003000, Color = "#df819c"
}
};
}
```





### Bubble shapes

The following types of shapes are available to render the bubbles in Maps.

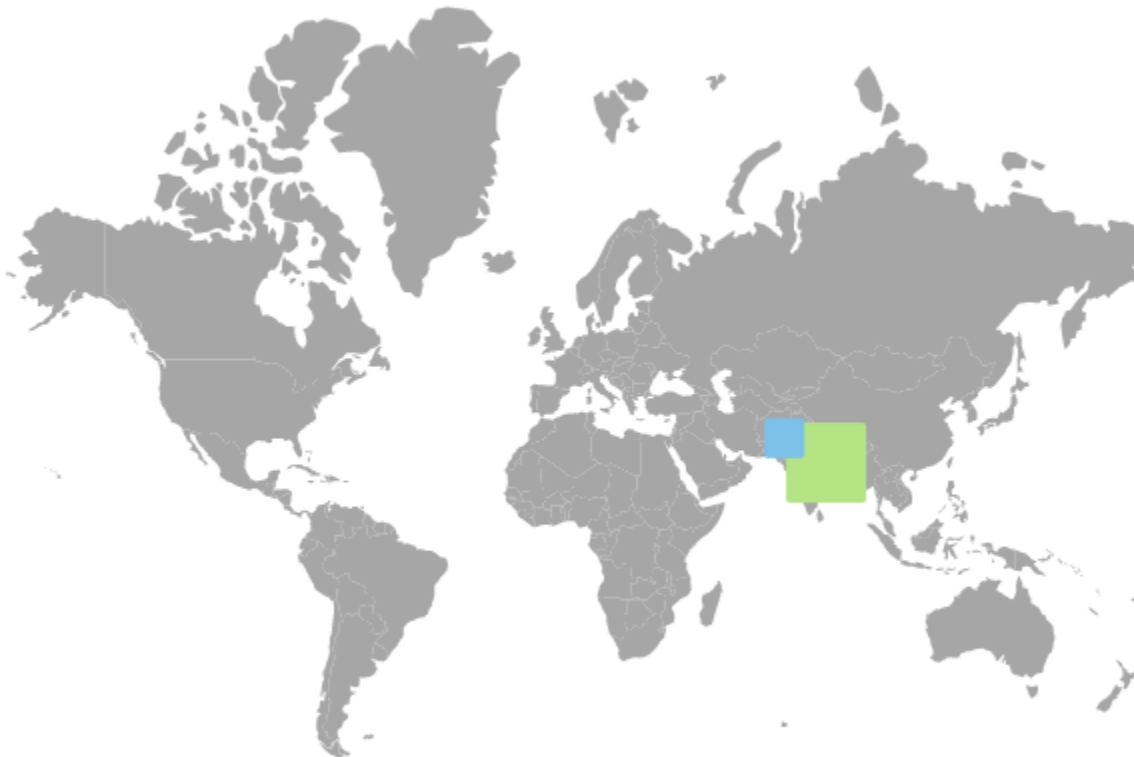
- Circle
- Square

By default, bubbles are rendered in the **Circle** type. To change the type of the bubble, set the [BubbleType](#) property of [MapsBubble](#) as **Square** to render the square shape bubbles.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      ShapeDataPath="Name" ShapePropertyPath='new string[] {"name"}'
      TValue="Country">
      @* To add bubbles based on population count *@
      <MapsBubbleSettings>
        <MapsBubble Visible="true" ValuePath="Population" ColorValuePath="Color"
          DataSource="PopulationDetails"
          BubbleType="Syncfusion.Blazor.Maps.BubbleType.Square" TValue="Country">
        </MapsBubble>
      </MapsBubbleSettings>
    </MapsLayer>
  </MapsLayers>
```

```
</SfMaps>
@code{
public class Country
{
public string Name { get; set; }
public double Population { get; set; }
public string Color { get; set; }
};
public List<Country> PopulationDetails = new List<Country> {
new Country
{
Name = "United States", Population = 325020000, Color = "#b5e485"
},
new Country
{
Name = "Russia", Population = 142905208, Color = "#7bc1e8"
},
new Country
{
Name = "India", Population=1198003000, Color = "#df819c"
}
};
}
```



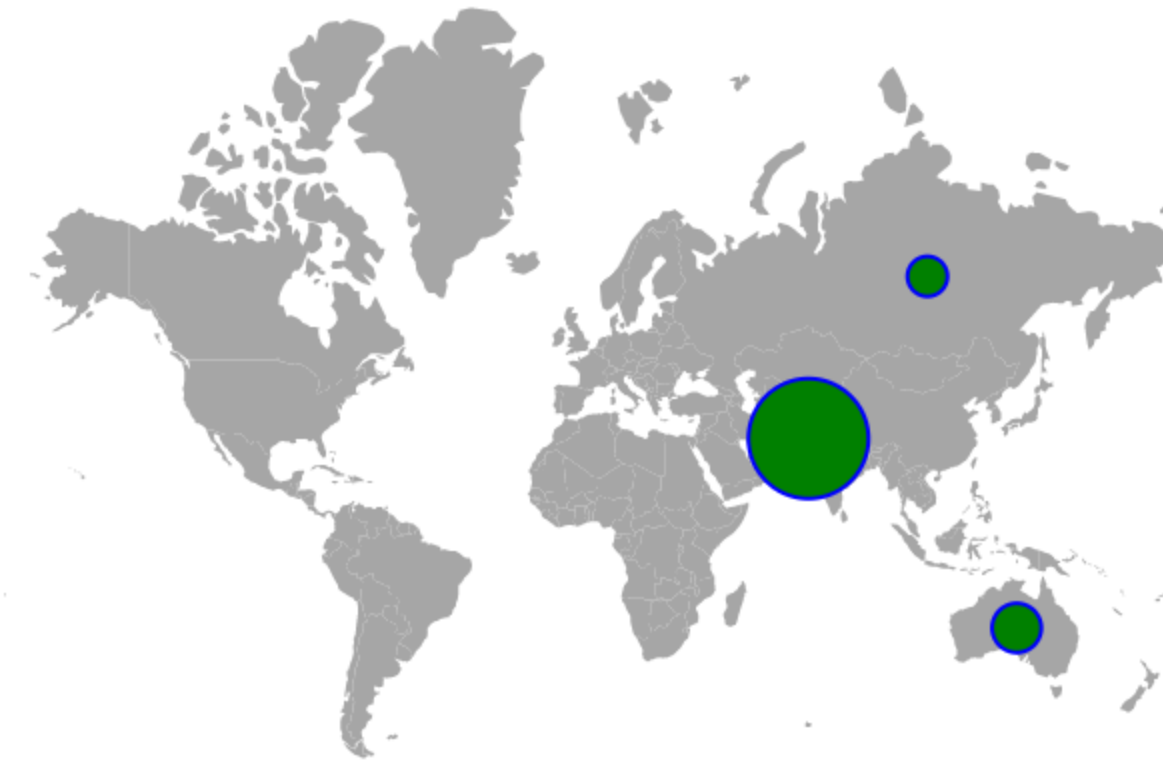
### Customization

The following properties and a class are available in [MapsBubble](#) to customize the bubbles of the Maps component.

- [MapsBubbleBorder](#) - To customize the color and width of the border of the bubbles in Maps.
- [Fill](#) - To apply the color for bubbles in Maps.
- [Opacity](#) - To apply opacity to the bubbles in Maps.
- [AnimationDelay](#) - To change the time delay in the transition for bubbles.
- [AnimationDuration](#) - To change the time duration of animation for bubbles.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapeDataPath="Name" ShapePropertyPath='new string[] {"name"}'
TValue="Country">
      @* To add bubbles based on population count *@
      <MapsBubbleSettings>
        <MapsBubble Visible="true" ValuePath="Population" Fill="green" MinRadius=5
MaxRadius=40 AnimationDelay=100
AnimationDuration=1000 Opacity=1 DataSource="PopulationDetails"
TValue="Country">
          <MapsBubbleBorder Color="blue" Width=2></MapsBubbleBorder>
        </MapsBubble>
      </MapsBubbleSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code{
public class Country
{
    public string Name { get; set; }
    public double Population { get; set; }
};
public List<Country> PopulationDetails = new List<Country> {
    new Country
    {
        Name = "United States", Population = 325020000
    },
    new Country
    {
        Name = "Russia", Population = 142905208
    },
    new Country
    {
        Name = "India", Population=1198003000
    }
};
}
```



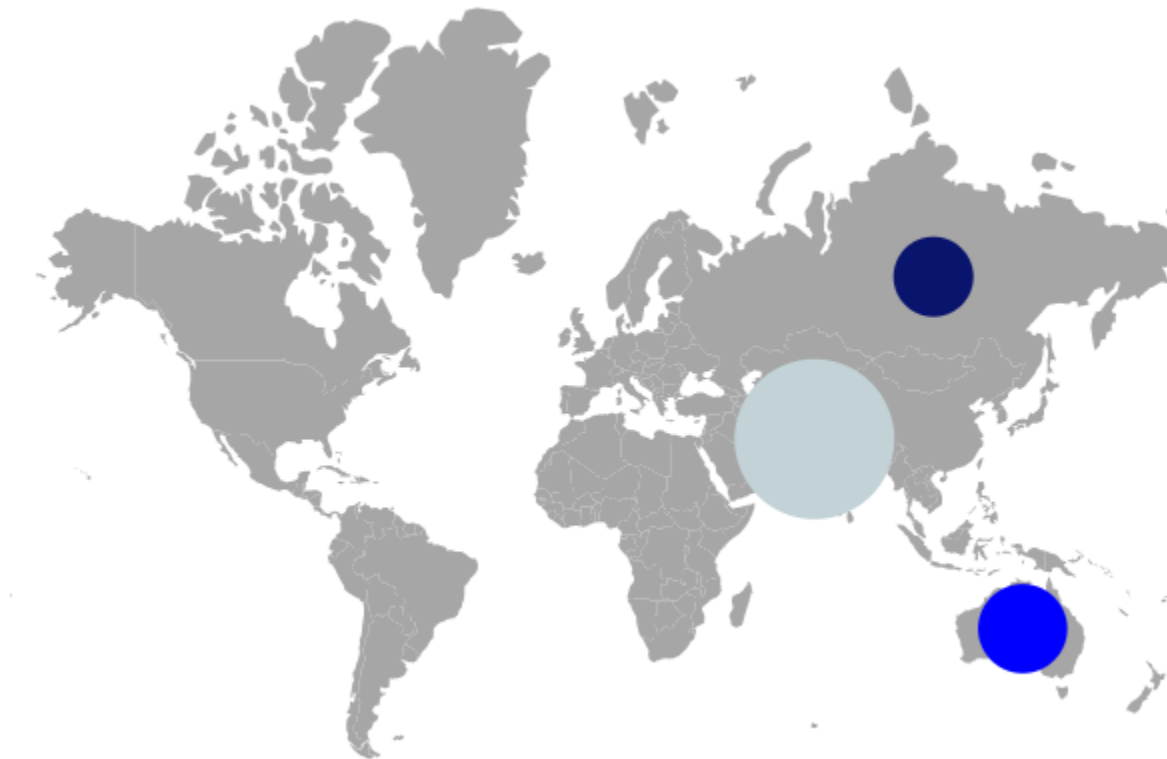
Setting colors to the bubbles from the data source

The color for each bubble in the Maps can be set using the [ColorValuePath](#) property of [MapsBubble](#). The value for the [ColorValuePath](#) property is the field name from the data source of the [MapsBubble](#) which contains the color values.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      ShapeDataPath="Name" ShapePropertyPath='new string[] { "name"}'
      TValue="Country">
      @* To add bubbles based on population count *@
      <MapsBubbleSettings>
        <MapsBubble Visible="true" ValuePath="Population" ColorValuePath="Color"
          MinRadius=20 MaxRadius=40
          DataSource="PopulationDetails" TValue="Country">
        </MapsBubble>
      </MapsBubbleSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code{
public class Country
```

```
{
public string Name { get; set; }
public double Population { get; set; }
public string Color { get; set; }
};
public List<Country> PopulationDetails = new List<Country> {
new Country
{
Name = "United States", Population = 325020000, Color = "#b5e485"
},
new Country
{
Name = "Russia", Population = 142905208, Color = "#7bc1e8"
},
new Country
{
Name = "India", Population = 1198003000, Color = "#df819c"
}
};
}
```

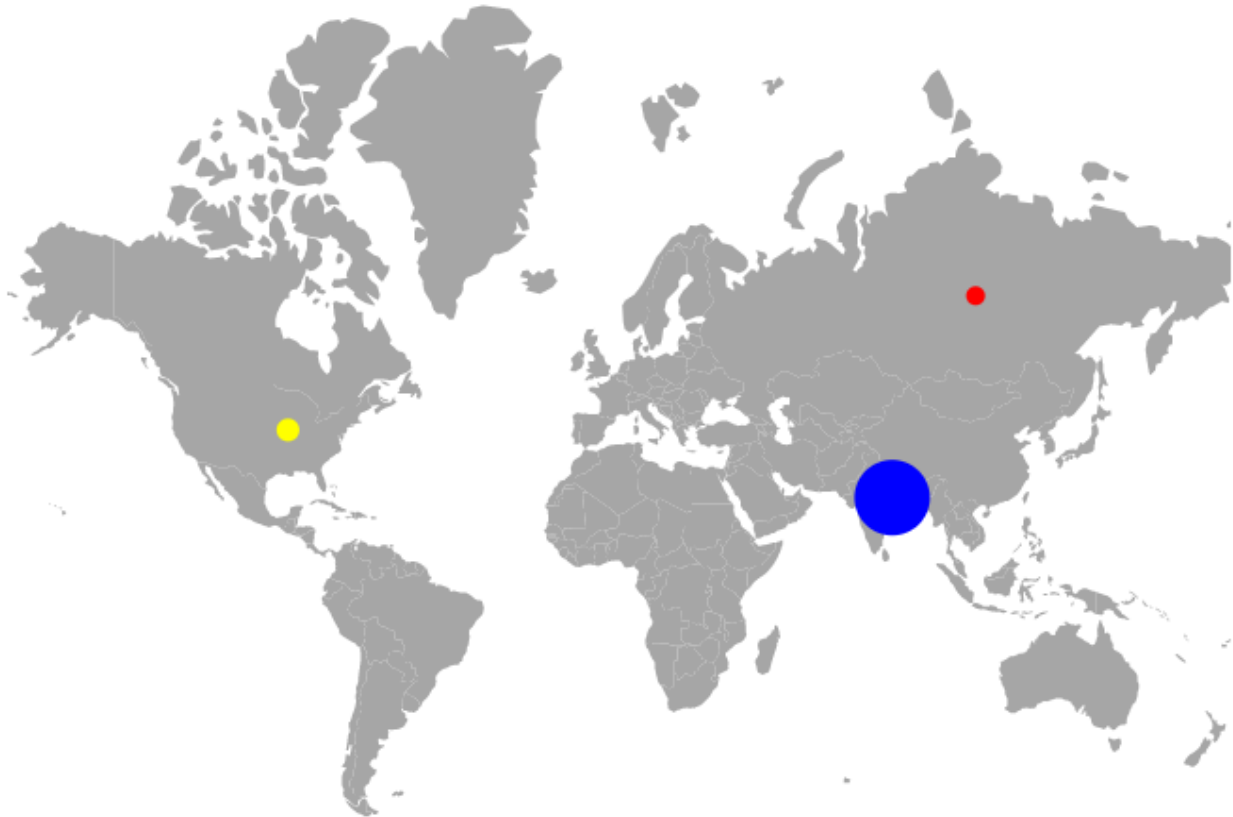


#### Setting the range of the bubble size

The size of the bubbles is calculated from the values got from the [ValuePath](#) property. The range for the radius of the bubbles can be modified using [MinRadius](#) and [MaxRadius](#) properties.

**ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
DataSource="PopulationDetails" ShapeDataPath="Name" ShapePropertyPath='new
string[] { "name"}' TValue="PopulationDetail">
    <MapsBubbleSettings>
      <MapsBubble Visible="true" ValuePath="Density" ColorValuePath="Color"
MinRadius="5" MaxRadius="20"
DataSource="PopulationDetails" TValue="PopulationDetail">
    </MapsBubble>
  </MapsBubbleSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class PopulationDetail
{
public string Name { get; set; }
public double Population { get; set; }
public double Density { get; set; }
public string Color { get; set; }
};
public List<PopulationDetail> PopulationDetails = new List<PopulationDetail>
{
new PopulationDetail
{
Name ="United States", Population = 325020000, Density = 33, Color="yellow"
},
new PopulationDetail
{
Name = "Russia", Population = 142905208, Density = 8.3, Color="red"
},
new PopulationDetail
{
Name="India", Population=1198003000, Density=364, Color="blue"
}
};
}
```



### Multiple bubble groups

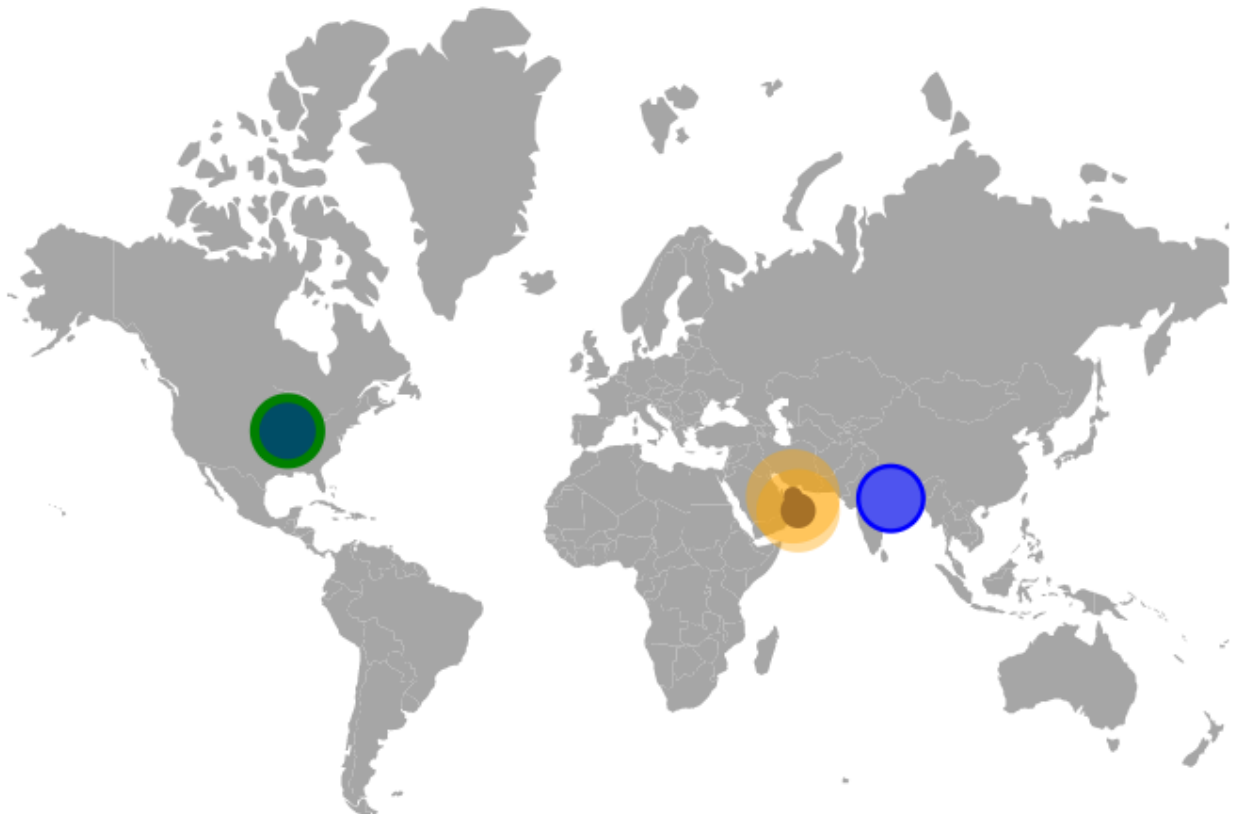
Multiple groups of bubbles can be added in the Maps by adding multiple [MapsBubble](#) in the [MapsBubbleSettings](#) and customization for the bubbles can be done with the [MapsBubble](#) class. In the following example, the gender-wise population ratio is demonstrated with two different bubble groups.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapePropertyPath='new string[] { "name"}' DataSource="GenderRatios"
ShapeDataPath="Country" TValue="GenderRatio">
      @* To add multiple bubble groups *@
      <MapsBubbleSettings>
        <MapsBubble Visible="true" MinRadius="5" MaxRadius="20"
ValuePath="FemaleRatio" ColorValuePath="FemaleRatioColor"
DataSource="GenderRatios" TValue="GenderRatio">
        </MapsBubble>
        <MapsBubble Visible="true" BubbleType="BubbleType.Circle" Opacity="0.4"
MinRadius="15" MaxRadius="25" ValuePath="MaleRatio"
ColorValuePath="MaleRatioColor" DataSource="GenderRatios"
TValue="GenderRatio">
        </MapsBubble>
      </MapsBubbleSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```

```
@code{
public class GenderRatio
{
    public string Country { get; set; }
    public double FemaleRatio { get; set; }
    public double MaleRatio { get; set; }
    public string FemaleRatioColor { get; set; }
    public string MaleRatioColor { get; set; }
}

public List<GenderRatio> GenderRatios = new List<GenderRatio> {
    new GenderRatio {
        Country = "United States", FemaleRatio = 50.50442726, MaleRatio = 49.49557274,
        FemaleRatioColor = "green", MaleRatioColor = "blue"
    },
    new GenderRatio {
        Country = "India", FemaleRatio = 48.18032713, MaleRatio = 51.81967287,
        FemaleRatioColor = "blue", MaleRatioColor = "#c2d2d6"
    },
    new GenderRatio {
        Country = "Oman", FemaleRatio = 34.15597234, MaleRatio = 65.84402766,
        FemaleRatioColor = "#09156d", MaleRatioColor = "orange"
    },
    new GenderRatio {
        Country = "United Arab Emirates", FemaleRatio = 27.59638942, MaleRatio =
        72.40361058, FemaleRatioColor = "#09156d", MaleRatioColor = "orange"
    }
};
}
```



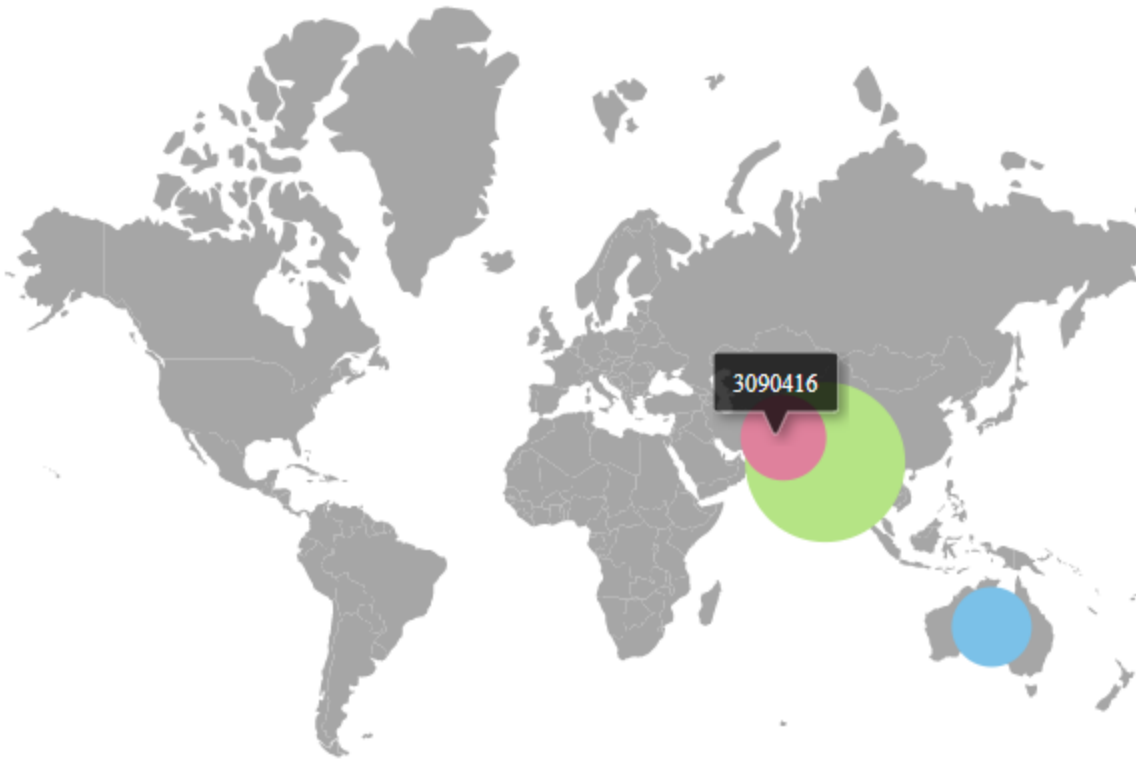


### Enable tooltip for bubble

The tooltip for the bubbles can be enabled by setting the [Visible](#) property of the [MapsBubbleTooltipSettings](#) as **true**. The content for the tooltip can be set using the [ValuePath](#) property in the [MapsBubbleTooltipSettings](#) of the [MapsBubble](#) where the value for the [ValuePath](#) property is the field name from the data source of the [MapsBubble](#). Any HTML element can be added as the template in tooltip using the [TooltipTemplate](#) property.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapeDataPath="Name" ShapePropertyPath='new string[] {"name"}'
TValue="Country">
      @* To add bubbles based on population count *@
      <MapsBubbleSettings>
        <MapsBubble Visible="true" ValuePath="Population" ColorValuePath="Color"
MinRadius=20 MaxRadius=40
DataSource="PopulationDetails" TValue="Country">
          <MapsBubbleTooltipSettings Visible="true"
ValuePath="Population"></MapsBubbleTooltipSettings>
        </MapsBubble>
      </MapsBubbleSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code{
public class Country
{
    public string Name { get; set; }
    public double Population { get; set; }
    public string Color { get; set; }
};
public List<Country> PopulationDetails = new List<Country> {
    new Country
    {
        Name ="United States", Population = 325020000, Color = "#b5e485"
    },
    new Country
    {
        Name = "Russia", Population = 142905208, Color = "#7bc1e8"
    },
    new Country
    {
        Name="India", Population=1198003000, Color = "#df819c"
    }
};
}
```



### Legend in Blazor Maps Component

A Legend is a visual representation of the symbols used on the Maps. It can be represented in various colors, shapes or other identifiers based on the data and provides valuable information for interpreting what the Maps are displaying. It explains what each symbol in the Maps represents. Legends are enabled by setting the [Visible](#) property of [MapsLegendSettings](#) to **true**.

#### Modes of legend

Legend had two types of mode.

1. [Default](#) mode
2. [Interactive](#) mode

#### Default mode

Default mode legends having symbols with legend labels, used to identify the shape or bubble or marker color. To enable this option by setting the [Mode](#) property of [MapsLegendSettings](#) as **Default**.

#### Interactive mode

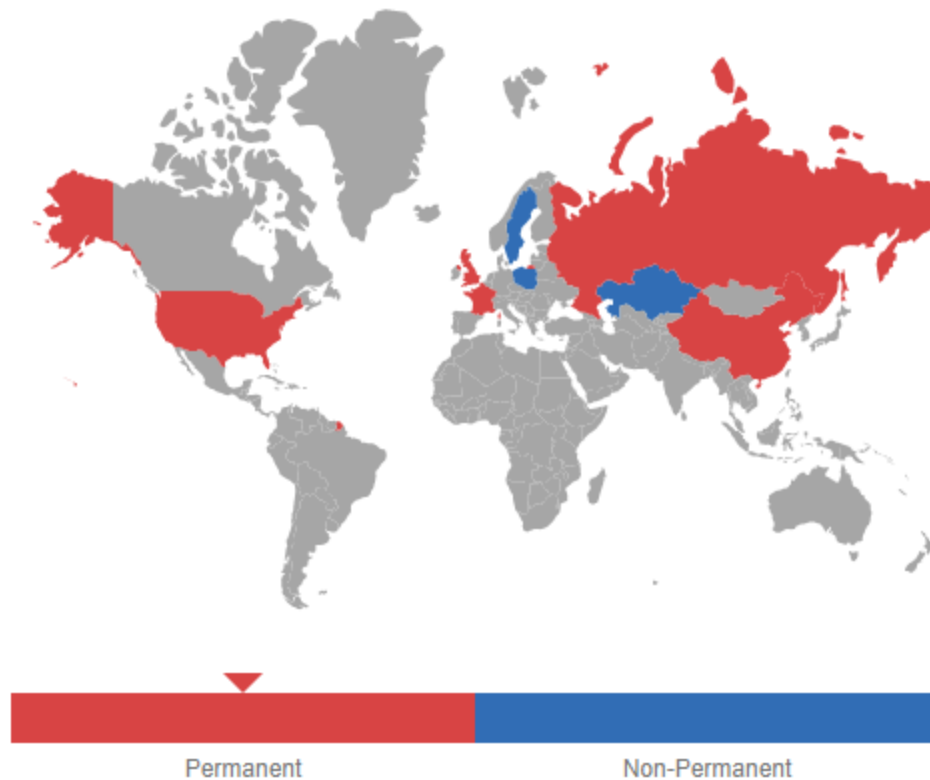
The legends can be made interactive with an arrow mark indicating the exact range color in the legend when the mouse hovers over the corresponding shapes. To enable this type of mode by setting the [Mode](#) property of [MapsLegendSettings](#) as **Interactive**. The [InvertedPointer](#) property is used to enable or disable the visibility of the inverted pointer in interactive legend in Maps.

#### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
@* To set legend mode as interactive *@
<MapsLegendSettings Visible="true" Mode="LegendMode.Interactive"
InvertedPointer="true">
</MapsLegendSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
TValue="UNCouncilCountry"
DataSource="SecurityCouncilDetails" ShapePropertyPath='new string[]
{"name"}' ShapeDataPath="Name">
<MapsShapeSettings ColorValuePath="Membership">
<MapsShapeColorMappings>
<MapsShapeColorMapping Value="Permanent" Color='new string[] {"#D84444"}' />
<MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
{"#316DB5"}' />
</MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
private List<UNCouncilCountry> SecurityCouncilDetails = new
List<UNCouncilCountry>{
new UNCouncilCountry { Name= "China", Membership= "Permanent"},
new UNCouncilCountry { Name= "France", Membership= "Permanent" },
new UNCouncilCountry { Name= "Russia", Membership= "Permanent"},
new UNCouncilCountry { Name= "Kazakhstan", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "Poland", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "Sweden", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "United Kingdom", Membership= "Permanent"},
new UNCouncilCountry { Name= "United States", Membership= "Permanent"}
};
public class UNCouncilCountry
{
public string Name { get; set; }
public string Membership { get; set; }
};
}

```



### Positioning of the legend

The legend can be positioned in the following two ways:

- Absolute position
- Dock position

#### <b>Absolute position</b>

The legend of the Maps can be positioned using the [X](#) and [Y](#) properties in the [MapsLegendSettings](#). For positioning the legend based on co-ordinates corresponding to a Maps, the [Position](#) property is set as **Float**.

#### <b>Dock position</b>

Legends are positioned in the following locations within the container. The [Position](#) property in [MapsLegendSettings](#) is used to set these options in Maps.

- Top
- Left
- Bottom
- Right

The above four positions can be aligned with combination of **Near**, **Center**, and **Far** using [Alignment](#) property in [MapsLegendSettings](#). So, the legend can be aligned to 12 positions.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
```

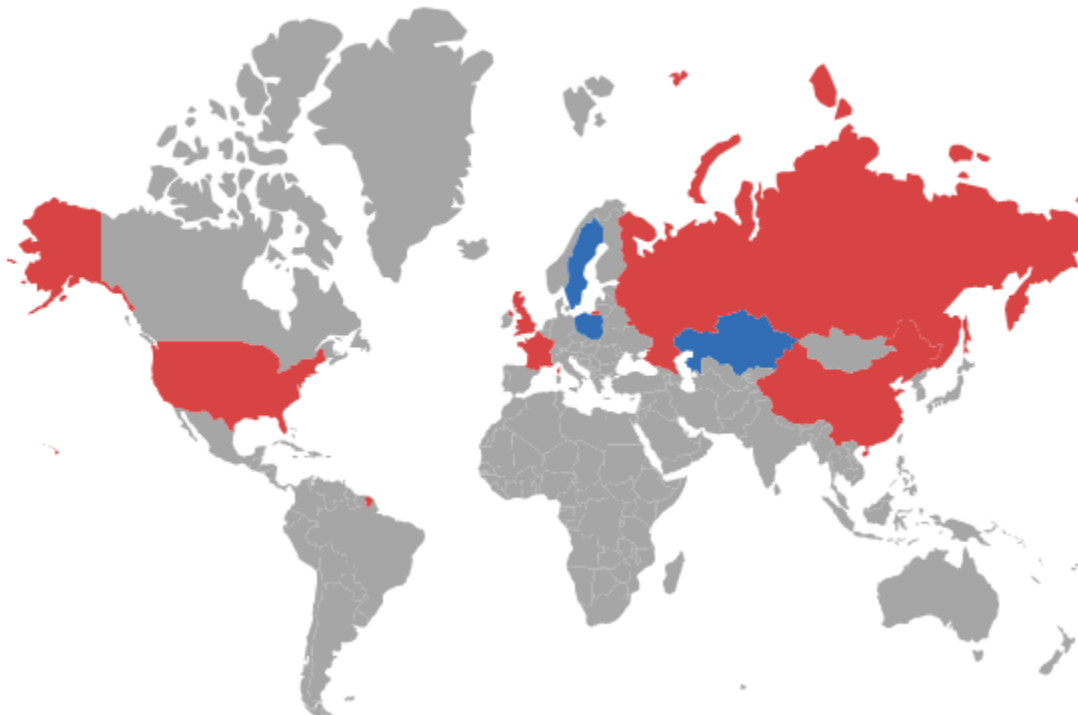
```

<SfMaps>
@* To position the legend *@
<MapsLegendSettings Visible="true" Position="LegendPosition.Top"
Alignment="Alignment.Near">
</MapsLegendSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapeDataPath="Name"
DataSource="SecurityCouncilDetails" ShapePropertyPath='new string[]
{"name"}' TValue="UNCouncilCountry">
<MapsShapeSettings ColorValuePath="Membership">
<MapsShapeColorMappings>
<MapsShapeColorMapping Value="Permanent" Color='new string[] {"#D84444"}' />
<MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
{"#316DB5"}' />
</MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>

```

Refer [code block](#) to know the property value of securityCouncilDetails.

● Permanent ● Non-Permanent



### Legend for shapes

Legend for shapes can be generated from color mapping types such as equal color mapping, range color mapping and desaturation color mapping.

The below code snippet demonstrate the equal color mapping legends for the shapes. To bind the **MembershipDetails** data to the [DataSource](#) property of [MapsLayer](#). Set the value of [ShapePropertyPath](#) to **name** and [ShapeDataPath](#) to **Country**. To enable equal color mapping, set the [MapsShapeColorMapping](#) in [MapsShapeSettings](#). Finally, set the [Visible](#) property of [MapsLegendSettings](#) as **true**. The [Label](#) property in [MapsColorMapping](#) is used to set the text name for legend in Maps.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLegendSettings Visible="true">
</MapsLegendSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapeDataPath="Country"
DataSource="MembershipDetails" ShapePropertyPath='new string[] {"name"}'
TValue="UNCouncil">
<MapsShapeSettings ColorValuePath="Membership" >
<MapsShapeColorMappings>
<MapsShapeColorMapping Value="Permanent" Color='new string[] {"#D84444"}' />
<MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
{"#316DB5"}' />
</MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class UNCouncil
{
public string Country { get; set; }
public string Membership { get; set; }
}
private List<UNCouncil> MembershipDetails = new List<UNCouncil> {
new UNCouncil { Country= "China", Membership= "Permanent" },
new UNCouncil { Country= "France",Membership= "Permanent" },
new UNCouncil { Country= "Russia",Membership= "Permanent" },
new UNCouncil { Country= "Kazakhstan",Membership= "Non-Permanent" },
new UNCouncil { Country= "Poland",Membership= "Non-Permanent" },
new UNCouncil { Country= "Sweden",Membership= "Non-Permanent" }
};
}
```



### Legend shape

Maps supports the following types of legend shapes. The [Shape](#) property in the [MapsLegendSettings](#) can be used to change the type of legend shapes.

- Circle
- Rectangle
- Triangle
- Diamond
- Cross
- Star
- HorizontalLine
- VerticalLine
- Pentagon
- InvertedTriangle

The shape of legends can be customized using the [ShapeHeight](#), [ShapeWidth](#), [ShapePadding](#) properties and [MapsLegendShapeBorder](#).

### Customization

The following properties and classes are available in legend to customize the legend and legend text in Maps.

- [Background](#) - To customize the background color of the Legend.

- [MapsLegendBorder](#) - To customize the color and width of the border for the Legend.
- [Fill](#) - To apply the color for the Legend.
- [LabelDisplayMode](#) - To customize the display mode for the Legend text.
- [LabelPosition](#) - To customize the position of the Legend text.
- [Orientation](#) - To customize the orientation of the Legend.
- [MapsLegendTextStyle](#) - To customize the text style for Legend.
- [MapsLegendTitle](#) - To apply the title for the Legend.
- [MapsLegendTitleStyle](#) - To customize the style of the title for the Legend.
- [Height](#) - To customize the height of the Legend.
- [Width](#) - To customize the width of the Legend.
- [Opacity](#) - To apply the opacity to the Legend.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
@* To customize the legend *@
<MapsLegendSettings Visible="true"
Shape="Syncfusion.Blazor.Maps.LegendShape.Star" ShapeHeight="30"
ShapeWidth="30" ShapePadding="10">
<MapsLegendShapeBorder Color="blue" Width="0.5">
</MapsLegendShapeBorder>
</MapsLegendSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapeDataPath="Country"
DataSource="MembershipDetails" ShapePropertyPath='new string[] {"name"}'
TValue="UNCouncilCountry">
<MapsShapeSettings ColorValuePath="Membership">
<MapsShapeColorMappings>
<MapsShapeColorMapping Value="Permanent" Color='new string[] {"#D84444"}' />
<MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
{"#316DB5"}' />
</MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class UNCouncil
{
public string Country { get; set; }
public string Membership { get; set; }
}
private List<UNCouncil> MembershipDetails = new List<UNCouncil>
{
new UNCouncil { Country= "China", Membership= "Permanent" },
new UNCouncil { Country= "France",Membership= "Permanent" },
new UNCouncil { Country= "Russia",Membership= "Permanent" },
new UNCouncil { Country= "Kazakhstan",Membership= "Non-Permanent" },
new UNCouncil { Country= "Poland",Membership= "Non-Permanent" },
new UNCouncil { Country= "Sweden",Membership= "Non-Permanent" }
};
}
```





#### *Legend for items excluded from color mapping*

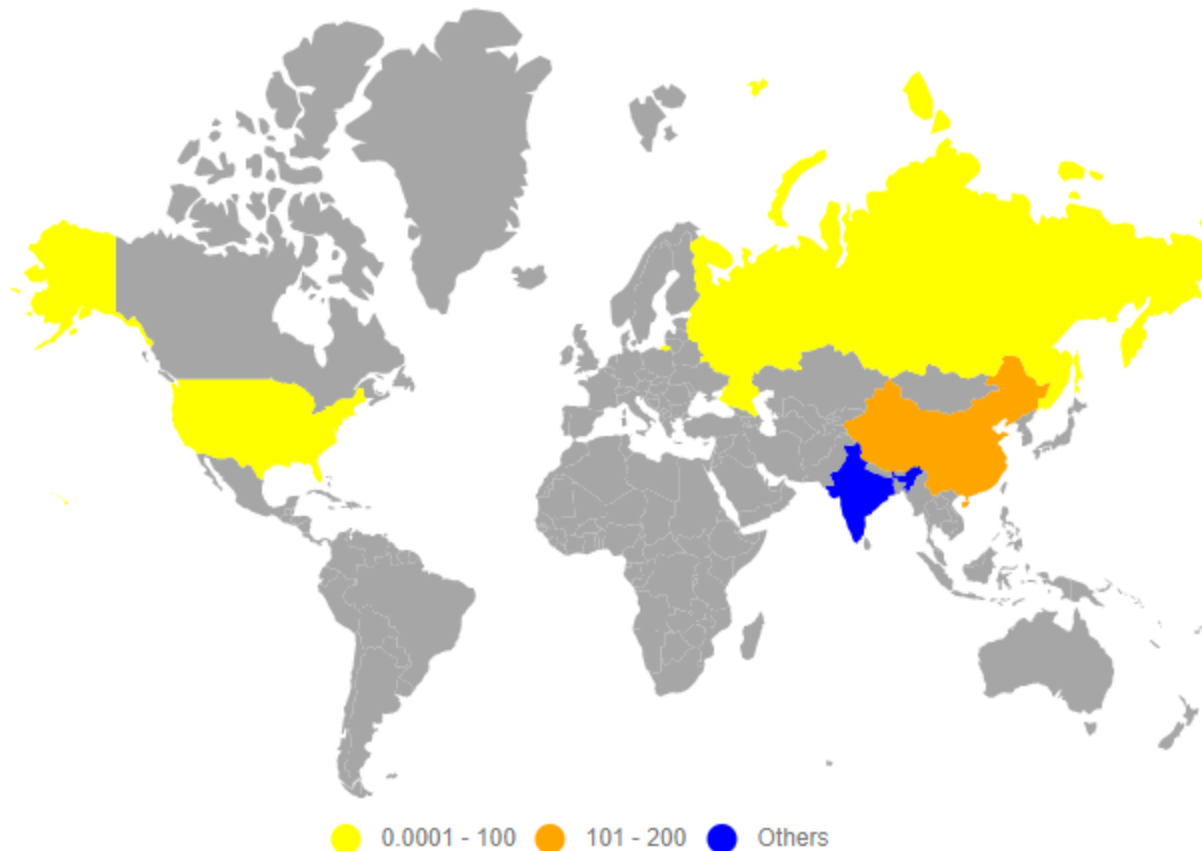
The legend can be enabled for items excluded from the color mapping using the [Color](#) property in [MapsShapeColorMapping](#).

In the following example, color mapping is added for the ranges from **0** to **200**. If there are any records in the data source that are outside of this range, the color mapping will not be applied. To apply the color for these excluded items, set the [Color](#) property alone in the [MapsShapeColorMapping](#). To enable legend for these items, set the [Visible](#) property of [MapsLegendSettings](#) to **true**.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLegendSettings Visible="true"></MapsLegendSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      ShapeDataPath="Name"
      DataSource="populationDetails" ShapePropertyPath='new string[] {"name"}'
      TValue="PopulationDetail">
      <MapsShapeSettings ColorValuePath="Density">
      <MapsShapeColorMappings>
      <MapsShapeColorMapping StartRange="0.0001" EndRange="100" Color='new
        string[] {"yellow"}' />
    </MapsShapeColorMapping>
  </MapsShapeColorMappings>
    </MapsShapeSettings>
  </MapsLayer>
  </MapsLayers>
</SfMaps>
```

```
<MapsShapeColorMapping StartRange="101" EndRange="200" Color='new string[]
{"orange"}' />
<MapsShapeColorMapping Color='new string[] {"blue"}' />
</MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class PopulationDetail
{
public string Name { get; set; }
public double Population { get; set; }
public double Density { get; set; }
};
private List<PopulationDetail> populationDetails = new
List<PopulationDetail> {
new PopulationDetail
{
Name ="United States", Population = 325020000, Density = 33
},
new PopulationDetail
{
Name = "Russia", Population = 142905208, Density = 8.3
},
new PopulationDetail
{
Name="India", Population=1198003000, Density=364
},
new PopulationDetail
{
Name="China", Population=1389750000, Density=144
}
};
}
```



#### *Hiding desired legend items*

Use the [ShowLegend](#) property in the [MapsShapeColorMapping](#) to show or hide the desired legend items in Maps. If the [ShowLegend](#) property is set to **false**, the legend item will be hidden. otherwise, it will be visible.

#### **ASPX-CS**

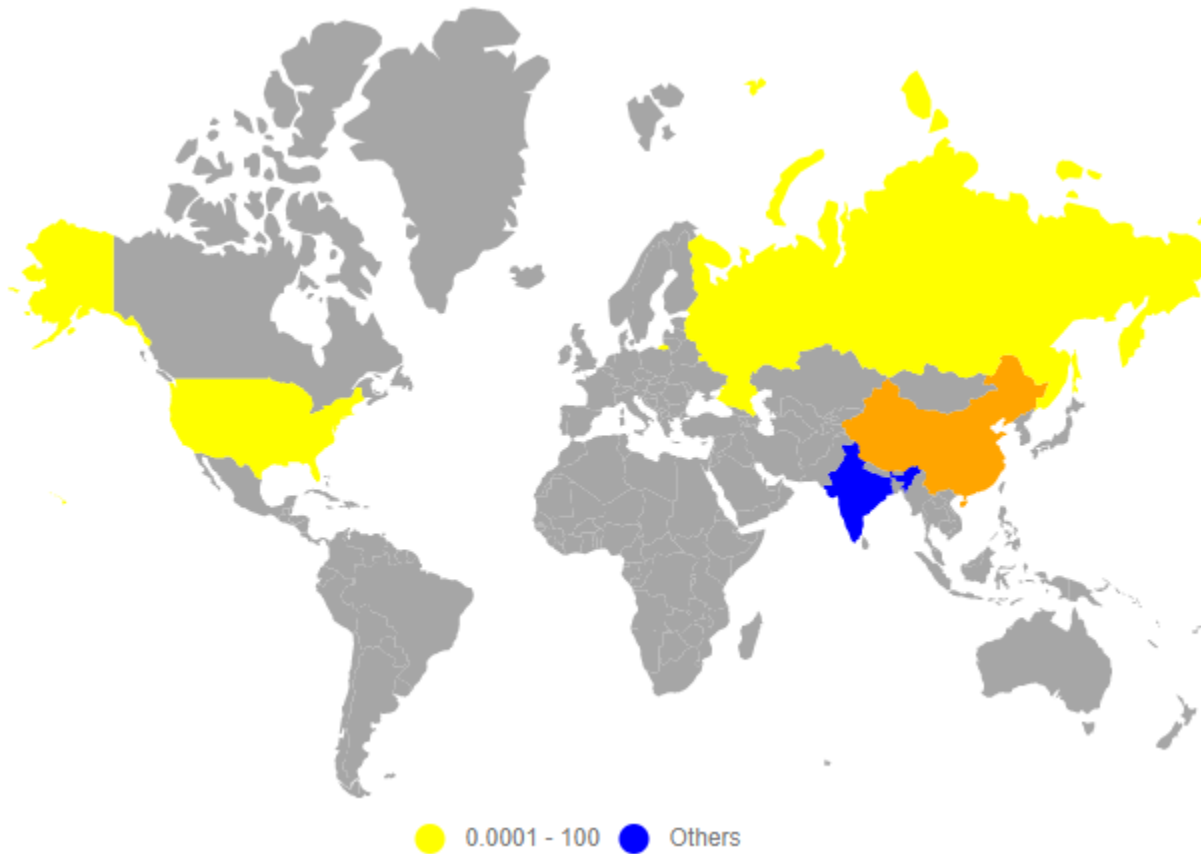
```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLegendSettings Visible="true"></MapsLegendSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      ShapeDataPath="Name"
      DataSource="populationDetails" ShapePropertyPath='new string[] {"name"}'
      TValue="PopulationDetail">
      <MapsShapeSettings ColorValuePath="Density">
        <MapsShapeColorMappings>
          <MapsShapeColorMapping StartRange="0.0001" EndRange="100" Color='new
            string[] {"yellow"}' ShowLegend="true" />
          @* hide legend for this range *@
          <MapsShapeColorMapping StartRange="101" EndRange="200" Color='new string[]
            {"orange"}' ShowLegend="false" />
          <MapsShapeColorMapping Color='new string[] {"blue"}' ShowLegend="true" />
        </MapsShapeColorMappings>
      </MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```

```

</MapsLayer>
</MapsLayers>
</SfMaps>

```

Refer [code block](#) to know the property value of PopulationDetails.



#### Hide legend items based on data source value

Depending on the boolean values provided in the data source, the legend items will be hidden or visible. Bind the field name that contains the visibility state in the data source to the [ShowLegendPath](#) property of the [MapsLegendSettings](#) to achieve this.

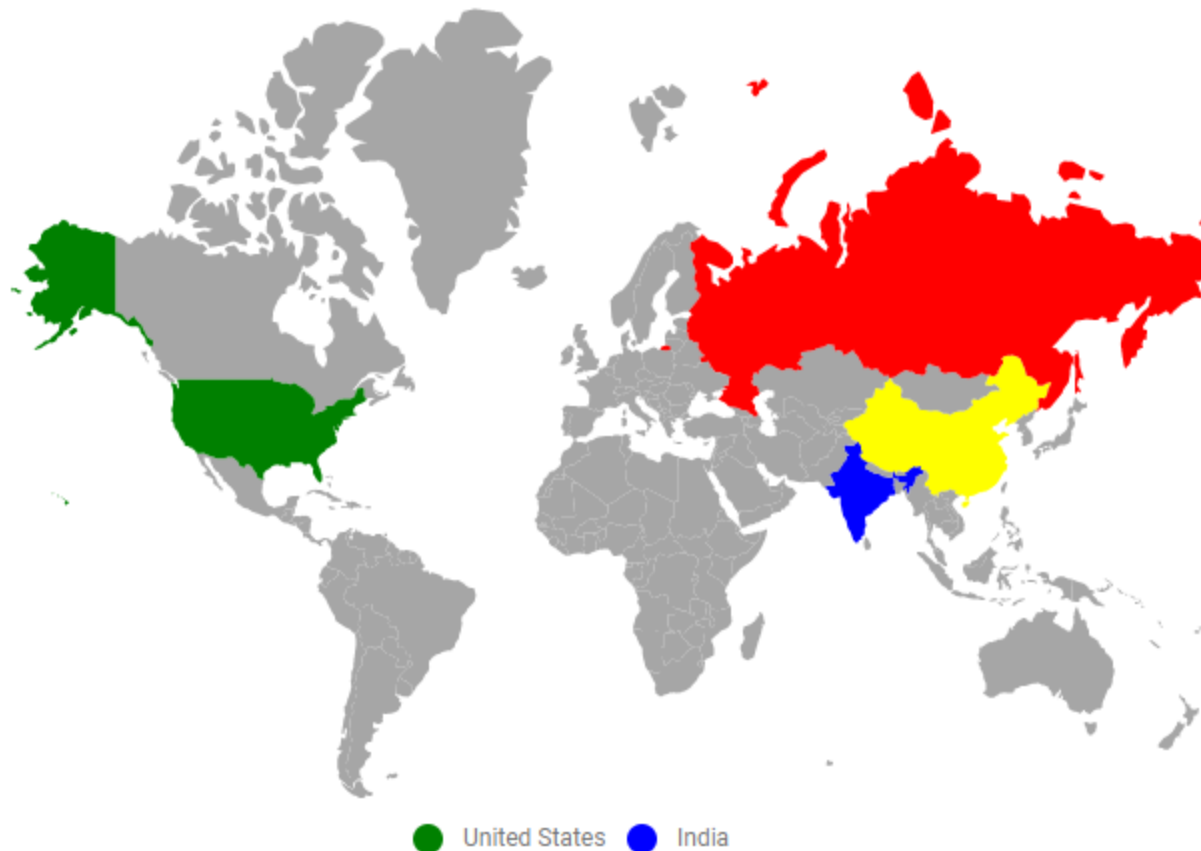
#### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
@* To hide legend based in data source fields *@
<MapsLegendSettings Visible="true" ShowLegendPath="LegendVisibility"/>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapeDataPath="Name"
DataSource="populationDetails" ShapePropertyPath='new string[] {"name"}'
TValue="PopulationDetail">
<MapsShapeSettings ColorValuePath="Color"> </MapsShapeSettings>
</MapsLayer>

```

```
</MapsLayers>
</SfMaps>
@code{
public class PopulationDetail
{
    public string Name { get; set; }
    public double Population { get; set; }
    public double Density { get; set; }
    public bool LegendVisibility { get; set; }
    public string Color { get; set; }
};
private List<PopulationDetail> populationDetails = new
List<PopulationDetail> {
    new PopulationDetail
    {
        Name = "United States", Population = 325020000, Density = 33,
        LegendVisibility = true, Color = "green"
    },
    new PopulationDetail
    {
        Name = "Russia", Population = 142905208, Density = 8.3, LegendVisibility =
        false, Color = "red"
    },
    new PopulationDetail
    {
        Name="India", Population=1198003000, Density=364, LegendVisibility = true,
        Color = "blue"
    },
    new PopulationDetail
    {
        Name="China", Population=1389750000, Density=144, LegendVisibility = false,
        Color = "orange"
    }
};
}
```



#### *Binding legend item text from data source*

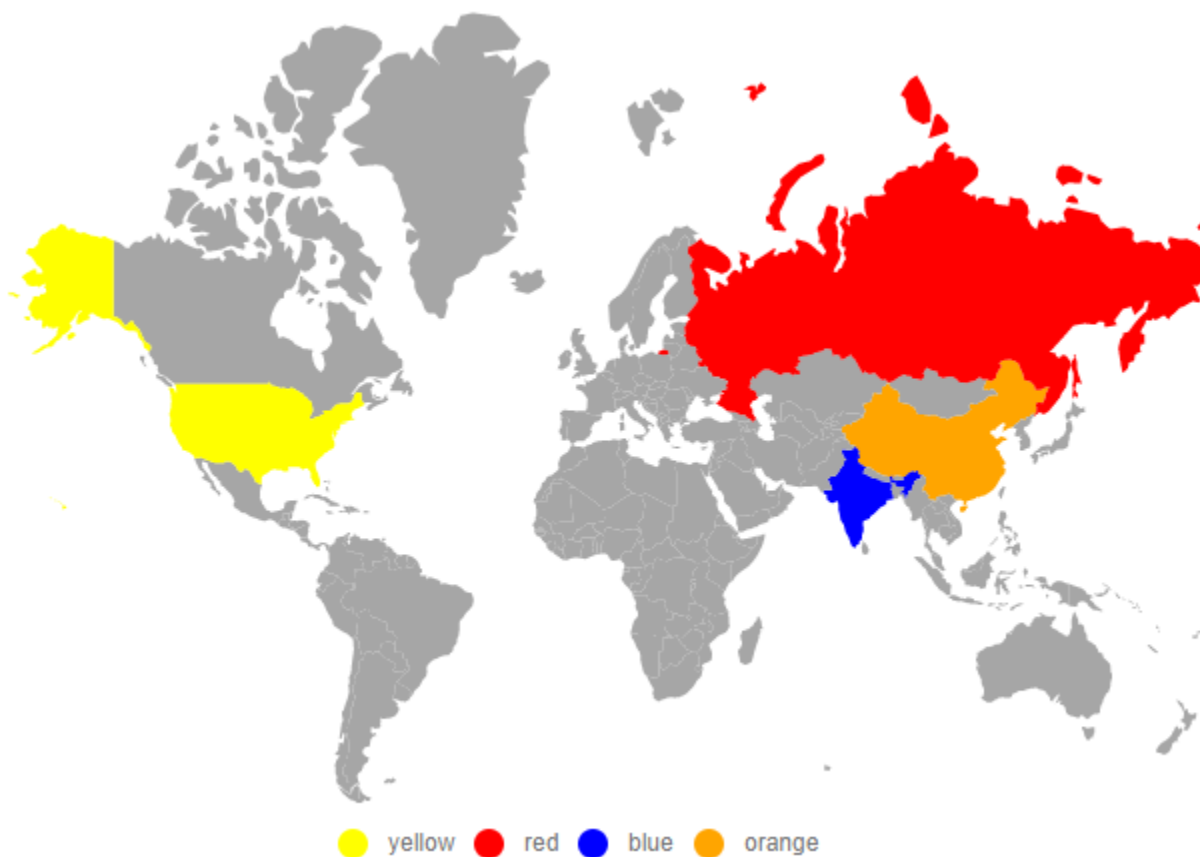
To show the legend text based on values provided in the data source, use the [ValuePath](#) property in the [MapsLegendSettings](#).

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLegendSettings Visible="true" ValuePath="Color"></MapsLegendSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      ShapeDataPath="Name"
      DataSource="PopulationDetails" ShapePropertyPath='new string[] { "name"}'
      TValue="PopulationDetail">
      <MapsShapeSettings ColorValuePath="Color"> </MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>

@code{
public class PopulationDetail
{
    public string Name { get; set; }
    public double Population { get; set; }
    public double Density { get; set; }
    public string Color { get; set; }
};
```

```
private List<PopulationDetail> PopulationDetails = new
List<PopulationDetail> {
    new PopulationDetail
    {
        Name = "United States", Population = 325020000, Density = 33, Color="yellow"
    },
    new PopulationDetail
    {
        Name = "Russia", Population = 142905208, Density = 8.3, Color="red"
    },
    new PopulationDetail
    {
        Name="India", Population=1198003000, Density=364, Color="blue"
    },
    new PopulationDetail
    {
        Name="China", Population=1389750000, Density=144, Color="orange"
    }
};
```



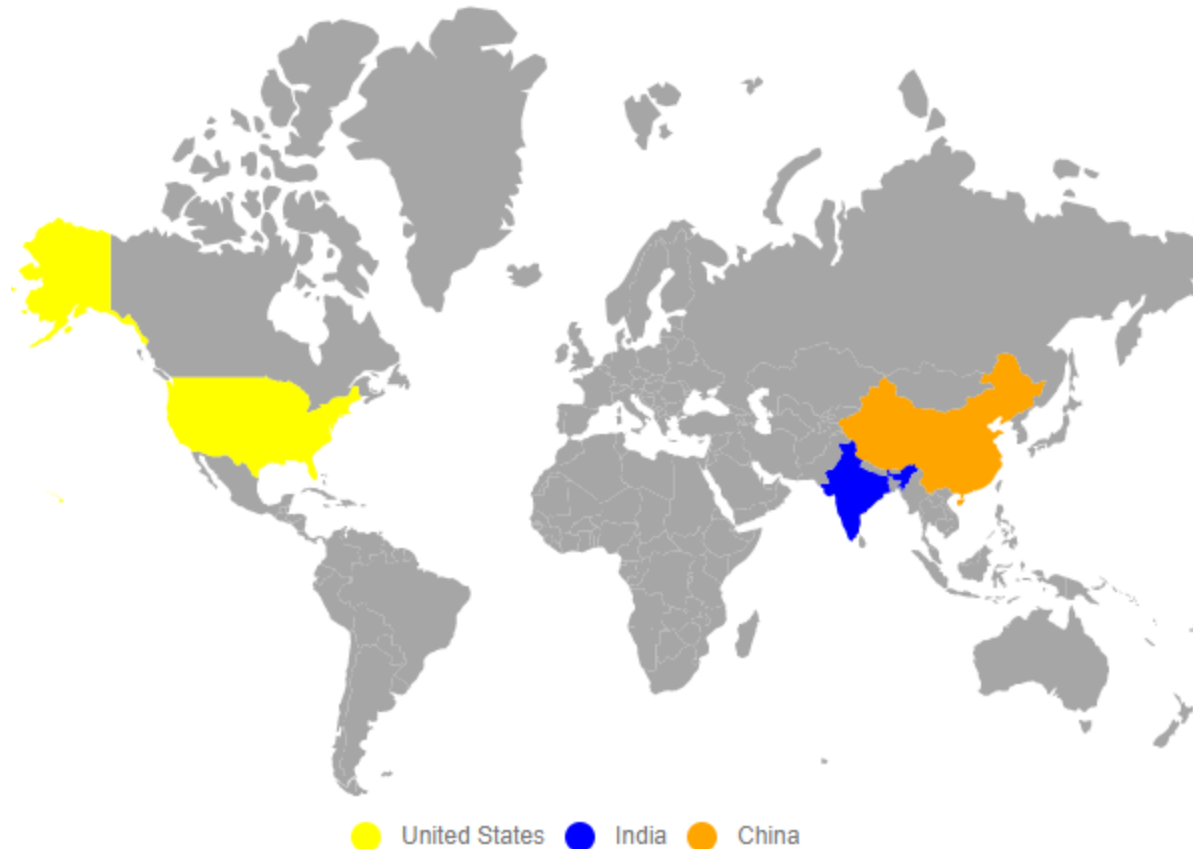
#### *Hiding duplicate legend items*

To hide the duplicate legend items in Maps, set the [RemoveDuplicateLegend](#) property to **true** in the [MapsLegendSettings](#).

**ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLegendSettings Visible="true"
  RemoveDuplicateLegend="true"></MapsLegendSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
    ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
    ShapeDataPath="Name"
    DataSource="populationDetails" TValue="PopulationDetail"
    ShapePropertyPath='new string[] { "name"}'>
      <MapsShapeSettings ColorValuePath="Color"> </MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code{
public class PopulationDetail
{
  public string Name { get; set; }
  public double Population { get; set; }
  public double Density { get; set; }
  public string Color { get; set; }
};
private List<PopulationDetail> populationDetails = new
List<PopulationDetail> {
  new PopulationDetail
  {
    Name = "United States", Population = 325020000, Density = 33, Color =
    "yellow"
  },
  new PopulationDetail
  {
    Name = "United States", Population = 325020000, Density = 33, Color =
    "yellow"
  },
  new PopulationDetail
  {
    Name = "India", Population = 1198003000, Density = 364, Color = "blue"
  },
  new PopulationDetail
  {
    Name = "China", Population = 1389750000, Density = 144, Color = "orange"
  }
};
}
```





#### *Toggle option in legend*

The toggle option has been provided for legend. If the legend can be toggled, the given color will be changed to the corresponding Maps shape item. To enable the toggle options in Legend, set the [Enable](#) property of the [MapsToggleLegendSettings](#) to **true**.

The following properties and components are available to customize the toggle option in legend.

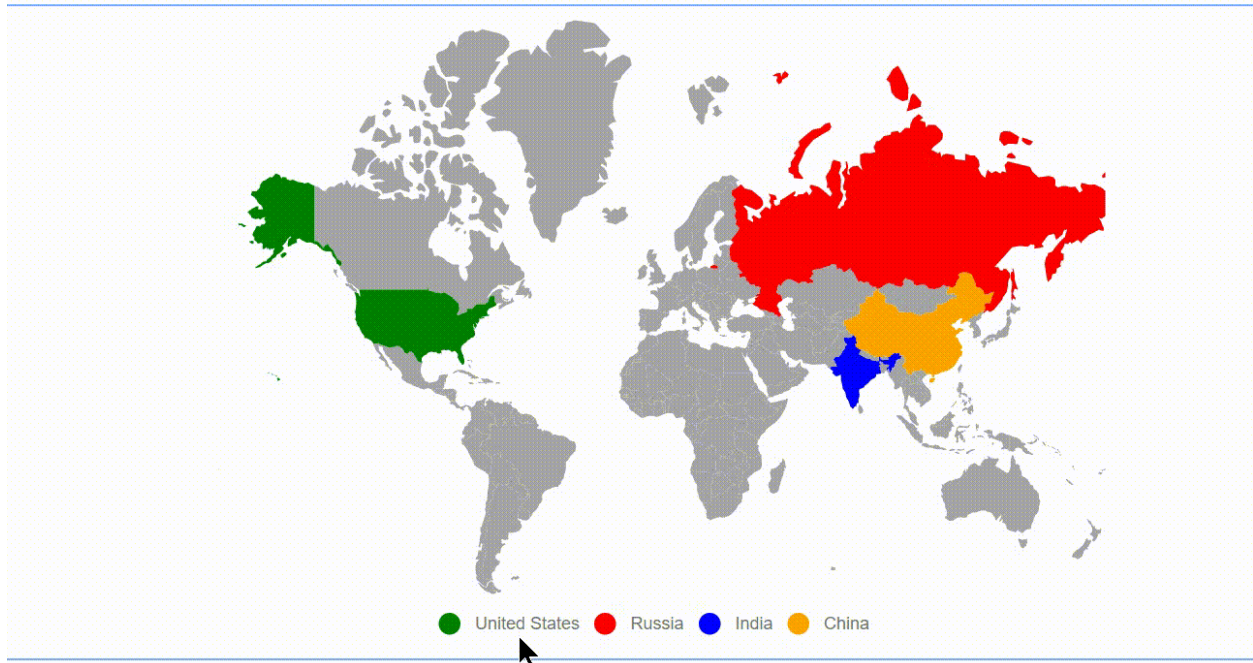
- [ApplyShapeSettings](#) – To apply the [Fill](#) property value of the shape of the Maps when toggling the legend items.
- [Fill](#) - To apply the color to the shape of the Maps for which legend item is toggled.
- [Opacity](#) – To customize the transparency for the shapes for which legend item is toggled.
- [MapsToggleLegendBorder](#) – To customize the color and width of the border of the shapes in Maps.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
@* To hide legend based in data source fields *@
<MapsLegendSettings Visible="true">
<MapsToggleLegendSettings Enable="true" ApplyShapeSettings="false">
<MapsLegendBorder Width="2"
Color="green"></MapsLegendBorder></MapsToggleLegendSettings>
</MapsLegendSettings>
<MapsLayers>
```

```
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapeDataPath="Name"
DataSource="populationDetails" TValue="PopulationDetail"
ShapePropertyPath='new string[] {"name"}'>
  <MapsShapeSettings ColorValuePath="Color"> </MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>

@code{
public class PopulationDetail
{
public string Name;
public double Population;
public double Density;
public bool LegendVisibility;
public string Color;
};
private List<PopulationDetail> populationDetails = new
List<PopulationDetail> {
new PopulationDetail
{
Name="United States", Population = 325020000, Density = 33,
LegendVisibility = true, Color = "green"
},
new PopulationDetail
{
Name = "Russia", Population = 142905208, Density = 8.3, LegendVisibility =
false, Color = "red"
},
new PopulationDetail
{
Name="India", Population=1198003000, Density=364, LegendVisibility = true,
Color = "blue"
},
new PopulationDetail
{
Name="China", Population=1389750000, Density=144, LegendVisibility = false,
Color = "orange"
}
};
}
```



Enable legend for bubbles

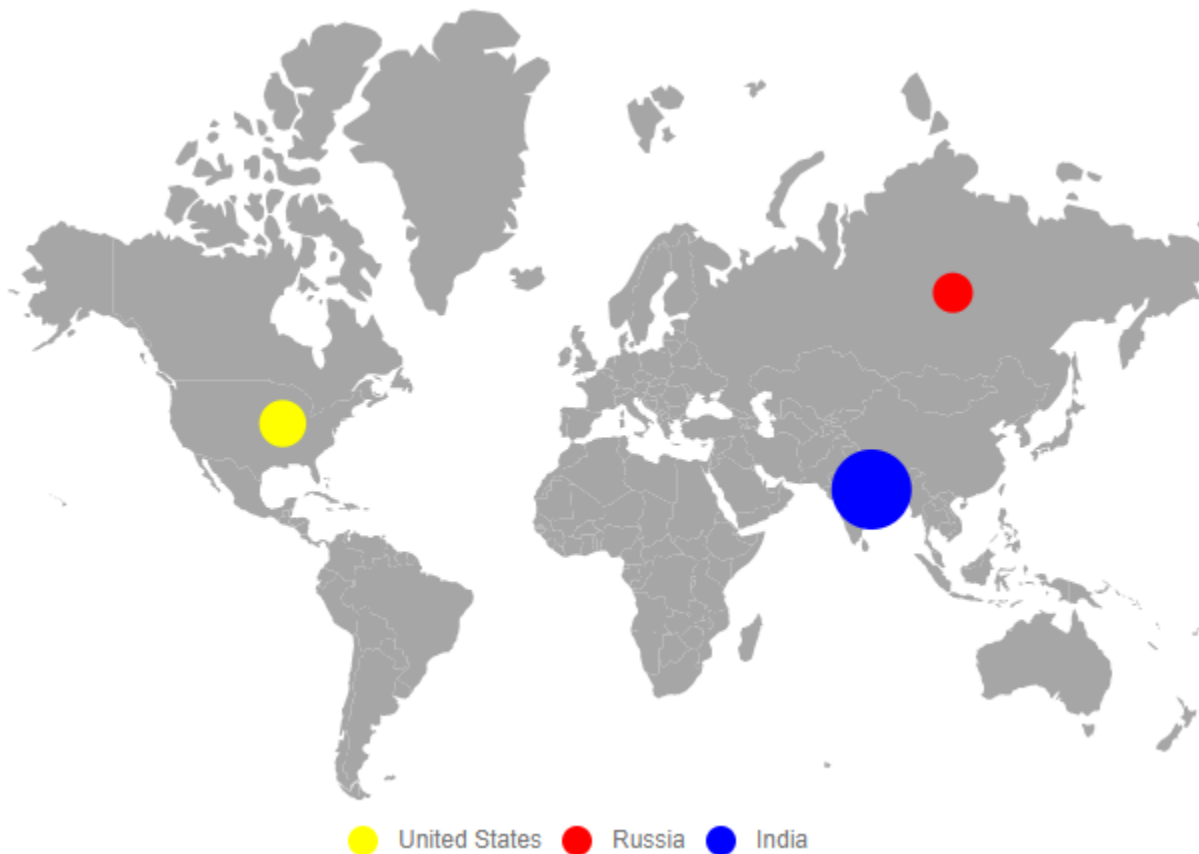
To enable the legend for the bubble by setting the [Visible](#) property of [MapsLegendSettings](#) as **true** and [Type](#) property of [MapsLegendSettings](#) as **Bubbles**.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  @* To enable legend for bubbles *@
  <MapsLegendSettings Visible="true"
    Type="LegendType.Bubbles"></MapsLegendSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      ShapeDataPath="Name"
      DataSource="PopulationDetails" ShapePropertyPath='new string[] { "name"}'
      TValue="PopulationDetail">
      <MapsBubbleSettings>
        <MapsBubble Visible="true" ValuePath="Population" ColorValuePath="Color"
          DataSource="PopulationDetails" TValue="PopulationDetail">
        </MapsBubble>
      </MapsBubbleSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>

@code {
  public class PopulationDetail
  {
    public string Name { get; set; }
    public double Population { get; set; }
    public double Density { get; set; }
    public string Color { get; set; }
  };
}
```

```
private List<PopulationDetail> PopulationDetails = new
List<PopulationDetail> {
    new PopulationDetail
    {
        Name ="United States", Population = 325020000, Density = 33, Color="yellow"
    },
    new PopulationDetail
    {
        Name = "Russia", Population = 142905208, Density = 8.3, Color="red"
    },
    new PopulationDetail
    {
        Name="India", Population=1198003000, Density=364, Color="blue"
    }
};
```



### Enable legend for markers

To enable legend for marker by setting the [Visible](#) property of [MapsLegendSettings](#) as **true** and [Type](#) property of [MapsLegendSettings](#) as **Markers**. The [LegendText](#) property in the [MapsMarker](#) can be used to show the legend text based on values provided in the data source.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
```

```
@* To enable legend for marker *@
<MapsLegendSettings Visible="true"
Type="LegendType.Markers"></MapsLegendSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
<MapsMarkerSettings>
<MapsMarker Visible="true" DataSource="Cities" Height="25" Width="15"
LegendText="Name" TValue="City">
</MapsMarker>
</MapsMarkerSettings>
<MapsShapeSettings Fill="lightgray"></MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
};
private List<City> Cities = new List<City> {
new City { Latitude=35.145083, Longitude=-117.960260, Name= "Californiya" },
new City { Latitude=40.724546, Longitude=-73.850344, Name="New York" },
new City { Latitude= 41.657782, Longitude=-91.533857, Name="Iowa" }
};
}
```



### Navigation Lines in Blazor Maps Component

The navigation lines are used to denote the path between two locations. This feature can be used to draw flight or sea routes. Navigation lines are enabled by setting the [Visible](#) property of the [MapsNavigationLine](#) to **true**.

#### Customization

The following properties and classes are available in [MapsNavigationLine](#) to customize the navigation line of the Maps component.

- [Color](#) - To apply the color for navigation lines in Maps.
- [DashArray](#) - To define the pattern of dashes and gaps that is applied to the outline of the navigation lines.
- [Width](#) - To customize the width of the navigation lines.
- [Angle](#) - To customize the angle of the navigation lines.
- [MapsNavigationLineHighlightSettings](#) - To customize the highlight settings of the navigation line.
- [MapsNavigationLineSelectionSettings](#) - To customize the selection settings of the navigation line.

To navigate the line between two cities on the world map, [Latitude](#) and [Longitude](#) values are used to indicate the start and end points of navigation lines drawn on Maps.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
```

```
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      TValue="string">
      <MapsNavigationLines>
        <MapsNavigationLine Visible="true" Color="black" Angle="90" Width="2"
          DashArray="4"
          Latitude="new double[] { 40.7128, 36.7783 }" Longitude="new double[] { -
            74.0060, -119.4179 }">
        </MapsNavigationLine>
      </MapsNavigationLines>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```



### Enabling the arrows

To enable the arrow in the navigation line, set the [ShowArrow](#) property of [MapsArrow](#) to **true**. The following properties are available in [MapsArrow](#) to customize the arrow of the navigation lines.

- [Color](#) - To apply the color for arrow of the navigation line.
- [Offset](#) - To customize the offset position of the arrow of the navigation line.
- [Position](#) - To customize the position of the arrow in navigation line. The possible values can be [Start](#) and [End](#).
- [Size](#) - To customize the size of the arrow in pixels.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      TValue="string">
      <MapsNavigationLines>
        <MapsNavigationLine Visible="true" Color="blue" Angle="90" Width="2"
          DashArray="4"
          Latitude="new double[]{ 40.7128, 36.7783 }" Longitude="new double[]{ -
            74.0060, -119.4179 }">
          @* To set arrow for navigation line *@
          <MapsArrow ShowArrow="true" Color="blue"></MapsArrow>
        </MapsNavigationLine>
      </MapsNavigationLines>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```



### Annotations in Blazor Maps component

Annotations are used to mark the specific area of interest in the Maps with texts, shapes, or images. Any number of annotations can be added to the Maps component.



### Annotation

By using the `ContentTemplate` property of [MapsAnnotation](#), text content or an HTML element can be specified to render a new HTML element in Maps.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsAnnotations>
    <MapsAnnotation X="0%" Y="50%">
      <ContentTemplate>
        <div>
          <img style="height: 30px; width: 40px" src='src/maps/images/wheel.png'>
        </div>
      </ContentTemplate>
    </MapsAnnotation>
  </MapsAnnotations>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapePropertyPath='new string[] {"name"}' TValue="string">
  </MapsLayer>
</MapsLayers>
</SfMaps>
```



## Annotation customization

### Changing the z-index

The stack order of an annotation element can be changed using the [ZIndex](#) property in the [MapsAnnotation](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsAnnotations>
    <MapsAnnotation X="0%" Y="50%" ZIndex= "-1">
      <ContentTemplate>
        <div>
          <div id="first"><h1>Maps</h1></div>
        </div>
      </ContentTemplate>
    </MapsAnnotation>
  </MapsAnnotations>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      ShapePropertyPath='new string[] {"name"}' TValue="string">
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```



### Positioning an annotation

Annotations can be placed anywhere in the Maps by specifying percentage values to the [X](#) and [Y](#) properties in the [MapsAnnotation](#).

**ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsAnnotations>
    <MapsAnnotation X="20%" Y="50%" ZIndex= "-1">
      <ContentTemplate>
        <div>
          <div id="first"><h1>Maps</h1></div>
        </div>
      </ContentTemplate>
    </MapsAnnotation>
  </MapsAnnotations>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapePropertyPath='new string[] { "name"}' TValue="string">
  </MapsLayer>
</MapsLayers>
</SfMaps>
```

*Alignment of an annotation*

Annotations can be aligned using the [HorizontalAlignment](#) and [VerticalAlignment](#) properties in the [MapsAnnotation](#). The possible values can be [Center](#), [Far](#), [Near](#) and [None](#).

**ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsAnnotations>
```

```

<MapsAnnotation X="20%" Y="10%" ZIndex= "-1"
VerticalAlignment="AnnotationAlignment.Center"
HorizontalAlignment="AnnotationAlignment.Center">
  <ContentTemplate>
    <div>
      <div id="first"><h1>Maps</h1></div>
    </div>
  </ContentTemplate>
</MapsAnnotation>
</MapsAnnotations>
<MapsLayers>
  <MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapePropertyPath='new string[] {"name"}' TValue="string">
  </MapsLayer>
</MapsLayers>
</SfMaps>

```



### Multiple Annotation

Multiple annotations can be added to the Maps by adding multiple [MapsAnnotation](#) in the [MapsAnnotations](#) and customization for the annotations can be done with the [MapsAnnotation](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsAnnotations>
    <MapsAnnotation X="0%" Y="50%">
      <ContentTemplate>
        <div>

```

```

<img style="height: 30px; width: 40px" src='src/maps/images/wheel.png'>
</div>
</ContentTemplate>
</MapsAnnotation>
<MapsAnnotation X="20%" Y="10%" ZIndex= "-1"
VerticalAlignment="AnnotationAlignment.Center"
HorizontalAlignment="AnnotationAlignment.Center">
<ContentTemplate>
<div>
<div id="first"><h1>Maps</h1></div>
</div>
</ContentTemplate>
</MapsAnnotation>
</MapsAnnotations>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapePropertyPath='new string[] {"name"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>

```



## User Interactions in Blazor Maps Component

### Zooming

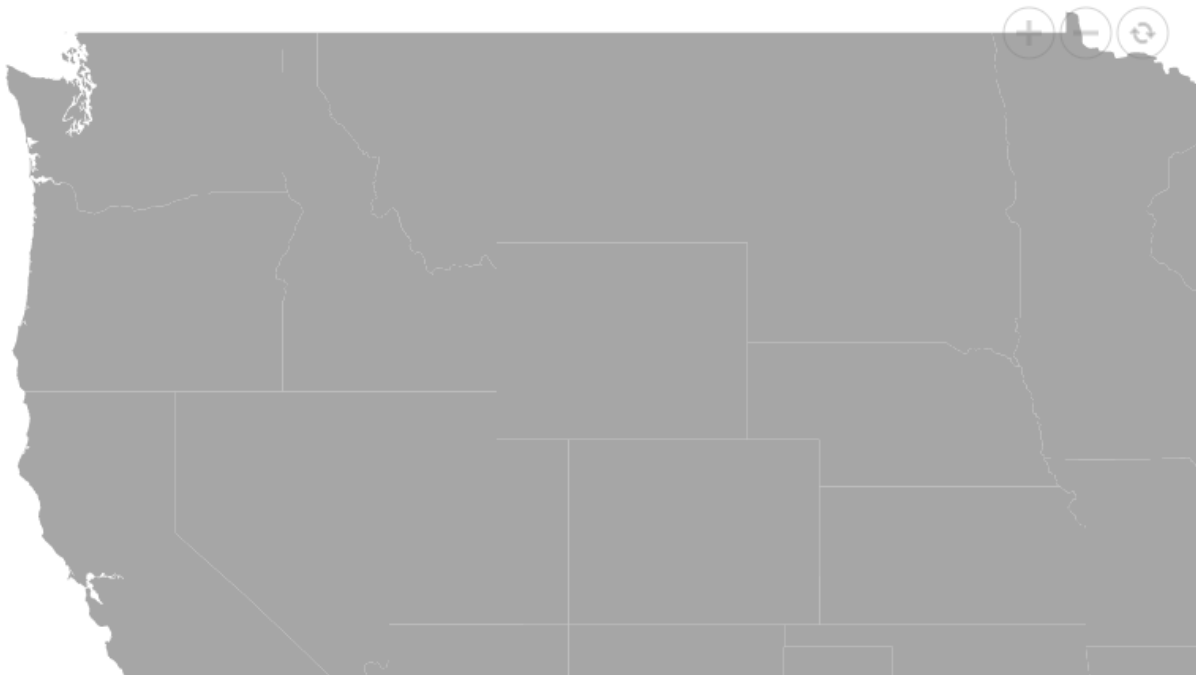
The zooming feature is used to zoom in and out of Maps to show in-depth information. It is controlled by the [ZoomFactor](#) property of the [MapsZoomSettings](#) class. The [ZoomFactor](#) is increased or decrease dynamically based on zoom in and out interaction.

**<b>Enable zooming</b>**

Zooming of Maps is enabled by setting the [Enable](#) property of [MapsZoomSettings](#) to **true**.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsZoomSettings Enable="true"></MapsZoomSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
```



**<b>Enable panning</b>**

To enable the panning feature, set the [EnablePanning](#) property of [MapsZoomSettings](#) to **true**.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsZoomSettings Enable="true" EnablePanning="true"></MapsZoomSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
```

### Various types of zooming

Zooming can be performed in the following types:

#### Zooming toolbar

A toolbar is available in the Maps component to perform zooming and panning operation. The zooming toolbar appears when the [Enable](#) property of [MapsZoomSettings](#) is set to **true**.

The following options are available in toolbar.

1. Zoom - Performs selection zooming.
2. Zoom In - Zooms in the Maps.
3. Zoom Out - Zooms out the Maps.
4. Pan - Switches to panning if selection zoom is enabled.
5. Reset - Restores the Maps to the default view.

By default, the toolbar is rendered with **Zoom In**, **Zoom Out**, and **Reset** options.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings Enable="true">
  </MapsZoomSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```

#### Customization of zooming toolbar

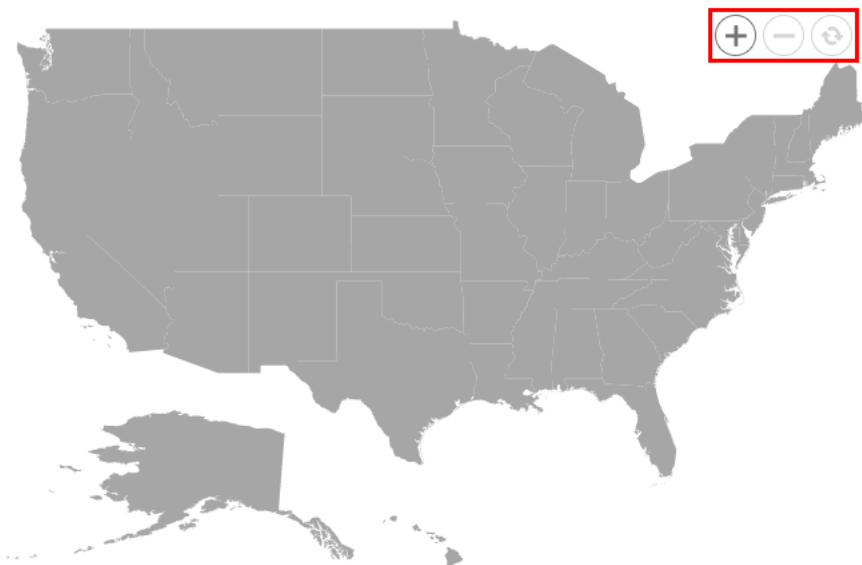
The [MapsZoomToolbarSettings](#) class can be used to customize the zooming toolbar. The following properties are available for customization in the [MapsZoomToolbarSettings](#) class.

- [Orientation](#) - To customize the orientation of the zooming toolbar.
- [HorizontalAlignment](#) - To customize the position type of toolbar when it is placed horizontally.
- [VerticalAlignment](#) - To customize the position type of toolbar when it is placed vertically.
- [BackgroundColor](#) - To customize the background color of the zooming toolbar.
- [BorderColor](#) - To apply the border color to the zooming toolbar.
- [BorderWidth](#) - To set width to the border of the zooming toolbar.
- [BorderOpacity](#) - To set opacity to the border of the zooming toolbar.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings Enable="true">
    <MapsZoomToolbarSettings BackgroundColor="black" BorderColor="red"
    BorderOpacity="1" BorderWidth="3" Orientation="Orientation.Horizontal"
    HorizontalAlignment="Alignment.Far"
    VerticalAlignment="Alignment.Near"></MapsZoomToolbarSettings>
  </MapsZoomSettings>
  <MapsLayers>
```

```
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
```



The [MapsZoomToolBarButton](#) class can be used to customize the buttons in the zooming toolbar. The [MapsZoomToolBarButton](#) class has the following properties.

- [ToolBarItems](#) - To customize the items that are to be shown in the zooming toolbar.
- [Color](#) - To apply the color to the icons in the buttons in the zooming toolbars.
- [Fill](#) - To apply the fill color to the buttons in the zooming toolbars.
- [Opacity](#) - To apply the opacity for the buttons in the zooming toolbars.
- [HighlightColor](#) - To apply the color for the buttons in the zooming toolbar when the mouse has hovered on the toolbar element.
- [SelectionColor](#) - To apply the color for the buttons in the zooming toolbar when clicking the zooming toolbar.
- [Radius](#) - To set the radius for the buttons in the zooming toolbars.
- [Padding](#) - To set the spacing between the buttons in the zooming toolbars.
- [BorderColor](#) - To set the color in the border of the buttons in the zooming toolbars.
- [BorderOpacity](#) - To set the opacity in the border of the buttons in the zooming toolbars.
- [BorderWidth](#) - To set the width of the border of the buttons in the zooming toolbars.

### ASPX-CS

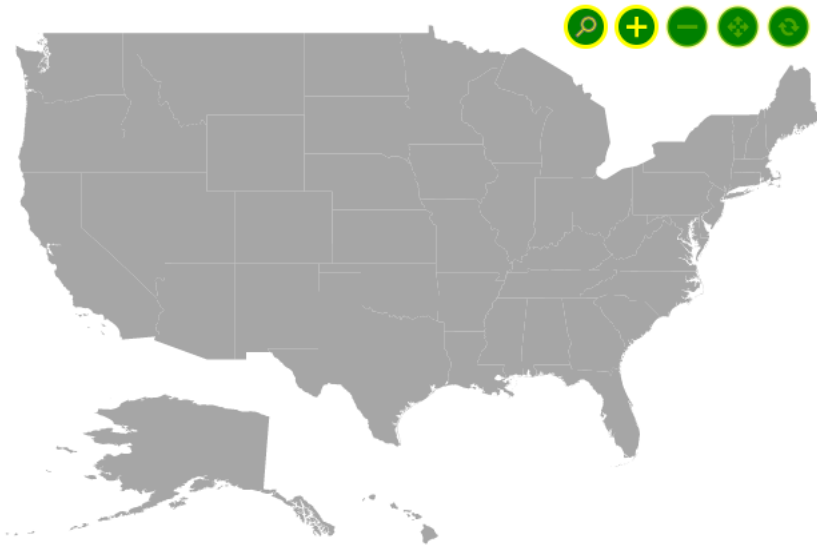
```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsZoomSettings Enable="true">
<MapsZoomToolBarSettings>
<MapsZoomToolBarButton BorderColor="yellow" BorderOpacity="1"
BorderWidth="3" Color="yellow" Fill="green" HighlightColor="red" Opacity="1">
```



```

Padding="5" Radius="30" SelectionColor="red" ToolbarItems="new
List<ToolbarItem>() { ToolbarItem.Zoom,
ToolbarItem.ZoomIn, ToolbarItem.ZoomOut, ToolbarItem.Pan, ToolbarItem.Reset
}"></MapsZoomToolbarButton>
</MapsZoomToolbarSettings>
</MapsZoomSettings>
<MapsLayers>
<MapLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
</MapLayer>
</MapsLayers>
</SfMaps>

```



The [MapsZoomToolbarTooltipSettings](#) class can be used to customize the tooltip in the zooming toolbar. The [MapsZoomToolbarTooltipSettings](#) class has the following properties.

- [Visible](#) - To enable or disable the tooltip in the zooming toolbar.
- [Fill](#) - To set the fill color to the tooltip in the zooming toolbar.
- [BorderColor](#) - To set the color in the border of the tooltip in the zooming toolbar.
- [BorderOpacity](#) - To set the opacity in the border of the tooltip in the zooming toolbar.
- [BorderWidth](#) - To set the width of the border of the tooltip in the zooming toolbar.
- [FontColor](#) - To set the text color in the tooltip of the zooming toolbar.
- [FontFamily](#) - To set the font family in the tooltip of the zooming toolbar.
- [FontStyle](#) - To set the font style in the tooltip of the zooming toolbar.
- [FontWeight](#) - To set the font weight in the tooltip of the zooming toolbar.
- [FontSize](#) - To set the font size in the tooltip of the zooming toolbar.
- [FontOpacity](#) - To set the font opacity in the tooltip of the zooming toolbar.

### ASPX-CS

```

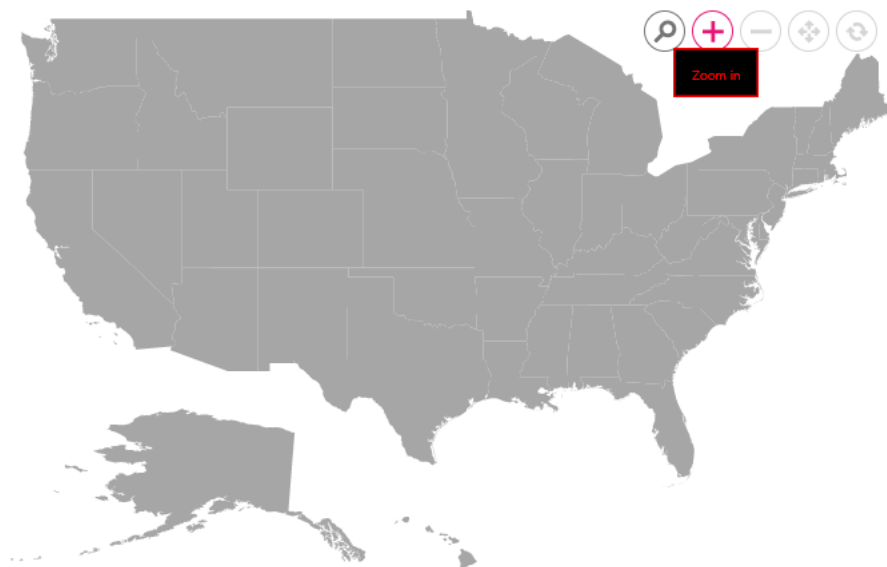
@using Syncfusion.Blazor.Maps
<SfMaps>

```

```

<MapsZoomSettings Enable="true">
  <MapsZoomToolBarSettings>
    <MapsZoomToolBarButton ToolbarItems="new List<ToolBarItem>() {
      ToolBarItem.Zoom,
      ToolBarItem.ZoomIn, ToolBarItem.ZoomOut, ToolBarItem.Pan, ToolBarItem.Reset
    }"></MapsZoomToolBarButton>
    <MapsZoomToolBarTooltipSettings Visible="true" BorderColor="red"
      BorderOpacity="1" BorderWidth="3" Fill="black" FontColor="red"
      FontFamily="Segoe UI"
      FontOpacity="1" FontSize="12px" FontStyle="normal"
      FontWeight="normal"></MapsZoomToolBarTooltipSettings>
  </MapsZoomToolBarSettings>
</MapsZoomSettings>
<MapsLayers>
  <MapsLayer ShapeData='new {dataOptions
    ="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
  </MapsLayer>
</MapsLayers>
</SfMaps>

```



### Pinch zooming

To enable or disable the pinch zooming, use the [PinchZooming](#) property in [MapsZoomSettings](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings Enable="true" PinchZooming="true"></MapsZoomSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
    </MapsLayer>
  </MapsLayers>
</SfMaps>

```

### Single-click zooming

To enable or disable the single-click zooming using mouse, use the [ZoomOnClick](#) property in [MapsZoomSettings](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings Enable="true" ZoomOnClick="true"></MapsZoomSettings>
  <MapsLayers>
    <MapLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
    </MapLayer>
  </MapsLayers>
</SfMaps>
```

### Double-click zooming

To enable or disable the double-click zooming using mouse, use the [DoubleClickZoom](#) property in [MapsZoomSettings](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings Enable="true" DoubleClickZoom="true"></MapsZoomSettings>
  <MapsLayers>
    <MapLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
    </MapLayer>
  </MapsLayers>
</SfMaps>
```

### Mouse wheel zooming

To enable or disable mouse wheel zooming, use the [MouseWheelZoom](#) property in [MapsZoomSettings](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings Enable="true" MouseWheelZoom="true"></MapsZoomSettings>
  <MapsLayers>
    <MapLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
    </MapLayer>
  </MapsLayers>
</SfMaps>
```

### Selection zooming

To enable or disable selection zooming, use the [EnableSelectionZooming](#) property in [MapsZoomSettings](#). The [EnablePanning](#) property must be set to **false** to enable the selection zooming in Maps.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
```

```
<SfMaps>
<MapsZoomSettings Enable="true" EnableSelectionZooming="true"
EnablePanning="true" Toolbars='new string[]{"Zoom", "ZoomIn", "ZoomOut",
"Pan", "Reset" }'></MapsZoomSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
```

#### Setting minimum and maximum values for zoom factor

The zoom factor range can be adjusted using the [MinZoom](#) and [MaxZoom](#) properties in [MapsZoomSettings](#). The [MinZoom](#) value is set to 1 by default, and the [MaxZoom](#) value is set to 10.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsZoomSettings Enable="true" MinZoom="2" MaxZoom="9"></MapsZoomSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
```

#### Zooming with animation

To zoom in or zoom out the Maps with animation, use the [AnimationDuration](#) property in [MapsLayer](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsZoomSettings Enable="true">
</MapsZoomSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string"
AnimationDuration="500">
</MapsLayer>
</MapsLayers>
</SfMaps>
```

#### Selection

Each shape in the Maps can be selected and deselected during interaction with the shapes. Selection is enabled by setting the [Enable](#) property of [MapsSelectionSettings](#) to **true**.

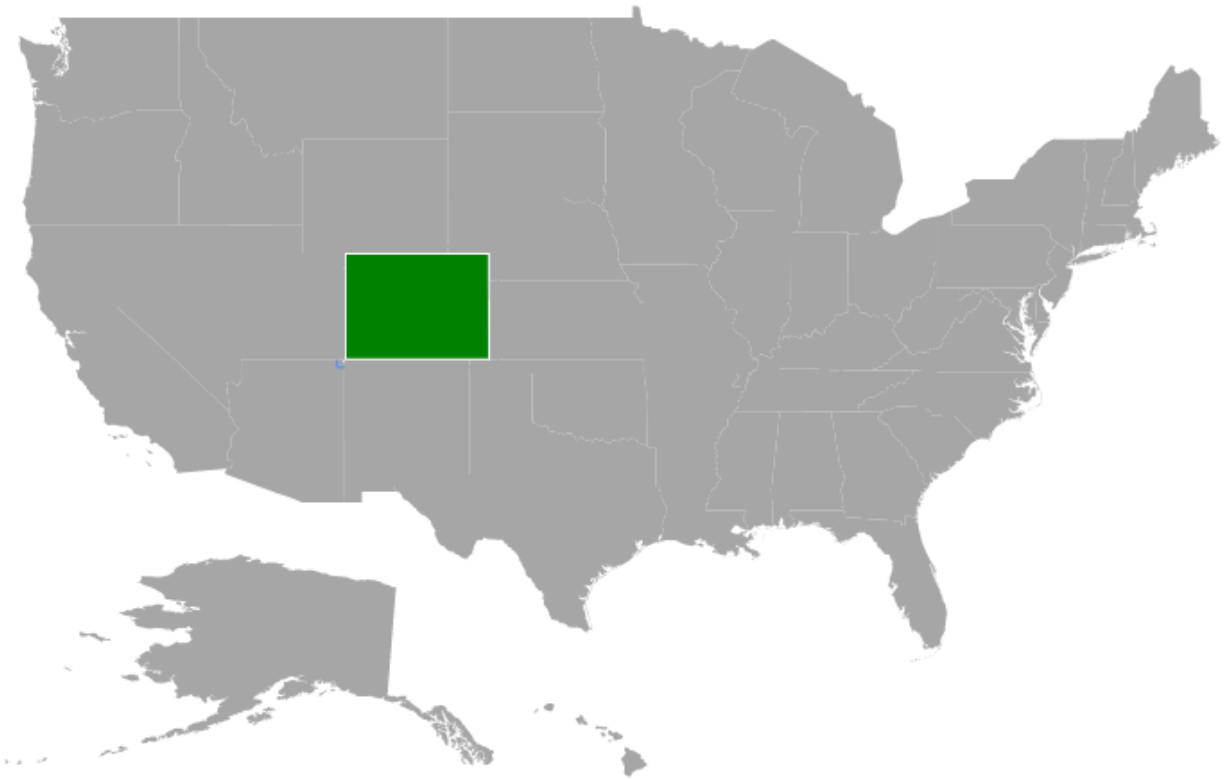
The following properties and class are available to customize the selection of Maps elements such as shapes, bubbles and markers.

- [MapsLayerSelectionBorder](#) - To customize the color and width of the border of which element is selected in Maps.
- [Fill](#) - Applies the color for the element that is selected.

- [Opacity](#) - To customize the transparency for the element that is selected.
- [EnableMultiSelect](#) - To enable or disable the selection for multiple shapes or markers or bubbles in the Maps.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
    ="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      <MapsLayerSelectionSettings Enable="true" Fill="green">
        <MapsLayerSelectionBorder Color="White"
        Width="2"></MapsLayerSelectionBorder>
      </MapsLayerSelectionSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```

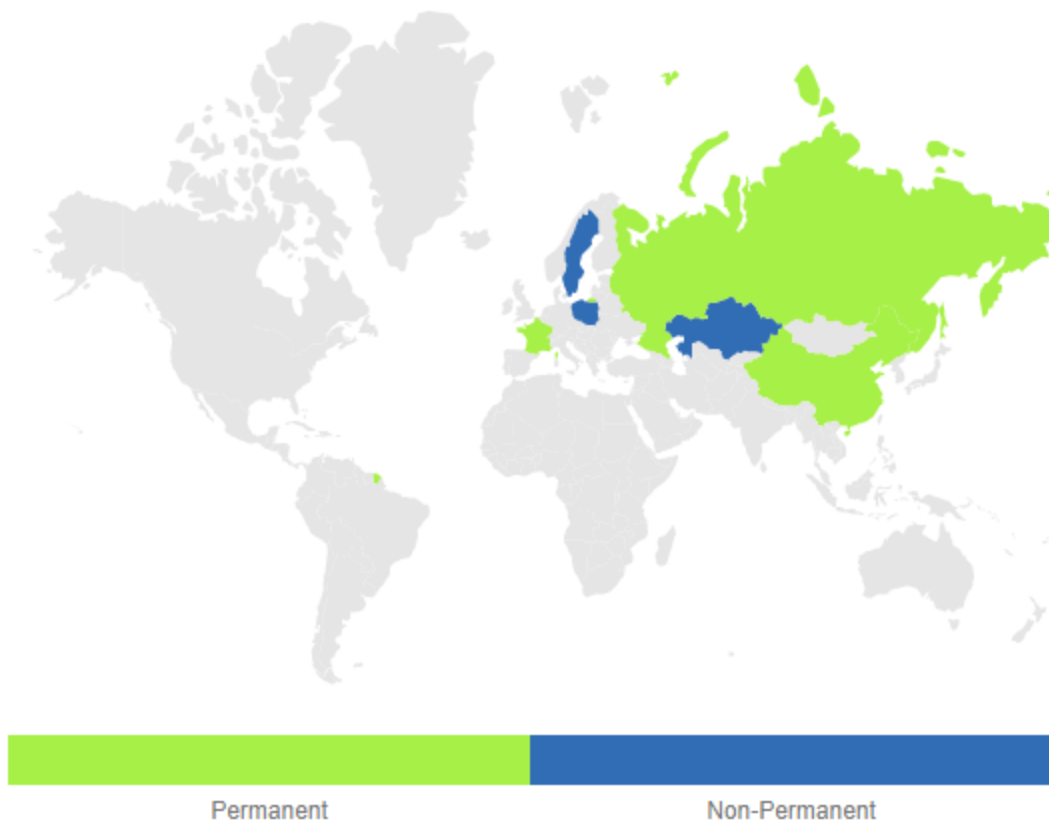


By tapping on the specific legend, the shapes which are bounded to the selected legend is also selected and vice versa.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLegendSettings Visible="true"
  Mode="LegendMode.Interactive"></MapsLegendSettings>
```

```
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapeDataPath="Country"
DataSource="CouncilMemberdetails" ShapePropertyPath='new string[] {"name"}'
TValue="UNCouncil">
<MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Membership">
<MapsShapeColorMappings>
<MapsShapeColorMapping Value="Permanent" Color='new string[] {"#D84444"}' />
<MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
{"#316DB5"}' />
</MapsShapeColorMappings>
</MapsShapeSettings>
<MapsLayerSelectionSettings Enable="true" Fill="#a7f047">
<MapsLayerSelectionBorder Color="White"
Width="2"></MapsLayerSelectionBorder>
</MapsLayerSelectionSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class UNCouncil
{
public string Country { get; set; }
public string Membership { get; set; }
};
private List<UNCouncil> CouncilMemberdetails = new List<UNCouncil>{
new UNCouncil { Country= "China", Membership= "Permanent" },
new UNCouncil { Country= "France",Membership= "Permanent" },
new UNCouncil { Country= "Russia",Membership= "Permanent" },
new UNCouncil { Country= "Kazakhstan",Membership= "Non-Permanent" },
new UNCouncil { Country= "Poland",Membership= "Non-Permanent" },
new UNCouncil { Country= "Sweden",Membership= "Non-Permanent" }
};
}
```



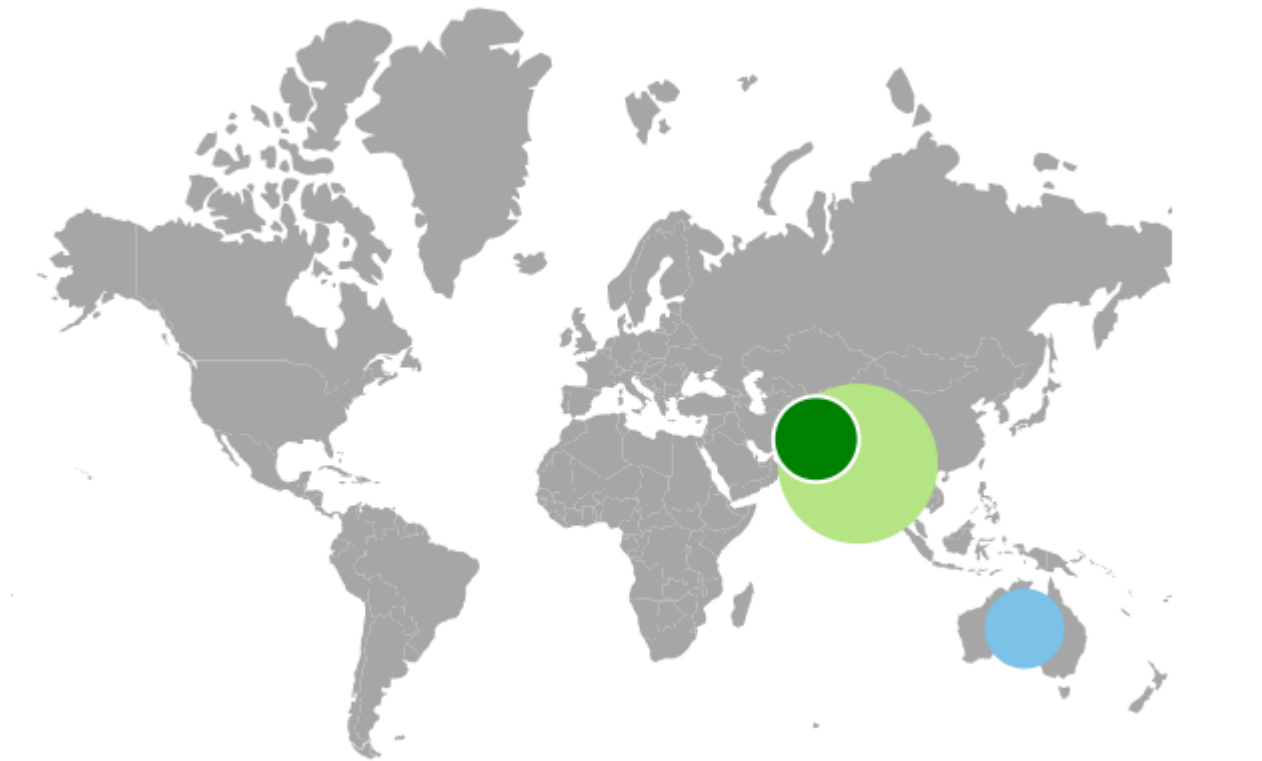
#### Enable selection for bubbles

To enable the selection for bubbles in Maps, set the [MapsBubbleSelectionSettings](#) in [MapsBubble](#) and set the [Enable](#) property of [MapsBubbleSelectionSettings](#) as **true**.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      ShapeDataPath="Name"
      ShapePropertyPath='new string[] { "name"}' TValue="Country">
      @* To add bubbles based on population count *@
      <MapsBubbleSettings>
        <MapsBubble Visible="true" ValuePath="Population" ColorValuePath="Color"
          MinRadius=20 MaxRadius=40
          DataSource="PopulationDetails" TValue="Country">
          <MapsBubbleSelectionSettings Enable="true"
            Fill="green"></MapsBubbleSelectionSettings>
        </MapsBubble>
      </MapsBubbleSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code{
  public class Country
  {
    public string Name { get; set; }
```

```
public double Population { get; set; }
public string Color { get; set; }
};
private List<Country> PopulationDetails = new List<Country> {
    new Country
    {
        Name = "United States", Population = 325020000, Color = "#b5e485"
    },
    new Country
    {
        Name = "Russia", Population = 142905208, Color = "#7bc1e8"
    },
    new Country
    {
        Name = "India", Population = 1198003000, Color = "#df819c"
    }
};
```



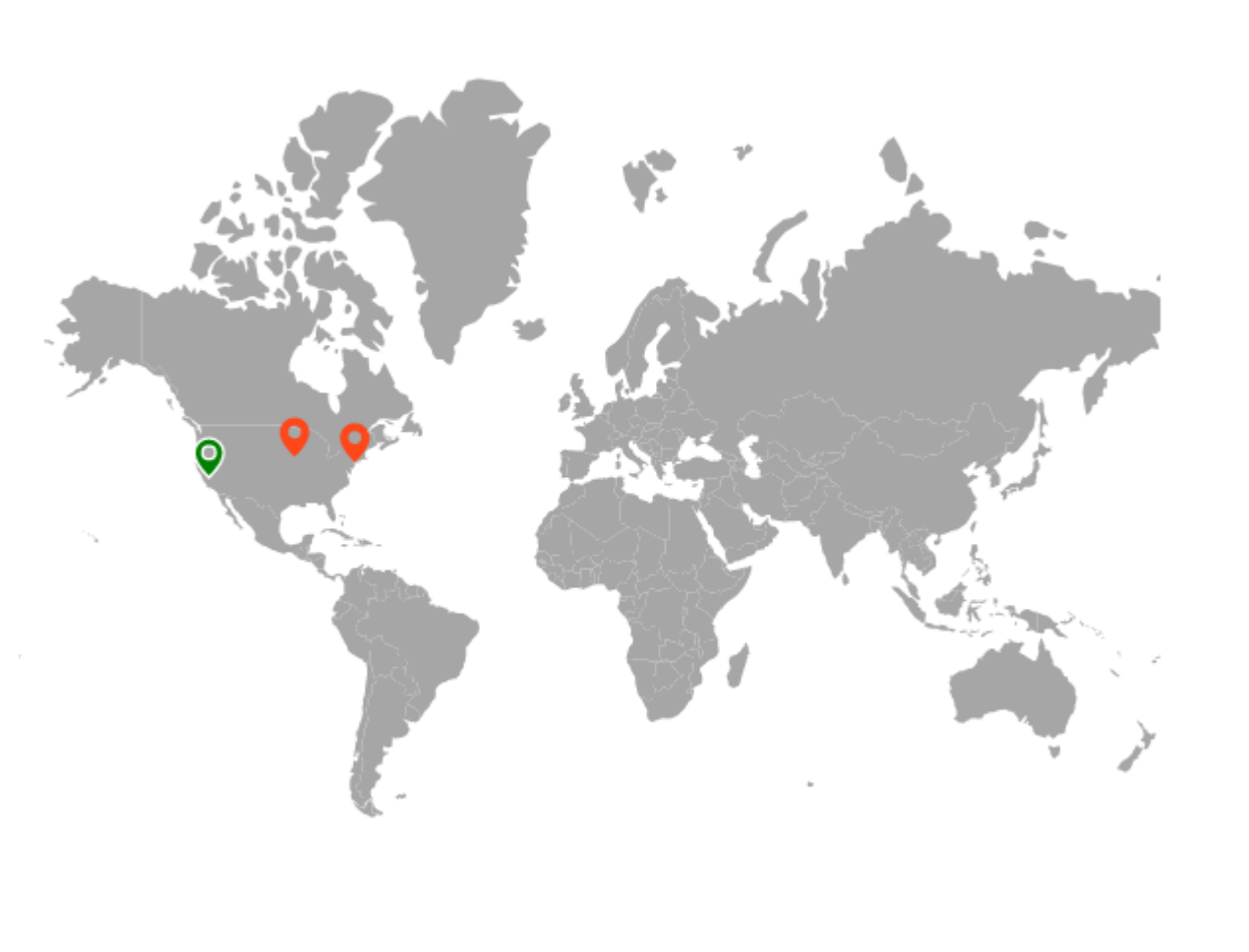


*Enable selection for markers*

To enable the selection for markers in Maps, set the [MapsMarkerSelectionSettings](#) in the [MapsMarkerSettings](#) and set the [Enable](#) property of the [MapsLayerSelectionSettings](#) as **true**.

**ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" DataSource="California" Height="25" Width="15"
          TValue="City">
          <MapsMarkerSelectionSettings Fill="green"
            Enable="true"></MapsMarkerSelectionSettings>
        </MapsMarker>
        <MapsMarker Visible="true" DataSource="NewYork" Height="25" Width="15"
          TValue="City">
          <MapsMarkerSelectionSettings Fill="green"
            Enable="true"></MapsMarkerSelectionSettings>
        </MapsMarker>
        <MapsMarker Visible="true" DataSource="Iowa" Height="25" Width="15"
          TValue="City">
          <MapsMarkerSelectionSettings Fill="green"
            Enable="true"></MapsMarkerSelectionSettings>
        </MapsMarker>
      </MapsMarkerSettings>
    <MapsShapeSettings Fill="lightgray"></MapsShapeSettings>
  </MapsLayer>
</MapsLayers>
</SfMaps>
@code {
  public class City
  {
    public double Latitude { get; set; }
    public double Longitude { get; set; }
  };
  public List<City> California = new List<City> {
    new City { Latitude=35.145083,Longitude=-117.960260 }
  };
  public List<City> NewYork = new List<City> {
    new City { Latitude=40.724546, Longitude=-73.850344 }
  };
  public List<City> Iowa = new List<City> {
    new City { Latitude= 41.657782, Longitude=-91.533857 }
  };
}
```



#### *Public method for the shape selection*

The [ShapeSelection](#) method can be used to select each shape in the Maps. **layerIndex**, **propertyName**, **shapeDataPath**, and **isSelected** boolean value to select or deselect the shape are the input parameters for this method.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<div>
<SfMaps @ref="mapsref">
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
<MapsLayerSelectionSettings Enable="true" Fill="green">
<MapsLayerSelectionBorder Color="white"
Width="2"></MapsLayerSelectionBorder>
</MapsLayerSelectionSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
</div>
<button id="select" @onclick="Select">select</button>
```

```
<button id="unselect" @onclick="Unselect">unselect</button>
@code{
    SfMaps mapsref;
    public void Select() {
        mapsref.ShapeSelection(0, "continent", "Asia", true);
    }
    public void Unselect() {
        mapsref.ShapeSelection(0, "continent", "Asia", false);
    }
}
```



#### Initial shape selection

The shape is initially selected using the [MapsInitialShapeSelection](#), and the values are mapped to the [ShapePath](#) and [ShapeValue](#).

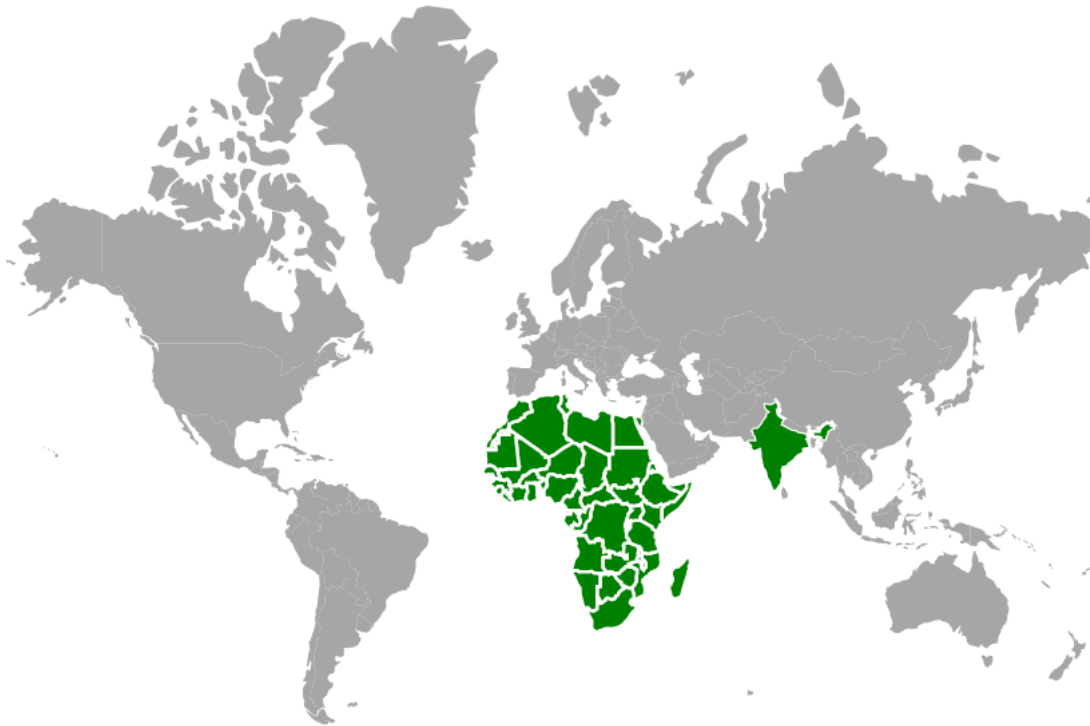
#### ASPX-CS

```
<SfMaps>
<MapsLayers>
<MapLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
<MapLayerSelectionSettings Enable="true" Fill="green">
<MapLayerSelectionBorder Color="white"
Width="2"></MapLayerSelectionBorder>
</MapLayerSelectionSettings>
<MapsInitialShapeSelectionSettings>
<MapsInitialShapeSelection ShapePath="continent"
ShapeValue="Africa"></MapsInitialShapeSelection>
```

```

<MapsInitialShapeSelection ShapePath="name"
ShapeValue="India"></MapsInitialShapeSelection>
</MapsInitialShapeSelectionSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>

```



#### Initial marker selection

Using the [InitialMarkerSelection](#), the marker shape can be selected initially. Markers render based on the [Latitude](#) and [Longitude](#) values.

#### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
<MapsMarkerSettings>
<MapsMarker Visible="true" DataSource="Cities" Height="25" Width="15"
TValue="City">
<MapsMarkerSelectionSettings Fill="green"
Enable="true"></MapsMarkerSelectionSettings>
<InitialMarkerSelectionSettings>
<InitialMarkerSelection Latitude="35.145083" Longitude="-
117.960260"></InitialMarkerSelection>
</InitialMarkerSelectionSettings>
</MapsMarker>
</MapsMarkerSettings>
<MapsShapeSettings Fill="lightgray"></MapsShapeSettings>
</MapsLayer>

```

```
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
};
private List<City> Cities = new List<City> {
new City {Latitude=35.145083,Longitude=-117.960260, Name= "Californiya"},
new City { Latitude=40.724546, Longitude=-73.850344, Name="New York"},
new City {Latitude= 41.657782, Longitude=-91.533857, Name="Iowa"}
};
}
```



### Highlight

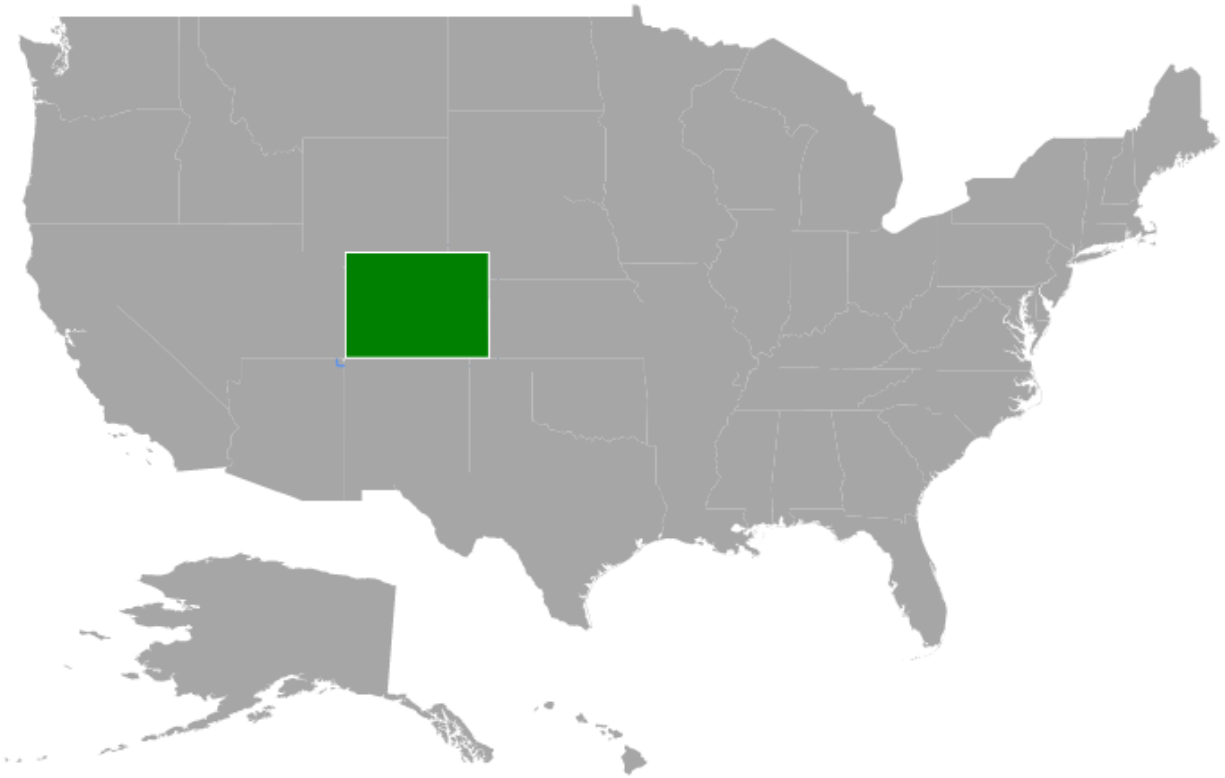
Each shape in the Maps can be highlighted during mouse hover on the Maps elements such as shapes, bubbles, markers and legends. Highlight is enabled by setting the [Enable](#) property of [MapsHighlightSettings](#) to **true**.

The following properties and classes are available to customize the highlight of Maps elements such as shapes, bubbles and markers.

- [Fill](#) - Applies the color for the element that is highlighted.
- [Opacity](#) - To customize the transparency for the element that is highlighted.
- [MapsLayerHighlightBorder](#) - To customize the color and width of the border of the layer when that is highlighted.
- [MapsBubbleHighlightBorder](#) - To customize the color and width of the border of the bubble when that is highlighted.
- [MapsMarkerHighlightBorder](#) - To customize the color and width of the border of the marker when that is highlighted.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      <MapsLayerHighlightSettings Enable="true" Fill="green">
        <MapsLayerHighlightBorder Color="white"
Width="2"></MapsLayerHighlightBorder>
      </MapsLayerHighlightSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```

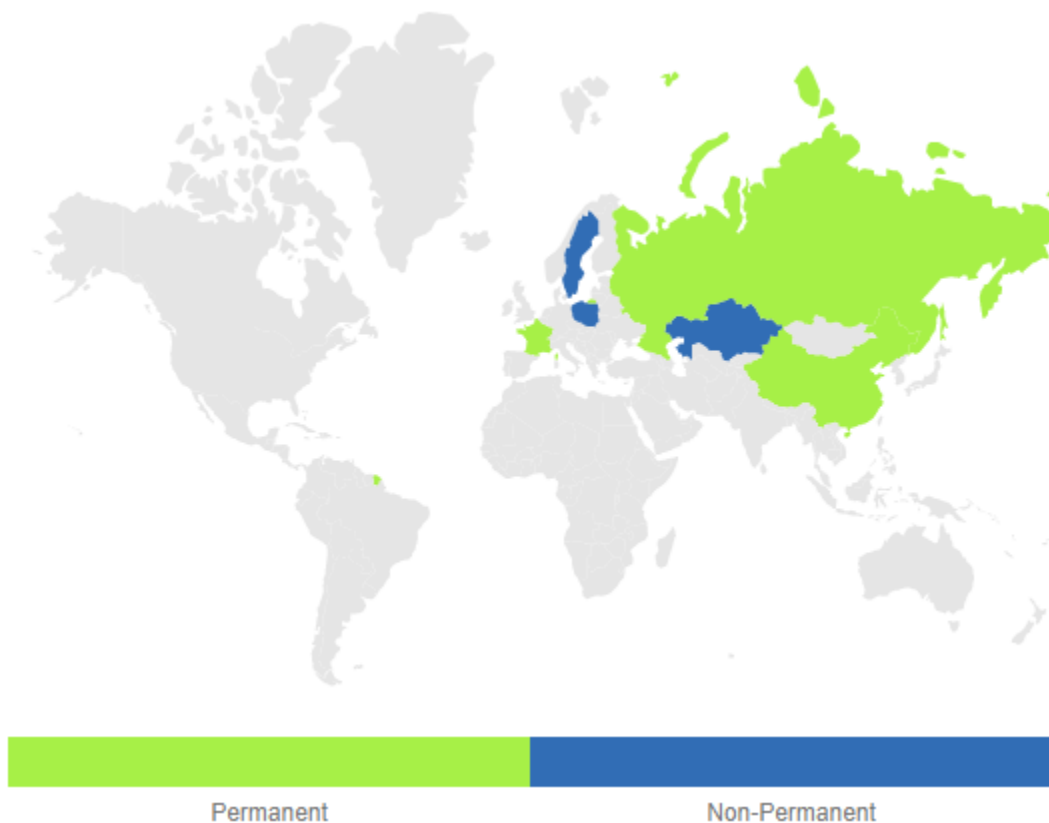


Hovering on the specific legend, the shapes which are bounded to the selected legend is also highlighted and vice versa.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLegendSettings Visible="true"
  Mode="LegendMode.Interactive"></MapsLegendSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
    ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
    ShapeDataPath="Country"
    DataSource="CouncilMemberDetails" ShapePropertyPath='new string[] {"name"}'
    TValue="UNCouncil">
      <MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Membership">
        <MapsShapeColorMappings>
          <MapsShapeColorMapping Value="Permanent" Color='new string[] {"#D84444"}' />
          <MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
          {"#316DB5"}' />
        </MapsShapeColorMappings>
      </MapsShapeSettings>
      <MapsLayerHighlightSettings Enable="true" Fill="#a7f047">
        <MapsLayerHighlightBorder Color="White"
        Width="2"></MapsLayerHighlightBorder>
      </MapsLayerHighlightSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```

```
@code{
public class UNCouncil
{
public string Country { get; set; }
public string Membership { get; set; }
};
private List<UNCouncil> CouncilMemberDetails = new List<UNCouncil>{
new UNCouncil { Country= "China", Membership= "Permanent"},
new UNCouncil { Country= "France",Membership= "Permanent" },
new UNCouncil { Country= "Russia",Membership= "Permanent"},
new UNCouncil { Country= "Kazakhstan",Membership= "Non-Permanent"},
new UNCouncil { Country= "Poland",Membership= "Non-Permanent"},
new UNCouncil { Country= "Sweden",Membership= "Non-Permanent"}
};
}
```



#### *Enable highlight for bubbles*

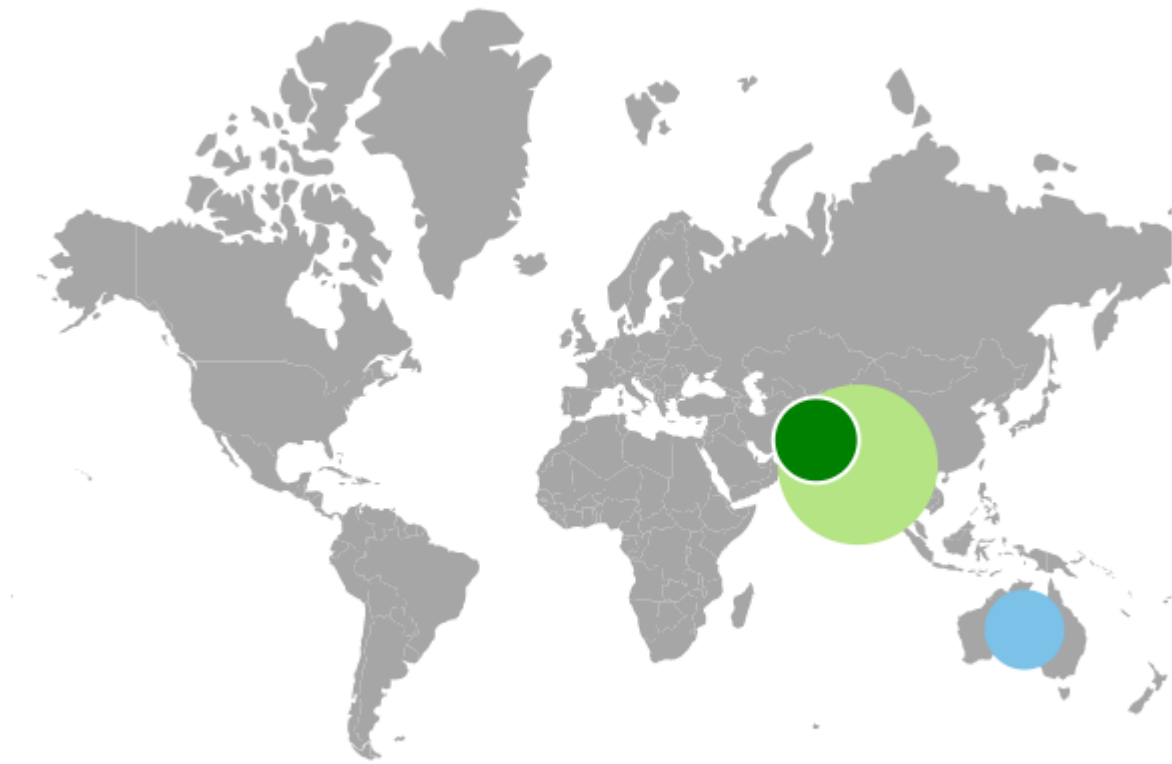
To enable the highlight for bubbles in Maps, set the [MapsBubbleHighlightSettings](#) in [MapsBubble](#) and set the [Enable](#) property of [MapsBubbleHighlightSettings](#) as **true**.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'>
```



```
ShapeDataPath="Name" ShapePropertyPath='new string[] { "name" }'  
TValue="Country">  
@* To add bubbles based on population count *@  
<MapsBubbleSettings>  
<MapsBubble Visible="true" ValuePath="Population" ColorValuePath="Color"  
MinRadius=20 MaxRadius=40  
DataSource="PopulationDetails" TValue="Country">  
<MapsBubbleHighlightSettings Enable="true" Fill="green">  
<MapsBubbleHighlightBorder Width="2"  
Color="orange"></MapsBubbleHighlightBorder>  
</MapsBubbleHighlightSettings>  
</MapsBubble>  
</MapsBubbleSettings>  
</MapsLayer>  
</MapsLayers>  
</SfMaps>  
@code{  
public class Country  
{  
public string Name { get; set; }  
public double Population { get; set; }  
public string Color { get; set; }  
};  
private List<Country> PopulationDetails = new List<Country> {  
new Country  
{  
Name = "United States", Population = 325020000, Color = "#b5e485"  
},  
new Country  
{  
Name = "Russia", Population = 142905208, Color = "#7bc1e8"  
},  
new Country  
{  
Name="India", Population=1198003000, Color = "#df819c"  
}  
};  
}
```



### *Enable highlight for markers*

To enable the highlight for markers in Maps, set the [MapsMarkerHighlightSettings](#) in [MapsMarker](#) and set the [Enable](#) property of [MapsMarkerHighlightSettings](#) as **true**.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" DataSource="Cities" Height="25" Width="15"
TValue="City">
          <MapsMarkerSelectionSettings Fill="green"
Enable="true"></MapsMarkerSelectionSettings>
        </MapsMarker>
      </MapsMarkerSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```

```
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
};
private List<City> Cities = new List<City> {
new City { Latitude=35.145083,Longitude=-117.960260, Name= "California" },
new City { Latitude=40.724546, Longitude=-73.850344, Name="New York" },
new City { Latitude= 41.657782, Longitude=-91.533857, Name="Iowa" }
};
}
```



### Tooltip

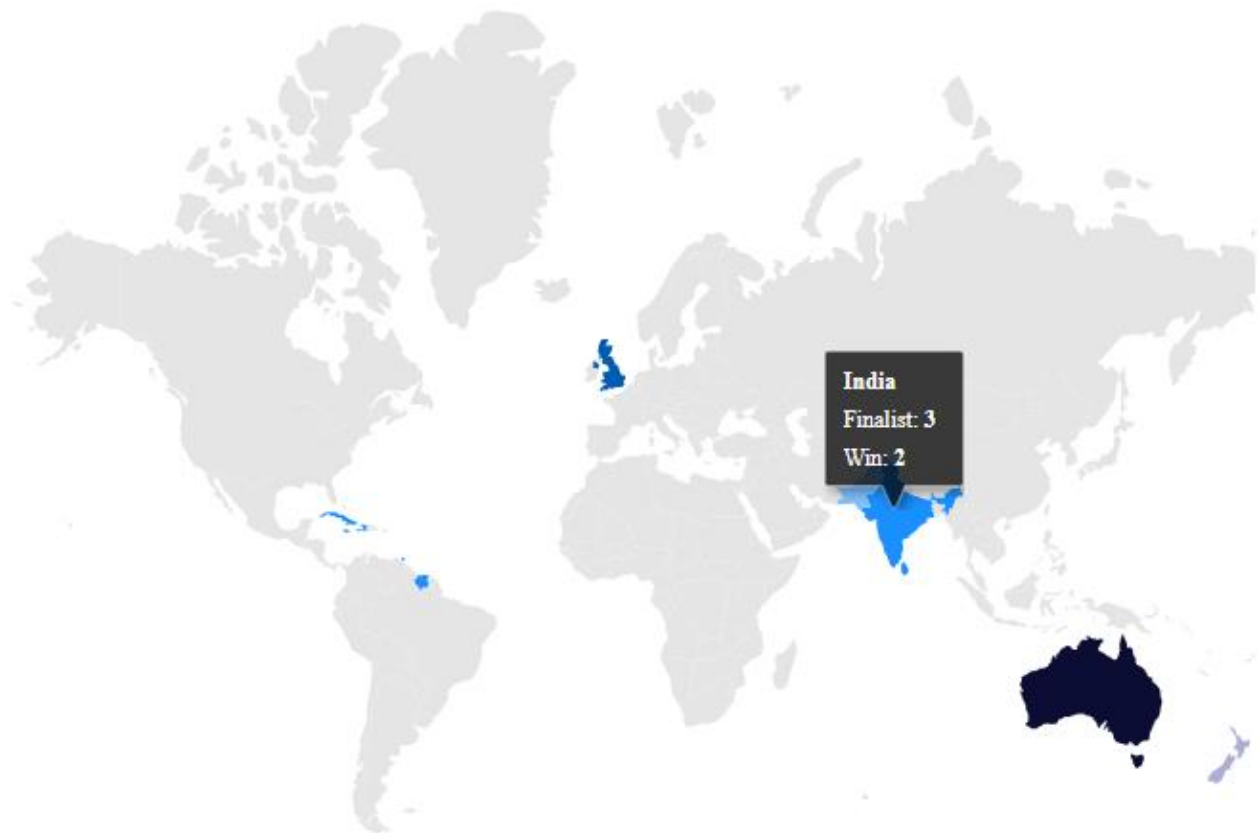
On mouse over or touch end event, the tooltip is used to get more information about the layer, bubble, or marker. It can be enabled separately for layer or bubble or marker by using the [Visible](#) property of [MapsLayerTooltipSettings](#) or [MapsBubbleTooltipSettings](#) or [MapsMarkerTooltipSettings](#) respectively. The [TooltipDisplayMode](#) property is used to change the display mode of the tooltip in Maps. Following

display modes of tooltip are available in the Maps component. By default, [TooltipDisplayMode](#) is set to **MouseMove**.

- MouseMove
- Click
- DoubleClick

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
      <MapsLayerTooltipSettings Visible="true" ValuePath="name">
    </MapsLayerTooltipSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```



### Customization

The following properties are available to customize the tooltip of the Maps component.

- [Fill](#) - Applies the color of the tooltip in layers, markers, and bubbles of Maps.
- [Format](#) - To customize the format of the tooltip in layers, markers, and bubbles of Maps.

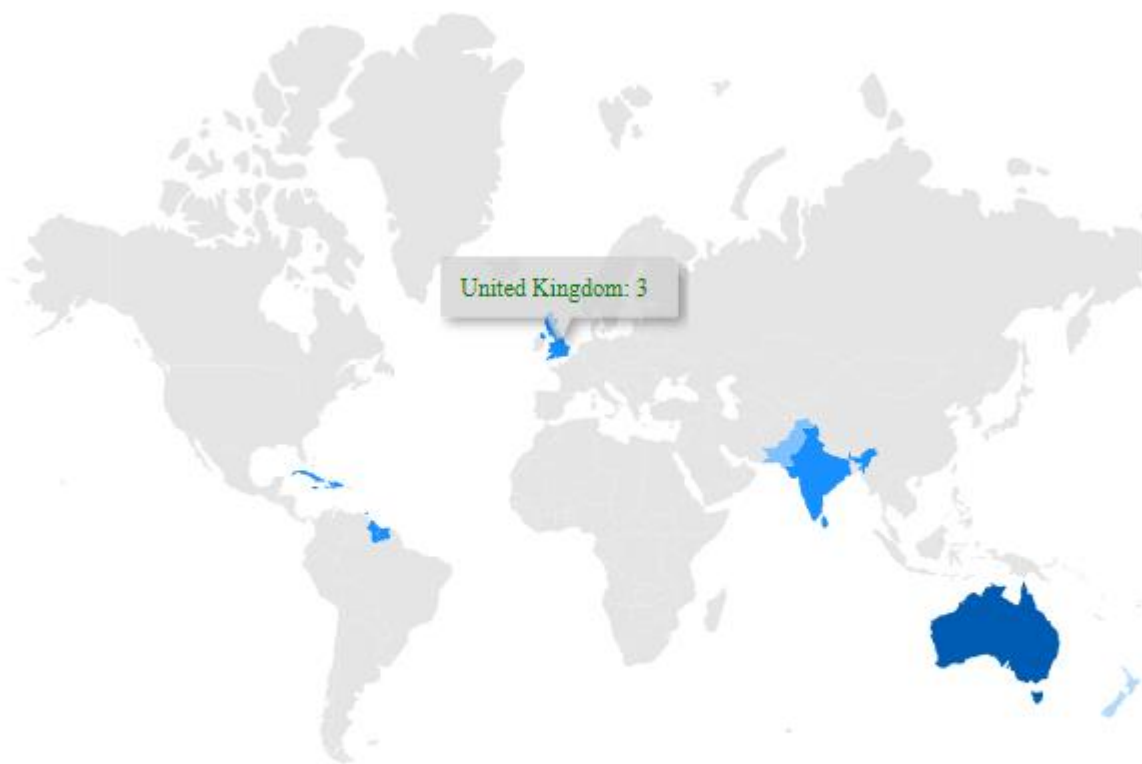
**ASPX-CS**

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapeDataPath="Name"
ShapePropertyPath='new string[] { "name" }' DataSource='PerformanceReport'
TValue="Country">
<MapsLayerTooltipSettings Visible="true" ValuePath="CountryName"
Format="<b>${CountryName}</b><br>Finalist: <b>${Winner}</b><br>Win:
<b>${Finalist}</b>">
</MapsLayerTooltipSettings>
<MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Finalist">
<MapsShapeColorMappings>
<MapsShapeColorMapping Value="1" Color='new string[]
{"#acaed8"}'></MapsShapeColorMapping>
<MapsShapeColorMapping Value="2" Color='new string[]
{"#80c1ff"}'></MapsShapeColorMapping>
<MapsShapeColorMapping Value="3" Color='new string[]
{"#1a90ff"}'></MapsShapeColorMapping>
<MapsShapeColorMapping Value="4" Color='new string[]
{"#005cb3"}'></MapsShapeColorMapping>
<MapsShapeColorMapping Value="7" Color='new string[]
{"#0b0d35"}'></MapsShapeColorMapping>
</MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public List<Country> PerformanceReport = new List<Country> {
new Country { CountryName="India", Name="India", Finalist="3", Winner="2" },
new Country { CountryName="United Kingdom", Name="United Kingdom",
Finalist="4", Winner="1" },
new Country { CountryName="Australia", Name="Australia", Finalist="7",
Winner="5" },
new Country { CountryName="Sri Lanka", Name="Sri Lanka", Finalist="3",
Winner="1"},
new Country { CountryName="Pakistan", Name="Pakistan", Finalist="2",
Winner="1" },
new Country { CountryName="New Zealand", Name="New Zealand", Finalist="1",
Winner="0"},
new Country { CountryName="West Indies", Name="Dominican Rep", Finalist="3",
Winner="2"},
new Country { CountryName="West Indies", Name="Cuba", Finalist="3",
Winner="2"},
new Country { CountryName="West Indies", Name="Jamaica", Finalist="3",
Winner="2"},
new Country { CountryName="West Indies", Name="Haiti", Finalist="3",
Winner="2"},
new Country { CountryName="West Indies",Name="Gayana", Finalist="3",
Winner="2"},
new Country { CountryName="West Indies", Name="Suriname", Finalist="3",
Winner="2"},

```

```
new Country { CountryName="West Indies", Name="Trinidad and Tobago",  
Finalist="3", Winner="2"}  
};  
public class Country  
{  
    public string Name { get; set; }  
    public string Winner { get; set; }  
    public string Finalist { get; set; }  
    public string CountryName { get; set; }  
}  
}
```



#### *Tooltip template*

The HTML element can be rendered in the tooltip of the Maps using the [TooltipTemplate](#) property of the [MapsLayerTooltipSettings](#).

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps  
<SfMaps>
```

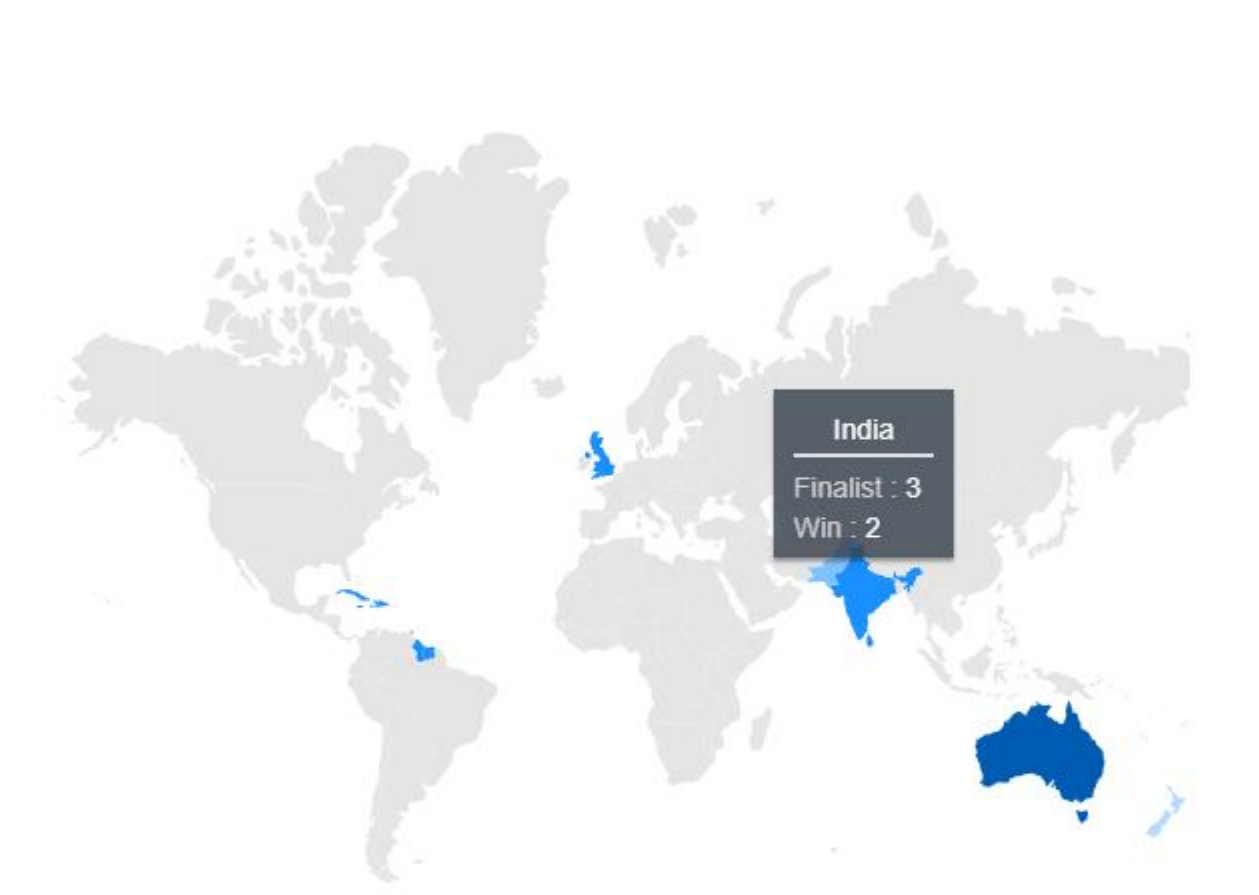
```

<MapsTitleSettings Text="Finalist in Cricket World Cup">
<MapsTitleTextStyle Size="16px" />
</MapsTitleSettings>
<MapsZoomSettings Enable="false" />
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
DataSource="@PerformanceReport" TValue="Country"
ShapePropertyPath="@ShapePropertyPath" ShapeDataPath="Name">
<MapsLayerTooltipSettings Visible="true" ValuePath="name">
<TooltipTemplate>
@{
var Data = context as Country;
<div id="template">
<div class="toolback">
<div class="listing2">
<center>
@Data.Name
</center>
</div>
<hr style="margin-top: 2px;margin-bottom:5px;border:0.5px solid #DDDDDD">
<div>
<span class="listing1">Finalist : </span><span
class="listing2">@Data.Winner</span>
</div>
<div>
<span class="listing1">Win : </span><span
class="listing2">@Data.Runner</span>
</div>
</div>
</div>
}
</TooltipTemplate>
</MapsLayerTooltipSettings>
<MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Winner">
<MapsShapeColorMappings>
<MapsShapeColorMapping Value="1" Color="@ShapeColorOne" />
<MapsShapeColorMapping Value="2" Color="@ShapeColorTwo" />
<MapsShapeColorMapping Value="3" Color="@ShapeColorThree" />
<MapsShapeColorMapping Value="7" Color="@ShapeColorFour" />
</MapsShapeColorMappings>
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public string[] ShapePropertyPath = { "name" };
public string[] ShapeColorOne = { "#b3daff" };
public string[] ShapeColorTwo = { "#80c1ff" };
public string[] ShapeColorThree = { "#1a90ff" };
public string[] ShapeColorFour = { "#005cb3" };
public List<Country> PerformanceReport = new List<Country> {
new Country { Name="India", Winner="3", Runner="2", City="India" },
new Country { Name="Dominican Rep.", Winner="3", Runner="2", City="West
Indies" },
new Country { Name="Cuba", Winner="3", Runner="2", City="West Indies" },
new Country { Name="Jamaica", Winner="3", Runner="2", City="West Indies" },

```

```
new Country { Name="Haiti", Winner="3", Runner="2", City="West Indies" },
new Country { Name="Guyana", Winner="3", Runner="2", City="West Indies" },
new Country { Name="Suriname", Winner="3", Runner="2", City="West Indies" },
new Country { Name="Trinidad and Tobago", Winner="3", Runner="2", City="West Indies" },
new Country { Name="Sri Lanka", Winner="3", Runner="1", City="Sri Lanka" },
new Country { Name="United Kingdom", Winner="3", Runner="0", City="England"
},
new Country { Name="Pakistan", Winner="2", Runner="1", City="Pakistan" },
new Country { Name="New Zealand", Winner="1", Runner="0", City="New Zealand"
},
new Country { Name="Australia", Winner="7", Runner="5", City="Australia" }
};
public class Country
{
public string Name { get; set; }
public string Winner { get; set; }
public string Runner { get; set; }
public string City { get; set; }
}
}
<style>
.toolbox {
width: 100px;
border-radius: 4px;
border: 1px #abb9c6;
background: rgba(53, 63, 76, 0.90);
box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.40);
padding-bottom: 5px;
padding-top: 10px;
padding-left: 10px;
padding-right: 10px
}
.listing1 {
font-size: 13px;
color: #cccccc
}
.listing2 {
font-size: 13px;
color: #ffffff;
font-weight: 500;
}
</style>
```





See also

- [Change center position on zooming](#)

## Print and export in Blazor Maps Component

### Print

The rendered Maps can be printed directly from the browser by calling the [Print](#) method. To use the print functionality, set the [AllowPrint](#) property to **true**.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<button @onclick="PrintMap">Print</button>
<SfMaps @ref="maps" AllowPrint="true">
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
      <MapsLayerTooltipSettings Visible="true" ValuePath="name">
      </MapsLayerTooltipSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
```

```
</MapsLayers>
</SfMaps>
@code {
    SfMaps maps;
    void PrintMap()
    {
        // using Maps component reference call 'Print' method
        this.maps.Print();
    }
}
```

Print



## Export

### Image Export

To use the image export functionality, set the [AllowImageExport](#) property as **true**. The rendered Maps can be exported as an image using the [Export](#) method. The method requires two parameters: image type and file name. The Maps can be exported as an image in the following formats.

- JPEG
- PNG
- SVG

## ASPX-CS

```
@using Syncfusion.Blazor.Maps
<button @onclick="ExportMap">Export</button>
```

```
<SfMaps @ref="Maps" AllowImageExport="true">
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
  SfMaps Maps;
  void ExportMap()
  {
    this.Maps.Export(ExportType.PNG, "Maps");
  }
}
```

Export



#### PDF Export

To use the PDF export functionality, set the [AllowPdfExport](#) property as **true**. The rendered Maps can be exported as PDF using the [Export](#) method. The [Export](#) method requires three parameters: file type, file name and orientation of the PDF document. The orientation of the PDF document can be set as **0** or **1**. **0** indicates the **portrait** and **1** indicates **landscape**.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<button @onclick="ExportMap">Export</button>
<SfMaps @ref="Maps" AllowPdfExport="true">
  <MapsLayers>
```

```
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
SfMaps Maps;
void ExportMap()
{
this.Maps.Export(ExportType.PDF, "Maps", 0);
}
}
```

Export



#### *Exporting Maps as base64 string of the file*

The image can be exported as base64 string for the JPEG, PNG and PDF formats. The rendered Maps can be exported to image as base64 string using the [Export](#) method. The arguments that are required for this method is image type, file name, orientation of the exported PDF document which must be set as **null** for image export and **0** or **1** for the PDF export and finally **allowDownload** which should be set as **false** to return base64 string.

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<button @onclick="export">Export</button>
<SfMaps @ref="Maps" AllowPdfExport="true">
<MapsLayers>
```

```
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
SfMaps Maps;
string exportString;
public async Task export()
{
exportString = await this.Maps.Export(ExportType.PDF, "Maps", 0, false);
Console.WriteLine(exportString);
}
}
```

---

Add the below service in startup.cs file if the size of the Maps is too large.

```
services.AddServerSideBlazor().AddHubOptions(o => { o.MaximumReceiveMessageSize = 102400000; });
```

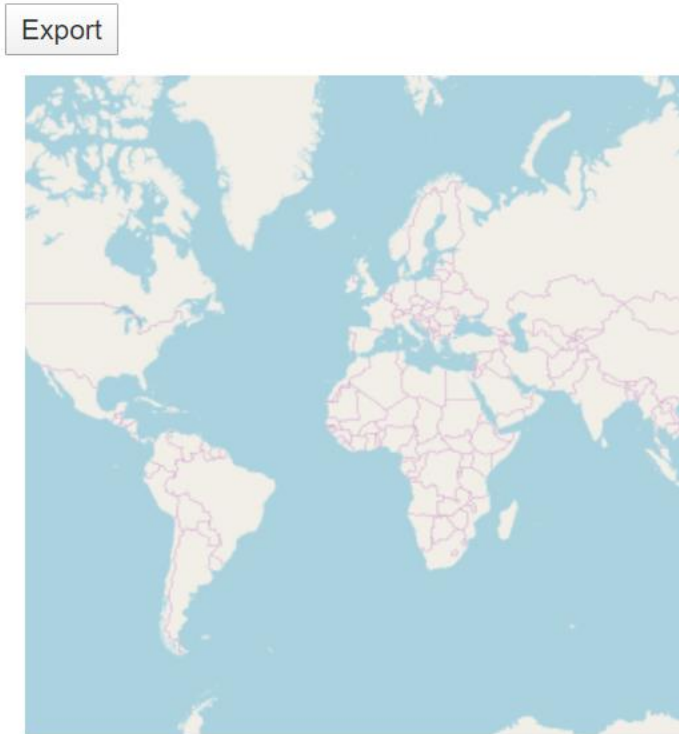
#### *Export the tile Maps*

The rendered Maps with providers such as OSM, Bing and other map providers can be exported using the [Export](#) method. It supports the following export formats.

- JPEG
- PNG
- PDF

#### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<button @onclick="ExportMap">Export</button>
<SfMaps @ref="Maps" AllowPdfExport="true" AllowImageExport="true">
<MapsLayers>
<MapsLayer
UrlTemplate="https://tile.openstreetmap.org/level/tileX/tileY.png"
TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
SfMaps Maps;
void ExportMap()
{
this.Maps.Export(ExportType.PNG, "OSM Map");
}
}
```



## State Persistence in Blazor Maps Component

### State Persistence

State persistence allows the Maps to retain the current model value in the browser cookies for state maintenance. This action is handled through the `enablePersistence` property which is set to **false** by default. When it is set to **true**, some of the Maps component model values will be retained even after refreshing the page.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps EnablePersistence ="true">
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
    </MapsLayer>
  </MapsLayers>
  <MapsZoomSettings Enable='true'></MapsZoomSettings>
</SfMaps>
```

## Accessibility in Blazor Maps Component

Maps provides built-in compliance with the [WAI-ARIA](#) specifications. The WAI-ARIA accessibility support is achieved through the attribute like `aria-label` in the SVG element. It helps to provide information about elements in a document for assistive technology. This attribute sets the text label with some default description for the following elements in Maps.

<!-- markdownlint-disable MD033 -->

Element	Default description
Maps container	Specifies the Maps component.
Maps title	Specifies the title of the Maps.
Maps subtitle	Specifies the sub-title of the Maps.
Legend title	Specifies the title of legend in the Maps.

To change this default description, use the [Description](#) property available in [MapsLegendTitle](#), [MapsTitleSettings](#), [MapsSubtitleSettings](#) and [SfMaps](#). It helps the screen reader to read for an assistive purpose.

### KeyBoard Navigation

All the Maps actions can be controlled via keyboard keys. The applicable key combinations and their relative functionalities are listed below for the appropriate UI features available in the component.

#### Interaction Keys | Description

**Tab** | Moves to the next focusable element on the map, such as the legend or shape.

**Shift + Tab** | Moves to the previous focusable element on the map, such as the legend or shape.

**+** | When zooming is enabled, zoom in operation can be performed.

**-** | When zooming is enabled, zoom out operation can be performed.

**Left arrow** | When zoomed in, the map can be scrolled to the left.

**Right arrow** | When zoomed in, the map can be scrolled to the right.

**Up arrow** | When zoomed in, the map can be scrolled upward.

**Down arrow** | When zoomed in, the map can be scrolled downward.

**R** | When zooming is enabled, reset operation can be performed.

**Enter** | The page can be navigated to the next and previous states in legend. Similarly, the selection can be made while navigating over the shape.

### Globalization in Blazor Maps Component

Maps provide support for internationalization for the below elements.

- Data label
- Tooltip

### Globalization

Globalization is the process of designing and developing a component that works in different cultures/locales. It can be provided to the Blazor server-side and client-side applications. Refer to [Blazor server-side](#) and [Blazor client-side](#) sections for configuring the globalization for the Maps component. It is used to globalize number, date, time values in

Maps component using [Format](#) property in the Maps component.

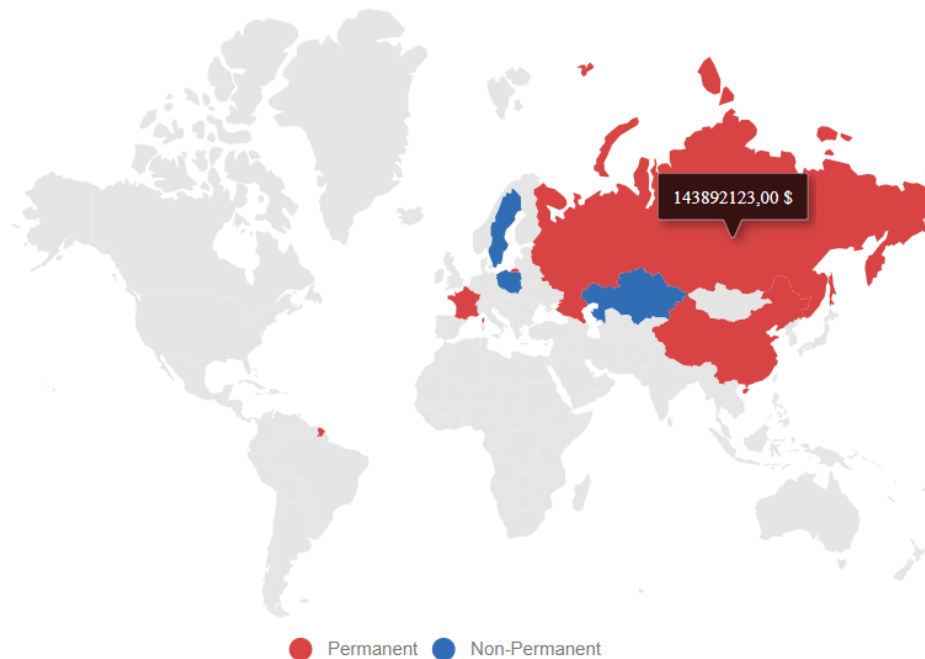
**ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps Format="n0">
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      DataSource="CountryData"
      ShapePropertyPath='new string[] { "name" }' ShapeDataPath="Country"
      TValue="MapDataSource" >
      <MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Membership">
        <MapsShapeColorMappings>
          <MapsShapeColorMapping Value="Permanent" Color='new string[] { "#D84444" }' />
          <MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
            { "#316DB5" }' />
        </MapsShapeColorMappings>
      </MapsShapeSettings>
      <MapsLayerTooltipSettings Visible="true"
        ValuePath="Population"></MapsLayerTooltipSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>

@code {
  public class MapDataSource
  {
    public string Country { get; set; }
    public string Membership { get; set; }
    public double Population { get; set; }
  };

  public List<MapDataSource> CountryData = new List<MapDataSource>{
    new MapDataSource { Country= "China", Membership= "Permanent",
      Population=20000 },
    new MapDataSource { Country= "France",Membership= "Permanent",
      Population=30000 },
    new MapDataSource { Country= "Russia",Membership= "Permanent",
      Population=40000 },
    new MapDataSource { Country= "Kazakhstan",Membership= "Non-Permanent",
      Population=50000 },
    new MapDataSource { Country= "Poland",Membership= "Non-Permanent",
      Population=60000 },
    new MapDataSource { Country= "Sweden",Membership= "Non-Permanent",
      Population=70000 }
  };
}
```





### Numeric Format

The numeric formats such as currency, percentage and so on can be displayed in the tooltip and data labels of the Maps using the [Format](#) property in the [SfMaps](#) class. In the below example, the tooltip is globalized to **German** culture. When setting the [UseGroupingSeparator](#) property as **true**, the numeric text in the Maps separates with the comma separator.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps Format="c" UseGroupingSeparator="true">
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      DataSource="CountryData" ShapePropertyPath='new string[] {"name"}'
      ShapeDataPath="Country" TValue="MapDataSource" >
      <MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Membership">
        <MapsShapeColorMappings>
          <MapsShapeColorMapping Value="Permanent" Color='new string[] {"#D84444"}' />
          <MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
            {"#316DB5"}' />
        </MapsShapeColorMappings>
      </MapsShapeSettings>
      <MapsLayerTooltipSettings Visible="true"
        ValuePath="Population"></MapsLayerTooltipSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>

@code {
  public class MapDataSource
  {
    public string Country { get; set; }
    public string Membership { get; set; }
    public double Population { get; set; }
  }
}
```

```
};  
public List<MapDataSource> CountryData = new List<MapDataSource>{  
    new MapDataSource { Country= "China", Membership= "Permanent", Population=  
        38332521},  
    new MapDataSource { Country= "France", Membership= "Permanent", Population=  
        19651127 },  
    new MapDataSource { Country= "Russia", Membership= "Permanent", Population=  
        3090416},  
    new MapDataSource { Country= "Kazakhstan", Membership= "Non-Permanent",  
        Population= 1232521},  
    new MapDataSource { Country= "Poland", Membership= "Non-Permanent",  
        Population= 90332521},  
    new MapDataSource { Country= "Sweden", Membership= "Non-Permanent",  
        Population= 383521}  
};  
}
```



See also

- [Localization in Blazor Maps component](#)

### Localization in Blazor Maps Component

The localization library allows localizing the default text content of the Maps component. The Maps component has the static text of some features such as tooltip of zoom toolbar, and that can be

changed to other cultures (Arabic, Deutsch, French, etc..) by referring to the Resource file. Refer to more details about localization [here](#).

<!-- markdownlint-disable MD033 -->

The following is the list of properties that is available in the **.resx** file under the **Resource** folder and its values used in the Maps component.

Name	Text to display
Maps_Zoom	Zoom
Maps_ZoomIn	Zoom In
Maps_ZoomOut	Zoom Out
Maps_Reset	Reset
Maps_Pan	Pan

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings Enable="true"></MapsZoomSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      ShapePropertyPath='new string[] { "name"}'
      DataSource="SecurityCouncilDetails"
      ShapeDataPath="Name" TValue="UNCouncilCountry">
      <MapsDataLabelSettings Visible="true" LabelPath="CountryName">
      </MapsDataLabelSettings>
      <MapsLayerTooltipSettings Visible="true" Format="{CountryName} -
      ${Membership}">
      </MapsLayerTooltipSettings>
      <MapsShapeSettings Fill="#E5E5E5" ColorValuePath="Membership">
      <MapsShapeColorMappings>
        <MapsShapeColorMapping Value="Permanent" Color='new string[] { "#EDB46F"}'>
        </MapsShapeColorMapping>
        <MapsShapeColorMapping Value="Nicht-Permanent" Color='new string[]
        { "#F1931B"}'>
        </MapsShapeColorMapping>
      </MapsShapeColorMappings>
    </MapsShapeSettings>
  </MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class UNCouncilCountry
{
public string Name { get; set; }
public string CountryName { get; set; }
public string Membership { get; set; }
};
// Set data source value in 'de' culture
```

```
private List<UNCouncilCountry> SecurityCouncilDetails = new
List<UNCouncilCountry>{
new UNCouncilCountry { Name= "China", CountryName= "China", Membership=
"Permanent"},
new UNCouncilCountry { Name= "France", CountryName= "Frankreich",
Membership= "Permanent" },
new UNCouncilCountry { Name= "Russia", CountryName= "Russland", Membership=
"Permanent"},
new UNCouncilCountry { Name= "Kazakhstan", CountryName= "Kasachstan",
Membership= "Nicht-Permanent"},
new UNCouncilCountry { Name= "Poland", CountryName= "Polen", Membership=
"Nicht-Permanent"},
new UNCouncilCountry { Name= "Sweden", CountryName= "Schweden", Membership=
"Nicht-Permanent"},
new UNCouncilCountry { Name= "United Kingdom", CountryName=
"Großbritannien", Membership= "Permanent"},
new UNCouncilCountry { Name= "United States", CountryName= "Vereinigte
Staaten", Membership= "Permanent"},
new UNCouncilCountry { Name= "Bolivia", CountryName= "Bolivien", Membership=
"Nicht-Permanent"},
new UNCouncilCountry { Name= "Eq. Guinea", CountryName= "Gl. Guinea",
Membership= "Nicht-Permanent"},
new UNCouncilCountry { Name= "Ethiopia", CountryName= "Äthiopien",
Membership= "Nicht-Permanent"},
new UNCouncilCountry { Name= "Côte d Ivoire", CountryName= "Elfenbeinküste",
Membership= "Permanent"},
new UNCouncilCountry { Name= "Kuwait", CountryName= "Kuwait", Membership=
"Nicht-Permanent"},
new UNCouncilCountry { Name= "Netherlands", CountryName= "Niederlande",
Membership= "Nicht-Permanent"},
new UNCouncilCountry { Name= "Peru", CountryName= "Peru", Membership=
"Nicht-Permanent"}
};
}
```



See also

- [Globalization in Blazor Maps component](#)

## Events in Blazor Maps Component

This section explains the list of events that will be triggered for appropriate actions in Maps. The events should be provided to the Maps using the [MapsEvents](#).

### AnimationCompleted

When the animation in the component is completed, the [AnimationCompleted](#) event will be triggered. To know more about the arguments of this event, refer [here](#)

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsEvents AnimationCompleted="@AnimationEvent"></MapsEvents>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
      TValue="string">
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
  public void AnimationEvent(Syncfusion.Blazor.Maps.AnimationCompleteEventArgs
    args)
  {
```

```
// Here you can customize your code
}  
}
```

### AnnotationRendering

Before the annotation is rendered in the Maps, the [AnnotationRendering](#) event will be triggered. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps  
<SfMaps>  
<MapsEvents AnnotationRendering="@AnnotationRenderingEvent"></MapsEvents>  
<MapsAnnotations>  
<MapsAnnotation X="20%" Y="10%" ZIndex="-1"  
VerticalAlignment="AnnotationAlignment.Center"  
HorizontalAlignment="AnnotationAlignment.Center">  
<ContentTemplate>  
<div>  
<div id="first"><h1>Maps</h1></div>  
</div>  
</ContentTemplate>  
</MapsAnnotation>  
</MapsAnnotations>  
<MapsLayers>  
<MapsLayer ShapeData='new {dataOptions  
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'  
TValue="string">  
</MapsLayer>  
</MapsLayers>  
</SfMaps>  
@code {  
public void  
AnnotationRenderingEvent(Syncfusion.Blazor.Maps.AnnotationRenderingEventArgs  
args)  
{  
// Here you can customize your code  
}  
}
```

### BubbleRendering

The [BubbleRendering](#) event is triggered before rendering each bubble. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps  
<SfMaps>  
<MapsEvents BubbleRendering="@BubbleRenderingEvent"></MapsEvents>  
<MapsLayers>  
<MapsLayer ShapeData='new {dataOptions  
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'  
DataSource="PopulationDetails"  
ShapeDataPath="Name" ShapePropertyPath='new string[] { "name" }'  
TValue="Country">  
</MapsLayer>  
</MapsLayers>  
</SfMaps>
```

```
@* To add bubbles based on population count *@
<MapsBubbleSettings>
<MapsBubble Visible="true" ValuePath="Population" ColorValuePath="Color"
DataSource="PopulationDetails" TValue="Country">
</MapsBubble>
</MapsBubbleSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class Country
{
public string Name { get; set; }
public double Population { get; set; }
public string Color { get; set; }
};
private List<Country> PopulationDetails = new List<Country> {
new Country
{
Name = "United States", Population = 325020000, Color = "#b5e485"
},
new Country
{
Name = "Russia", Population = 142905208, Color = "#7bc1e8"
},
new Country
{
Name="India", Population=1198003000, Color = "#df819c"
}
};
void BubbleRenderingEvent(Syncfusion.Blazor.Maps.BubbleRenderingEventArgs
args)
{
// Here you can customize your code
}
}
```

### DataLabelRendering

The [DataLabelRendering](#) event is triggered before rendering each label. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLayers>
<MapsEvents DataLabelRendering="@DataLabelRenderingEvent"></MapsEvents>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
@* To add data labels *@
<MapsDataLabelSettings Visible="true"
LabelPath="name"></MapsDataLabelSettings>
<MapsShapeSettings Autofill="true"></MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
```

```
@code {
public void
DataLabelRenderingEvent(Syncfusion.Blazor.Maps.LabelRenderingEventArgs args)
{
// Here you can customize your code
}
}
```

### LayerRendering

The [LayerRendering](#) event is triggered before rendering each layer. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsEvents LayerRendering="@LayerRenderingEvent"></MapsEvents>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public void
LayerRenderingEvent(Syncfusion.Blazor.Maps.LayerRenderingEventArgs args)
{
// Here you can customize your code
}
}
```

### LegendRendering

The [LegendRendering](#) event is triggered before rendering the legend in the component. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsEvents LegendRendering="@LegendRenderingEvent"></MapsEvents>
@* To set legend mode as interactive *@
<MapsLegendSettings Visible="true" Mode="LegendMode.Interactive">
</MapsLegendSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
ShapeDataPath="Name"
DataSource="SecurityCouncilDetails" ShapePropertyPath='new string[]
{"name"}' TValue="UNCouncilCountry">
<MapsShapeSettings ColorValuePath="Membership">
<MapsShapeColorMappings>
<MapsShapeColorMapping Value="Permanent" Color='new string[] { "#D84444"}' />
<MapsShapeColorMapping Value="Non-Permanent" Color='new string[]
{"#316DB5"}' />
</MapsShapeColorMappings>
```



```

</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
private List<UNCouncilCountry> SecurityCouncilDetails = new
List<UNCouncilCountry>{
new UNCouncilCountry { Name= "China", Membership= "Permanent"},
new UNCouncilCountry { Name= "France", Membership= "Permanent" },
new UNCouncilCountry { Name= "Russia", Membership= "Permanent"},
new UNCouncilCountry { Name= "Kazakhstan", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "Poland", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "Sweden", Membership= "Non-Permanent"},
new UNCouncilCountry { Name= "United Kingdom", Membership= "Permanent"},
new UNCouncilCountry { Name= "United States", Membership= "Permanent"}
};
public class UNCouncilCountry
{
public string Name { get; set; }
public string Membership { get; set; }
};
public void
LegendRenderingEvent(Syncfusion.Blazor.Maps.LegendRenderingEventArgs args)
{
// Here you can customize your code
}
}

```

### Loaded

The [Loaded](#) event is triggered after the Maps component has been loaded. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsEvents Loaded="@LoadedEvent"></MapsEvents>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public void LoadedEvent(Syncfusion.Blazor.Maps.LoadedEventArgs args)
{
// Here you can customize your code
}
}

```

### MarkerRendering

The [MarkerRendering](#) event is triggered before rendering each marker. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsEvents MarkerRendering="@MarkerRenderingEvent"></MapsEvents>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
<MapsMarkerSettings>
<MapsMarker Visible="true" DataSource="California" Height="25" Width="15"
TValue="City"></MapsMarker>
<MapsMarker Visible="true" DataSource="NewYork" Height="25" Width="15"
TValue="City"></MapsMarker>
<MapsMarker Visible="true" DataSource="Iowa" Height="25" Width="15"
TValue="City"></MapsMarker>
</MapsMarkerSettings>
<MapsShapeSettings Fill="lightgray"></MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
};
public List<City> California = new List<City> {
new City {Latitude=35.145083,Longitude=-117.960260}
};
public List<City> NewYork = new List<City> {
new City { Latitude=40.724546, Longitude=-73.850344 }
};
public List<City> Iowa = new List<City> {
new City {Latitude= 41.657782, Longitude=-91.533857}
};
public void
MarkerRenderingEvent(Syncfusion.Blazor.Maps.MarkerRenderingEventArgs args)
{
// Here you can customize your code
}
}

```

### MarkerClusterClick

The [MarkerClusterClick](#) event is triggered after clicking the marker cluster. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsEvents MarkerClusterClick="@MarkerClusterClickEvent"></MapsEvents>
<MapsZoomSettings Enable="true"></MapsZoomSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
<MapsMarkerSettings>
<MapsMarker Visible="true" DataSource="LargestCities" Height="25" Width="15"
TValue="City">

```

```

</MapsMarker>
</MapsMarkerSettings>
<MapsMarkerClusterSettings AllowClustering="true" Shape="MarkerType.Circle"
Fill="#008CFF" Height="25" Width="25">
<MapsLayerMarkerClusterLabelStyle
Color="white"></MapsLayerMarkerClusterLabelStyle>
</MapsMarkerClusterSettings>
<MapsShapeSettings Fill="lightgray">
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
public double Area { get; set; }
};
private List<City> LargestCities = new List<City> {
new City { Latitude=40.6971494, Longitude= -74.2598747, Name="New York",
Area=8683 },
new City { Latitude=40.0024137, Longitude= -75.2581194, Name="Philadelphia",
Area=4661 },
new City { Latitude=42.3142647, Longitude= -71.11037, Name="Boston",
Area=4497 },
new City { Latitude=42.3526257, Longitude= -83.239291, Name="Detroit",
Area=3267 },
new City { Latitude=47.2510905, Longitude= -123.1255834, Name="Washington",
Area=2996 },
new City { Latitude=25.7823907, Longitude= -80.2994995, Name="Miami",
Area=2891 },
new City { Latitude=19.3892246, Longitude= -70.1305136, Name="San Juan",
Area=2309 }
};
public void
MarkerClusterClickEvent(Syncfusion.Blazor.Maps.MarkerClusterClickEventArgs
args)
{
// Here you can customize your code
}
}

```

### MarkerClusterMouseMove

The [MarkerClusterMouseMove](#) event will be triggered when the cursor moves over the marker cluster. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsEvents
MarkerClusterMouseMove="@MarkerClusterMouseMoveEvent"></MapsEvents>
<MapsZoomSettings Enable="true"></MapsZoomSettings>
<MapsLayers>

```

```

<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
<MapsMarkerSettings>
<MapsMarker Visible="true" DataSource="LargestCities" Height="25" Width="15"
TValue="City">
</MapsMarker>
</MapsMarkerSettings>
<MapsMarkerClusterSettings AllowClustering="true" Shape="MarkerType.Circle"
Fill="#008CFF" Height="25" Width="25">
<MapsLayerMarkerClusterLabelStyle
Color="white"></MapsLayerMarkerClusterLabelStyle>
</MapsMarkerClusterSettings>
<MapsShapeSettings Fill="lightgray">
</MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
public double Area { get; set; }
};
private List<City> LargestCities = new List<City> {
new City { Latitude=40.6971494, Longitude= -74.2598747, Name="New York",
Area=8683 },
new City { Latitude=40.0024137, Longitude= -75.2581194, Name="Philadelphia",
Area=4661 },
new City { Latitude=42.3142647, Longitude= -71.11037, Name="Boston",
Area=4497 },
new City { Latitude=42.3526257, Longitude= -83.239291, Name="Detroit",
Area=3267 },
new City { Latitude=47.2510905, Longitude= -123.1255834, Name="Washington",
Area=2996 },
new City { Latitude=25.7823907, Longitude= -80.2994995, Name="Miami",
Area=2891 },
new City { Latitude=19.3892246, Longitude= -70.1305136, Name="San Juan",
Area=2309 }
};
public void
MarkerClusterMouseMoveEvent(Syncfusion.Blazor.Maps.MarkerClusterMoveEventArgs args)
{
// Here you can customize your code
}
}

```

### OnBubbleClick

The [OnBubbleClick](#) event will be triggered when clicking on the bubbles. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
```

```

<SfMaps>
  <MapsEvents OnBubbleClick="@OnBubbleClickEvent"></MapsEvents>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
DataSource="PopulationDetails"
ShapeDataPath="Name" ShapePropertyPath='new string[] {"name"}'
TValue="Country">
    @* To add bubbles based on population count *@
    <MapsBubbleSettings>
      <MapsBubble Visible="true" ValuePath="Population" ColorValuePath="Color"
DataSource="PopulationDetails" TValue="Country">
    </MapsBubble>
    </MapsBubbleSettings>
  </MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class Country
{
    public string Name { get; set; }
    public double Population { get; set; }
    public string Color { get; set; }
};
private List<Country> PopulationDetails = new List<Country> {
    new Country
    {
        Name="United States", Population = 325020000, Color = "#b5e485"
    },
    new Country
    {
        Name = "Russia", Population = 142905208, Color = "#7bc1e8"
    },
    new Country
    {
        Name="India", Population=1198003000, Color = "#df819c"
    }
    };
public void OnBubbleClickEvent(Syncfusion.Blazor.Maps.BubbleClickEventArgs
args)
{
    // Here you can customize your code
}
}

```

### OnBubbleMouseMove

The [OnBubbleMouseMove](#) event will be triggered when the cursor moves over the bubbles. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsEvents OnBubbleMouseMove="@OnBubbleMouseMoveEvent"></MapsEvents>
  <MapsLayers>

```

```

<MapsLayer ShapeData='new {dataOptions
="https://cdn.syncfusion.com/maps/map-data/world-map.json"}'
DataSource="PopulationDetails"
ShapeDataPath="Name" ShapePropertyPath='new string[] {"name"}'
TValue="Country">
    /* To add bubbles based on population count */
    <MapsBubbleSettings>
    <MapsBubble Visible="true" ValuePath="Population" ColorValuePath="Color"
    DataSource="PopulationDetails" TValue="Country">
    </MapsBubble>
    </MapsBubbleSettings>
    </MapsLayer>
</MapsLayers>
</SfMaps>
@code{
public class Country
{
    public string Name { get; set; }
    public double Population { get; set; }
    public string Color { get; set; }
};
private List<Country> PopulationDetails = new List<Country> {
    new Country
    {
        Name = "United States", Population = 325020000, Color = "#b5e485"
    },
    new Country
    {
        Name = "Russia", Population = 142905208, Color = "#7bc1e8"
    },
    new Country
    {
        Name="India", Population=1198003000, Color = "#df819c"
    }
};
public void
OnBubbleMouseMoveEvent(Syncfusion.Blazor.Maps.BubbleMoveEventArgs args)
{
    // Here you can customize your code
}
}

```

### OnClick

The [OnClick](#) event will be triggered after the Maps is clicked. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
    <MapsEvents OnClick="@OnClickEvent"></MapsEvents>
    <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
    "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
    </MapsLayer>
    </MapsLayers>

```

```

</SfMaps>
@code {
public void OnClickEvent(Syncfusion.Blazor.Maps.MouseEventArgs args)
{
// Here you can customize your code
}
}

```

### OnDoubleClick

When performing the double click operation on an element in Maps, the [OnDoubleClick](#) will be triggered. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsEvents OnDoubleClick="@OnDoubleClickEvent"></MapsEvents>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public void OnDoubleClickEvent(Syncfusion.Blazor.Maps.MouseEventArgs args)
{
// Here you can customize your code
}
}

```

### OnItemHighlight

The [OnItemHighlight](#) event occurs when the cursor moves over the shapes. To know more about the arguments of this event, refer [here](#).

@using Syncfusion.Blazor.Maps

#### ASPX-CS

```

<SfMaps>
<MapsEvents OnItemHighlight="@OnItemHighlightEvent"></MapsEvents>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
<MapsLayerHighlightSettings Enable="true" Fill="green">
<MapsLayerHighlightBorder Color="white"
Width="2"></MapsLayerHighlightBorder>
</MapsLayerHighlightSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public void OnItemHighlightEvent(Syncfusion.Blazor.Maps.SelectionEventArgs args)
{
// Here you can customize your code
}
}

```

```
}  
}
```

### OnItemSelect

The [OnItemSelect](#) event occurs when selecting the shapes in the Maps. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps  
<SfMaps>  
<MapsEvents OnItemSelect="@OnItemSelectEvent"></MapsEvents>  
<MapsLayers>  
<MapsLayer ShapeData='new {dataOptions=  
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">  
<MapsLayerSelectionSettings Enable="true" Fill="green">  
<MapsLayerSelectionBorder Color="White"  
Width="2"></MapsLayerSelectionBorder>  
</MapsLayerSelectionSettings>  
</MapsLayer>  
</MapsLayers>  
</SfMaps>  
@code {  
public void OnItemSelectEvent(Syncfusion.Blazor.Maps.SelectionEventArgs  
args)  
{  
// Here you can customize your code  
}  
}
```

### OnLoad

[OnLoad](#) event will be triggered before rendering the Maps. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps  
<SfMaps>  
<MapsEvents OnLoad="@OnLoadEvent"></MapsEvents>  
<MapsLayers>  
<MapsLayer ShapeData='new {dataOptions=  
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">  
</MapsLayer>  
</MapsLayers>  
</SfMaps>  
@code {  
public void OnLoadEvent(Syncfusion.Blazor.Maps.LoadEventArgs args)  
{  
// Here you can customize your code  
}  
}
```



### OnMarkerClick

[OnMarkerClick](#) event will be triggered by clicking the markers in the Maps. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsEvents OnMarkerClick="@OnMarkerClickEvent"></MapsEvents>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" DataSource="California" Height="25" Width="15"
          TValue="City"></MapsMarker>
        <MapsMarker Visible="true" DataSource="NewYork" Height="25" Width="15"
          TValue="City"></MapsMarker>
        <MapsMarker Visible="true" DataSource="Iowa" Height="25" Width="15"
          TValue="City"></MapsMarker>
      </MapsMarkerSettings>
      <MapsShapeSettings Fill="lightgray"></MapsShapeSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
  public class City
  {
    public double Latitude { get; set; }
    public double Longitude { get; set; }
  };
  public List<City> California = new List<City> {
    new City {Latitude=35.145083,Longitude=-117.960260}
  };
  public List<City> NewYork = new List<City> {
    new City { Latitude=40.724546, Longitude=-73.850344 }
  };
  public List<City> Iowa = new List<City> {
    new City {Latitude= 41.657782, Longitude=-91.533857}
  };
  public void OnMarkerClickEvent(Syncfusion.Blazor.Maps.MarkerClickEventArgs
  args)
  {
    // Here you can customize your code
  }
}
```

### OnMarkerMouseLeave

The [OnMarkerMouseLeave](#) event is triggered when the cursor moves away from the marker. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsEvents OnMarkerMouseLeave="@OnMarkerMouseLeaveEvent"></MapsEvents>
  <MapsLayers>
```

```

<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
  <MapsMarkerSettings>
    <MapsMarker Visible="true" DataSource="California" Height="25" Width="15"
    TValue="City"></MapsMarker>
    <MapsMarker Visible="true" DataSource="NewYork" Height="25" Width="15"
    TValue="City"></MapsMarker>
    <MapsMarker Visible="true" DataSource="Iowa" Height="25" Width="15"
    TValue="City"></MapsMarker>
  </MapsMarkerSettings>
  <MapsShapeSettings Fill="lightgray"></MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
};
public List<City> California = new List<City> {
new City {Latitude=35.145083,Longitude=-117.960260}
};
public List<City> NewYork = new List<City> {
new City { Latitude=40.724546, Longitude=-73.850344 }
};
public List<City> Iowa = new List<City> {
new City {Latitude= 41.657782, Longitude=-91.533857}
};
public void
OnMarkerMouseLeaveEvent(Syncfusion.Blazor.Maps.MarkerMoveEventArgs args)
{
// Here you can customize your code
}
}

```

### OnMarkerMouseMove

The [OnMarkerMouseMove](#) event is triggered when the cursor moves over the marker. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsEvents OnMarkerMouseMove="@OnMarkerMouseMoveEvent"></MapsEvents>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" DataSource="California" Height="25" Width="15"
        TValue="City"></MapsMarker>
        <MapsMarker Visible="true" DataSource="NewYork" Height="25" Width="15"
        TValue="City"></MapsMarker>
        <MapsMarker Visible="true" DataSource="Iowa" Height="25" Width="15"
        TValue="City"></MapsMarker>
      </MapsMarkerSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>

```

```

<MapsShapeSettings Fill="lightgray"></MapsShapeSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
};
public List<City> California = new List<City> {
new City {Latitude=35.145083,Longitude=-117.960260}
};
public List<City> NewYork = new List<City> {
new City { Latitude=40.724546, Longitude=-73.850344 }
};
public List<City> Iowa = new List<City> {
new City {Latitude= 41.657782, Longitude=-91.533857}
};
public void
OnMarkerMouseMoveEvent(Syncfusion.Blazor.Maps.MarkerMoveEventArgs args)
{
// Here you can customize your code
}
}

```

### OnPan

When panning the Maps, the [OnPan](#) event will be triggered. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsZoomSettings Enable="true" ToolBarOrientation="Orientation.Vertical"
EnablePanning="true"
Toolbars='new string[]{"Zoom", "ZoomIn", "ZoomOut", "Pan", "Reset" }'>
</MapsZoomSettings>
<MapsEvents OnPan="@OnPanEvent"></MapsEvents>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public void OnPanEvent(Syncfusion.Blazor.Maps.MapPanEventArgs args)
{
// Here you can customize your code
}
}

```

### OnPanComplete

The [OnPanComplete](#) event will be triggered after panning the Maps. To know more about the arguments of this event, refer [here](#).

**ASPX-CS**

```

@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings Enable="true" ToolBarOrientation="Orientation.Vertical"
  EnablePanning="true"
  Toolbars='new string[] {"Zoom", "ZoomIn", "ZoomOut", "Pan", "Reset" }'>
  </MapsZoomSettings>
  <MapsEvents OnPanComplete="@OnPanCompleteEvent"></MapsEvents>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
    "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
public void OnPanCompleteEvent(Syncfusion.Blazor.Maps.MapPanEventArgs args)
{
  // Here you can customize your code
}
}

```

**OnPrint**

The [OnPrint](#) event will be triggered before the print operation is started. To know more about the arguments of this event, refer [here](#).

**ASPX-CS**

```

<button @onclick="PrintMap">Print</button>
@using Syncfusion.Blazor.Maps
<SfMaps @ref="maps" AllowPrint="true">
  <MapsEvents OnPrint="@GetGEOLocation"></MapsEvents>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
    "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
SfMaps maps;
public void PrintMap()
{
  // using Maps component reference call 'Print' method
  this.maps.Print();
}
public void GetGEOLocation(Syncfusion.Blazor.Maps.PrintEventArgs args)
{
  // Here you can customize your code
}
}

```

**OnRightClick**

The [OnRightClick](#) event will be triggered when performing the right click operation on an element in Maps. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsEvents OnRightClick="@OnRightClickEvent"></MapsEvents>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
    "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
  </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
public void OnRightClickEvent(Syncfusion.Blazor.Maps.MouseEventArgs args)
{
  // Here you can customize your code
}
}
```

### OnZoom

The [OnZoom](#) event will be triggered before zooming-in or zooming-out the Maps. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings Enable="true" ToolBarOrientation="Orientation.Vertical"
  Toolbars='new string[]{"Zoom", "ZoomIn", "ZoomOut", "Pan", "Reset" }'>
  </MapsZoomSettings>
  <MapsEvents OnZoom="@OnZoomEvent"></MapsEvents>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
    "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
  </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
public void OnZoomEvent(Syncfusion.Blazor.Maps.MapZoomEventArgs args)
{
  // Here you can customize your code
}
}
```

### OnZoomComplete

The [OnZoomComplete](#) event will be triggered after performing zooming operation. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsZoomSettings Enable="true" ToolBarOrientation="Orientation.Vertical"
  Toolbars='new string[]{"Zoom", "ZoomIn", "ZoomOut", "Pan", "Reset" }'>
  </MapsZoomSettings>
  <MapsEvents OnZoomComplete="@OnZoomCompleteEvent"></MapsEvents>
```

```

<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public void OnZoomCompleteEvent(Syncfusion.Blazor.Maps.MapZoomEventArgs
args)
{
// Here you can customize your code
}
}

```

### Resizing

The [Resizing](#) event will be triggered when resizing the Maps. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsEvents Resizing="@ResizingEvent"></MapsEvents>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public void ResizingEvent(Syncfusion.Blazor.Maps.ResizeEventArgs args)
{
// Here you can customize your code
}
}

```

### ShapeHighlighted

The [ShapeHighlighted](#) event is triggered when mouse move on the shape in Maps and before the shape gets highlighted. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsEvents ShapeHighlighted="@ShapeHighlightedEvent"></MapsEvents>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
<MapsLayerHighlightSettings Enable="true" Fill="green">
<MapsLayerHighlightBorder Color="white"
Width="2"></MapsLayerHighlightBorder>
</MapsLayerHighlightSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>

```

```
@code {
public void
ShapeHighlightedEvent(Syncfusion.Blazor.Maps.ShapeSelectedEventArgs args)
{
// Here you can customize your code
}
}
```

### ShapeRendering

The [ShapeRendering](#) event will be triggered before rendering a shape of the Maps. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsEvents ShapeRendering="@ShapeRenderingEvent"></MapsEvents>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public void
ShapeRenderingEvent(Syncfusion.Blazor.Maps.ShapeRenderingEventArgs args)
{
// Here you can customize your code
}
}
```

### ShapeSelected

The [ShapeSelected](#) event is triggered when select a shape in Maps. To know more about the arguments of this event, refer [here](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsEvents ShapeSelected="@ShapeSelectedEvent"></MapsEvents>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
<MapsLayerSelectionSettings Enable="true" Fill="green">
<MapsLayerSelectionBorder Color="White"
Width="2"></MapsLayerSelectionBorder>
</MapsLayerSelectionSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public void ShapeSelectedEvent(Syncfusion.Blazor.Maps.ShapeSelectedEventArgs
args)
{
// Here you can customize your code
}
```

```
}
}
```

### TooltipRendering

The [TooltipRendering](#) event is triggered before the tooltip gets rendered. To know more about the arguments of this event, refer [here](#).

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsEvents TooltipRendering="@TooltipRenderingEvent"></MapsEvents>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
      <MapsLayerTooltipSettings Visible="true"
        ValuePath="name"></MapsLayerTooltipSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
  public void
  TooltipRenderingEvent(Syncfusion.Blazor.Maps.TooltipRenderEventArgs args)
  {
    // Here you can customize your code
  }
}
```

## Methods in Blazor Maps Component

This section explains the methods used in the Maps component.

### ShapeSelectionAsync

The [ShapeSelectionAsync](#) method can be used to select a shape dynamically in the shape layer of the Maps. The following are the arguments for this method.

Argument name	Description
layerIndex	Specifies the index number of layer in which the shape is to be selected.
propertyName	Specifies the property path for map shape data to select the shape.
name	Specifies the shape data path for the data source of the layer.
enable	Specifies whether to select or unselect the shape.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<button @onclick="ShapeSelectAsync">Select Shape</button>
<SfMaps @ref="maps">
  <MapsZoomSettings Enable="true" EnablePanning="true">
  </MapsZoomSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
```



```

<MapsLayerSelectionSettings Enable="true">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
SfMaps maps;
public async Task ShapeSelectAsync()
{
await maps.ShapeSelectionAsync(0, "Argentina", "Argentina");
}
}

```

### Refresh

The [Refresh](#) method can be used to change the state of the component and render it again.

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<button @onclick="Refresh">Refresh</button>
<SfMaps @ref="maps">
<MapsZoomSettings Enable="true" EnablePanning="true">
</MapsZoomSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
SfMaps maps;
public void Refresh()
{
await maps.Refresh();
}
}

```

### PanByDirectionAsync

[PanByDirectionAsync](#) method pans the Maps dynamically by specifying direction. The following are the arguments for this method.

Argument name	Description
direction	Specifies to the direction of panning operation.
mouseLocation	Specifies the position of the panning within the Maps.

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<button @onclick="PanByDirectionAsync">Pan by Direction</button>
<SfMaps @ref="maps">
<MapsZoomSettings Enable="true" EnablePanning="true">
</MapsZoomSettings>
<MapsLayers>

```

```

<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
SfMaps maps;
void PanByDirectionAsync()
{
Syncfusion.Blazor.Maps.Internal.Point position = new
Syncfusion.Blazor.Maps.Internal.Point();
position.X = 120;
position.Y = 200;
maps.PanByDirectionAsync(Syncfusion.Blazor.Maps.PanDirection.Bottom,
position);
}
}

```

### ZoomByPosition

[ZoomByPosition](#) method zooms the Maps by specifying the center position for the map. The following are the arguments for this method.

Argument name	Description
centerPosition	Specifies the position of the maps.
zoomFactor	Specifies the zoom level of maps.

### ASPX-CS

```

@using Syncfusion.Blazor.Maps
<button @onclick="ZoomByPosition">Print</button>
<SfMaps @ref="maps">
<MapsZoomSettings Enable="true">
</MapsZoomSettings>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
SfMaps maps;
public void ZoomByPosition()
{
MapsCenterPosition centerPosition = new MapsCenterPosition();
centerPosition.Latitude = 35.145083;
centerPosition.Longitude = -117.960260;
maps.ZoomByPosition(centerPosition, 2);
}
}

```

### ZoomToCoordinates

[ZoomToCoordinates](#) zooms the map to the center point of the provided minimum and maximum coordinates. The following are the arguments for this method.

Argument name	Description
minLatitude	Specifies the minimum latitude of the coordinate for the zooming operation.
minLongitude	Specifies the minimum longitude of the coordinate for the zooming operation.
maxLatitude	Specifies the maximum latitude of the coordinate for the zooming operation.
maxLongitude	Specifies the maximum longitude of the coordinate for the zooming operation.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<button onclick="ZoomToCoordinates">Print</button>
<SfMaps @ref="maps">
  <MapsZoomSettings Enable="true">
  </MapsZoomSettings>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
    "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
  SfMaps maps;
  public void ZoomToCoordinates()
  {
    maps.ZoomToCoordinates(0, 0, 100, 100);
  }
}
```

### How To

#### Display geometry shapes in Bing maps in Blazor Maps Component

Usually, the Bing Maps displays the maps in satellite view, in which you cannot make changes as you need. To overcome this, add maps shape as sub layer over the Bing Maps and customize it. The following steps explain how to add geometry shapes as sublayer in Bing Maps.

#### <b>Step 1</b>

To render Bing Maps in the Maps component, set the `UrlTemplate` property with the Bing Maps URL link passed and generated by the `GetBingUrlTemplate` method.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer UrlTemplate="@UrlTemplate" TValue="string"></MapsLayer>
  </MapsLayers>
</SfMaps>
```

```
@code {
public string UrlTemplate;
protected override async Task OnInitializedAsync()
{
    UrlTemplate = await
    SfMaps.GetBingUrlTemplate("https://dev.virtualearth.net/REST/V1/Imagery/Meta
data/CanvasGray?output=json&uriScheme=https&key=");
}
}
```

In the above URL passed to the `GetBingUrlTemplate` method, specify the Bing Maps key.



### <b>Step 2</b>

Add geometry shape in the Bing Maps using sublayer concept. To add geometry shape, import shape data, set type as subLayer, and assign your shape data to the ShapeData API.

### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
<MapsLayers>
<MapLayer UrlTemplate="@UrlTemplate" TValue="string"></MapLayer>
/* To add geometry shape as sublayer */
<MapLayer ShapeData='new {dataOptions =
"https://cdn.syncfusion.com/maps/map-data/africa.json"}'
Type="Syncfusion.Blazor.Maps.Type.SubLayer" TValue="string">
<MapShapeSettings Fill="blue"></MapShapeSettings>
</MapLayer>
</MapsLayers>
</SfMaps>
@code {
public string UrlTemplate;
protected override async Task OnInitializedAsync()
{
    UrlTemplate = await
    SfMaps.GetBingUrlTemplate("https://dev.virtualearth.net/REST/V1/Imagery/Meta
data/CanvasGray?output=json&uriScheme=https&key=");
}
}
```

```
}
```

In the above URL passed to the `GetBingUrlTemplate` method, specify the Bing Maps key.

The above code renders Africa continent as sublayer in the Bing Maps.



### Add different types of markers in Blazor Maps Component

You can add different types of markers in the Maps component using the [MapsMarkerSettings](#). The following steps describe how to add different types of markers.

#### <b>Step 1</b>

Initialize the Maps component with marker settings. Here, a marker is added with specified latitude and longitude of California using the [DataSource](#) property. You can customize the shape of the marker using the [Shape](#) property and change the border color and width of the marker using the [MapsMarkerBorder](#).

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions=
      "https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true"
          Shape="MarkerType.Circle"
```

```
Fill="white"
Width="15"
DataSource="Cities" TValue="City">
<MapsMarkerBorder Width="2" Color="#333"></MapsMarkerBorder>
</MapsMarker>
</MapsMarkerSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
};
private List<City> Cities = new List<City> {
new City{ Latitude = 40.7424509, Longitude = -74.0081468, Name = "New York"
}
};
}
```



### <b>Step 2</b>

Customize the above option for n number of markers as demonstrated in the following code example.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps
```

```
<SfMaps>
<MapsLayers>
<MapsLayer ShapeData='new {dataOptions=
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}' TValue="string">
<MapsMarkerSettings>
<MapsMarker Visible="true" Shape="MarkerType.Circle"
Fill="white"
Width="20"
DataSource="HighestPopulation" TValue="City">
<MapsMarkerBorder Width="2" Color="#333"></MapsMarkerBorder>
</MapsMarker>
<MapsMarker Visible="true" Shape="MarkerType.Rectangle"
Fill="yellow"
Width="20"
Height="5"
DataSource="LowestPopulation" TValue="City">
<MapsMarkerBorder Width="2" Color="#333"></MapsMarkerBorder>
</MapsMarker>
</MapsMarkerSettings>
</MapsLayer>
</MapsLayers>
</SfMaps>
@code {
public class City
{
public double Latitude { get; set; }
public double Longitude { get; set; }
public string Name { get; set; }
};
public List<City> HighestPopulation = new List<City> {
new City { Latitude = 40.7424509, Longitude = -74.0081468, Name = "New York"
}
};
public List<City> LowestPopulation = new List<City> {
new City { Latitude=33.5302186, Longitude=-117.7418381, Name="Laguna Niguel"
}
};
}
```



### *Tooltip for marker*

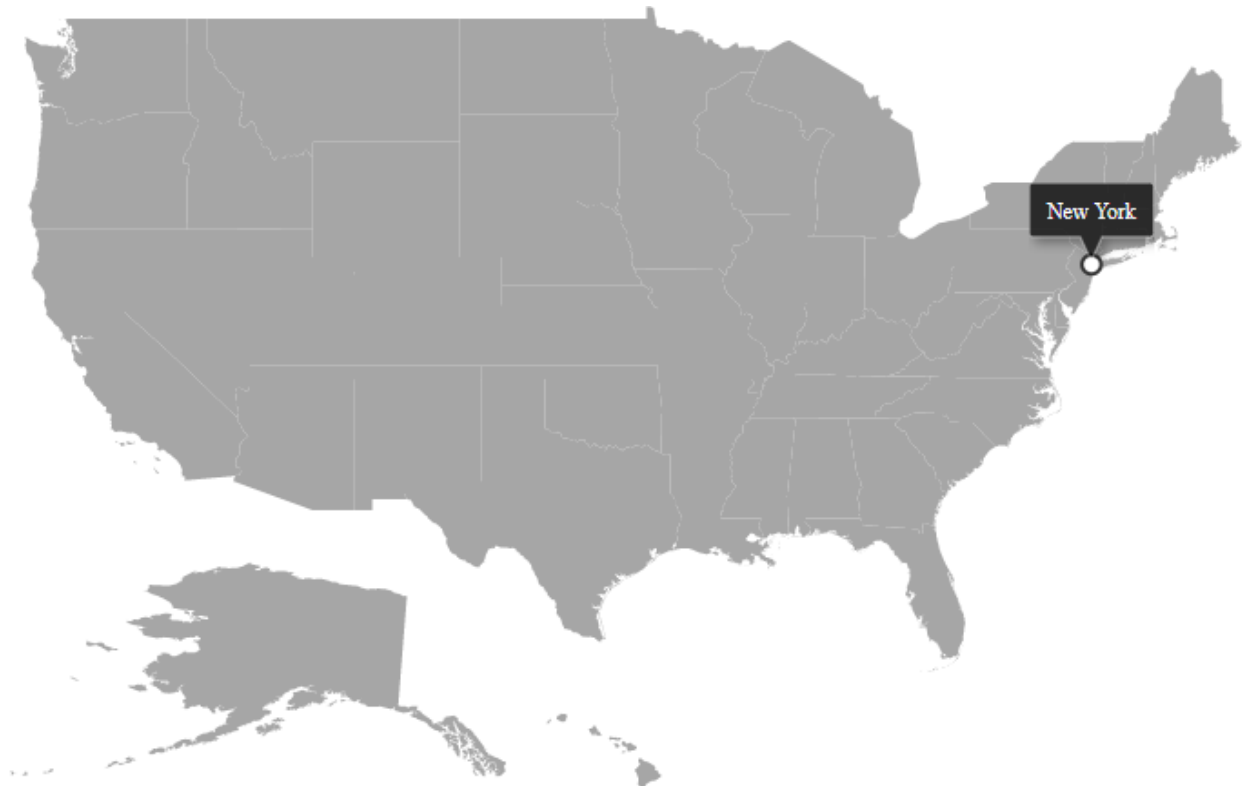
Tooltip is used to display more information about marker on mouse over or touch-end event. This can be enabled separately for layer or marker using the [MapsMarkerTooltipSettings](#). The [ValuePath](#) property in tooltip takes the field name that presents in dataSource and displays that value as tooltip text.

### **ASPX-CS**

```
@using Syncfusion.Blazor.Maps
<SfMaps>
  <MapsLayers>
    <MapsLayer ShapeData='new {dataOptions
      ="https://cdn.syncfusion.com/maps/map-data/usa.json"}' TValue="string">
      <MapsMarkerSettings>
        <MapsMarker Visible="true" Shape="MarkerType.Circle"
          Fill="white"
          Width="20"
          DataSource="HighestPopulation" TValue="City">
          <MapsMarkerBorder Width="2" Color="#333"></MapsMarkerBorder>
          <MapsMarkerTooltipSettings Visible="true"
            ValuePath="Name"></MapsMarkerTooltipSettings>
        </MapsMarker>
      </MapsMarkerSettings>
    </MapsLayer>
  </MapsLayers>
</SfMaps>
@code {
  public class City
```



```
{  
    public double Latitude { get; set; }  
    public double Longitude { get; set; }  
    public string Name { get; set; }  
};  
public List<City> HighestPopulation = new List<City> {  
    new City { Latitude = 40.7424509, Longitude = -74.0081468, Name = "New York"  
    }  
};  
}
```



### Change center position on zooming in Blazor Maps Component

Blazor Maps component provides support to change the center position of the Maps. This can be achieved by setting the coordinates of the location in the [MapsCenterPosition](#). The [ZoomFactor](#) property in the [MapsZoomSettings](#) can be used to focus the provided center position in the Maps.

#### ASPX-CS

```
@using Syncfusion.Blazor.Maps  
<SfMaps>  
    @* To change center position *@  
    <MapsCenterPosition Latitude="25.54244147012483" Longitude="-  
89.62646484375"></MapsCenterPosition>  
    <MapsZoomSettings Enable="false" ZoomFactor="13"></MapsZoomSettings>  
    <MapsLayers>  
        <MapsLayer ShapeData='new {dataOptions=  
"https://cdn.syncfusion.com/maps/map-data/world-map.json"}'  
TValue="string"></MapsLayer>  
    </MapsLayers>
```

</SfMaps>

