

USER GUIDE

# Essential Studio for WPF

---

Version - v19.3.0.43 | Release Date - September 30, 2021

Welcome to Syncfusion WPF UI controls.....	68
How to best read this user guide .....	68
Controls List .....	68
table .....	68
anchor .....	69
title .....	69
Licensing.....	70
Additional help resources .....	70
Support and feedback.....	70
Installation and Upgrade.....	71
System Requirements for WPF .....	71
Download WPF Installer.....	71
Download the Trial Version .....	71
Download the License Version.....	74
Installation using Web Installer .....	74
Installation .....	75
Uninstallation.....	86
Installation using Offline Installer .....	96
Installing with UI .....	96
Installing in silent mode .....	105
Common Installation Errors .....	106
Unlocking the license installer using the trial key.....	107
License has expired .....	107
Unable to find a valid license or trial .....	108
Unable to install because of another installation.....	109
Unable to install due to controlled folder access .....	111
Upgrading Syncfusion WPF .....	112
Upgrading to the latest version .....	112
Upgrade from trial version to license version.....	112
Licensing.....	113
license { .....	113
Syncfusion Licensing in WPF .....	113
License Key Generation.....	113
License Key Registration .....	114
Common Licensing Errors .....	114



License key not registered .....	114
Invalid key .....	115
Trial Expired .....	115
Platform Mismatch .....	116
Version Mismatch .....	116
Could not load Syncfusion.Licensing.dll assembly version...? .....	117
Licensing FAQ.....	117
How to upgrade from Trial version after purchasing a license?.....	117
Where can I get a license key?.....	117
Registering Syncfusion account for direct NuGet.org user.....	117
Applying the Patches .....	118
Installing the Patch installer.....	118
Patch Assembly Version Format .....	121
Visual Studio Integration.....	122
Syncfusion WPF Extension .....	122
Toolbox Configuration .....	123
Configuring toolbox for WPF .NET 5.0 projects .....	126
Configuring toolbox for .NET Core 3.1 projects .....	127
Visual Studio Extensions .....	128
Download and Installation .....	128
Create WPF application.....	130
Add Syncfusion window to WPF application .....	136
Troubleshoot the project .....	142
Add Reference for WPF.....	148
Check for Updates in Syncfusion Essential WPF .....	155
NuGet Packages in Syncfusion Essential WPF.....	156
Installing NuGet Packages.....	157
Common.....	159
Control Dependencies in Syncfusion's WPF Controls .....	159
AutoComplete .....	159
BusyIndicator .....	159
ButtonAdv .....	160
Calculate.....	160
CalendarEdit.....	160
CardView .....	160

Carousel .....	160
CheckListBox .....	160
ChromelessWindow .....	160
Clock.....	160
ColorPicker .....	161
ColorPickerPalette .....	161
ComboBoxAdv.....	161
CurrencyTextBox.....	161
DateTimeEdit .....	161
DockingManager .....	161
DocumentContainer.....	161
DoubleTextBox.....	162
DropDownButtonAdv.....	162
Edit Control .....	162
Gantt .....	162
Grid Control.....	162
GroupBar .....	162
HierarchyNavigator .....	163
IntegerTextBox.....	163
ImageEditor .....	163
MaskedTextBox.....	163
MenuAdv.....	163
NotifyIcon.....	163
OlapChart .....	163
OlapClient.....	164
OlapGauge.....	164
OlapGrid .....	164
PDFViewer.....	165
PercentTextBox.....	165
PinnableListBox.....	165
PivotGrid .....	165
PropertyGrid .....	166
ReportDesigner .....	166
ReportViewer .....	166
ReportWriter.....	167

Ribbon .....	167
RichTextBoxAdv .....	167
RichTextDocIOParser .....	167
RichTextRibbon .....	168
SfAccordion .....	168
SfAreaSparkline .....	168
SfBadge .....	168
SfBarcode .....	168
SfBulletGraph .....	168
SfBusyIndicator .....	168
SfCalculator .....	169
SfChart .....	169
SfChromelessWindow .....	169
SfCircularProgressBar .....	169
SfColorPalette .....	169
SfColumnSparkline .....	169
SfDataGrid .....	169
SfDataPager .....	170
SfDatePicker .....	170
SfDateTimeRangeNavigator .....	170
SfDiagram .....	170
SfDomainUpDown .....	171
SfGauge .....	171
SfGridSplitter .....	171
SfHeatMap .....	171
SfHubTile .....	171
SfKanban .....	171
SfLinearProgressBar .....	172
SfLineSparkline .....	172
SfMaps .....	172
SfMaskedEdit .....	172
SfNavigationDrawer .....	172
SfPulsingTile .....	172
SfRadialMenu .....	172
SfRadialSlider .....	173

SfRangeSlider .....	173
SfRating .....	173
SfRichTextBoxAdv .....	173
SfRichTextRibbon .....	173
SfScheduler .....	174
SfSmithChart .....	174
SfSpellChecker.....	174
SfSpreadsheet .....	174
SfSunburstChart .....	175
SfTextboxExt.....	175
SfTextInputLayout.....	175
SfTimePicker.....	175
SfTreeGrid .....	175
SfTreeView .....	176
SfTreeMap.....	176
SfTreeNavigator .....	176
SfWinLossSparkline .....	176
SkinManager .....	176
SplitButton .....	176
TabControlExt .....	176
TabNavigation .....	177
TabSplitter.....	177
TaskBar.....	177
TileView.....	177
TimespanEdit.....	177
ToolBarAdv.....	177
TreeViewAdv .....	177
UpDown .....	178
Wizard .....	178
SfSkinManager Dependencies .....	178
Add Syncfusion WPF Controls.....	181
Through Designer.....	181
Through Code-Behind .....	183
Through Project Template .....	184
Localization of Syncfusion WPF Controls.....	189

Changing application culture .....	189
Creating .resx files .....	189
Editing default culture strings .....	192
Getting Started for Blend Support .....	192
Edit Control Style in Expression Blend .....	192
Edit ControlTemplate in Expression Blend.....	194
Right to Left support in Syncfusion WPF Controls .....	195
Pattern and Practices .....	195
Getting Started with MVVM .....	195
MVVM Commands .....	199
Featured Samples for Syncfusion WPF Controls.....	201
Syncfusion WPF Controls Panel .....	201
WPF Sample Browser.....	203
Offline Samples .....	203
Online Samples .....	205
Download demos from online (Clone from github repository) .....	205
Themes and Appearance .....	205
Getting Started with WPF Skin Manager .....	205
Themes list .....	205
Apply a theme to a control .....	207
Apply a theme globally in the application .....	209
Customize theme colors and fonts in the application .....	210
Apply themes to the controls derived from Syncfusion controls .....	211
Clearing SkinManager instance in an application .....	212
How to.....	212
Touch Support in Syncfusion WPF Controls.....	215
Size mode .....	215
Gesture.....	215
Getting Started with Windows 10 Compact ScrollBar (Touch ScrollBars) .....	223
Getting Started with Keyboard Focus Visual for WPF Controls .....	224
Getting Started with WPF Theme Studio .....	225
Supported themes .....	225
Supported palettes .....	225
Creating custom theme.....	227
Testing.....	237

Coded UI Testing in WPF.....	237
Prerequisites and compatibility .....	237
Create, record, and run the tests.....	237
Supported controls .....	243
Coded UI Test extension support.....	244
Steps to work with Coded UI .....	247
Overview in UFT Testing .....	251
Prerequisites and Compatibility.....	251
Installation .....	251
Enabling add-on support in UFT.....	256
Configuring add-in assemblies for different .NET framework version .....	256
Create, Record and Run the tests .....	257
Supported Controls.....	265
SfDataGrid.....	265
SfMulticolumnDropDownControl .....	266
SfDataPager.....	267
GridDataControl .....	268
GridTreeControl .....	271
Grid Control.....	272
Chart.....	274
SfChart.....	275
PivotGrid .....	276
Section 508 Compliance - Voluntary Product Accessibility Template .....	277
Subpart B – Technical Standards Section 1194.21.....	278
Subpart C - Section 1194.31 Functional Performance Criteria .....	281
Subpart D - Section 1194.41 Information, documentation, and support.....	282
SfAccordion .....	282
WPF Accordion (SfAccordion) Overview.....	282
Features .....	283
Getting Started with WPF Accordion (SfAccordion) .....	283
Assembly deployment.....	283
Create a simple application with SfAccordion .....	283
Create a project .....	283
Add control through designer.....	283
Add control manually in XAML .....	283

Add control manually in C\# .....	284
Add items using SfAccordionItem .....	284
Bind data .....	285
Apply template to item header.....	289
Set content to children .....	290
Theme .....	291
Populating Items in WPF Accordion (SfAccordion).....	291
Using Items.....	291
Using ItemsSource .....	294
Selection Mode in WPF Accordion (SfAccordion).....	297
SelectionMode - One .....	297
SelectionMode - OneOrMore .....	298
SelectionMode - ZeroOrOne .....	298
SelectionMode - ZeroOrMore.....	298
Selecting Items in WPF Accordion (SfAccordion).....	299
Selecting item using SelectedIndex .....	299
Selecting item using SelectedItem .....	299
Retrieving the selected items .....	299
Retrieving the selected item indices.....	299
Selecting item using IsSelected.....	301
Checking the lock state of an item .....	303
Select All Items.....	303
Unselect All Items .....	304
Notifying selected item change .....	304
Notifying an item selection .....	307
Notifying an item un-selection.....	307
Notifying selection change.....	308
Appearance and Styling in WPF Accordion (SfAccordion) .....	308
Applying accent colors .....	308
Accordion header style .....	309
Accordion expander style .....	311
AccordionItem header height customization .....	313
AccordionItem content height customization .....	314
Enable or disable the animation behaviour .....	315
SfBarcode .....	319

WPF Barcode (SfBarcode) Overview .....	319
Structure of the Control .....	319
Getting Started with WPF Barcode (SfBarcode) .....	319
Add Barcode control to an Application .....	319
Text .....	320
Rotation .....	321
Supported Barcode types in WPF Barcode (SfBarcode) .....	321
Barcode Customization in WPF Barcode (SfBarcode) .....	321
Symbology Settings in WPF Barcode (SfBarcode) .....	322
1D Barcode settings .....	322
2D Barcode Settings .....	323
Version .....	325
Error correction level .....	325
Input mode .....	325
SfBulletGraph .....	325
WPF Bullet Graph (SfBulletGraph) Overview .....	325
Use cases .....	326
Key Features in WPF Bullet Graph (SfBulletGraph) .....	326
Ticks .....	326
Label .....	326
Caption .....	326
Range .....	326
Performance measure .....	326
Bullet graphs type .....	326
Data Binding Support ## .....	327
MVVM support ## .....	327
SfBulletGraph Elements ## .....	327
Getting Started with WPF Bullet Graph (SfBulletGraph) .....	327
Configuring SfBulletGraph using Syncfusion Reference Manager .....	327
Configuring SfBulletGraph .....	332
Adding Caption .....	333
Configuring Ticks and Labels .....	333
Adding Ranges .....	335
Theme .....	336
Orientation in WPF Bullet Graph (SfBulletGraph) .....	336



Flow Direction in WPF Bullet Graph (SfBulletGraph) .....	338
Quantitative Scale in WPF Bullet Graph (SfBulletGraph) .....	340
Ticks in WPF Bullet Graph (SfBulletGraph) .....	341
Customizing Ticks .....	341
TickPosition .....	342
Labels in WPF Bullet Graph (SfBulletGraph) .....	342
Customizing Labels .....	342
Label Position .....	343
Caption in WPF Bullet Graph (SfBulletGraph) .....	343
Caption Position .....	344
Measures in WPF Bullet Graph (SfBulletGraph) .....	344
Featured Measure .....	344
Comparative Measure .....	346
Ranges in WPF Bullet Graph (SfBulletGraph) .....	347
Customizing Range .....	347
Binding RangeStroke to Ticks and Labels .....	348
Tooltip in WPF Bullet Graph (SfBulletGraph) .....	350
Customizing FeaturedMeasureToolTip .....	351
Customizing ComparativeMeasureToolTip .....	352
Customizing QualitativeRangeToolTip .....	354
SfBusyIndicator .....	355
WPF Busy Indicator (SfBusyIndicator) Overview .....	355
Key Features .....	356
Getting Started with WPF Busy Indicator (SfBusyIndicator) .....	356
Theme .....	356
IsBusy in WPF Busy Indicator (SfBusyIndicator) .....	357
Header in WPF Busy Indicator (SfBusyIndicator) .....	357
Header .....	357
Header Template .....	358
Sizing in WPF Busy Indicator (SfBusyIndicator) .....	358
ViewBoxHeight .....	358
ViewBoxWidth .....	359
AnimationTypes in WPF Busy Indicator (SfBusyIndicator) .....	360
SfBadge .....	367
WPF Badge (SfBadge) Overview .....	367

Badge control .....	367
Key features .....	367
Getting Started with WPF Badge (SfBadge) .....	368
Structure of SfBadge control .....	368
Assembly deployment .....	368
Adding WPF SfBadge via XAML .....	368
Adding WPF SfBadge via C# .....	368
Adding badge for a Button .....	369
Setting Badge display content .....	370
Alignment of Badge .....	370
Positioning of Badge .....	371
Place the Badge any where on the container .....	371
Adding badge without BadgeContainer .....	372
Predefined colors for displaying the badges .....	374
Predefined shapes for displaying the Badge .....	375
Animate when content changes .....	376
Custom Content Formats .....	377
Theme .....	378
Customization in WPF Badge (SfBadge) .....	378
Adding badge for a Button .....	378
Adding badge without BadgeContainer .....	379
Setting Badge display content .....	381
Custom UI for Badge content .....	381
Predefined colors for displaying the badges .....	382
Custom colors for displaying the badges .....	383
Predefined shapes for displaying the Badge .....	383
Custom shape for displaying the Badge .....	384
Animate when content changes .....	385
Stroke customization .....	386
Display number formatting .....	387
Change Badge size .....	388
Text formatting .....	389
Change opacity of Badge .....	390
Hide the Badge .....	390
Alignment and positioning in WPF Badge (SfBadge) .....	391

Alignment of Badge.....	391
Positioning of Badge .....	392
Place the Badge any where on the container .....	393
Custom alignment and positioning of Badge.....	394
Badge content alignment.....	395
ButtonAdv .....	395
WPF Button (ButtonAdv) Overview .....	395
Key features .....	395
Getting Started with WPF Button (ButtonAdv) .....	396
Control structure.....	396
Assembly deployment.....	396
Creating simple application with Button .....	396
Setting label .....	398
Setting size mode.....	399
Setting icon template.....	400
Setting icon template selector .....	402
Setting image .....	404
Setting icon width and height.....	405
IsDefault mode.....	405
IsCancel mode.....	406
Theme .....	406
MVVM in WPF Button (ButtonAdv) .....	406
Multiline Text in WPF Button (ButtonAdv) .....	409
Toggle State in WPF Button (ButtonAdv).....	409
Events in WPF Button (ButtonAdv).....	410
Click .....	410
Checked.....	410
Styles and Templates in WPF Button (ButtonAdv) .....	410
Edit appearance in Expression Blend.....	411
Edit appearance in Visual Studio.....	413
Themes in WPF Button (ButtonAdv).....	415
SfCalculator .....	415
WPF Calculator (SfCalculator) Overview.....	415
Key features .....	415
Getting Started with WPF Calculator (SfCalculator) .....	416

Assembly deployment.....	416
Creating Application with SfCalculator control.....	416
Creating project .....	416
Add control via designer .....	416
Add control manually in XAML .....	417
Add control manually in C#.....	417
Setting watermark .....	418
Setting value.....	419
Theme .....	420
Memory in WPF Calculator (SfCalculator) .....	420
CalendarEdit.....	421
WPF Calendar (CalendarEdit) Overview .....	421
Control structure.....	421
Features .....	421
Getting Started with WPF Calendar (CalendarEdit).....	422
Structure of CalendarEdit.....	422
Assembly deployment.....	422
Adding WPF CalendarEdit via designer .....	422
Adding WPF CalendarEdit via XAML .....	422
Adding WPF CalendarEdit via C#.....	423
Select a date.....	424
Select multiple dates.....	425
Restrict date selection .....	426
Block dates .....	427
Special days.....	428
Display week numbers .....	429
Change default view (Month, Year, Decade) .....	430
Setting Culture .....	431
Show full month and week name .....	432
Tooltip for particular days.....	432
Selected date changed notification .....	433
Theme .....	433
Date-Selection in WPF Calendar (CalendarEdit) .....	434
Select a date.....	434
Select multiple dates.....	435

Select multiple dates using key navigation .....	437
Highlight selected date .....	439
Get today date .....	440
Display today date .....	440
Differentiate current month days from other days .....	442
Change default view (Month, Year, Decade) .....	443
Display week numbers .....	444
Highlight week numbers .....	444
Special days .....	445
Setting Culture .....	447
Show full month and week name .....	448
Tooltip for particular days .....	448
Custom appearance of day cell .....	449
Custom UI for day name cell using style .....	450
Selected date changed notification .....	451
Restrict Date Selection in WPF Calendar (CalendarEdit) .....	451
Restrict date selection within minimum and maximum date .....	451
Restrict date selection .....	453
Block dates .....	453
Custom appearance of blocked days .....	454
Hide previous and next month days .....	456
Readonly support .....	457
Date Navigation in WPF Calendar (CalendarEdit) .....	457
Navigate to the day, month or year modes using header .....	457
Navigate to previous or next month .....	459
Custom UI for previous and next navigation buttons .....	460
Change navigation direction .....	461
Touch support for CalendarEdit .....	462
Using CalendarEdit Object in WPF Calendar (CalendarEdit) .....	462
Appearance in WPF Calendar (CalendarEdit) .....	468
Setting the foreground .....	468
Setting the background .....	469
Change flow direction .....	469
Theme .....	470
CardView .....	470

WPF Card View Overview .....	470
Control Structure .....	471
Key features .....	471
Getting Started with WPF Card View .....	471
Structure of CardView control .....	471
Assembly deployment.....	472
Adding WPF CardView control via designer .....	472
Adding WPF CardView control via XAML.....	472
Adding WPF CardView control via C# .....	473
Populating items using CardViewItem.....	474
Populating items using ItemsSource.....	475
Select a CardViewItem .....	477
Select CardViewItem programmatically using property .....	478
Group the CardViewItems .....	479
Sort the CardViewItems .....	482
Edit the CardViewItems .....	482
Orientation of CardViewItems .....	484
Selected item changed notification .....	485
Theme .....	485
Data Binding and Customization in WPF Card View .....	486
Data binding to objects .....	486
Custom UI for CardViewItem header.....	489
Custom UI for CardViewItem content.....	490
Custom UI for edit mode CardViewItem.....	492
Different UI styles for specific CardViewItem.....	493
Change flow direction .....	496
Theme .....	496
Editing Mode in WPF Card View .....	498
Enable/disable the editing mode.....	498
Card editing using keyboard and mouse interaction.....	499
Start card editing programmatically .....	502
Custom UI for edit mode CardViewItem.....	504
Grouping Mode in WPF Card View .....	506
Enable/disable the grouping mode.....	506
Group the CardViewItems .....	507

Group the cards programmatically .....	509
Nested grouping of CardViewItems .....	510
Hide the grouping header .....	511
Sorting Mode in WPF Card View .....	512
Enable/disable the sorting mode .....	512
Sort the CardViewItems .....	512
Sort the grouped CardViewItems .....	515
Hide the sorting header .....	515
Filtering Mode in WPF Card View .....	516
Filter the CardViewItems .....	516
Reset the filter .....	518
Hide the filtering header .....	519
Localization in WPF Card View .....	520
Carousel .....	522
WPF Carousel Overview .....	522
Structure of the Carousel Control .....	523
Features .....	523
Getting Started with WPF Carousel .....	523
Structure of Carousel .....	524
Assembly deployment .....	524
Adding WPF Carousel via designer .....	524
Adding WPF Carousel via XAML .....	525
Adding WPF Carousel via C# .....	525
Populating items using CarouselItem .....	526
Populating items using collection binding .....	529
Select carousel item .....	531
Rotate carousel item .....	532
Resize the carousel items .....	533
Change radius of carousel item .....	534
Custom UI of carousel item .....	535
Custom display path for carousel items .....	536
Number of items to be visible in Page .....	537
Selected item changed notification .....	538
Theme .....	538
Populating Items in WPF Carousel .....	539

Populating items using CarouselItem .....	539
Populating items using collection binding .....	542
Custom UI for carousel item using template .....	544
Custom UI for carousel item using style .....	545
Custom UI for specific carousel item using style selector .....	548
Virtualization support .....	550
Select carousel item .....	551
Select carousel item programmatically using property .....	551
Select carousel item programmatically using command and methods.....	552
Selected item changed notification .....	552
Change flow direction .....	553
Items Navigation in WPF Carousel .....	553
Navigate carousel item using keyboard navigation .....	554
Navigate carousel item using scroll bar .....	555
Navigate carousel item using mouse wheel .....	556
Navigate using commands .....	557
Looping items in Custom path view .....	558
Standard Path in WPF Carousel .....	559
Load carousel items in standard path.....	559
Change radius of carousel control .....	561
Change rotation speed.....	562
Disable rotation animation .....	563
Resize carousel item .....	564
Opacity for carousel item.....	565
Skewing the carousel item .....	566
Custom Path in WPF carousel (Carousel).....	567
Load carousel items in custom path .....	567
Number of items to be visible in a Page .....	569
Resize carousel items.....	570
Resize specific carousel item .....	570
Opacity for carousel items .....	571
Opacity for specific carousel item.....	572
Skewing the carousel items .....	573
Skewing the specific carousel item .....	573
Syntax Editor .....	575



WPF Syntax Editor Overview .....	575
Introduction to Essential Edit WPF .....	575
Key features .....	576
Getting Started with WPF Syntax Editor .....	576
Assembly deployment.....	576
Creating simple application with EditControl .....	577
Adding EditControl via designer.....	577
Adding WPF EditControl via XAML.....	578
Adding WPF EditControl via C# .....	578
Loading a file into document .....	579
Syntax highlighting .....	582
Theme .....	584
Syntax Editor Members in WPF Syntax Editor .....	585
Edit Commands in WPF Syntax Editor .....	590
Text Selection in WPF Syntax Editor .....	591
Selection using Mouse .....	591
Selection using Keyboard with shortcuts.....	591
Selection using Touch .....	591
Text Navigation in WPF Syntax Editor.....	593
Programmatic Navigation .....	593
Keyboard Navigation.....	594
Line Background Customization in WPF Syntax Editor .....	594
Applying line background customization.....	594
Resetting line background customization.....	595
On demand line background customization .....	596
Printing in WPF Syntax Editor .....	597
Print Preview .....	597
Print Settings.....	599
Setting Header and Footer.....	602
Basic Editing .....	605
File Support in WPF Syntax Editor .....	609
Status Bar in WPF Syntax Editor.....	616
Language Support .....	637
Supported Languages in WPF Syntax Editor .....	637
Syntax Highlighting in WPF Syntax Editor .....	638

Expand Collapse Support in WPF Syntax Editor .....	639
Custom Language Support in WPF Syntax Editor .....	648
IntelliSense Support in WPF Syntax Editor .....	658
RoutedUICommands in WPF Syntax Editor .....	671
SfChart.....	672
WPF Charts (SfChart) Overview .....	672
Key features .....	672
Getting Started with WPF Charts (SfChart).....	673
Adding chart reference .....	673
Initialize chart.....	673
Initialize view model .....	674
Populate chart with data .....	676
Add title.....	677
Enable data labels .....	677
Enable legend.....	678
Enable tooltip.....	679
See also .....	682
Theme .....	683
Area in WPF Charts (SfChart) .....	683
Customization .....	684
Multiple Area .....	684
Column Span and Row Span .....	686
Clone or copy the chart.....	688
Header in WPF Charts (SfChart).....	688
Axis in WPF Charts (SfChart) .....	690
Header.....	691
Axis Labels.....	694
Grid lines .....	713
Tick lines.....	715
Customize individual axis elements .....	721
AxisLine .....	722
Origin Customization.....	724
Types of Axis .....	728
Inverting axis.....	746
Customizing the Intervals.....	746

Apply Padding to the Range.....	751
Applying Padding to the Axis .....	761
AutoScrollingDelta .....	764
Auto Interval Calculation on Zooming .....	765
Multiple Axes .....	766
Multi-level Labels .....	767
Events.....	785
See also .....	786
Legend in WPF Charts (SfChart) .....	786
Legend Icon .....	787
Label.....	791
Checkbox.....	792
ToggleSeriesVisibility .....	793
Positioning the Legend.....	793
Legend Header .....	796
Multiple Legends.....	797
Legends for Accumulation Series .....	798
Series visibility on legend.....	799
Legend Orientation .....	800
Customization .....	801
Troubleshooting .....	803
See also .....	803
Series in WPF Charts (SfChart) .....	804
Fast in WPF Charts (SfChart) .....	804
Sorting in WPF Charts (SfChart) .....	804
Sorting for category(non-linear) axis .....	805
Sorting for linear axis .....	809
Data Binding in WPF Charts (SfChart) .....	813
See also .....	816
Data Markers in WPF Charts (SfChart).....	817
See also .....	818
Annotations in WPF Charts (SfChart).....	818
Adding Annotation .....	818
Positioning the Annotation .....	819
Text Annotation .....	823

Shape Annotation .....	825
Image Annotation .....	832
Interaction.....	833
ToolTip .....	835
Annotation Clipping .....	837
Annotation based on axis.....	838
Events.....	839
See also .....	840
Vertical Charts in WPF Charts (SfChart).....	840
OpposedPosition.....	840
IsTransposed .....	841
Striplines in WPF Charts (SfChart).....	844
Positioning the Striplines .....	844
Label.....	845
Multiple Striplines.....	848
Segmented Stripline.....	850
Customization .....	852
See also .....	854
Scale Breaks in WPF Charts (SfChart).....	854
Positioning the Breaks .....	854
Break Position Customization .....	855
Multiple Breaks .....	858
Customization .....	859
Technical Indicators in WPF Charts (SfChart) .....	860
Adding Technical Indicators to the Chart.....	860
Average True Range .....	862
Simple Average .....	862
RSI .....	863
Momentum .....	864
Stochastic .....	865
Exponential Average .....	866
Triangular Average.....	867
Accumulation Distribution .....	868
Bollinger Band.....	868
MACD .....	869

Trendlines in WPF Charts (SfChart).....	870
Types of Trendlines .....	872
Forecasting.....	878
Customization .....	880
See also .....	881
Interactive Features in WPF Charts (SfChart) .....	881
Appearance in WPF Charts (SfChart) .....	881
Palettes .....	882
Custom Palette.....	884
Gradient Colors .....	886
SegmentColorPath .....	888
Customize Legends .....	889
Customize ToolTip.....	890
See also .....	892
Animation in WPF Charts (SfChart).....	892
Exporting in WPF Charts (SfChart) .....	895
Methods.....	895
See also .....	897
Printing in WPF Charts (SfChart).....	897
Print() .....	897
Watermark in WPF Charts (SfChart) .....	899
Adding text watermark .....	899
Adding image watermark.....	900
See also .....	901
Performance in WPF Charts (SfChart).....	901
Deferred real-time updates .....	902
See also .....	902
Serialization in WPF Charts (SfChart).....	902
Coded UI in WPF Charts (SfChart).....	904
Levels.....	904
Requirements.....	905
Configuration .....	905
Getting Started.....	905
Run Tests.....	909
Tables of Properties .....	910

Migrating from Chart to SfChart in WPF Charts (SfChart) .....	911
Adding Reference .....	911
Initialization .....	911
Legend .....	913
Axis .....	915
Series .....	920
Adornments .....	924
Interactive Cursor .....	926
Zooming and Panning .....	927
StripLines .....	928
Watermark .....	930
Annotation .....	931
SfChart .....	937
SyncChartAreas .....	937
SyncChartAreas .....	938
Column Row Definition .....	939
Exporting and Printing .....	940
Printing Chart .....	940
How to .....	941
Add custom labels to track ball behavior .....	941
Transform axis value to pixel value and vice versa .....	942
Add range of points dynamically .....	942
Print the SfChart in WPF .....	942
Serialization and Deserialization: .....	943
SfChart3D .....	944
WPF SfChart3D Overview .....	944
Key features .....	945
Getting Started with WPF SfChart3D .....	945
Adding chart reference .....	945
Initialize chart .....	945
Initialize view model .....	946
Populate chart with data .....	947
Add Title .....	948
Enable data labels .....	948
Enable legend .....	949

Enable tooltip.....	950
Theme .....	952
Axis in WPF SfChart3D .....	953
Types of Axis .....	953
Depth Axis .....	955
3D Manhattan Chart .....	957
Series in WPF SfChart3D .....	958
Column Charts.....	959
Bar Charts.....	959
Line Charts .....	962
Scatter Chart .....	963
Area Chart .....	964
Stacking Charts.....	965
Pie Chart.....	969
Doughnut Chart.....	970
Dynamic explode.....	972
Data Markers in WPF SfChart3D .....	973
Interactive Features in WPF SfChart3D.....	974
Dynamic rotation .....	974
Segment Selection .....	974
Series Selection .....	975
Appearance in WPF SfChart3D .....	977
Palettes .....	977
Custom Palette.....	980
SegmentColorPath .....	982
SfSmithChart .....	983
WPF Smith Chart (SfSmithChart) Overview .....	983
Key features .....	984
Getting Started with WPF Smith Chart (SfSmithChart).....	984
Steps.....	984
Create a simple smith chart from XAML.....	985
Create a simple smith chart from code behind (C#) .....	992
Theme .....	996
Rendering Type in WPF Smith Chart (SfSmithChart) .....	997
Impedance .....	997

Admittance.....	998
Axes in WPF Smith Chart (SfSmithChart) .....	999
Horizontal Axis .....	999
AxisLine .....	1004
Radial Axis .....	1007
Events.....	1015
Series in WPF Smith Chart (SfSmithChart) .....	1016
Animation.....	1018
Series Visibility .....	1019
Data Markers in WPF Smith Chart (SfSmithChart).....	1020
Add Shapes.....	1020
Customizing Marker .....	1021
Add Labels .....	1023
Legend in WPF Smith Chart (SfSmithChart) .....	1027
Positioning the Legend.....	1029
Legend Icon .....	1031
Customizing Legend .....	1033
VisibilityOnLegend .....	1034
Series Visibility .....	1036
Appearance in WPF Smith Chart (SfSmithChart) .....	1037
SmithChart Palette .....	1037
Chart Area Customization .....	1039
Circle Radius.....	1040
Get smith chart properties.....	1041
User Interactions in WPF Smith Chart (SfSmithChart) .....	1042
ToolTip .....	1042
SfSunburstChart .....	1045
WPF Sunburst Chart (SfSunburstChart) Overview .....	1045
Key Features.....	1045
Getting Started with WPF Sunburst Chart (SfSunburstChart) .....	1045
Adding assembly reference.....	1046
Adding SfSunburstChart from Toolbox .....	1046
Initialize view model .....	1047
Populate Sunburst chart with data .....	1050
Add header.....	1051



Add legend .....	1051
Add data labels.....	1051
Theme .....	1053
Region in WPF Sunburst Chart (SfSunburstChart) .....	1054
Start and End angle .....	1054
Sunburst radius .....	1055
Sunburst inner radius.....	1056
Levels in WPF Sunburst Chart (SfSunburstChart) .....	1057
GroupMemberPath .....	1057
Legend in WPF Sunburst Chart (SfSunburstChart).....	1058
Legend Icon .....	1059
Positioning the Legend.....	1061
Customize the Legend.....	1062
Legend Interactivity .....	1063
Data Label in WPF Sunburst Chart (SfSunburstChart) .....	1065
LabelOverflowMode .....	1066
LabelRotationMode .....	1068
Font customization .....	1069
Customizing the data labels .....	1070
Tooltip in WPF Sunburst Chart (SfSunburstChart) .....	1071
Aligning the ToolTip .....	1072
VerticalOffset and HorizontalOffset .....	1074
Show Delay.....	1075
Animation for Tooltip.....	1076
Customize the Tooltip .....	1076
Selection in WPF Sunburst Chart (SfSunburstChart) .....	1077
SelectionDisplayMode .....	1078
SelectionMode .....	1079
SelectionType .....	1080
Selection Cursor .....	1083
Zooming in WPF Sunburst Chart (SfSunburstChart) .....	1084
Zooming Toolbar .....	1085
Animation in WPF Sunburst Chart (SfSunburstChart) .....	1087
Animation types .....	1087
Palette in WPF Sunburst Chart (SfSunburstChart) .....	1089

Applying palette to the chart .....	1089
Custom Palette .....	1091
Events in WPF Sunburst Chart (SfSunburstChart) .....	1093
SegmentCreated Event .....	1093
SelectionChanged .....	1094
SfSurfaceChart .....	1096
WPF Surface Chart (SfSurfaceChart) Overview .....	1096
Key Feature of Surface Chart .....	1096
Getting Started with WPF Surface Chart (SfSurfaceChart) .....	1096
Visual Structure .....	1096
Create simple surface chart from XAML .....	1097
Add Surface chart from Toolbox .....	1098
Create a simple surface chart from Code behind. ....	1102
Data Binding in WPF Surface Chart (SfSurfaceChart) .....	1105
Surface Types in WPF Surface Chart (SfSurfaceChart) .....	1107
Surface Axis in WPF Surface Chart (SfSurfaceChart) .....	1112
Axis customization. ....	1112
ColorBar in WPF Surface Chart (SfSurfaceChart) .....	1115
Palettes in WPF Surface Chart (SfSurfaceChart) .....	1116
Surface Area in WPF Surface Chart (SfSurfaceChart) .....	1118
Properties .....	1118
User interaction in WPF Surface Chart (SfSurfaceChart) .....	1121
Zooming .....	1121
Rotation .....	1122
Tilt .....	1123
SfDateTimeRangeNavigator .....	1124
WPF Range Selector (SfDateTimeRangeNavigator) Overview .....	1124
Key features .....	1124
Getting Started with WPF Range Selector (SfDateTimeRangeNavigator) .....	1124
Visual structure .....	1124
Create SfDateTimeRangeNavigator .....	1125
Create SfDateTimeRangeNavigator from code-behind .....	1127
Theme .....	1129
Interactivity in WPF Range Selector (SfDateTimeRangeNavigator) .....	1129
Properties .....	1129

Events.....	1130
Thumb style customization .....	1133
Higher and lower bars customization .....	1135
Customization in WPF Range Selector (SfDateTimeRangeNavigator) .....	1135
Interval customization .....	1135
Label style customization.....	1137
RangePadding Customization .....	1139
ToolTip Support in WPF Range Selector (SfDateTimeRangeNavigator) .....	1140
Properties.....	1140
SfSparkline .....	1141
WPF Sparkline (SfSparkline) Overview.....	1141
Getting Started with WPF Sparkline (SfSparkline) .....	1141
Creating sparkline .....	1141
Theme .....	1143
Sparkline Types in WPF Sparkline (SfSparkline).....	1144
Line Sparkline .....	1144
Column Sparkline .....	1144
Area sparkline .....	1145
WinLoss Sparkline .....	1146
Range Band in WPF Sparkline (SfSparkline).....	1147
Track ball in WPF Sparkline (SfSparkline).....	1148
Markers in WPF Sparkline (SfSparkline).....	1150
Marker Customization in WPF Sparkline (SfSparkline) .....	1150
Highlight Segment in WPF Sparkline (SfSparkline) .....	1152
Customize segment brush in WPF Sparkline (SfSparkline) .....	1153
ShowHide Axis in WPF Sparkline (SfSparkline) .....	1154
How to.....	1156
Customize the marker for specific data point.....	1156
Add labels for track ball .....	1157
CheckBox .....	1157
WPF CheckedListBox (CheckBox) Overview .....	1157
Features .....	1158
Getting Started with WPF CheckedListBox (CheckBox) .....	1158
Control Structure .....	1158
Assembly deployment.....	1158

Creating simple application with CheckListBox control.....	1158
Creating project .....	1158
Adding control via designer .....	1159
Adding control manually in XAML.....	1159
Adding control manually in C# .....	1159
Populating items using CheckListBoxItem .....	1160
Populating items by DataBinding .....	1161
Check or Uncheck items.....	1162
Checked event notification .....	1163
Get list of checked items.....	1163
Localization support.....	1164
Theme .....	1165
Sorting Items in WPF CheckedListBox (CheckListBox) .....	1166
Grouping Items in WPF CheckedListBox (CheckListBox) .....	1167
Nested Grouping.....	1169
Custom grouping.....	1171
Check or uncheck all items in WPF CheckedListBox (CheckListBox).....	1174
Check Items in WPF CheckedListBox (CheckListBox).....	1175
Check items programmatically.....	1176
Check items using Mouse .....	1179
Check items using Keyboard .....	1180
Checked state changed notification.....	1182
Selection changed notification .....	1182
Virtualization in WPF CheckedListBox (CheckListBox).....	1183
Disable the Virtualization.....	1185
Appearance in WPF CheckedListBox (CheckListBox) .....	1185
Setting the Foreground .....	1185
Change flow direction .....	1187
Item Template.....	1187
Group Template .....	1189
SelectAll Template .....	1191
ItemContainerStyle .....	1192
ItemTemplateSelector .....	1194
ItemContainerStyleSelection .....	1196
Theme .....	1198

How to.....	1199
Check the Item in CheckListBox when Initiating.....	1199
ChromelessWindow .....	1199
WPF Chromeless Window Overview .....	1199
Visual Structure in WPF Chromeless Window .....	1199
Getting Started with WPF Chromeless Window.....	1200
Assembly deployment.....	1200
Creating simple application with ChromelessWindow .....	1200
Add ChromelessWindow.....	1200
Customizing title bar .....	1201
Title bar background .....	1201
Title bar font .....	1202
Title bar height.....	1203
Title bar text alignment.....	1204
Title bar icon .....	1205
Title bar icon alignment .....	1206
Customizing the border of ChromelessWindow.....	1207
Theme .....	1208
Set Visual Styles in WPF Chromeless Window .....	1209
Theme .....	1209
Corner Radius in WPF Chromeless Window .....	1210
UseNativeChrome in WPF Chromeless Window .....	1211
Title bar Customization in WPF Chromeless Window .....	1212
Customizing the background .....	1212
Customizing the height .....	1212
Customizing the font size .....	1213
Show or hide the title bar text .....	1214
Show or hide the maximize and minimize buttons .....	1215
Adding controls in the Title bar in WPF Chromeless Window .....	1216
Customizing Border of the ChromelessWindow in WPF Chromeless Window .....	1223
End user capabilities in WPF Chromeless Window.....	1225
Maximize/Minimize .....	1225
Restore .....	1226
Close.....	1226
Resize .....	1227

Styling the ChromelessWindow in WPF Chromeless Window .....	1229
Custom template for the TitleBar .....	1230
Custom template for the TitleButton .....	1231
Overriding the default style .....	1232
SfCircularGauge.....	1234
WPF Radial Gauge (SfCircularGauge) Overview.....	1234
Key features .....	1234
Getting Started with WPF Radial Gauge (SfCircularGauge) .....	1235
Adding gauge references .....	1235
Initialize gauge .....	1235
Adding headers .....	1236
Configuring scales .....	1237
Adding ranges .....	1237
Adding a needle pointer.....	1238
Adding a range pointer .....	1238
Adding a symbol pointer .....	1239
Theme .....	1242
See also .....	1243
Header in WPF Radial Gauge (SfCircularGauge) .....	1243
Setting Header for Circular Gauge .....	1243
Setting alignment for header .....	1244
Setting position for header .....	1245
Customization of header font .....	1246
Scales in WPF Radial Gauge (SfCircularGauge) .....	1247
Scale .....	1247
Setting start and end values for scale .....	1248
Setting start and sweep angles for scale .....	1249
Setting interval for scale .....	1250
Setting sweep direction for scale.....	1251
Setting multiple scales for circular gauge .....	1252
Rim in WPF Radial Gauge (SfCircularGauge).....	1254
Rim customization.....	1255
Setting a position for rim .....	1256
Ticks in WPF Radial Gauge (SfCircularGauge) .....	1259
Tick customization .....	1260

Setting shape for tick .....	1262
Setting position for tick .....	1263
Labels in WPF Radial Gauge (SfCircularGauge) .....	1267
Label stroke customization .....	1267
Label font customization .....	1268
Setting a position for labels .....	1269
Label rotation .....	1270
Setting a smart labels .....	1271
Setting a number of fraction digits for labels .....	1272
Setting a postfix and prefix for labels .....	1273
Events .....	1276
Ranges in WPF Radial Gauge (SfCircularGauge) .....	1278
Setting a start and end values for range .....	1278
Range customization .....	1278
Setting a width for range .....	1280
Binding range color to scale tick and labels .....	1281
Setting a position for range .....	1282
Setting a multiple ranges .....	1285
Pointers in WPF Radial Gauge (SfCircularGauge) .....	1289
Needle pointer .....	1289
Range pointer .....	1296
Symbol pointer .....	1304
Setting animation for pointer .....	1312
Circular pointer dragging .....	1313
Events .....	1315
See also .....	1319
Annotations in WPF Radial Gauge (SfCircularGauge) .....	1319
Setting a view annotation .....	1319
SfCircularProgressBar .....	1321
WPF Circular ProgressBar (SfCircularProgressBar) Overview .....	1321
Key features .....	1322
Getting Started with WPF Circular ProgressBar (SfCircularProgressBar) .....	1322
Assembly deployment .....	1323
Adding control through designer .....	1323
Adding control manually in XAML .....	1323

Adding control through code behind .....	1324
Theme .....	1324
States in WPF circular progressbar (SfCircularProgressBar) .....	1325
Determinate .....	1325
Indeterminate .....	1325
Buffer .....	1326
Segment in WPF circular progressbar (SfCircularProgressBar).....	1327
Appearance in WPF circular progressbar (SfCircularProgressBar) .....	1327
Angle .....	1327
Radius/Thickness.....	1328
Color Customization.....	1331
Range Colors .....	1331
Gradient .....	1332
Corner radius.....	1333
AnimationDuration .....	1334
AnimationEasing .....	1335
Custom Content in WPF circular progressbar (SfCircularProgressBar) .....	1336
SfStepProgressBar.....	1340
WPF Step ProgressBar (SfStepProgressBar) Overview .....	1340
Key features .....	1340
Getting Started with WPF Step ProgressBar (SfStepProgressBar) .....	1341
Assembly deployment.....	1341
Adding control through designer .....	1341
Adding control manually in XAML.....	1341
Select item using Selected Index .....	1341
Adding control through code behind.....	1343
Theme .....	1344
Data Binding in WPF Step ProgressBar (SfStepProgressBar) .....	1344
Data binding to Objects .....	1344
Data-Binding with XML .....	1347
Selected Item in WPF Step ProgressBar (SfStepProgressBar).....	1347
SelectedItemStatus .....	1347
SelectedItemProgress .....	1348
AnimationDuration .....	1349
Appearance in WPF Step ProgressBar (SfStepProgressBar) .....	1351



Step Shape .....	1351
Orientation.....	1352
Connector Customization .....	1353
MarkerClicked Event .....	1354
Customizing Data Templates in WPF Step ProgressBar (SfStepProgressBar).....	1355
Item Template.....	1355
Item Template Selector.....	1356
Marker Template Selector .....	1357
Secondary content in Step Progressbar .....	1360
Secondary Content Template Selector .....	1360
Secondary Content Template in Step View Item .....	1362
Layouts in WPF Step ProgressBar (SfStepProgressBar) .....	1363
ItemsStretch.....	1363
MinimumItemSpacing.....	1368
SfColorPalette .....	1369
WPF Color Palette (SfColorPalette) Overview .....	1369
Key features .....	1370
Getting Started with WPF Color Palette (SfColorPalette).....	1370
Assembly deployment.....	1370
Creating Application with SfColorPalette control.....	1370
Select a Color .....	1373
Binding a selected color .....	1373
Navigate to the list of swatches.....	1375
Theme .....	1376
Appearance in WPF Color Palette (SfColorPalette) .....	1377
Setting the Foreground .....	1377
Setting the Background.....	1378
Change flow direction .....	1379
Theme .....	1380
ColorPicker.....	1381
WPF color picker (ColorPicker) Overview .....	1381
Features .....	1382
Getting Started with WPF color picker (ColorPicker) .....	1382
Structure of ColorPicker.....	1382
Assembly deployment.....	1382

Adding WPF ColorPicker via designer .....	1382
Adding WPF ColorPicker via XAML.....	1383
Adding WPF ColorPicker via C#.....	1384
Select a Color .....	1385
Color and Brush changed notification .....	1388
Change opacity of the color .....	1389
Switch between Solid, Linear and Gradient brush mode .....	1389
Theme .....	1391
Select solid color in WPF color picker (ColorPicker) .....	1391
How to select your solid color .....	1392
Select RGB and HSV color .....	1393
Get solid color using Hexadecimal code .....	1395
Pick a color from anywhere (Eye Dropper).....	1395
Select a standard color.....	1395
Solid color changed notification .....	1396
Get color name from color property.....	1397
Select gradient color in WPF color picker (ColorPicker) .....	1398
Create Gradient colors using GradientStops Editor.....	1399
Create Linear Gradient colors .....	1402
Create Radial Gradient colors .....	1406
Reverse the Gradient Colors .....	1411
Show selected gradient color name .....	1412
Show gradient color value editor.....	1413
Switch between Solid, Gradient mode .....	1414
Disable Switching between Solid, Linear and Gradient brush mode at runtime.....	1416
Gradient color changed notification .....	1416
Appearance in WPF color picker (ColorPicker) .....	1417
Change Header Template .....	1417
Change flow direction .....	1418
Setting ToolTip .....	1418
Theme .....	1419
ColorPickerPalette .....	1419
WPF Color Picker Palette Overview .....	1419
Getting Started with WPF Color Picker Palette .....	1420
Control Structure .....	1421

Assembly deployment.....	1422
Adding WPF ColorPickerPalette via designer.....	1422
Adding WPF ColorPickerPalette via XAML.....	1423
Adding WPF ColorPickerPalette via C\#.....	1424
Accessing a Color programmatically .....	1424
Select color from color palette .....	1425
Add your own color in the palette .....	1426
Recently used color items .....	1428
Choosing a color from MoreColor window .....	1429
Reset selected color .....	1430
Selected brush or color changed notification.....	1431
Localization support.....	1432
Theme .....	1432
Working with ColorPickerPalette in WPF .....	1433
Accessing a Color programmatically .....	1433
Accessing a color brush programmatically .....	1433
Setting automatic color.....	1434
Select transparent color programmatically .....	1435
Select a predefined colors.....	1436
Show white and black color variants .....	1439
Add your own colors in the palette .....	1440
Recently used color items .....	1441
Choosing a color from MoreColor window .....	1442
Clear the colour you picked with a transparent colour .....	1445
Selected brush or color changed notification.....	1446
Customize the header .....	1447
Tooltip support.....	1448
Expanded mode .....	1448
ColorPickerPalette as a command button .....	1449
Change color item size .....	1451
Change color palette size .....	1452
Change header and more color icons .....	1453
Hide the drop down button .....	1454
Appearance in WPF Color Picker Palette .....	1455
Change flow direction .....	1455

Theme .....	1456
ComboBoxAdv.....	1457
WPF ComboBox (ComboBoxAdv) Overview .....	1457
Getting Started with WPF ComboBox (ComboBoxAdv).....	1457
Assembly deployment.....	1457
Creating Application with ComboBoxAdv control .....	1457
Creating project .....	1458
Adding control via designer .....	1458
Adding control manually in XAML.....	1458
Adding control manually in C# .....	1459
Adding items in ComboBoxAdv.....	1459
Add items using ComboBoxItemAdv .....	1459
Adding items by DataBinding.....	1460
Defining ItemTemplate .....	1463
Selection.....	1464
Editing .....	1464
Theme .....	1465
See Also.....	1465
Editable Support in WPF ComboBox (ComboBoxAdv).....	1465
IsEditable.....	1465
Adding IsEditable property to an application .....	1465
Auto Complete Support in WPF ComboBox .....	1466
MultiSelection Support in WPF ComboBox (ComboBoxAdv) .....	1466
Properties.....	1467
Adding multiple selections to an application.....	1467
Selecting an item through programmatically .....	1467
Override selected items programmatically.....	1469
Multiselect edit using tokens .....	1471
Watermark Support in WPF ComboBox (ComboBoxAdv) .....	1472
Delimiter String Customization .....	1473
Blendability in WPF ComboBox (ComboBoxAdv) .....	1474
Appearance .....	1475
CurrencyTextBox.....	1476
WPF Currency TextBox Overview .....	1476
Control structure.....	1476

Features .....	1477
Getting Started with WPF Currency TextBox.....	1477
Assembly deployment.....	1477
Adding WPF CurrencyTextBox via designer .....	1477
Adding WPF CurrencyTextBox via XAML .....	1477
Adding WPF CurrencyTextBox via C\# .....	1478
Setting Value .....	1479
Value Changed Notification .....	1480
Min Max Value Restriction.....	1480
Step Interval to increase or decrease the value .....	1481
Formatting the value.....	1481
Setting the Culture .....	1482
Theme .....	1482
Changing Currency Value in WPF Currency TextBox .....	1482
Change currency value by pasting the clipboard's text .....	1484
Show UpDown Button.....	1485
Value Changed Event .....	1485
Setting the Null value.....	1485
Setting Watermark Text.....	1486
Step Interval in WPF Currency TextBox .....	1487
Change Value on Up, Down arrow key .....	1487
Change Value on Mouse Wheel.....	1488
Change Value on Click and Drag .....	1488
Allow or restrict selection on focus .....	1489
Restriction or Validation in WPF Currency TextBox.....	1489
Restrict the value within minimum and maximum value .....	1489
Restrict number of decimal digits .....	1491
Read only mode .....	1492
Culture and Formatting in WPF Currency TextBox .....	1492
Culture based formatting.....	1492
NumberFormatInfo based formatting .....	1493
Formatting with dedicated properties.....	1494
Appearance in WPF Currency TextBox .....	1497
Setting the Foreground .....	1497
Setting the Background.....	1499

Setting the Corner Radius .....	1499
Apply Background for Selection .....	1500
Align Value .....	1500
Setting ToolTip .....	1500
Theme .....	1501
Range Adorner in WPF Currency TextBox.....	1501
Changing background of range-adorner .....	1501
SfDataGrid .....	1502
WPF DataGrid (SfDataGrid) Overview .....	1502
Choose between different Grid's .....	1502
Getting Started with WPF DataGrid (SfDataGrid).....	1505
Assembly deployment.....	1506
Creating simple application with SfDataGrid .....	1506
Defining Columns .....	1511
Selection.....	1513
Sorting, Grouping, and Filtering .....	1513
Editing .....	1516
Theme .....	1516
Data Binding in WPF DataGrid (SfDataGrid) .....	1517
Binding with IEnumerable.....	1517
Binding with DataTable .....	1517
Binding with dynamic data object .....	1518
Binding Complex properties.....	1518
Binding Indexer properties .....	1518
Defining source data type .....	1519
Events.....	1519
View .....	1520
Binding data from WCF service.....	1522
Binding data from ADO.NET Entity Framework .....	1529
Binding data from LINQ to SQL .....	1534
Binding data from ADO.NET.....	1540
Binding data from Microsoft Access .....	1545
How To .....	1546
Columns in WPF DataGrid (SfDataGrid).....	1547
Defining columns .....	1547

Column manipulation.....	1555
DataGrid column resizing.....	1556
DataGrid column drag and drop .....	1557
DataGrid freeze columns .....	1560
Binding column properties with ViewModel .....	1561
Column Types in WPF DataGrid (SfDataGrid) .....	1562
GridColumn .....	1563
Column CellTemplate.....	1564
Column Formatting .....	1570
Column styling.....	1572
End-user interaction .....	1575
Column width, alignment and padding.....	1575
GridTextColumnBase .....	1575
GridEditorColumn .....	1576
GridTextColumn .....	1576
GridNumericColumn .....	1577
GridCurrencyColumn .....	1579
GridPercentColumn.....	1580
GridDateTimeColumn .....	1581
GridCheckBoxColumn .....	1586
GridTemplateColumn.....	1586
GridComboBoxColumn .....	1590
GridMultiColumnDropDownList .....	1596
GridHyperlinkColumn.....	1602
GridImageColumn .....	1603
GridMaskColumn .....	1605
GridTimeSpanColumn .....	1607
GridCheckBoxSelectorColumn .....	1608
Custom column support.....	1612
How To .....	1628
AutoSize Columns in WPF DataGrid (SfDataGrid).....	1629
Fill remaining width for any column instead of last column when ColumnSizer is AutoLastColumnFill or AutoWithLastColumnFill.....	1631
Refreshing autosize calculation at runtime .....	1631
Resetting column width to apply autosize calculation .....	1632

Customizing built-in column auto-sizing logic .....	1632
Auto width calculation based on font settings .....	1633
Star column sizer ratio support.....	1634
Change the width of DataGrid ComboBoxColumn based on it's ItemsSource .....	1636
Stacked Headers in WPF DataGrid (SfDataGrid) .....	1637
Stacked Headers using Data Annotation .....	1638
Adding ChildColumns .....	1639
Removing ChildColumns .....	1639
Changing stacked header row height .....	1640
Sorting in WPF DataGrid (SfDataGrid) .....	1640
Sort column in double click .....	1642
Sorting order .....	1642
Multi column sorting.....	1642
Programmatic sorting .....	1644
Custom sorting .....	1645
Sorting the underlying collection .....	1647
Handling events .....	1648
See Also .....	1648
Grouping in WPF DataGrid (SfDataGrid) .....	1649
UI grouping.....	1649
Programmatic grouping .....	1651
Group based on display text .....	1651
Clearing or removing group .....	1653
Hiding the column when grouped .....	1654
Freezing caption rows when scrolling.....	1655
Expanding or collapsing the groups .....	1656
Customize indent column width .....	1657
GroupDropArea customization .....	1657
Custom grouping.....	1659
Sorting groups based on summary values .....	1662
Grouping events.....	1663
See Also .....	1664
Summaries in WPF DataGrid (SfDataGrid) .....	1665
Table Summary .....	1665
Group Summary .....	1680



Caption Summaries .....	1693
Formatting summary .....	1708
Aggregate Types.....	1711
Calculate summary for selected rows.....	1711
Custom summaries .....	1713
Overriding Summary Renderer .....	1715
See Also .....	1718
Filtering in WPF DataGrid (SfDataGrid).....	1719
Programmatic filtering .....	1719
Excel like UI Filtering .....	1721
Choose between built-in UI Views .....	1722
Advanced Filter UI.....	1724
Performance tips.....	1726
Filtering null values .....	1727
Instant Filtering.....	1729
Filtering based on DisplayText .....	1731
Events.....	1733
Getting the filtered records .....	1734
Show image in CheckBoxFilterControl instead of image path .....	1734
Apply ICollectionView.Filter and DataView.RowFilter on initial loading .....	1736
Functionality Customization .....	1736
Appearance customization .....	1738
See Also .....	1742
Filter Row in WPF DataGrid (SfDataGrid).....	1742
Built-in Editors.....	1743
Filter options .....	1744
Filtering null values .....	1749
Instant Filtering.....	1751
Disable filtering for a particular FilterRowCell .....	1751
Styling.....	1752
Customizing filter row cell .....	1753
Customizing Filter row editors .....	1755
Customizing GridFilterRowMultiSelectRenderer .....	1761
See Also .....	1763
Search in WPF DataGrid (SfDataGrid) .....	1763

Navigating cells based on search text .....	1766
Move CurrentCell when FindNext and FindPrevious .....	1768
Clear Search .....	1768
Search operation on Master-Details View .....	1768
Search customization .....	1769
See Also .....	1773
Editing in WPF DataGrid (SfDataGrid) .....	1774
Edit mode .....	1774
Edit cursor placement .....	1775
Retain editing on lost focus .....	1775
Working with IEditableObject interface .....	1775
Edit events .....	1778
Programmatically edit the cell .....	1780
Cell click events .....	1781
Mouse and Keyboard operations for UIElement inside Template .....	1782
How to .....	1783
See Also .....	1786
Data Validation in WPF DataGrid (SfDataGrid) .....	1787
Built-in validations .....	1787
Built-in validation using IDataErrorInfo / INotifyDataErrorInfo .....	1787
Built-in validation using Data Annotation .....	1789
Cell validation .....	1791
Row validation .....	1791
Data validation error icon customization .....	1792
Data validation error tip (help tip) customization .....	1801
Showing error details in RowHeader .....	1803
Data validation with Master-details view .....	1805
Data validation with checkbox column .....	1810
Show validation errors when using UseDrawing .....	1810
Limitations .....	1814
See Also .....	1814
CRUD Operations in WPF DataGrid (SfDataGrid) .....	1814
Managing data updates .....	1814
Add new rows .....	1816
Delete row .....	1825

Selection in WPF DataGrid (SfDataGrid) .....	1828
Multiple Row or Cell Selection .....	1829
CheckBox column selection .....	1830
Get Selected Rows and Cells .....	1831
Programmatic selection .....	1831
Selection in Master-Details View .....	1835
Scrolling Rows or Columns .....	1842
Mouse and Keyboard Behaviors .....	1843
Events Processed on Selection .....	1845
Appearance .....	1848
Binding Selection Properties .....	1852
Customizing Selection Behaviors .....	1854
See Also .....	1858
Clipboard Operations in WPF DataGrid (SfDataGrid) .....	1859
Copy to Clipboard in DataGrid .....	1859
Paste from Clipboard in DataGrid .....	1860
Cut to Clipboard in DataGrid .....	1861
Events .....	1862
Handling Programmatically .....	1865
Customizing Copy Paste Behavior in WPF DataGrid .....	1867
See Also .....	1870
Master Details View in WPF DataGrid (SfDataGrid) .....	1870
Generating Master-Details view from IEnumerable .....	1871
Generating Master-Details view from DataTable .....	1879
Populating Master-Details view through events .....	1882
Defining properties for DetailsViewDataGrid .....	1884
Defining columns for DetailsViewDataGrid .....	1887
Handling events for DetailsViewDataGrid .....	1890
Column sizing .....	1895
Selection .....	1899
Appearance customization .....	1903
Expanding and collapsing the DetailsViewDataGrid programmatically .....	1906
Hiding expander when parent record's relation property has an empty collection or null .....	1907
Hiding GridDetailsViewIndentCell in SfDataGrid .....	1908
Hiding the details view expander icon based on child items count .....	1909

Change DetailsViewDataGrid ItemsSource at runtime using LiveDataUpdateMode property .....	1910
Refreshing UI while adding records to relation property at run time .....	1911
Handling Events .....	1911
Master-Details View limitations.....	1913
See Also .....	1914
Record Template View in WPF DataGrid (SfDataGrid) .....	1914
Defining row template .....	1915
Defining RowTemplateSelector .....	1917
Height customization .....	1919
Keyboard navigation support for DetailsViewTemplate.....	1920
Populating record template view using events .....	1920
Expanding and collapsing row template programmatically .....	1921
Handling events .....	1921
Limitations.....	1922
Paging in WPF DataGrid (SfDataGrid) .....	1923
Getting started .....	1923
Load data in on-demand .....	1925
Data Virtualization in WPF DataGrid (SfDataGrid).....	1933
Data Virtualization .....	1934
Working with GridVirtualizingCollectionView .....	1934
Incremental Loading .....	1935
Paging.....	1940
Styles and Templates in WPF DataGrid (SfDataGrid).....	1940
Control Structure of SfDataGrid.....	1940
Customizing Default Containers .....	1940
Editing Style in Visual Studio Designer.....	1946
Editing DataGrid Elements Style in Visual Studio Designer .....	1946
Writing Style by TargetType.....	1948
Styling Record cell .....	1948
Styling Record row .....	1950
Alternating Row Style.....	1951
Selection.....	1952
Styling Column Header.....	1953
Setting Default Style for one column.....	1957
Styling CaptionSummary .....	1958

Styling GroupSummary .....	1960
Styling TableSummary.....	1962
Styling UnboundRows .....	1964
Styling AddNewRow.....	1966
Styling RowHeader .....	1967
Template Selectors .....	1968
Styling DetailsViewDataGrid .....	1971
Styling Filter popup .....	1972
Styling Sort icon .....	1972
Styling GroupDropArea .....	1978
Showing busy indicator before loading records .....	1979
Conditional Styling in WPF DataGrid (SfDataGrid).....	1980
Cell style .....	1981
Row style.....	1985
Alternate row style .....	1988
Caption summary cell style .....	1989
Caption summary row style .....	1993
Group summary cell style .....	1998
Group summary row style.....	2004
Table summary cell .....	2008
Table summary row style.....	2014
Table summary cell alignment based on column .....	2017
Row header style.....	2019
Grid Lines customization in WPF DataGrid (SfDataGrid) .....	2020
Record rows .....	2021
Header rows.....	2023
Grid lines for Master-Details view .....	2024
Limitations.....	2025
Themes in WPF DataGrid (SfDataGrid) .....	2025
Rows in WPF DataGrid (SfDataGrid) .....	2025
Row Header.....	2026
Header Row.....	2031
Freeze panes .....	2032
Row Height Customization in WPF DataGrid (SfDataGrid) .....	2036
QueryRowHeight event.....	2037

Fit the Row Height based on its content.....	2038
Reset Row Height at runtime.....	2041
Change HeaderRow Height based on its Content.....	2042
Change StackedHeaderRow Height based on its content .....	2043
Change TableSummaryRow Height.....	2045
Row drag and drop in WPF DataGrid (SfDataGrid) .....	2046
Dragging multiple rows .....	2047
Drag and drop events.....	2048
Changing the row drop indicator .....	2050
Customizing row drag and drop operation .....	2051
Row drag and drop between DataGrid and ListView.....	2055
Row drag and drop between two DataGrid's .....	2058
Row drag and drop between DataGrid and TreeGrid .....	2060
Merge Cells in WPF DataGrid (SfDataGrid) .....	2063
Merging cells .....	2064
Merging cells based on the content.....	2067
Merge cells in Master-Details View .....	2070
Refreshing the merged cells at runtime .....	2072
Exporting merged cells.....	2072
How to.....	2073
Unbound Rows in WPF DataGrid (SfDataGrid) .....	2073
Positioning unbound rows .....	2074
Populating data for unbound rows .....	2075
Refreshing the Unbound Rows at runtime .....	2076
Editing in unbound rows .....	2077
Styling in Unbound rows .....	2077
Customize the Unbound Row's behavior .....	2079
Templating unbound row cells .....	2084
Changing unbound row height .....	2086
Exporting Unbound rows .....	2087
Get unbound rows .....	2087
Merging with Unbound rows .....	2087
Unbound row for Master-details view.....	2088
Unbound Column in WPF DataGrid (SfDataGrid) .....	2090
Populating data for unbound column.....	2091

Refreshing the unbound column at runtime .....	2094
Editing unbound column .....	2094
Styling unbound column .....	2095
Customize the Unbound column behavior .....	2096
Templating unbound column.....	2098
Performance in WPF DataGrid (SfDataGrid) .....	2098
Improving scrolling performance.....	2098
Improving loading performance .....	2098
Improving performance when doing batch updates .....	2099
Adding columns efficiently.....	2100
Optimizing summary calculation performance.....	2100
Improving UI Filter loading time .....	2101
Improving performance while adding multiple FilterPredicates to the column in loop .....	2102
Interactive Features in WPF DataGrid (SfDataGrid) .....	2102
Column Chooser.....	2102
ToolTip in WPF DataGrid (SfDataGrid) .....	2108
Record cell tooltip .....	2108
Header tooltip.....	2109
ToolTip Customization .....	2110
Events.....	2114
Context menu in WPF DataGrid (SfDataGrid) .....	2115
Context menu for record rows.....	2117
Context menu for column header.....	2118
Context menu for group drop area.....	2121
Context menu for group item .....	2122
Context menu for caption summary row .....	2123
Context menu for group summary row .....	2125
Context menu for table summary row.....	2126
Events.....	2128
Customization of context menu.....	2128
Serialization and Deserialization in WPF DataGrid .....	2130
Serialization.....	2130
Serialization options.....	2130
Deserialization.....	2132
Deserialization options .....	2133

Customizing Serialization and Deserialization Operations .....	2135
Export To Excel in WPF DataGrid (SfDataGrid) .....	2139
Excel exporting options.....	2140
Saving options.....	2143
Opening exported excel without saving .....	2144
Export DataGrid pages to Excel.....	2145
Export DataGrid SelectedItems to Excel .....	2146
Export DataGrid to HTML.....	2146
Export DataGrid to Mail .....	2147
Export DataGrid to XML .....	2147
Export DataGrid to CSV .....	2148
Row Height and Column Width customization.....	2148
Styling cells based on CellType in Excel .....	2148
Cell customization in Excel while exporting.....	2149
Customize exported workbook and worksheet.....	2152
Exporting DetailsView .....	2156
Performance .....	2159
Export To PDF in WPF DataGrid (SfDataGrid) .....	2161
PDF exporting options.....	2161
Setting Header and Footer.....	2164
Change PDF page orientation .....	2165
Export DataGrid SelectedItems to PDF .....	2166
Saving options.....	2166
Opening exported PDF without saving .....	2167
Exporting Customization .....	2168
Cell customization in PDF while exporting.....	2170
Exporting DetailsView .....	2174
Printing in WPF DataGrid (SfDataGrid) .....	2179
Print Preview.....	2179
Print Settings.....	2180
Page Settings.....	2183
Setting Header and Footer.....	2186
Different modes of printing for better performance.....	2189
Printing Customization.....	2191
Creating custom PrintPreview window .....	2202



MVVM in WPF DataGrid (SfDataGrid).....	2204
DataGrid SelectedItem Binding.....	2204
DataGrid SelectedItems Binding .....	2205
Button command binding to ViewModel .....	2206
Binding ComboBoxColumn ItemsSource from ViewModel .....	2206
Binding ViewModel ItemsSource to ComboBox inside data template.....	2207
Binding DataGrid Columns from ViewModel.....	2208
Localization in WPF DataGrid (SfDataGrid).....	2209
Localize when the resource file present in different assembly or different namespace?.....	2212
Editing default culture resource .....	2212
UI Automation in WPF DataGrid (SfDataGrid) .....	2214
Coded UI Test.....	2214
Quick Test Professional (QTP).....	2221
Helpers in WPF DataGrid (SfDataGrid) .....	2221
IndexResolver.....	2221
Example: You can find the record index from row index using ResolveToRecordIndex method...	2221
Dispose.....	2222
SfDataPager.....	2222
WPF DataPager (SfDataPager) Overview .....	2222
Getting Started with WPF DataPager (SfDataPager) .....	2223
Control Structure .....	2223
Create Simple Application with SfDataPager .....	2224
Theme .....	2227
DataBinding in WPF DataPager (SfDataPager).....	2228
Source and PagedSource .....	2228
On DemandPaging .....	2230
Page Size .....	2232
How To .....	2233
Appearance in WPF DataPager (SfDataPager).....	2234
AutoEllipsisMode .....	2234
AccentBrush .....	2236
Display Modes.....	2238
Orientation.....	2239
PageNavigation in WPF DataPager (SfDataPager) .....	2239
How To .....	2240

How to Interact with User before Page Changes.....	2240
UIAutomation in WPF DataPager (SfDataPager) .....	2240
Coded UI.....	2240
Quick Test Professional.....	2242
Styles and Templates in WPF DataPager (SfDataPager) .....	2242
Edit Appearance in Expression Blend.....	2242
Edit Appearance in Visual Studio .....	2244
SfSkinManager .....	2245
Getting Started with WPF Skin Manager .....	2245
Themes list .....	2245
Apply a theme to a control .....	2247
Apply a theme globally in the application .....	2249
Customize theme colors and fonts in the application .....	2249
Apply themes to the controls derived from Syncfusion controls .....	2251
Clearing SkinManager instance in an application .....	2252
How to.....	2252
DateTimeEdit .....	2255
WPF DateTimePicker (DateTimeEdit) Overview .....	2255
Key features .....	2255
Getting Started with WPF DateTimePicker (DateTimeEdit) .....	2256
Visual Structure.....	2256
Assembly deployment.....	2256
Creating an application with the DateTimeEdit control .....	2256
Creating a project.....	2256
Adding control via designer .....	2256
Adding control manually in XAML.....	2257
Adding control manually in C# .....	2257
Setting date time value .....	2258
Binding date time value .....	2259
Value Changed Notification .....	2260
Applying built-in pattern.....	2260
Applying custom pattern .....	2261
Editing date time.....	2262
Restrict date range.....	2262
Show watermark when value is null .....	2263

Change month names .....	2263
Change week day names.....	2265
Block particular dates .....	2266
Theme .....	2267
DateTime Editing in WPF DateTimePicker (DateTimeEdit).....	2267
Mask editing.....	2267
DateTime field navigation .....	2268
Change date time programmatically .....	2269
Binding date time value .....	2270
Free form editing .....	2270
Change date time on mouse wheel .....	2271
Change date time using up-down button .....	2271
Disable up-down button .....	2272
Customize up-down appearance .....	2272
Change month using alpha keys .....	2274
Delete and edit the date time value .....	2274
Change date time using custom calendar and clock.....	2275
Setting the null value .....	2277
Show watermark when value is null .....	2278
Select the date time field on focus .....	2279
Restrict automatic focus to next field.....	2279
Value Changed Notification .....	2280
Restricting date value in WPF DateTimePicker (DateTimeEdit) .....	2280
Restrict the datetime within minimum and maximum datetime.....	2280
Restrict date selection .....	2282
Block particular dates .....	2282
ReadOnly support .....	2283
DateTime formatting in WPF DateTimePicker (DateTimeEdit) .....	2284
Predefined display patterns.....	2284
Change datetime format.....	2286
Custom display pattern .....	2286
Change culture .....	2287
Dropdown Popup in WPF DateTimePicker (DateTimeEdit) .....	2288
Open and close the datetime selector popup using short-cut keys .....	2289
Dropdown date time selector .....	2289

Custom drop down date time selector .....	2290
Custom UI for drop down button .....	2290
Select today date .....	2290
Reset the selected date .....	2292
Change month names .....	2292
Change weekday names .....	2293
Open the popup date time selector with time delay.....	2294
Disable dropdown date time selector.....	2295
Hide Today button .....	2295
Appearance in WPF DateTimePicker (DateTimeEdit) .....	2296
Setting the Foreground .....	2296
Setting the Background.....	2296
Change focus border color.....	2297
Change flow direction .....	2297
Theme .....	2297
Localization in WPF DateTimePicker (DateTimeEdit) .....	2298
UI Automation in WPF DateTimePicker (DateTimeEdit).....	2300
Quick Test Professional (QTP).....	2300
Coded UI.....	2301
SfDatePicker.....	2301
WPF DatePicker (SfDatePicker) Overview .....	2301
Getting Started with WPF DatePicker (SfDatePicker).....	2302
Structure of SfDatePicker.....	2303
Assembly deployment.....	2303
Add control through designer.....	2303
Adding control manually in XAML.....	2304
Add control manually in C\# .....	2304
Setting the Date .....	2305
Date changed notification.....	2306
Display the date using the FormatString .....	2306
Specifying format for the DateSelector .....	2306
Set selected value on lost focus.....	2307
Localization support.....	2308
Theme .....	2308
Setting Date in WPF DatePicker (SfDatePicker) .....	2309

Setting Date using property .....	2309
Setting Null Value.....	2309
Setting WaterMark text .....	2310
Set selected value on lost focus.....	2311
Setting the date using editing .....	2311
Setting the Input Scope for the On-Screen Keyboard.....	2312
Restrict selecting date limit .....	2312
Date changed notification.....	2313
Date Formatting in WPF DatePicker (SfDatePicker) .....	2314
Display the date using the FormatString .....	2314
Specifying format for the DateSelector .....	2314
Customizing DropDown in WPF DatePicker (SfDatePicker).....	2315
Change DropDown height.....	2315
Show or hide DropDown button.....	2316
Date Selector in WPF DatePicker (SfDatePicker) .....	2316
Change the Cell templates .....	2317
Change the DayCell Template.....	2317
Change the MonthCell Template.....	2318
Change the YearCell Template.....	2319
Change size of cells .....	2320
DateSelector item spacing .....	2321
Appearance in WPF DatePicker (SfDatePicker) .....	2322
Setting the Foreground .....	2322
Setting the Background.....	2323
Change flow direction .....	2324
Theme .....	2325
SfDiagram.....	2326
WPF Diagram (SfDiagram) Overview .....	2326
Key features of SfDiagram are as follows: .....	2327
Getting Started with WPF Diagram (SfDiagram).....	2328
Assembly deployment.....	2328
Creating the project .....	2328
Basic Diagram elements.....	2330
Flow chart .....	2330
Organization layout.....	2342

Theme .....	2345
Node in WPF Diagram (SfDiagram) .....	2346
Create node.....	2346
Visualize a node .....	2348
Position .....	2354
Flip.....	2355
Padding .....	2356
Appearance .....	2357
Interaction.....	2358
Events.....	2361
Constraints .....	2361
See Also .....	2361
Shapes in WPF Diagram (SfDiagram) .....	2362
Basic Shapes.....	2362
BPMN Shapes in WPF Diagram (SfDiagram) .....	2375
Connector in WPF Diagram (SfDiagram).....	2379
Connector types .....	2379
Create connector .....	2382
Create connectors through connection points.....	2382
Create connection between nodes.....	2382
Connections with ports .....	2384
Draw connectors .....	2386
Connectors through data source .....	2386
Add connectors from stencil.....	2386
See Also .....	2386
Port in WPF Diagram (SfDiagram).....	2387
Connections .....	2387
Node port.....	2389
Connector port.....	2393
Dock port.....	2395
Padding .....	2397
HitPadding.....	2398
Connection direction.....	2400
Appearance .....	2401
Events.....	2402

Constraints .....	2402
See Also .....	2402
Annotation in WPF Diagram (SfDiagram) .....	2402
Define annotation .....	2402
Multiple Annotations .....	2404
Group in WPF Diagram (SfDiagram) .....	2405
Create Group.....	2405
Interaction.....	2406
See Also .....	2406
Swimlane in WPF Diagram (SfDiagram) .....	2406
Create a swimlane.....	2407
Orientation.....	2412
Interaction.....	2413
Gridlines in WPF Diagram (SfDiagram) .....	2414
Change grid lines appearance.....	2416
Change grid spacing .....	2418
Snapping in WPF Diagram (SfDiagram).....	2420
Snap-to-objects.....	2420
How to change the snap indication style.....	2426
Snap to Lines .....	2427
Page Settings in WPF Diagram (SfDiagram) .....	2428
How to enable the multiple page .....	2430
How to change the page appearance .....	2431
How to change the margin around the pages .....	2433
How to change the unit of the page .....	2434
How to change the scaling of Page .....	2436
How to customize the page origin .....	2437
Scroll-Settings in WPF Diagram (SfDiagram).....	2442
Get current scroll status.....	2442
How to update the scroll status.....	2443
AutoScroll.....	2444
Autoscroll border .....	2445
Interaction.....	2446
Selection in WPF Diagram (SfDiagram).....	2446
Quick Command in WPF Diagram (SfDiagram).....	2452

Keyboard support in WPF Diagram.....	2456
Dragging based on DragLimit in WPF Diagram (SfDiagram) .....	2458
Serialization in WPF Diagram (SfDiagram) .....	2460
Save .....	2461
Load.....	2461
How to serialize custom properties .....	2462
How to serialize a custom class .....	2463
How to load SfDiagram's old version in new version.....	2463
Virtualization in WPF Diagram (SfDiagram) .....	2463
Deferred Scrolling .....	2464
Outline customization .....	2465
Rulers in WPF Diagram (SfDiagram).....	2467
Define Rulers.....	2467
Customizing the Ruler .....	2468
Commands .....	2468
Alignment Commands in WPF Diagram (SfDiagram).....	2468
Spacing Commands in WPF Diagram (SfDiagram) .....	2471
Sizing Commands in WPF Diagram (SfDiagram) .....	2473
Clipboard Commands in WPF Diagram (SfDiagram) .....	2474
Grouping Commands in WPF Diagram (SfDiagram) .....	2476
Flip Command in WPF Diagram (SfDiagram) .....	2477
Z-Order Commands in WPF Diagram (SfDiagram) .....	2478
Zoom Command in WPF Diagram (SfDiagram).....	2480
Undo Redo Commands in WPF Diagram (SfDiagram) .....	2485
Nudge Commands in WPF Diagram (SfDiagram) .....	2486
FitToPage Command in WPF Diagram (SfDiagram) .....	2487
Expand Collapse Command in WPF Diagram (SfDiagram).....	2490
Rotate Command in WPF Diagram (SfDiagram) .....	2491
SelectByType Command in WPF Diagram (SfDiagram).....	2492
SelectTool Command in WPF Diagram (SfDiagram) .....	2493
SetShapeStyle Commands in WPF Diagram (SfDiagram).....	2495
Edit and Format Text Commands in WPF Diagram (SfDiagram).....	2496
Command Manager in WPF Diagram (SfDiagram).....	2498
Undo and Redo in WPF Diagram (SfDiagram).....	2500
Undo and Redo actions .....	2500



History actions .....	2501
Entry mode.....	2501
Stack limit.....	2502
Start group action .....	2502
End group action .....	2502
How to view Undo/Redo stack .....	2503
How to log custom actions in undo/redo stack .....	2505
How to restrict Undo/Redo.....	2508
History changed event .....	2510
Events for Undo Redo Process .....	2510
See Also .....	2511
DataSource in WPF Diagram (SfDiagram) .....	2511
Defining DataSource .....	2511
Defining layout.....	2513
Root.....	2514
Layout with multiple parents.....	2515
FlowchartDataSourceSettings.....	2516
Automatic Layout in WPF Diagram (SfDiagram) .....	2520
Defining layout.....	2523
Updating layout.....	2524
Customize spacing between nodes in layout.....	2525
Customize tree orientation in layout .....	2526
Avoiding connector segment overlapping in layout .....	2527
Customize margin in layout .....	2528
See Also .....	2528
Overview of Essential WPF Diagram (SfDiagram) .....	2528
Usage scenario .....	2529
Define overview .....	2529
ZoomSlider .....	2530
Interaction.....	2530
Deferred scrolling.....	2533
Event .....	2533
Stencil in WPF Diagram (SfDiagram) .....	2533
Add symbols in a Stencil .....	2535
Using the Diagram Elements.....	2535

Using the SymbolViewModel .....	2537
Constraints .....	2539
See also .....	2539
Printing in WPF Diagram (SfDiagram) .....	2539
Direct print .....	2539
Print preview .....	2540
Print settings .....	2540
Header and Footer .....	2545
Skip empty pages .....	2546
Printing event .....	2547
Custom paper size .....	2547
Classic PrintPreview .....	2550
See Also .....	2551
Exporting in WPF Diagram (SfDiagram) .....	2551
Export settings .....	2552
See Also .....	2556
Localization in WPF Diagram (SfDiagram) .....	2556
Localize the Annotations using ResourceManager .....	2556
Tools in WPF Diagram (SfDiagram) .....	2558
Tool Selection .....	2558
Drawing Tools .....	2558
How to override the default tool of diagram elements .....	2562
How to override the default cursors while interaction .....	2563
Constraints in WPF Diagram (SfDiagram) .....	2565
Graph Constraints .....	2565
Node Constraints .....	2567
Connector Constraints .....	2569
Port Constraints .....	2571
Annotation Constraints .....	2572
Selector Constraints .....	2574
Snap Constraints .....	2575
Bitwise Operations .....	2576
Context Menu in WPF Diagram (SfDiagram) .....	2577
Default Context Menu .....	2577
Customize Context Menu .....	2577

Events.....	2578
See Also .....	2579
Diagram Ribbon in WPF Diagram (SfDiagram).....	2579
Assembly Deployment .....	2580
Adding Diagram Ribbon .....	2580
Simple Diagram Designer.....	2581
Ribbon Customization .....	2583
SfHeatMap .....	2584
WPF HeatMap (SfHeatMap) Overview .....	2584
Getting Started with WPF HeatMap (SfHeatMap).....	2585
Initialize the SfHeatMap.....	2585
Prepare data.....	2586
Populate data .....	2586
Map data into SfHeatMap.....	2587
Color Mapping.....	2588
Legend.....	2589
Theme .....	2591
Items Mapping in WPF HeatMap (SfHeatMap) control.....	2592
Color Mapping in WPF HeatMap (SfHeatMap) control .....	2592
Legend in WPF HeatMap (SfHeatMap) control .....	2593
Create Legend .....	2593
Legend Mode .....	2593
Orientation.....	2594
SfDigitalGauge.....	2595
WPF Digital Gauge (SfDigitalGauge) Overview .....	2595
Key Features in WPF Digital Gauge (SfDigitalGauge).....	2595
Getting Started with WPF Digital Gauge (SfDigitalGauge).....	2595
Adding gauge references .....	2595
Initialize gauge .....	2596
Displaying Values .....	2596
Setting character type.....	2597
Configuring properties .....	2597
Theme .....	2598
See also .....	2598
Digital Characters in WPF Digital Gauge (SfDigitalGauge) .....	2598

7-Segments .....	2599
14-Segments .....	2599
16-Segments .....	2600
8*8 Dot Matrix Segments .....	2600
Transformation of Characters in WPF Digital Gauge (SfDigitalGauge) .....	2601
Scaling .....	2601
Skewing .....	2602
Settings in WPF Digital Gauge (SfDigitalGauge) .....	2603
Character Spacing .....	2603
Character Stroke .....	2603
Segment Thickness .....	2604
RTL (Right to Left) support .....	2604
Dimmed Brush stroke .....	2605
Dimmed Brush opacity .....	2605
DockingManager .....	2606
WPF Docking (DockingManager) Overview .....	2606
Key features .....	2606
Getting Started with WPF Docking (DockingManager) .....	2607
Assembly deployment .....	2607
Creating Application with DockingManager control .....	2607
Creating project .....	2608
Adding control via designer .....	2608
Adding control manually in XAML .....	2608
Adding control manually in C# .....	2609
Set Header for each child window .....	2610
Set States for each child window .....	2611
Set Sides for children .....	2612
Save / Load the layout .....	2613
Set Visual Styles .....	2614
ToolTip for child window .....	2615
Theme .....	2616
Docking Window in WPF Docking (DockingManager) .....	2617
Configuring window in Different Sides .....	2618
Docking window in various Targets .....	2619
Maximize/Minimize Support .....	2620

Hot Drag the window .....	2624
Enabling or Disabling the Dock functionality .....	2624
Enabling or Disabling the Header Visibility .....	2625
Custom context menu items for docking window .....	2625
Auto Hide Window in WPF Docking (DockingManager) .....	2628
Configuring window in Different Side .....	2628
Side panel Customization.....	2630
Excel-like Scrollable panel .....	2632
Changing pinning behavior .....	2633
Enabling and disabling the AutoHide functionality.....	2636
Change AutoHide behavior like Visual Studio 2013 .....	2638
AutoHide Animation enabled on Mouse Click .....	2639
Allow or restrict dragging the AutoHide Window.....	2639
Pinning / UnPinning All Window .....	2640
Changing dock state using context menu .....	2641
Floating Window in WPF Docking (DockingManager) .....	2641
Rolling Up support .....	2642
Displaying Float Windows in Taskbar .....	2643
Multiple Monitor functionalities.....	2644
Enabling or Disabling the float functionality.....	2644
Enabling and Disabling the float functionality Operation on Double Click.....	2645
Maximize/Minimize float window .....	2645
Float or Maximize on double clicking the header .....	2646
Positioning on desire location.....	2647
Snapping Float window .....	2647
Custom context menu items for floating window .....	2648
Tabbed Window in WPF Docking (DockingManager) .....	2651
Tab alignments.....	2651
Closing a Tabbed window .....	2655
Tabbed window order changed notification.....	2656
Restrict tabbed window reordering .....	2657
Data Binding in WPF Docking (DockingManager) .....	2659
Adding Docking Window child through ItemsSource: .....	2659
Docking Window in Different side .....	2660
Configure the Docking window through ItemsSource.....	2661

Dealing with Windows in WPF Docking (DockingManager) .....	2662
Activating a window.....	2662
Adding Window Programmatically .....	2663
Hiding Window Programmatically .....	2667
Restore Window Programmatically .....	2668
Detect the closing of a DockingManager child .....	2668
Removing Window Programmatically .....	2669
Event to notify when a child is added or removed .....	2669
Restricting Docking in Float Window .....	2671
Customizing a window .....	2671
Customizing FloatWindow .....	2672
Enable/Disable Dragging a Window.....	2673
Drag Shadow of a Window.....	2673
Drag Border of a Window .....	2674
Customizing a resizing behaviors.....	2675
Configuring window sizing .....	2678
Occupy whole window.....	2683
Applying Context Menu .....	2686
DockBehavior .....	2691
Hosting a client control between windows .....	2693
DockingManager as FlatLayoutControl.....	2694
MDI/TDI functionalities in WPF Docking (DockingManager).....	2695
Setting MDI Window state.....	2697
Getting state of the MDI window .....	2699
Detecting the maximized state of the MDI window .....	2699
Resizing MDI.....	2699
Closing TDI tab items on mouse middle click.....	2699
Different Keyboard Navigation Modes .....	2700
Setting MDI Layout.....	2704
Closing a MDI Windows .....	2706
Indexing an Item in TDI .....	2707
Drag / Drop support in TDI.....	2708
Rearrange position of document items with auto scrolling .....	2708
TDI window's order changed notification .....	2709
Restrict TDI window reordering .....	2710

Customizing Close Menu.....	2711
Creating Document Tab Group .....	2712
Disable TabGroups .....	2714
VS2010 Behavior of TDI.....	2714
TDI Header Renaming Support .....	2716
Hiding TDI Header .....	2716
Add New button in Header Panel .....	2717
Pin and Unpin tab items.....	2717
Custom context menu items for tab item.....	2724
Custom tab list context menu item .....	2726
Interaction with DragProvider in WPF Docking (DockingManager) .....	2729
Dock a window to desired direction .....	2730
Restrict docking by disabling inner and outer dock hints .....	2731
Restrict outer dockability.....	2741
Restrict docking at run-time .....	2741
Linked Manager and Nested Docking in WPF Docking .....	2743
Linked Manager .....	2743
Nested Docking .....	2746
Hosting in WPF Docking (DockingManager) .....	2747
Hosting a Windows Form control .....	2747
Interaction with control hosted by Win32 Host .....	2748
State Persistence in WPF Docking (DockingManager).....	2748
Auto Save / Load functionalities .....	2749
Manipulating Save / Load functionalities .....	2750
Serialize a complex layout.....	2750
Serialize the dynamically added children .....	2750
Various formats to Save / Load states .....	2750
Restrict state persistence for specific child .....	2752
Styling and Templates in WPF Docking (DockingManager) .....	2753
Theme .....	2753
Dock Window Style .....	2753
Float Window Style .....	2765
Auto Hide Window Style .....	2768
Drag Provider Style .....	2772
DocumentTabControlStyle.....	2777

DocumentTabItemStyle .....	2778
DocumentMDIHeaderStyle .....	2779
TabControl style .....	2780
Patterns and Practices in WPF Docking (DockingManager) .....	2782
MVVM .....	2782
MVVMLight .....	2785
Practice with PRISM .....	2786
Configuring DockingManager with Prism 6.1 .....	2790
Configuring DockingManager with Prism 7.1 .....	2794
DocumentContainer.....	2801
WPF Tabbed MDI Form (DocumentContainer) Overview .....	2801
Features .....	2801
Getting Started with WPF Tabbed MDI Form (DocumentContainer).....	2801
Assembly deployment.....	2802
Create a simple application with DocumentContainer.....	2802
Create a project .....	2802
Add control through designer.....	2802
Add control manually in XAML .....	2802
Add control manually in C\# .....	2803
Add document windows .....	2803
Set header to document .....	2804
Set TDI/MDI document mode.....	2805
Minimizing MDI window .....	2807
Theme .....	2807
Adding and Removing Items from the WPF Tabbed MDI Form control.....	2809
Adding items .....	2809
Remove item.....	2809
setting mode for document container in WPF Tabbed MDI Form .....	2809
Setting Window State in WPF Tabbed MDI Form.....	2810
Setting Window State .....	2810
Notify event for MDIWindow State Changes.....	2811
Setting MDIBounds in WPF Tabbed MDI Form (DocumentContainer).....	2812
Setting Header of the Document container in WPF Tabbed MDI Form .....	2812
Maximizing MDI window in WPF Tabbed MDI Form (DocumentContainer).....	2813
Restrict Maximizing the MDI window .....	2814



Minimizing MDI window in WPF Tabbed MDI Form (DocumentContainer) .....	2815
Restrict Minimizing the MDI window .....	2815
MDI Resize in WPF Tabbed MDI Form (DocumentContainer) .....	2816
Setting Window Switchers in WPF Tabbed MDI Form .....	2816
State Persistence in WPF Tabbed MDI Form (DocumentContainer) .....	2818
Load and Save in Registry .....	2818
Load and Save in Isolated Storage .....	2818
Load and Save in XML .....	2818
Load and Save in Bin .....	2819
Reset the states in Document Container .....	2819
Delete the State .....	2820
Disabling TDI items Drag-Drop in WPF DockingManager & Tabbed MDI Form .....	2820
Rearrange position of document items with auto scrolling .....	2820
TDI item's order changed notification in DocumentContainer .....	2821
Restrict TDI item reordering in DocumentContainer .....	2822
Creating Tab Groups in WPF Tabbed MDI Form (DocumentContainer) .....	2823
Creating tab groups using context menu item .....	2823
Creating tab groups using tabitem dragging .....	2824
Creating tab groups programmatically .....	2825
Layout Related Features in WPF Tabbed MDI Form (DocumentContainer) .....	2826
Theme .....	2826
Pin and Unpin TabItems in WPF Tabbed MDI Form (DocumentContainer) .....	2828
Enabling/disabling pinning behavior .....	2828
Pin and Unpin tab items using PinButton .....	2829
Pin and Unpin the tab items programmatically .....	2832
Pin and Unpin tab items through ContextMenu .....	2832
Full Screen in DocumentContainer in WPF Tabbed MDI Form .....	2836
Toolbar in DocumentContainer .....	2837
SizeToContent for MDI Window in DocumentContainer .....	2838
Localization in WPF Tabbed MDI Form (DocumentContainer) .....	2838

## Welcome to Syncfusion WPF UI controls

Syncfusion Essential Studio for WPF is a comprehensive collection of over 90+ essential WPF controls like DataGrid, Chart, Diagram, and PDF Viewer for building powerful line-of-business Windows applications faster. Syncfusion WPF controls provides unparalleled performance, stunning built-in themes, touch-friendly UI, localization and seamless integration with Visual Studio.

### How to best read this user guide

- The best way to get started would be to read the “Getting Started” section of the documentation for the component that you would like to start using first. The “Getting Started” guide gives just enough information that you need to know before starting to write code. This is the only section that is recommended to be read end-to-end before starting to write code, since all other information can be referred when needed.
- Now that you are familiar with the basics of using the component, the next step would be to start integrating the component into your application. A good starting point is to refer to the code examples in the online or offline sample browser that contains hundreds of code examples, as it is very likely that you may find a code example that resembles your intended usage scenario.
- After integrated the component into application using one of the code examples as a starting point, it is likely that you may want additional information on specific features and API. The best option is to search the specific topic using the search box that is available at the top of the user guide.
- Another valuable resource is the API reference that provides detailed information on the object hierarchy as well as the settings available on every object.

### Controls List

---

**Note:** The name of the Visual Studio Toolbox entry provided when the common control name and toolbox name differs.

---

<style>

table

{

border:0 !important;

line-height: 2!important;

}

tr

{

border:0 !important;

}

td

{

border:0 !important;

}

anchor

{

text-decoration: none!important;

font-family: Caros!important;

font-size: 14px!important;

color: #0079F3!important;

letter-spacing: 0.47px!important;

text-align: left!important;

}

title

{

font-family: CarosMedium!important;

font-size: 14px!important;

color: #22252A!important;

letter-spacing: 0.47px!important;

text-align: left!important;

font-weight: bold!important;

border:0 !important;

background-color:transparent!important;

}

</style>

GRIDS	LAYOUT	NAVIGATION	INPUT CONTROLS
<a href="#">DataGrid</a>	<a href="#">Docking</a>	<a href="#">TabControl</a>	<a href="#">MaskedTextBox</a>
<a href="#">TreeGrid</a>	<a href="#">Carousel</a>	<a href="#">Ribbon</a>	<a href="#">Color Picker</a>
<a href="#">PropertyGrid</a>	<a href="#">Card View</a>	<a href="#">TreeView</a>	<a href="#">Color Palette</a>
<a href="#">Excel-like Grid</a>	<a href="#">Tabbed MDI Form</a>	<a href="#">BreadCrumb</a>	<a href="#">Color Picker Palette</a>
<a href="#">DataPager</a>	<a href="#">Chromeless Window</a>	<a href="#">Accordion</a>	<a href="#">Currency TextBox</a>
	<a href="#">Tab Splitter</a>	<a href="#">Navigation Pane</a>	<a href="#">Integer TextBox</a>
DATA VISUALIZATION	<a href="#">GridSplitter</a>	<a href="#">Navigation Drawer</a>	<a href="#">Double TextBox</a>
<a href="#">Charts</a>	<a href="#">Tile View</a>	<a href="#">Menu</a>	<a href="#">Percent TextBox</a>

<a href="#">Diagram</a>		<a href="#">TaskBar</a>	<a href="#">Domain Updown</a>
<a href="#">Map</a>	FILE VIEWERS & EDITORS	<a href="#">Tab Navigation</a>	<a href="#">NumericUpdown</a>
<a href="#">Surface Chart</a>	<a href="#">PDF Viewer</a>	<a href="#">Radial Menu</a>	<a href="#">Radial Slider</a>
<a href="#">Smith Chart</a>	<a href="#">ImageEditor</a>	<a href="#">Toolbar</a>	<a href="#">Range Slider</a>
<a href="#">Gantt</a>	<a href="#">Spreadsheet</a>	<a href="#">Tree Navigator</a>	<a href="#">Rating</a>
<a href="#">Radial Gauge</a>	<a href="#">RichTextBox</a>	<a href="#">Wizard</a>	<a href="#">Calculator</a>
<a href="#">Digital Gauge</a>	<a href="#">Syntax Editor</a>		
<a href="#">Linear Gauge</a>			
<a href="#">Kanban Board</a>	FILE FORMAT FRAMEWORKS		
<a href="#">Barcode</a>	<a href="#">Excel</a>		
<a href="#">Bullet Graph</a>	<a href="#">PDF</a>		
<a href="#">Sunburst Chart</a>	<a href="#">Word</a>		
<a href="#">Sparkline</a>	<a href="#">PowerPoint</a>		
<a href="#">Range Selector</a>			
<a href="#">TreeMap</a>			

---

**Note:** Using Classic labeled controls in new projects is not recommended. In Classic labeled controls, new features and enhancements will not be included. You can only use it if the current control features meets the requirements of your application.

---

## Licensing

Refer [licensing documentation](#) to learn about registering Syncfusion license key in your WPF application to use Syncfusion controls without license message dialog at runtime.

## Additional help resources

The [Knowledge Base](#) section contains responses to some of the most common questions that other customers have asked in the past, so this would be a good place to search for topics that are not covered in the user guide.

Similar to the [Knowledge Base](#), the [Forum](#) section also contains responses to questions that other customers have asked in the past.

## Support and feedback

If you are unable to find the information that you are looking for in the self-help resources mentioned above then you contact us by creating a [support ticket](#).

Don't see what you need? Please request it in our [feedback portal](#).

## Installation and Upgrade

### System Requirements for WPF

The system requirements for using our Syncfusion WPF platform are as follows

#### *Operating Systems*

- Windows XP
- Windows Vista SP2
- Windows 7 SP1
- Windows 8, 8.1
- Windows 10

#### *Hardware Environment*

- Processor: x86 or x64
- RAM : 512 MB (minimum), 1 GB (recommended)
- Hard disc: up to 3 GB of free space may be required

#### *Development Environment*

- Microsoft Visual Studio 2008/2010/2012/2013/2015/2017/2019/2022
- .NET Framework 3.5/4.0/4.5/4.5.1/4.6
- .NET Core 3.1
- .NET 5.0

### Download WPF Installer

The Syncfusion WPF installer can be downloaded from the [Syncfusion.com](https://www.syncfusion.com) website. You can either download the licensed installer or try our trial installer depending on your license.

- Trial Installer - Licensed Installer

#### Download the Trial Version

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

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

#### *Download Free Trial Setup*

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

### Thank you for choosing to evaluate Essential Studio for WPF!

Please choose your preferred evaluation option to proceed.

#### Full installation evaluation experience

Download and install the evaluation version of the full product along with all the required assemblies and demo code.

Choose Download Format

☒ Web Installer ☐ Offline Installer

[Download](#)

#### NuGet evaluation experience

No installation required. Get started in 2 minutes

[Install assemblies from NuGet](#)

&

Download demo code from [GitHub](#) | Open demo code in [Visual Studio](#)

Version: 19.2.0.44, Released: June 30, 2021

[Release Notes](#) | [License Agreement](#) | [System Requirements](#) | [Installation Instructions](#)

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

## Trial Downloads and Unlock Keys

Essential Studio

Trial SKU Name

Trial Version :

[What's New](#) | [Generate License File](#) ? | [Get Unlock Key](#) ?

[Download](#) 1

[More Download Options](#) 2

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

## Downloads


Version : 19.2.0.44  
Released on : Jun 30, 2021


Windows Mac


GENERAL INFORMATION NEW

DesktopOffline InstallerWeb Installer

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

 .EXE 634 MB  
Checksum Value

 .ZIP 634 MB  
Checksum Value

 .EXE 3 MB  
Checksum Value

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

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

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

### DESKTOP

WinForms  
Includes 100+ enterprise-class WinForms controls for developing modern desktop applications.

START TRIAL

WPF  
Includes 100+ enterprise-class WPF controls for developing modern desktop applications.

START TRIAL

WinUI (Preview)  
Syncfusion WinUI platform is a set of ever-growing UI controls for developing rich UWP apps.

START TRIAL

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

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

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

## Trial Downloads and Unlock Keys

With version 18.1.0.36 (2020 Volume 1 beta), we're changing the Syncfusion Blazor namespace, component names, and NuGet package name. Please refer to this [help topic](#) for more information.

WPF - Trial License [REDACTED]

Trial Version : [REDACTED]

[What's New](#) | [Release Notes](#) | [Get License Key](#) ⓘ | [Get Unlock Key](#) ⓘ

Download 1

More Download Options 2

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

### Download the License Version

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

## License Downloads and Unlock Keys

License SKU Name	
Essential Studio [REDACTED]	<a href="#">Download Older Versions</a> 2
Latest Official Release : [REDACTED]	<div>Download 1</div> <div>More Download Options 3</div>
<a href="#">What's New</a>   <a href="#">Generate License File</a> ⓘ   <a href="#">Get Unlock Key</a> ⓘ	

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

### Installation using Web Installer

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

#### Overview

For the Essential Studio WPF product, Syncfusion offers a Web Installer. This installer alleviates the burden of downloading a larger installer. You can simply download and run the online installer, which

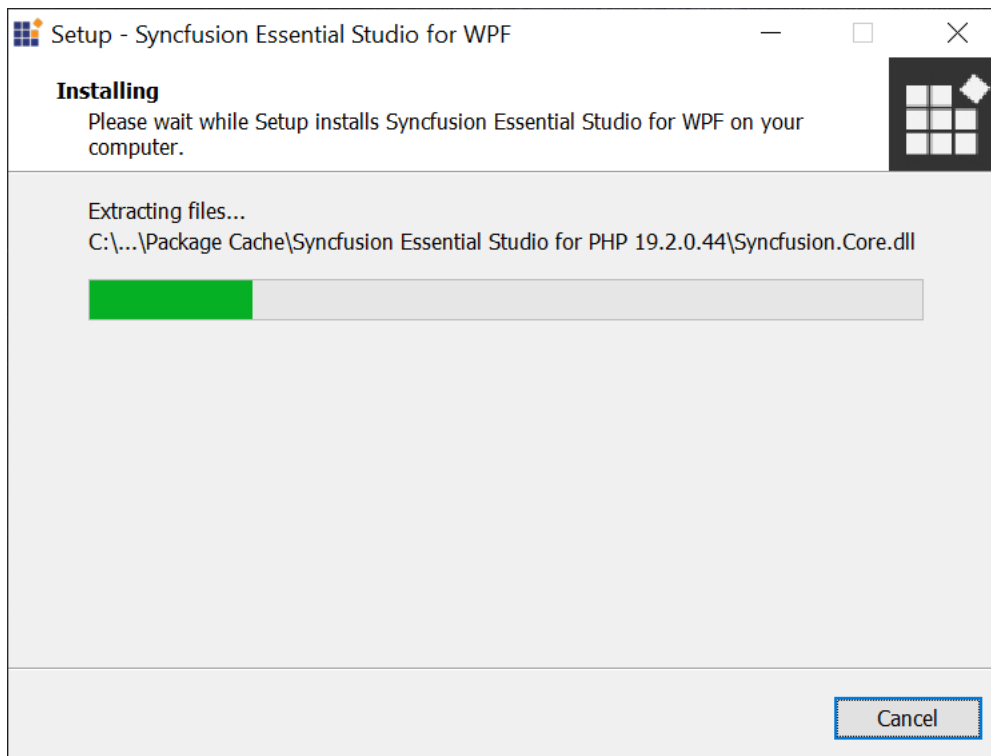


will be smaller in size and will download and install the Essential Studio products you have chosen. You can get the most recent version of Essential Studio Web Installer [here](#).

### Installation

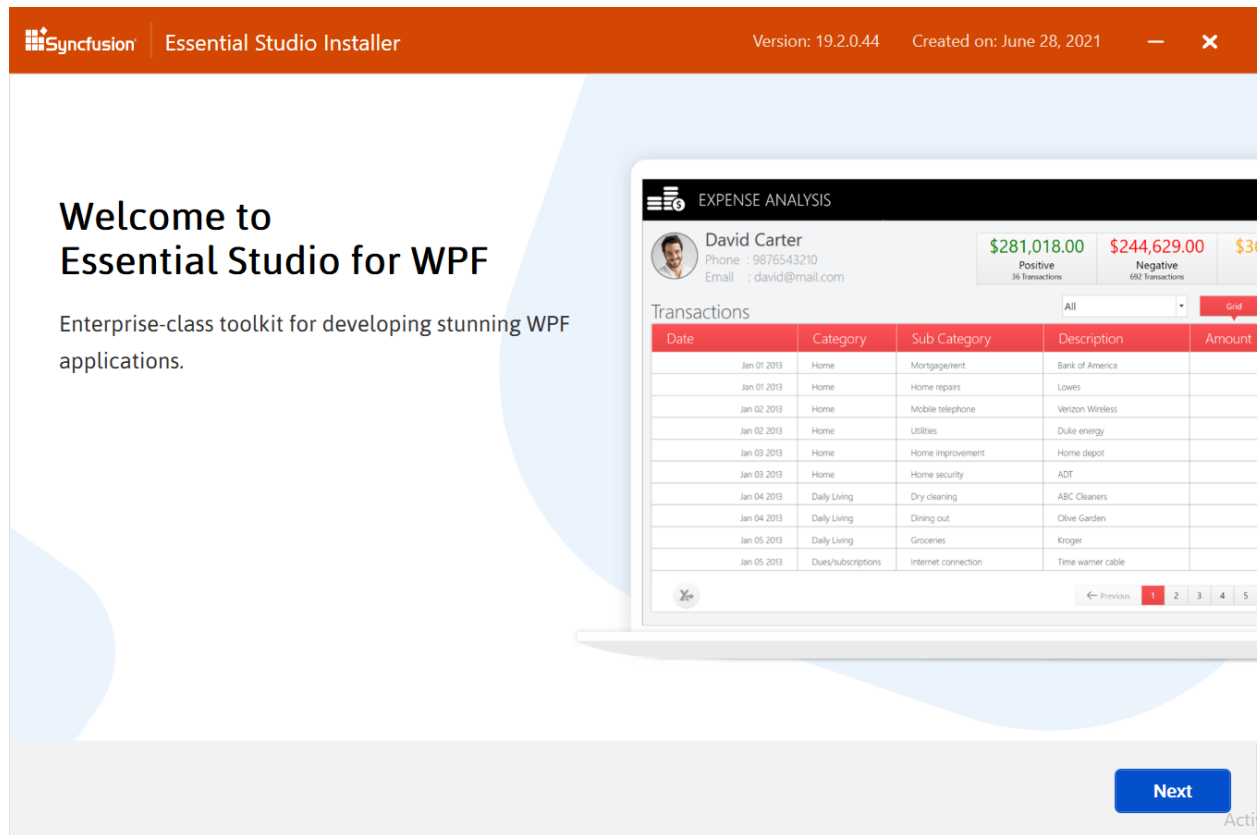
The steps below show how to install Essential Studio WPF Web Installer.

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



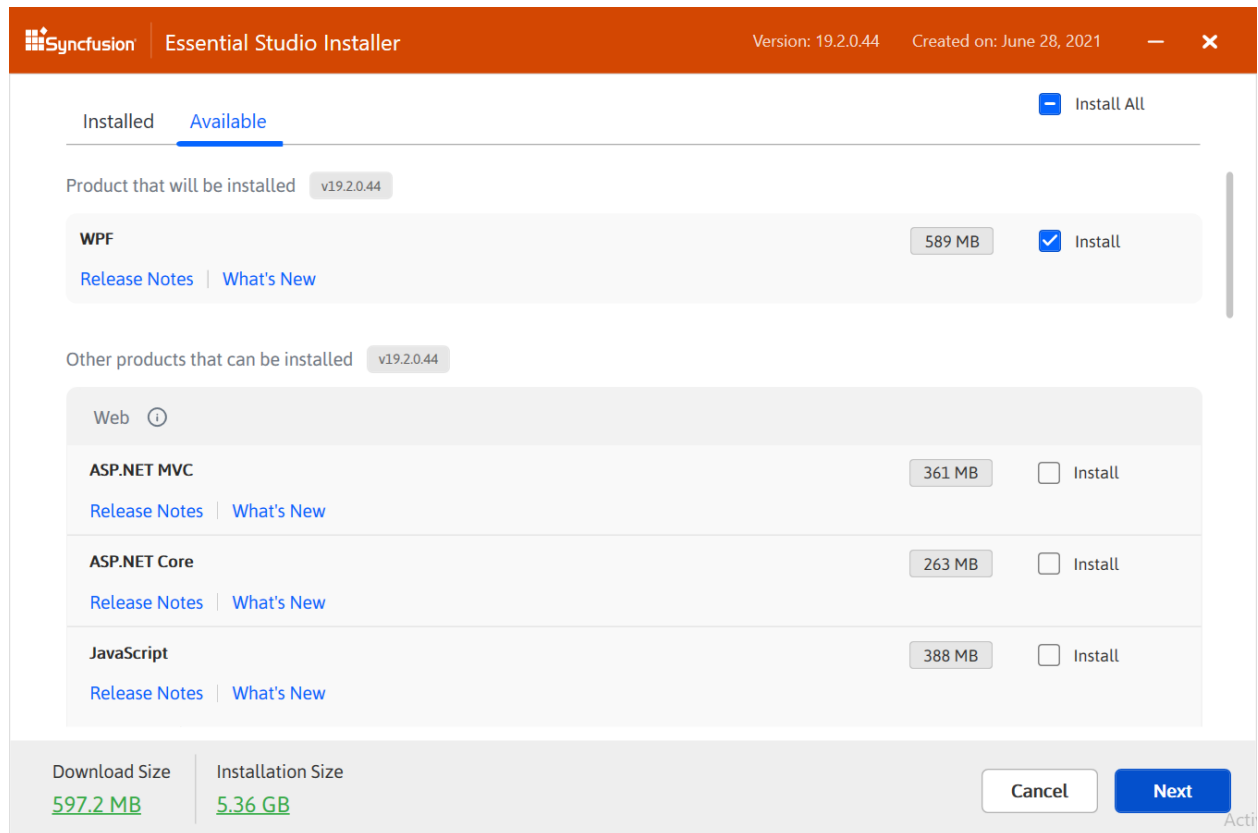
**Note:** The installer wizard extracts the `syncfusionessentialwpfwebinstaller_{version}.exe` dialog, which displays the package's unzip operation.

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



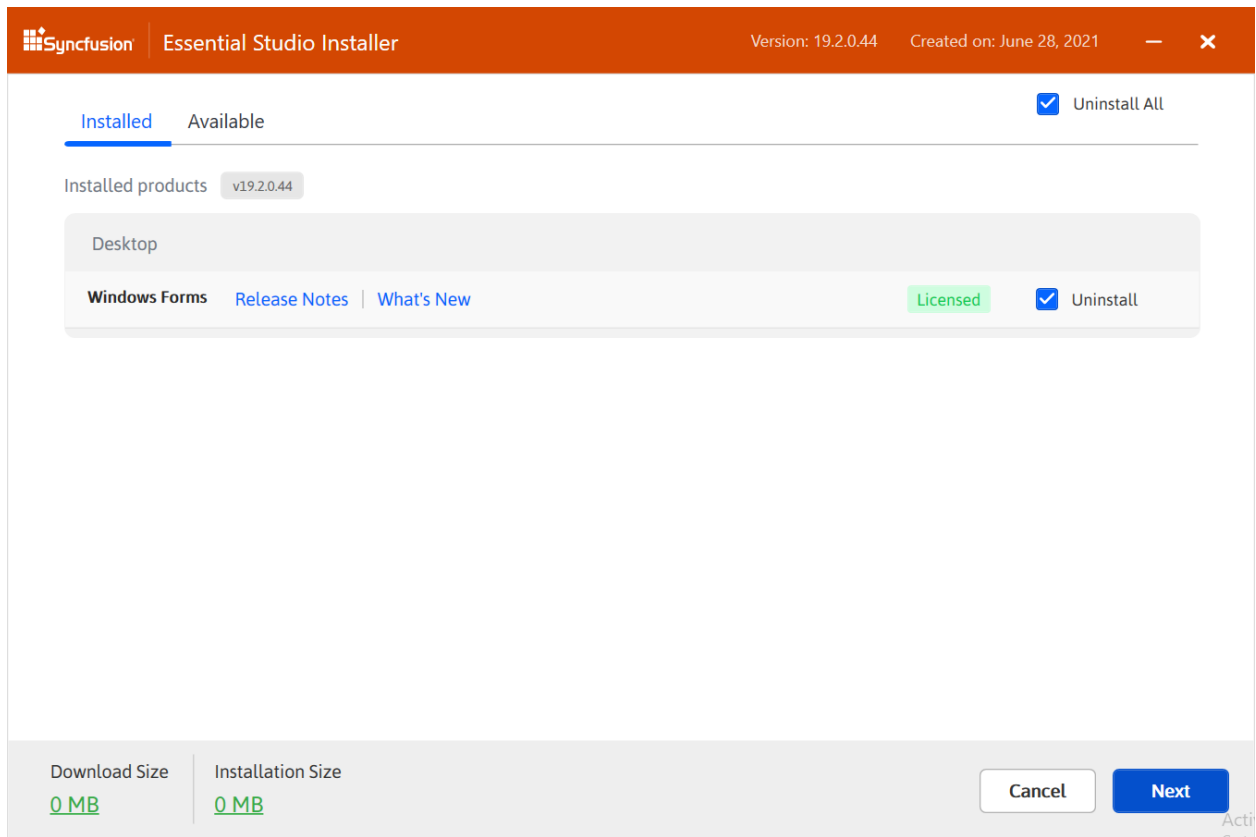
- The Platform Selection Wizard will appear. From the **Available** tab, select the products to be installed. Select the **Install All** checkbox to install all products.

<em>Available</em>



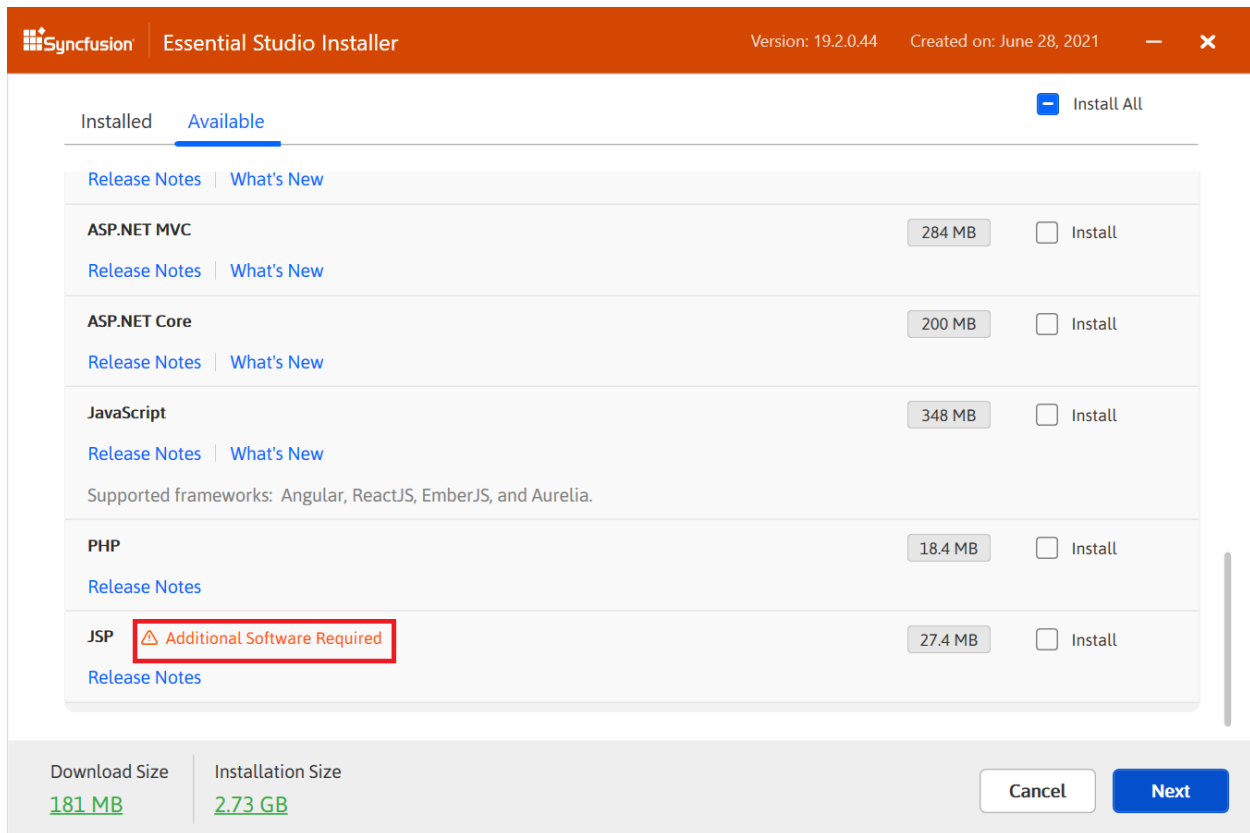
If you have multiple products installed in the same version, they will be listed under the **Installed** tab. You can also select which products to uninstall from the same version. Click the Next button.

**Installed**

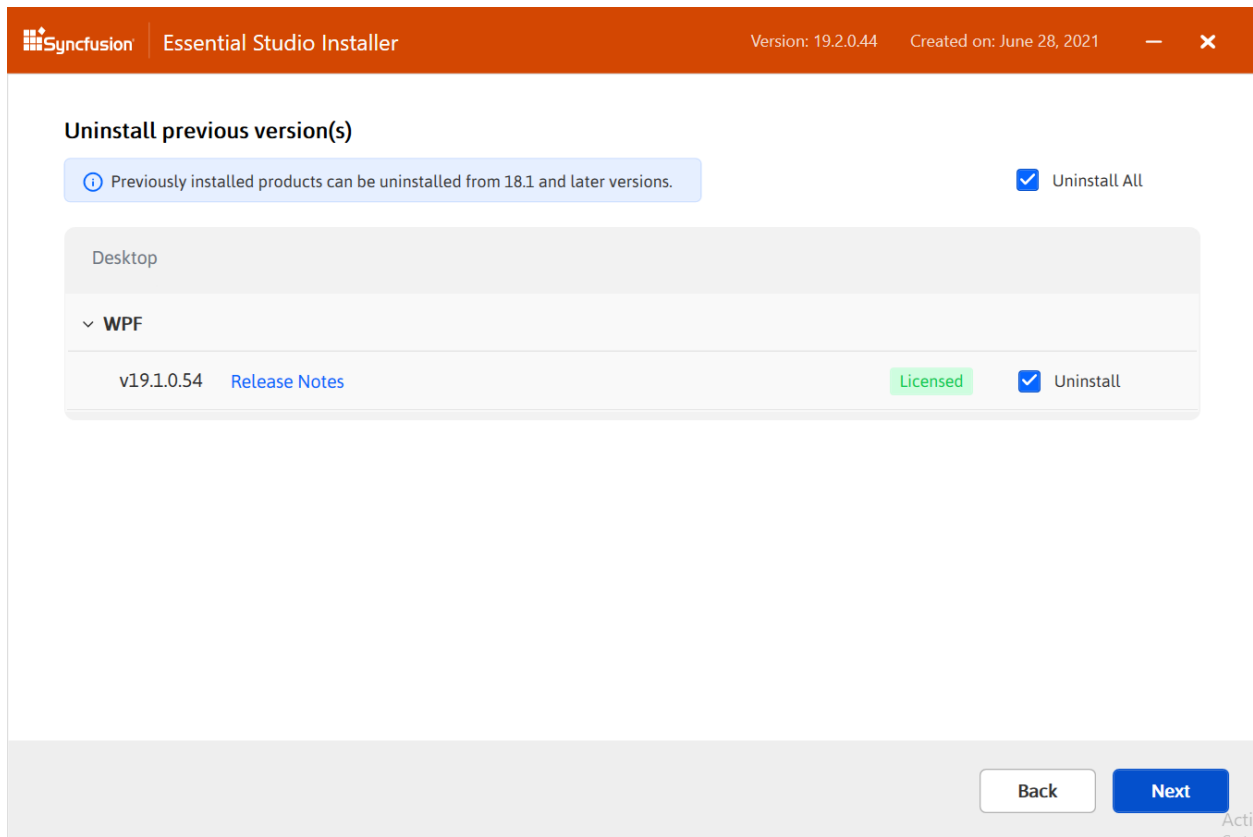


**Information:** If the required software for the selected product isn't already installed, the **Additional Software Required** alert will appear. You can, however, continue the installation and install the necessary software later.

## Required Software

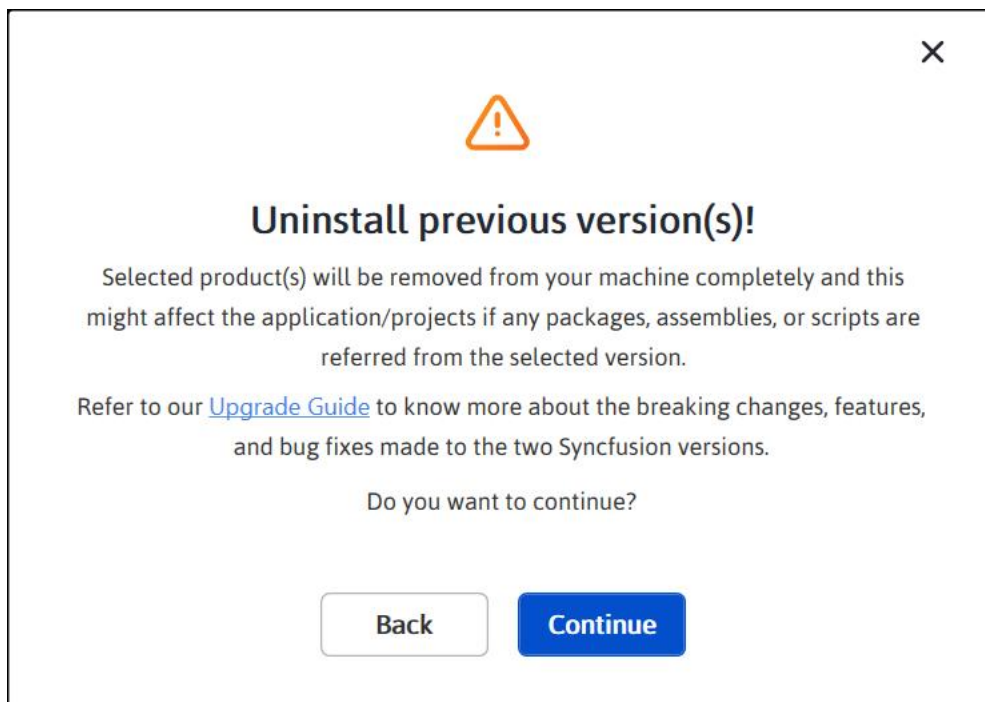


4. If previous version(s) for the selected products are installed, the Uninstall previous version wizard will be displayed. You can see the list of previously installed versions for the products you've chosen here. To remove all versions, check the **Uninstall All** checkbox. Click the Next button.



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

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



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

Syncfusion Essential Studio Installer Version: 19.2.0.44 Created on: June 28, 2021

## Products to Be Installed and Uninstalled

Install Product (1) Uninstall Product (1)


WPF v19.2.0.44	UWP v19.2.0.44

Download Size	Installation Size
597.2 MB	5.36 GB

Back Next

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

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


**Essential Studio Installer**
Version: 19.2.0.44
Created on: June 28, 2021

## Configuration

**Download Location**  

[Browse](#)

**Installation Location**  

[Browse](#)

**Demos Location**  

[Browse](#)

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

**Additional Settings**

- ☒ Install Demos
  - ☒ WPF
- ☒ Register Syncfusion assemblies in GAC
  - ☒ WPF
- ☒ Configure Syncfusion Controls in Visual Studio
  - ☒ WPF
- ☒ Configure Syncfusion Extensions in Visual Studio
  - ☒ WPF
- ☒ Create Desktop Shortcut(s)
  - ☒ WPF
- ☒ Create Start Menu Shortcut(s)
  - ☒ WPF

Download Size  
**597.2 MB**

Installation Size  
**5.36 GB**

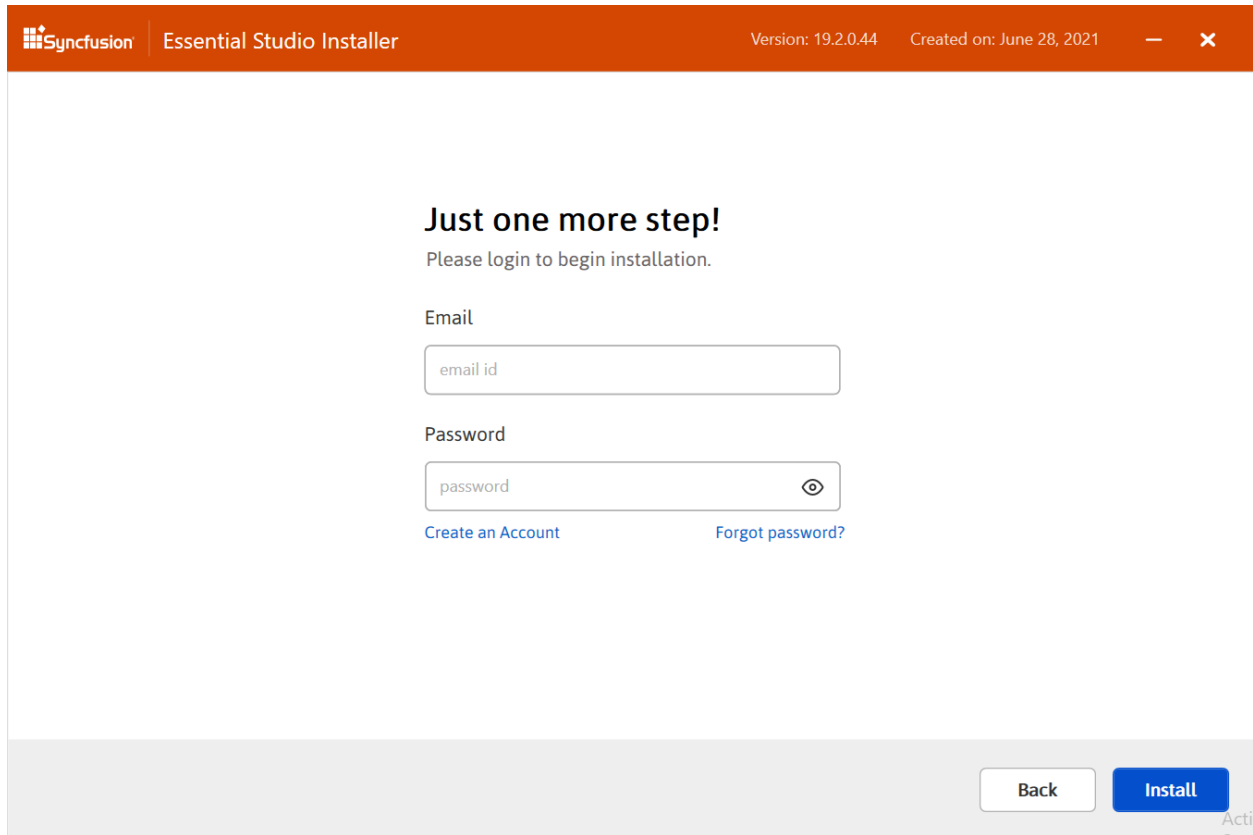
[Back](#)
[Next](#)

### Additional settings

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



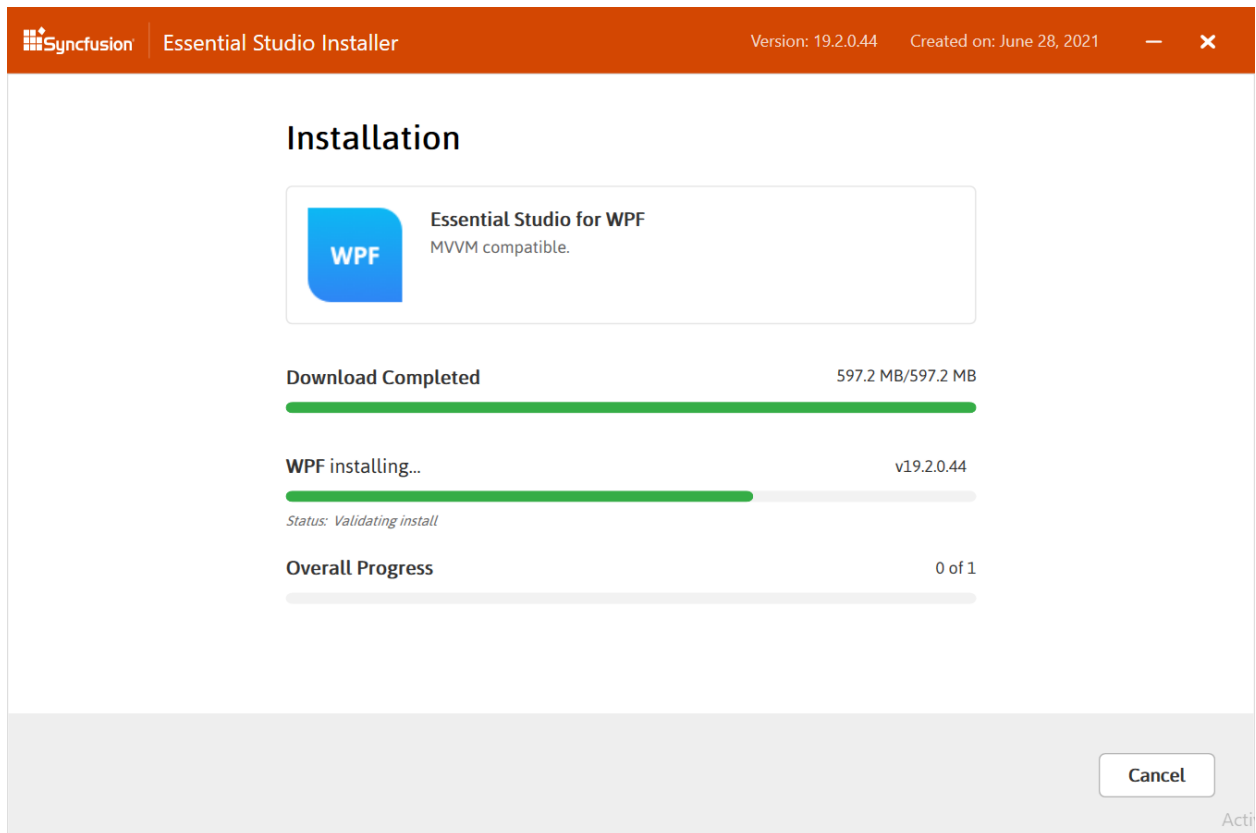
If you have forgotten your password, click **Forgot Password** to create a new one. Click the Install button.



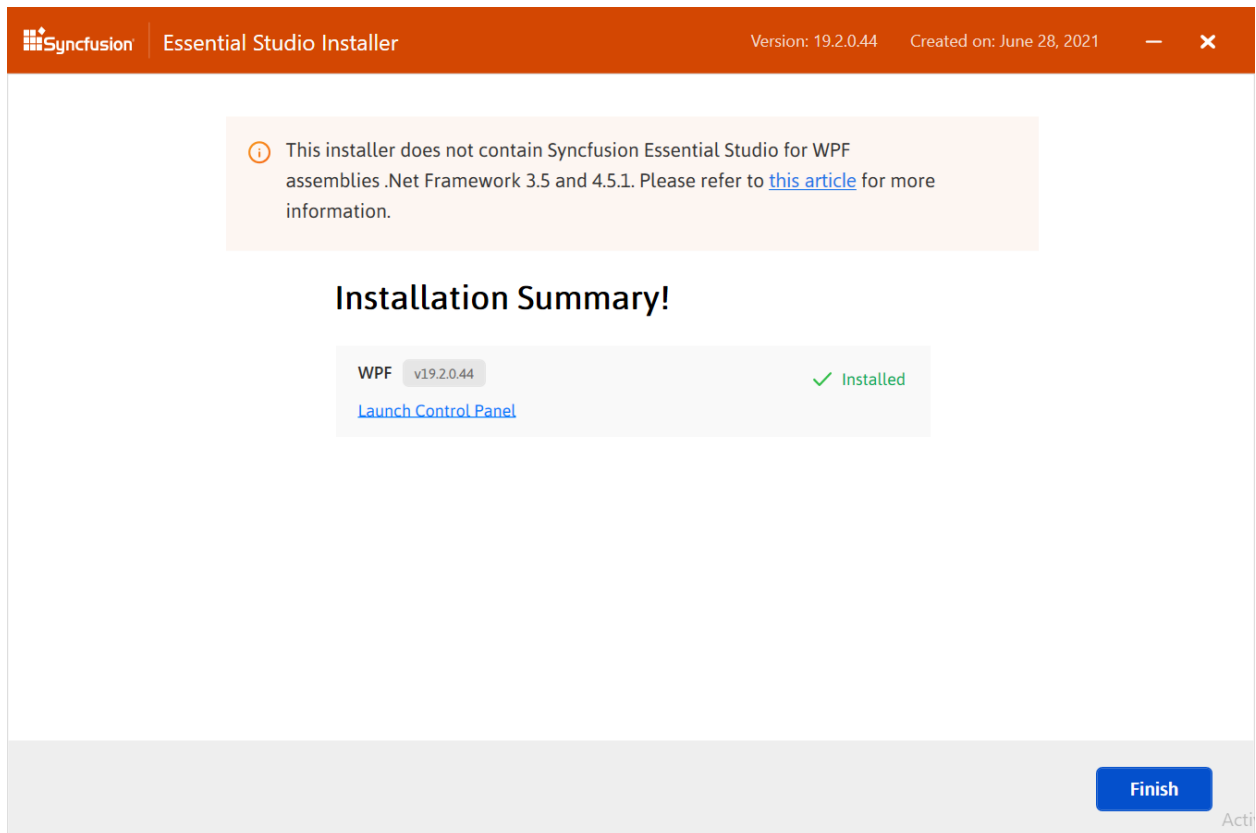
The screenshot shows the Syncfusion Essential Studio Installer window. The title bar is orange and contains the Syncfusion logo, the text 'Essential Studio Installer', the version 'Version: 19.2.0.44', and the creation date 'Created on: June 28, 2021'. The main content area has a white background with the heading 'Just one more step!' and the instruction 'Please login to begin installation.' Below this are two input fields: 'Email' with a placeholder 'email id' and 'Password' with a placeholder 'password' and a toggle icon. At the bottom of the form are two links: 'Create an Account' and 'Forgot password?'. At the bottom right of the window are two buttons: 'Back' and 'Install'.

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

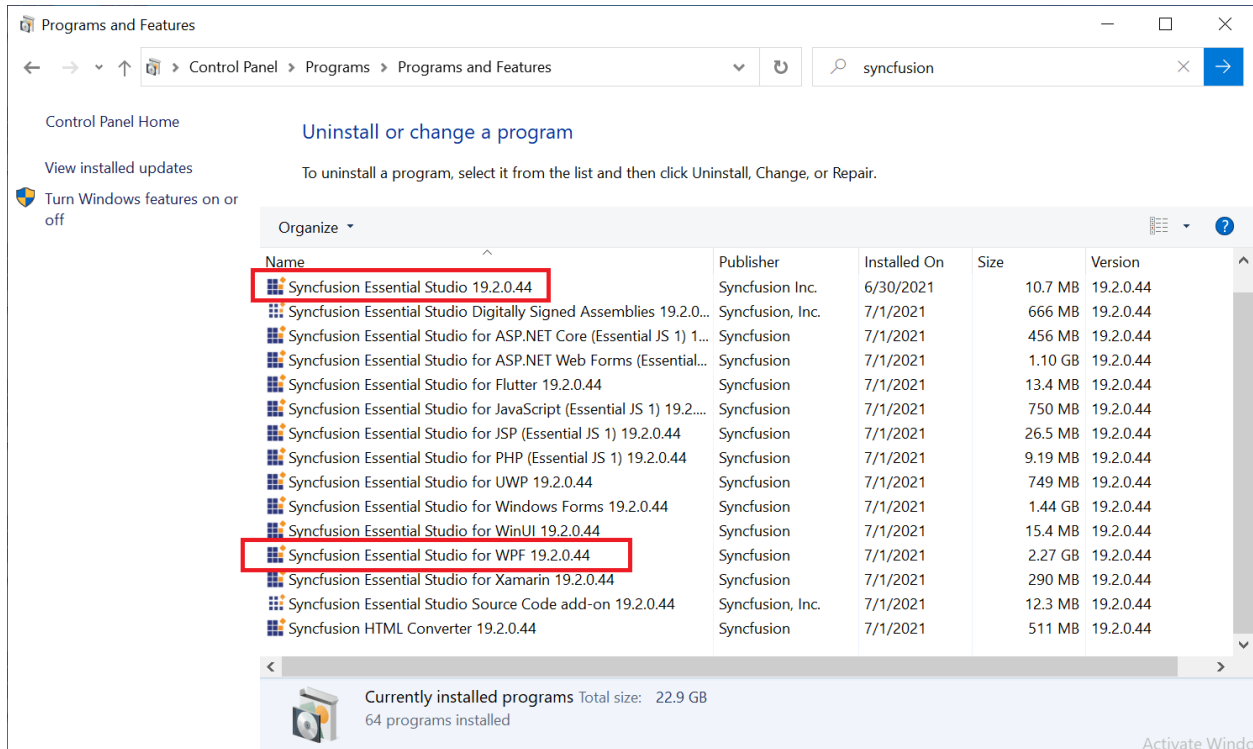
10. The download and installation\uninstallation progress will be displayed as shown below.



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



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



## Uninstallation

Syncfusion WPF installer can be uninstalled in two ways.

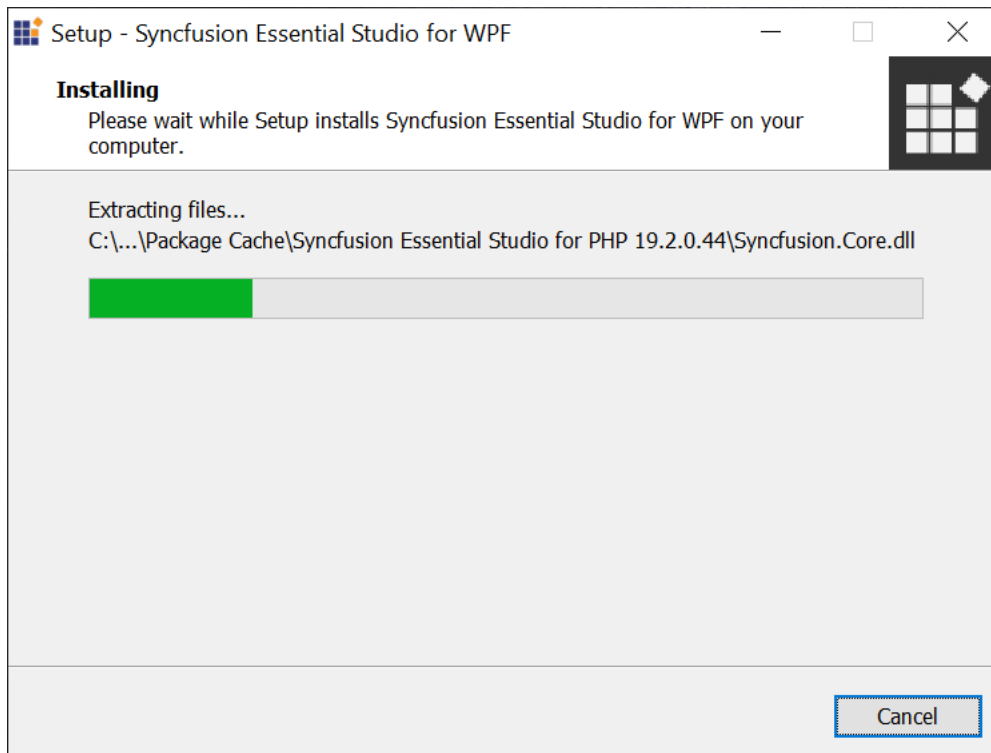
- Uninstall the WPF using the Syncfusion WPF web installer
- Uninstall the WPF from Windows Control Panel

Follow either one of the option below to uninstall Syncfusion Essential Studio WPF installer.

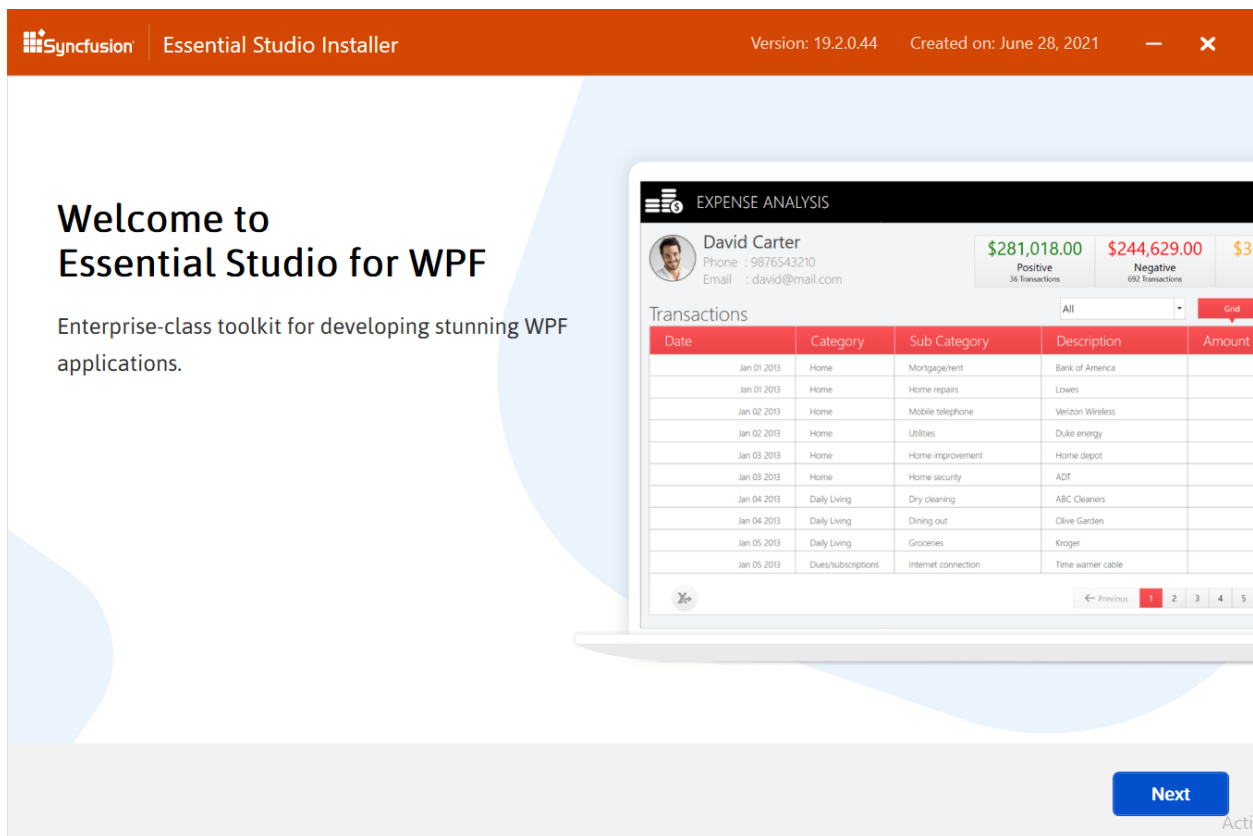
### Option 1: Uninstall the WPF using the Syncfusion WPF web installer

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

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

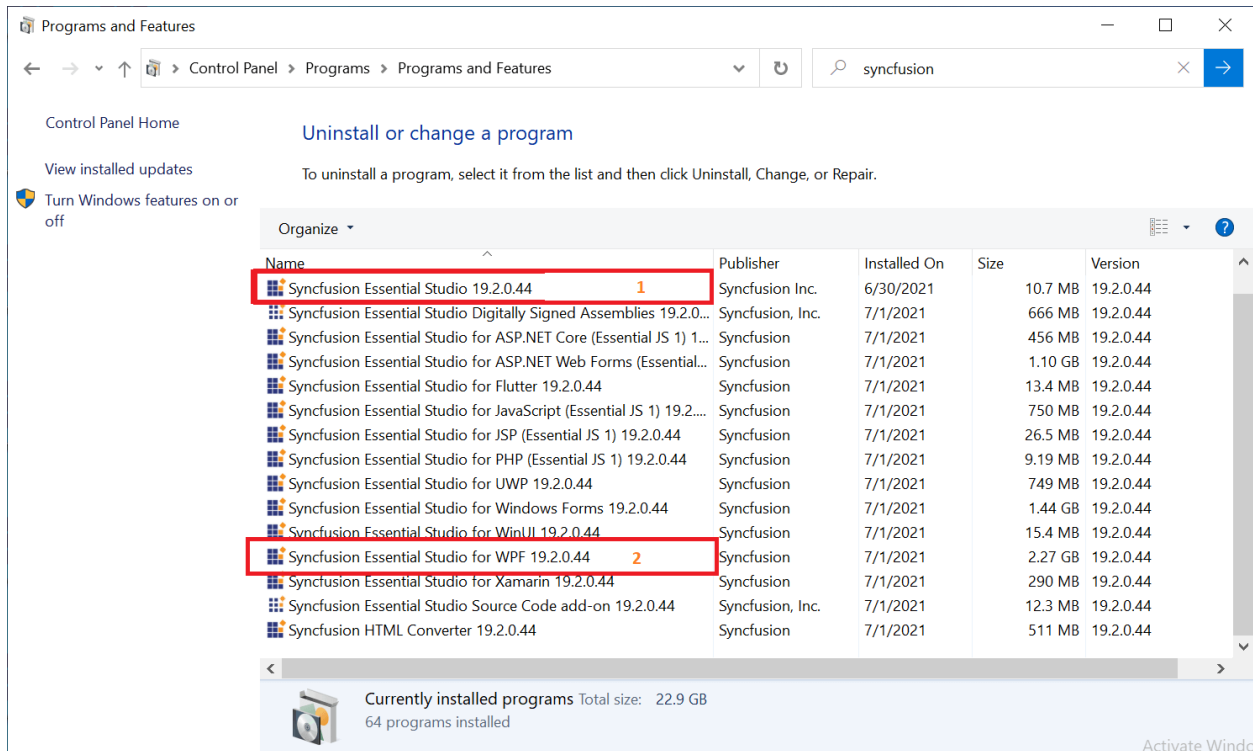


The Syncfusion WPF Web Installer's welcome wizard will be displayed. Click the Next button



**Option 2: Uninstall the WPF from Windows Control Panel**

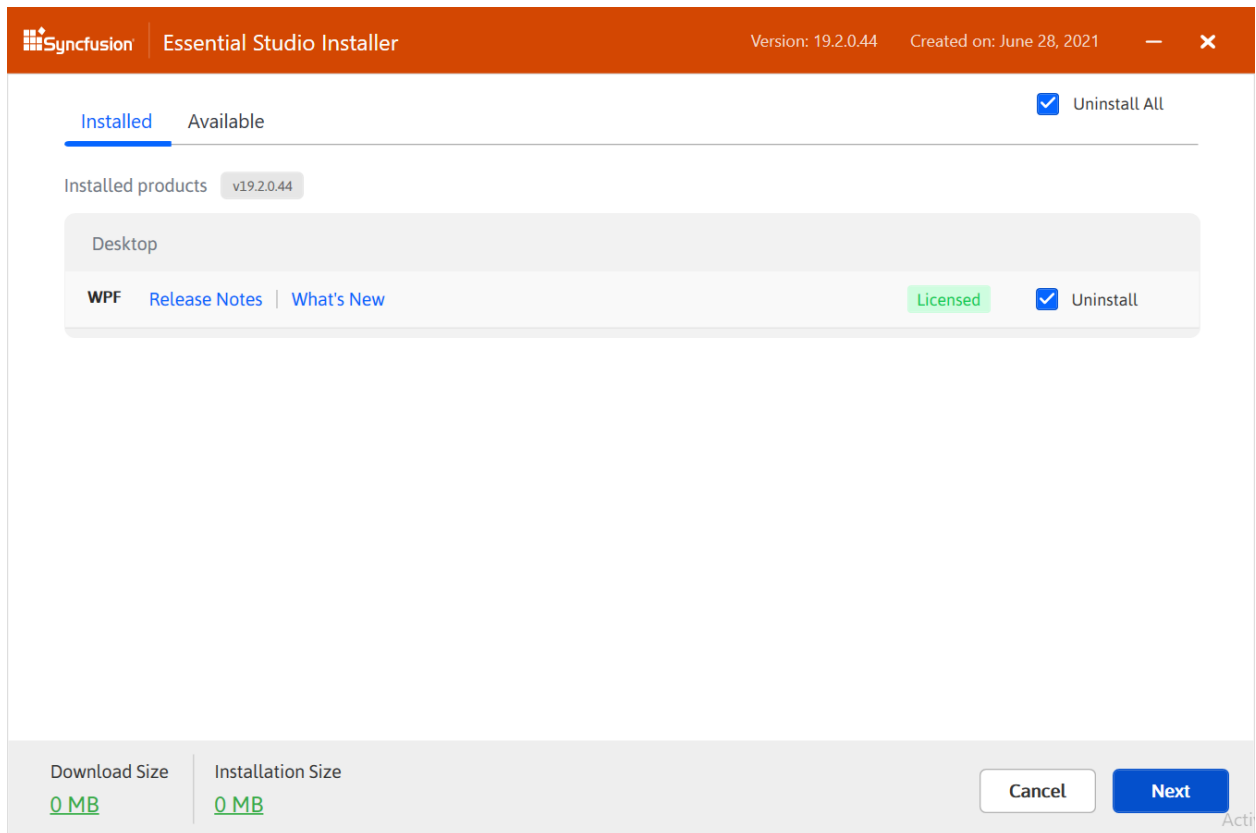
You can uninstall all the installed products by selecting the **Syncfusion Essential Studio {version}** entry (element 1 in the below screenshot) from the Windows control panel, or you can uninstall WPF alone by selecting the **Syncfusion Essential Studio for WPF {version}** entry (element 2 in the below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



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

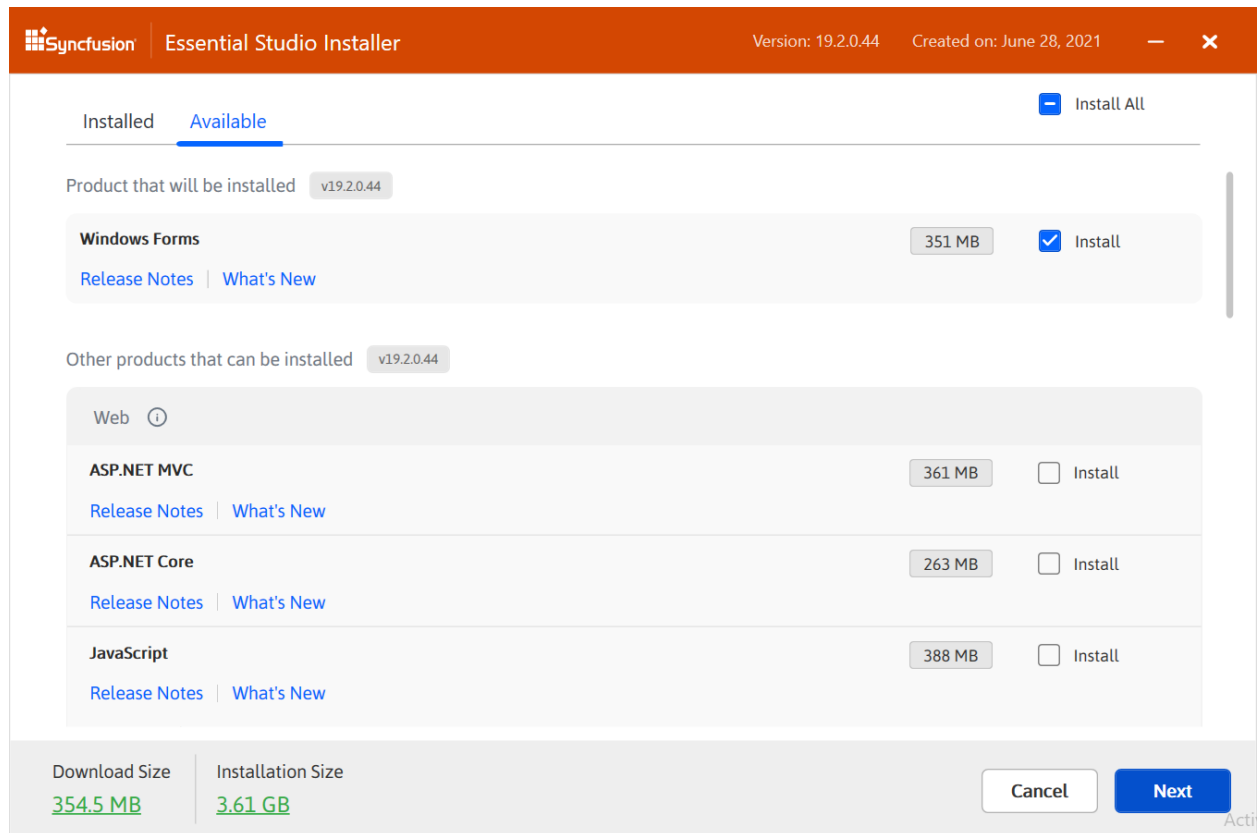
1. The Platform Selection Wizard will appear. From the **Installed** tab, select the products to be uninstalled. To select all products, check the **Uninstall All** checkbox. Click the Next button.

<em>Installed</em>



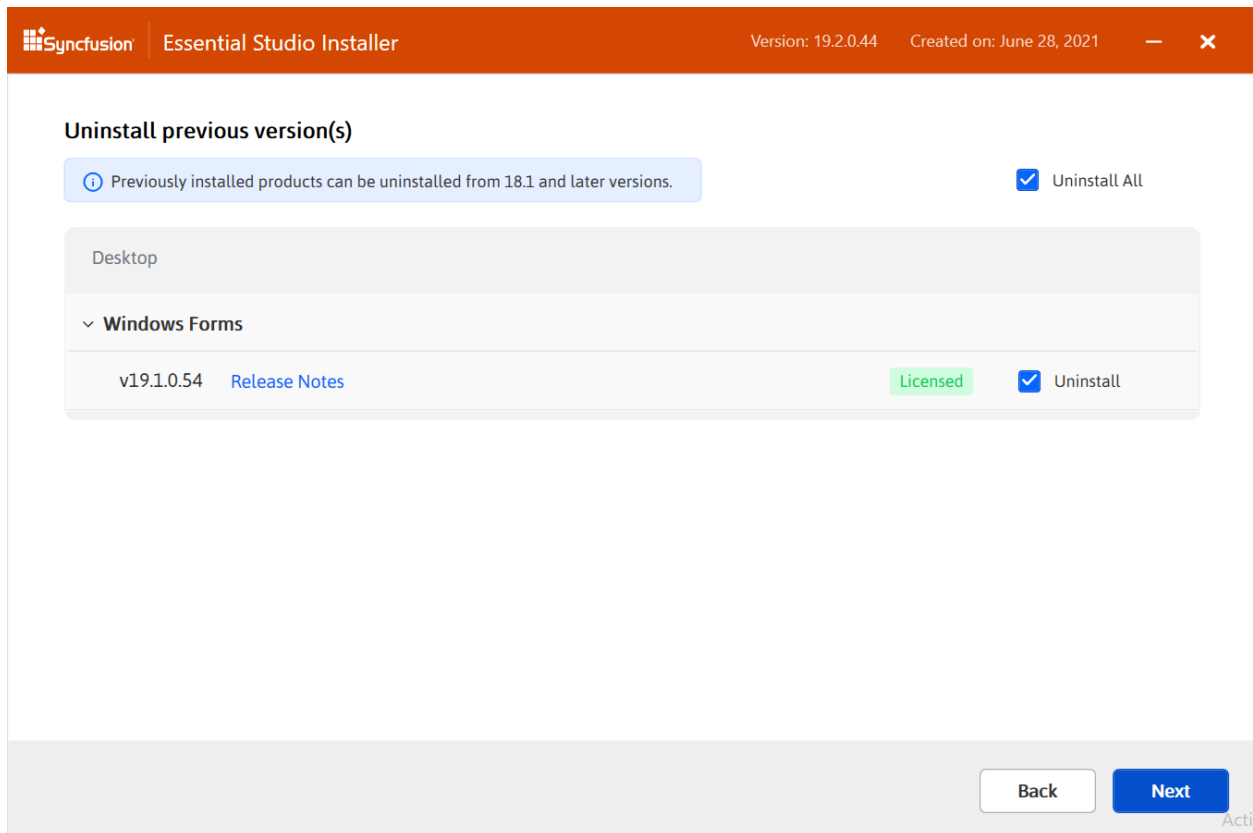
You can also select the products to be installed from the **Available** tab. Click the Next button.

<em>Available</em>

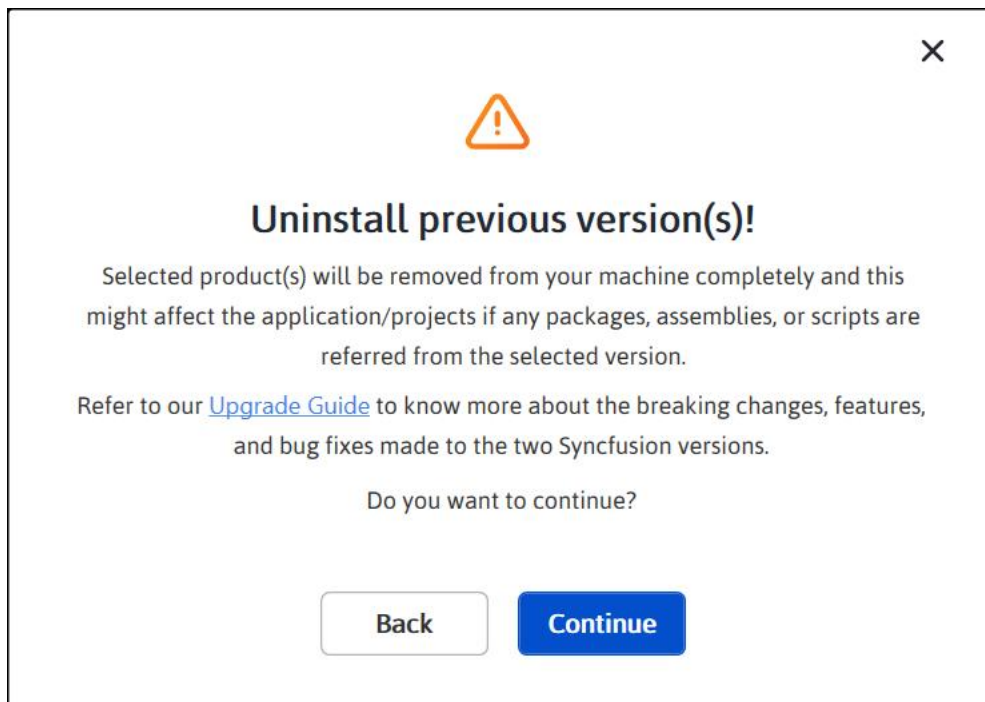


2. If any other products selected for installation, Uninstall previous version wizard will be displayed with previous version(s) installed for the selected products. Here you can view the list of installed previous versions for the selected products. Select **Uninstall All** checkbox to select all the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






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



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

 Essential Studio Installer

Version: 19.2.0.44 Created on: June 28, 2021

Products to Be Installed and Uninstalled

Install Product (1)

WPF v19.2.0.44

Uninstall Product (1)

UWP v19.2.0.44

Download Size

597.2 MB

Installation Size


5.36 GB

Back

Next

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

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


**Essential Studio Installer**
Version: 19.2.0.44
Created on: June 28, 2021
—
×

## Configuration

**Download Location**

[Browse](#)

**Installation Location**

[Browse](#)

**Demos Location**

[Browse](#)

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

**Additional Settings**


- ☒ Install Demos
  - ☒ Windows Forms
- ☒ Register Syncfusion assemblies in GAC
  - ☒ Windows Forms
- ☒ Configure Syncfusion Controls in Visual Studio
  - ☒ Windows Forms
- ☒ Configure Syncfusion Extensions in Visual Studio
  - ☒ Windows Forms
- ☒ Create Desktop Shortcut(s)
  - ☒ Windows Forms
- ☒ Create Start Menu Shortcut(s)
  - ☒ Windows Forms

Download Size  
**354.5 MB**

Installation Size  
**3.61 GB**

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

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

 Essential Studio Installer

Version: 19.2.0.44 Created on: June 28, 2021

## Just one more step!

Please login to begin installation.

Email

Password

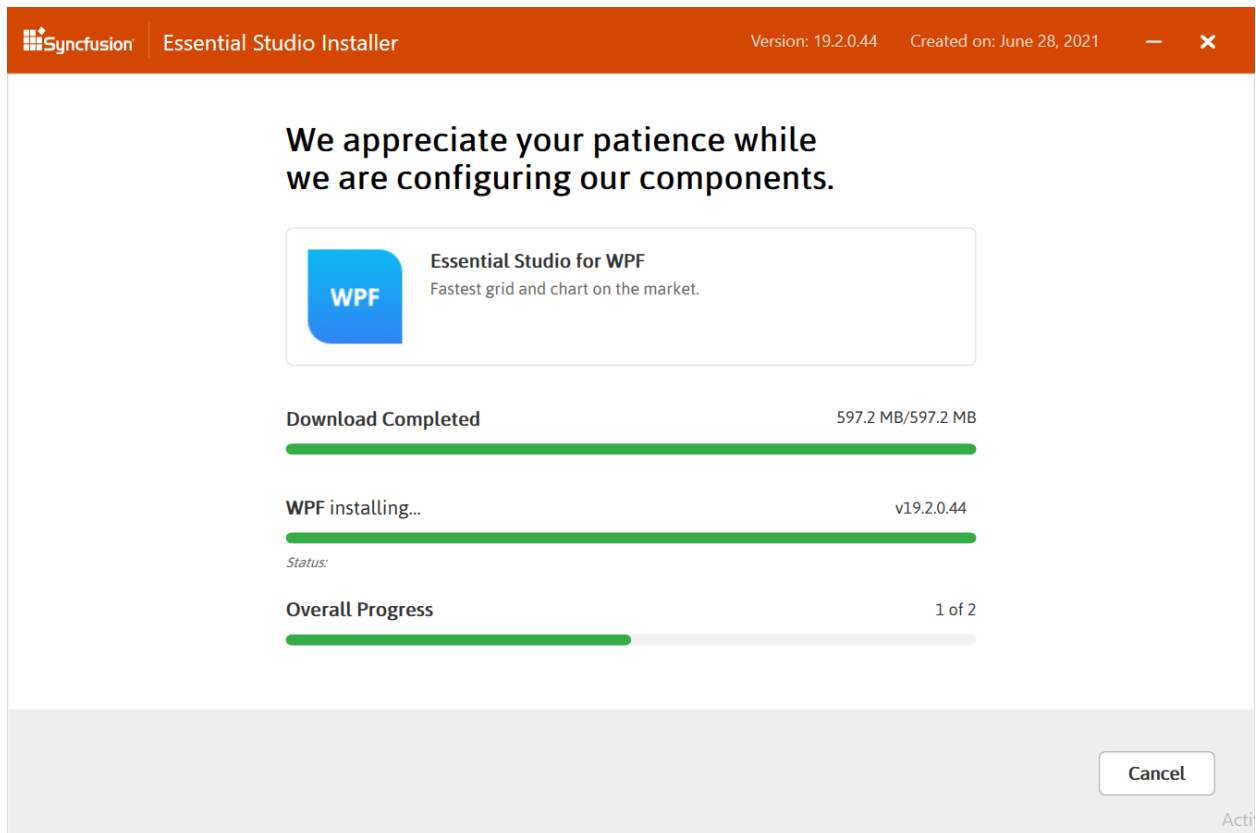
[Create an Account](#) [Forgot password?](#)

Back

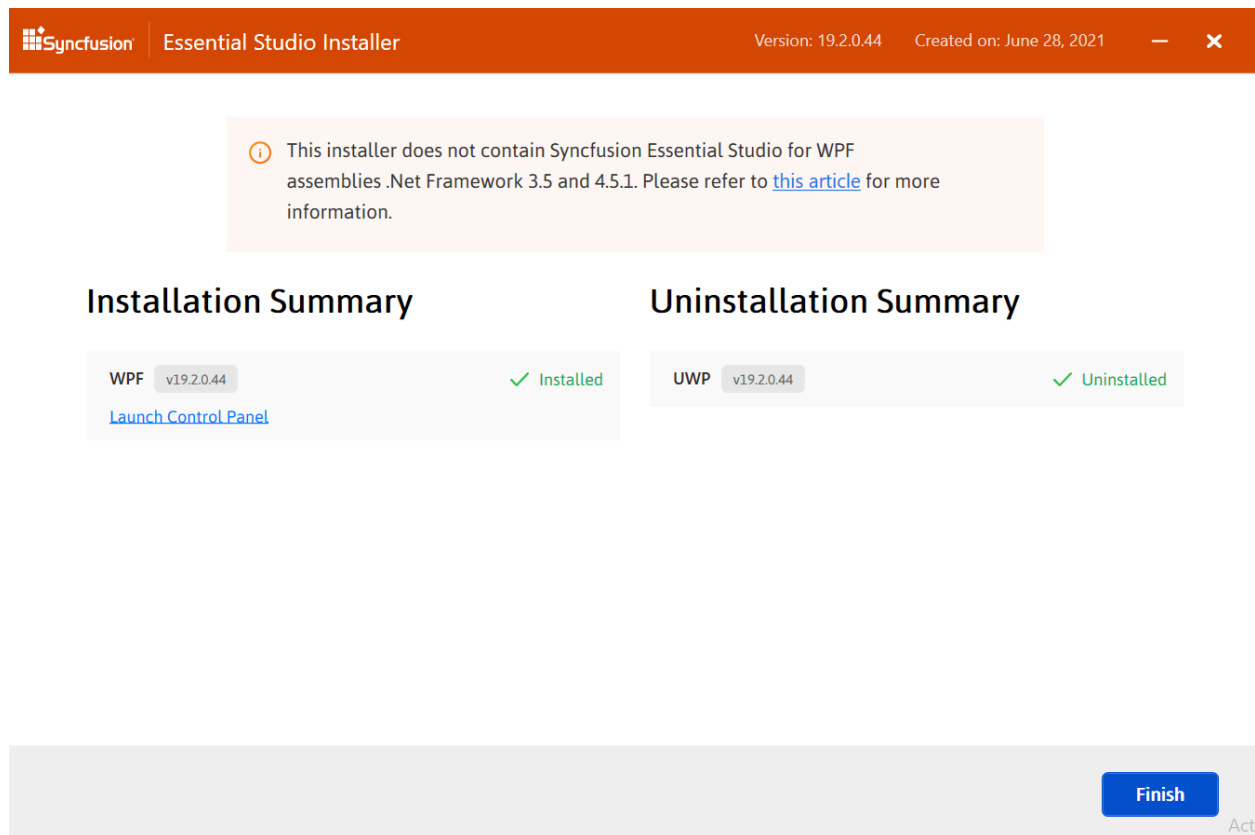
Install

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

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



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



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

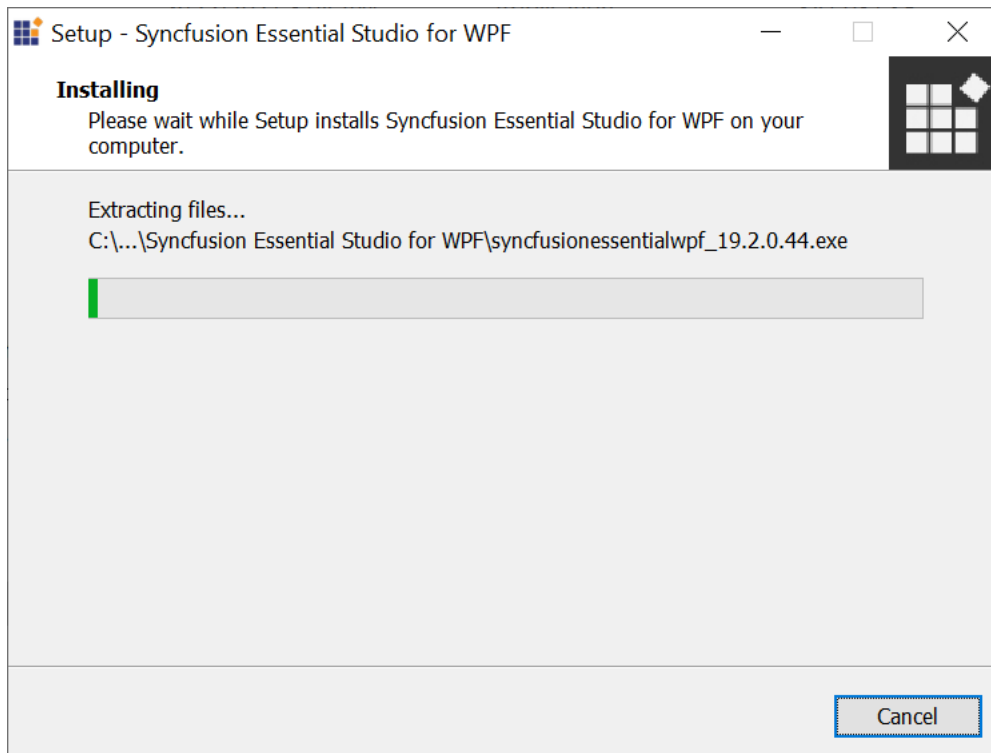
## Installation using Offline Installer

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

### Installing with UI

The steps below show how to install the Essential Studio WPF installer.

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



---

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

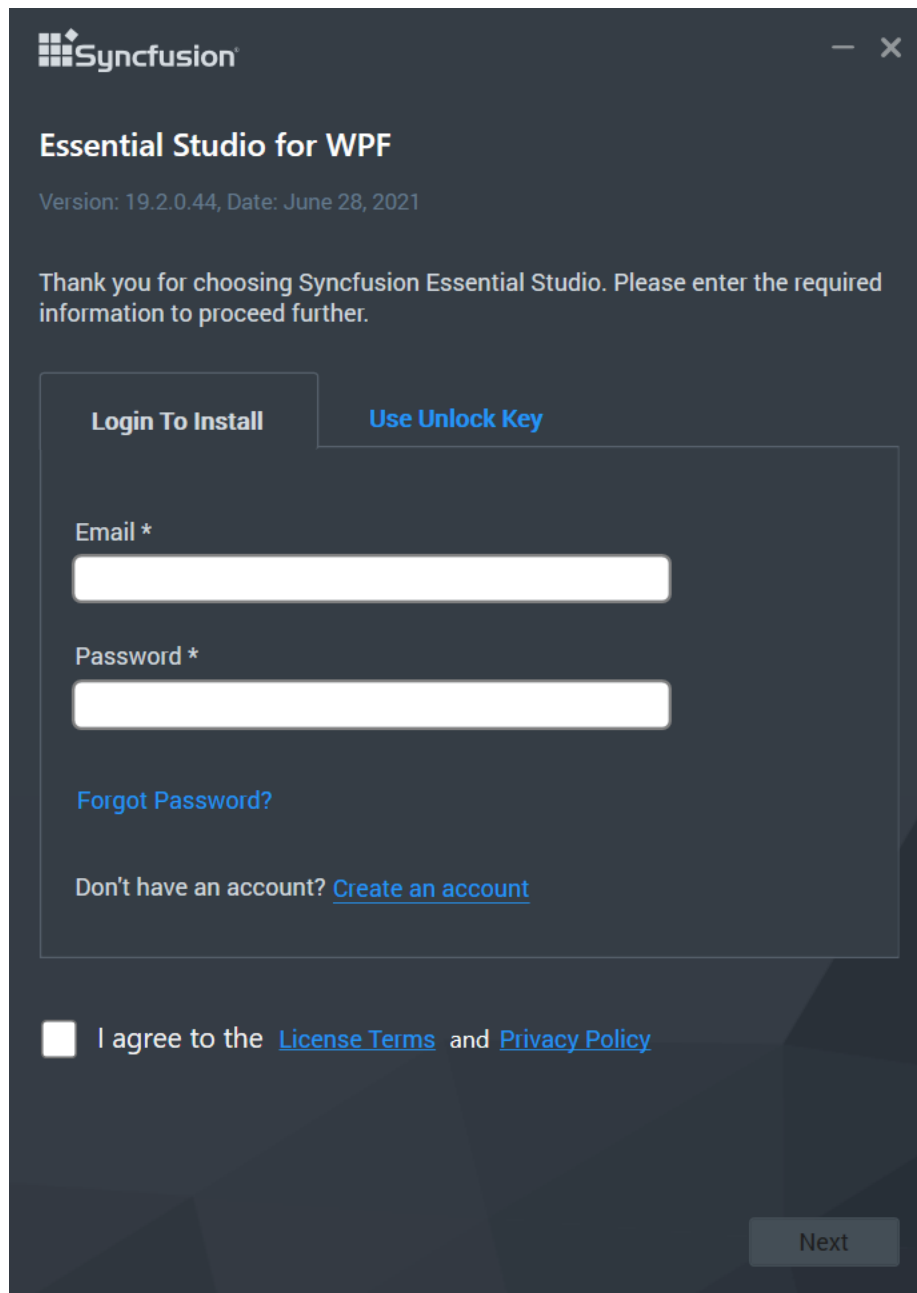
---

2. To unlock the Syncfusion offline installer, you have two options:

- *Login To Install*
- *Use Unlock Key*

### Login To Install

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



**Syncfusion**

## Essential Studio for WPF

Version: 19.2.0.44, Date: June 28, 2021

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

**Login To Install**    [Use Unlock Key](#)

Email \*

Password \*

[Forgot Password?](#)

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

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

Next

### Use Unlock Key

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

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

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



Syncfusion

## Essential Studio for WPF

Version: 19.2.0.44, Date: June 28, 2021

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

[Login To Install](#) **Use Unlock Key**

If you do not have an unlock key, [request one from Syncfusion](#).

Unlock Key \*

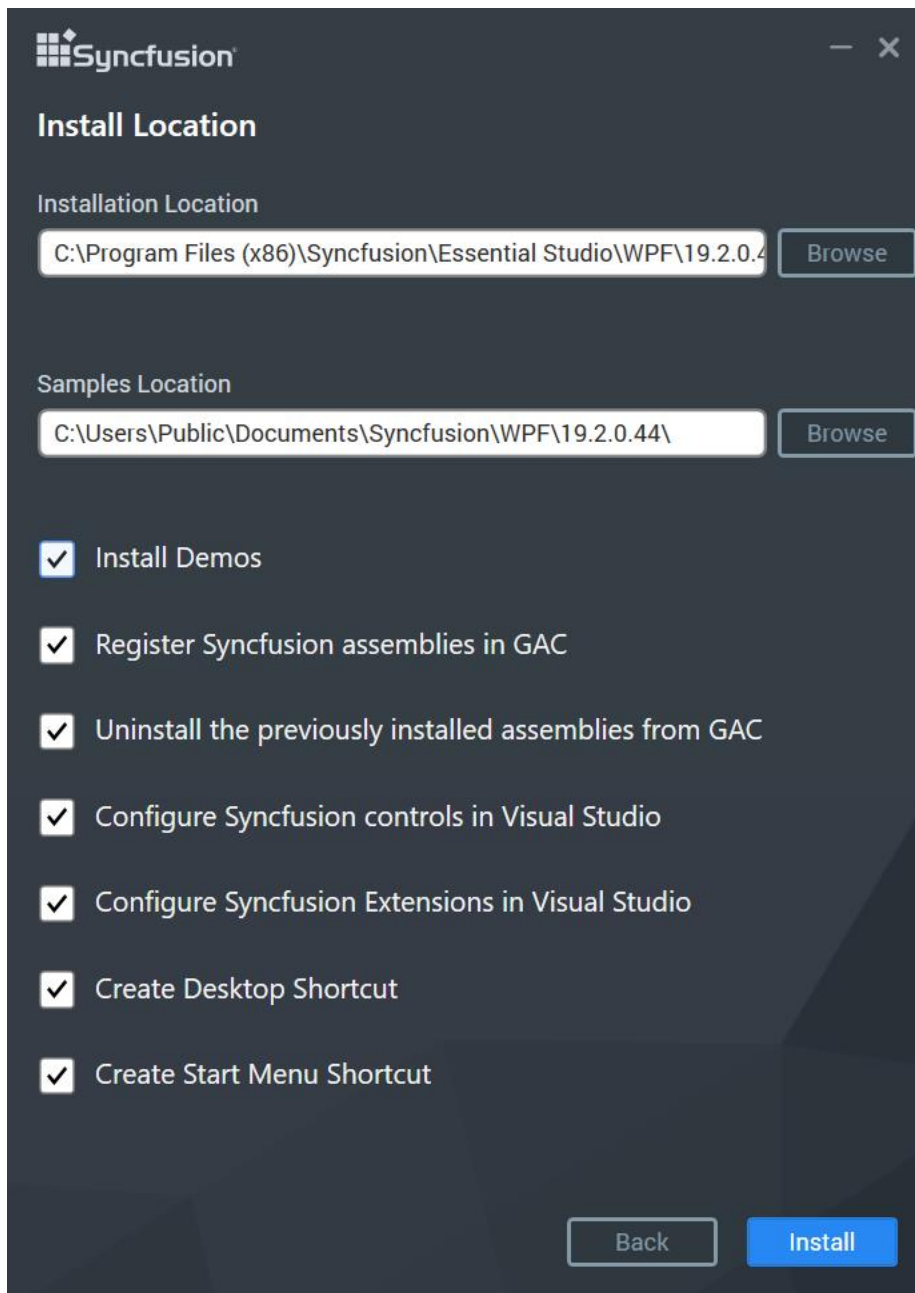
[Free Evaluation Key](#)

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

Next

3. After reading the License Terms and Privacy Policy, check the **“I agree to the License Terms and Privacy Policy”** check box. Click the Next button.

4. Change the install and sample locations here. You can also change the Additional settings. Click Next\Install to install with the default settings.

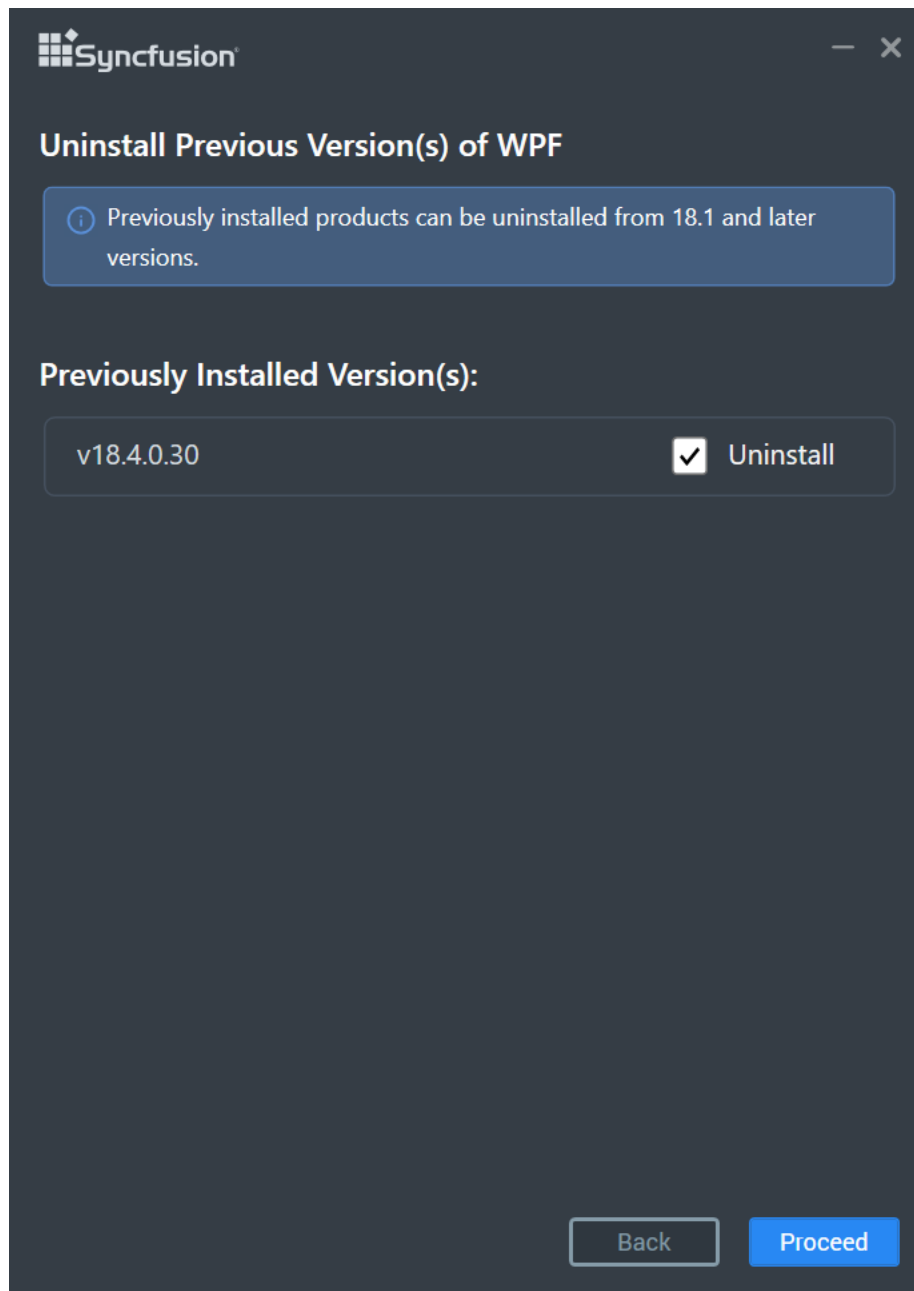


### Additional Settings

- Select the **Install Demos** check box to install Syncfusion samples, or leave the check box unchecked, if you do not want to install Syncfusion samples
- Select the **Register Syncfusion Assemblies in GAC** check box to install the latest Syncfusion assemblies in GAC, or clear this check box when you do not want to install the latest assemblies in GAC.
- Select the **Configure Syncfusion controls in Visual Studio** check box to configure the Syncfusion controls in the Visual Studio toolbox, or clear this check box when you do not want to configure the Syncfusion controls in the Visual Studio toolbox during installation. Note that you must also select the Register Syncfusion assemblies in GAC check box when you select this check box.

- Select the **Configure Syncfusion Extensions controls in Visual Studio** checkbox to configure the Syncfusion Extensions in Visual Studio or clear this check box when you do not want to configure the Syncfusion Extensions in Visual Studio.
- Check the **Create Desktop Shortcut** checkbox to add a desktop shortcut for Syncfusion Control Panel
- Check the **Create Start Menu Shortcut** checkbox to add a shortcut to the start menu for Syncfusion Control Panel

5. If any previous versions of the current product is installed, the Uninstall Previous Version(s) wizard will be opened. Select **Uninstall** checkbox to uninstall the previous versions and then click the Proceed button.



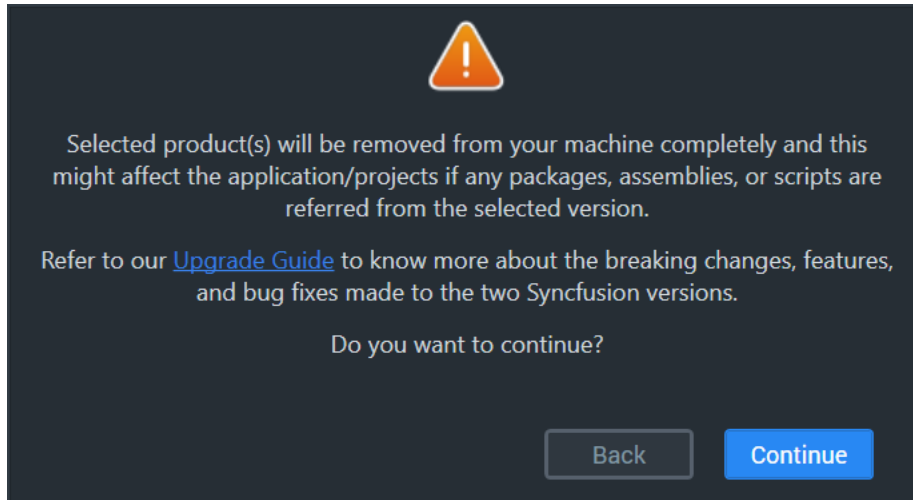
---

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

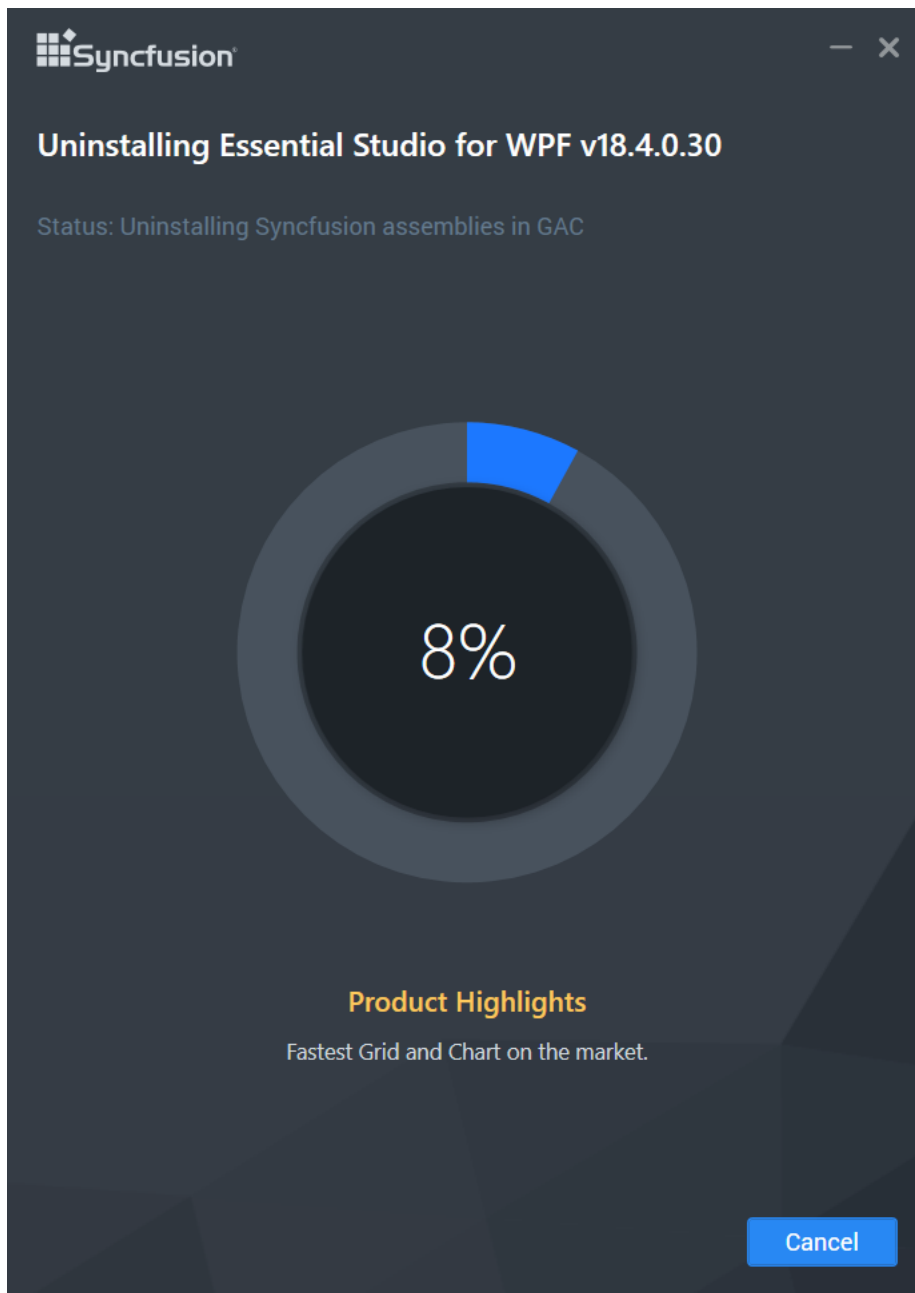
**Note:** If any version is selected to uninstall, a confirmation screen will appear; if continue is selected, the Progress screen will display the uninstall and install progress, respectively. If none of the versions are chosen to be uninstalled, only the installation progress will be displayed.

---

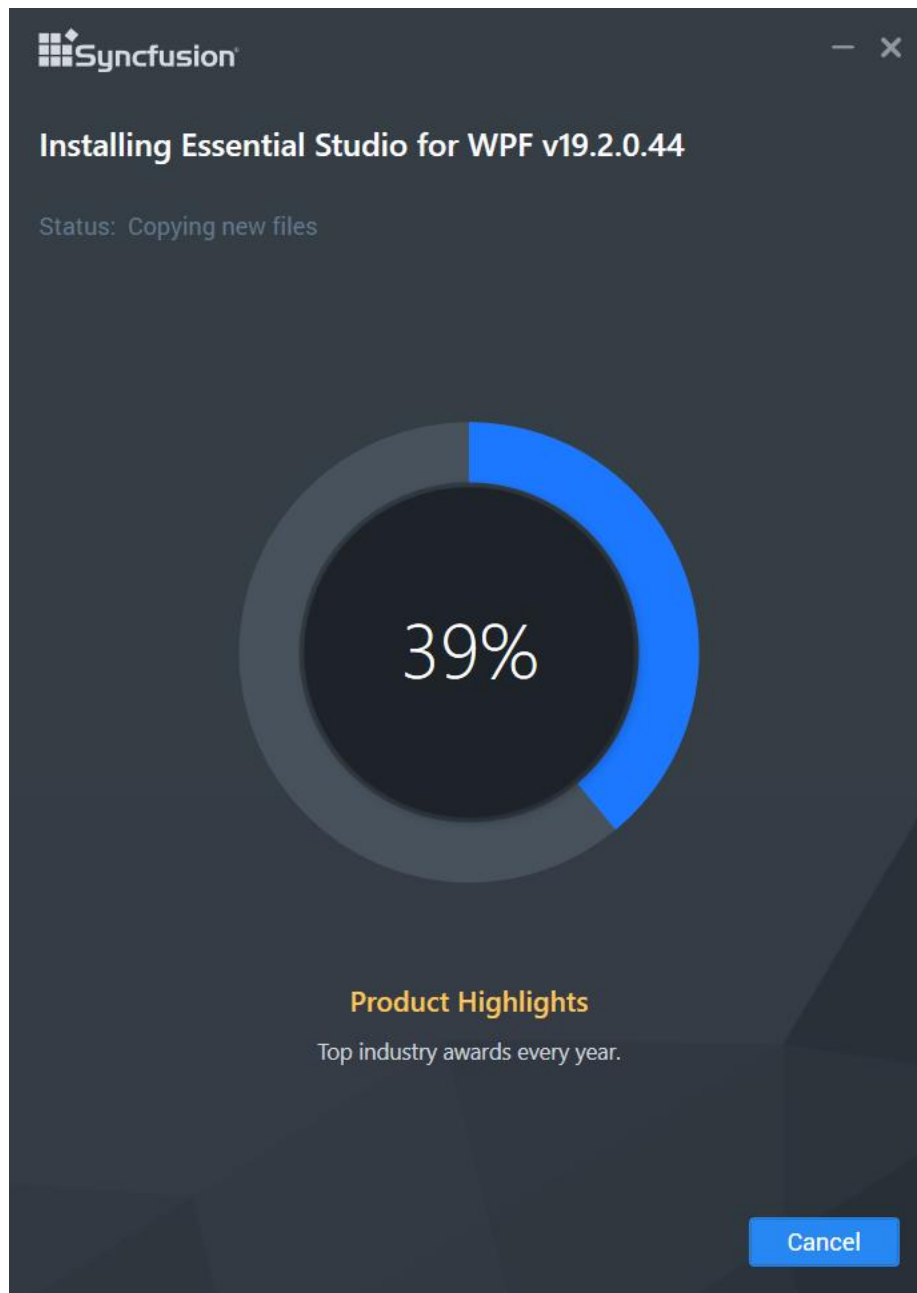
#### Confirmation Alert



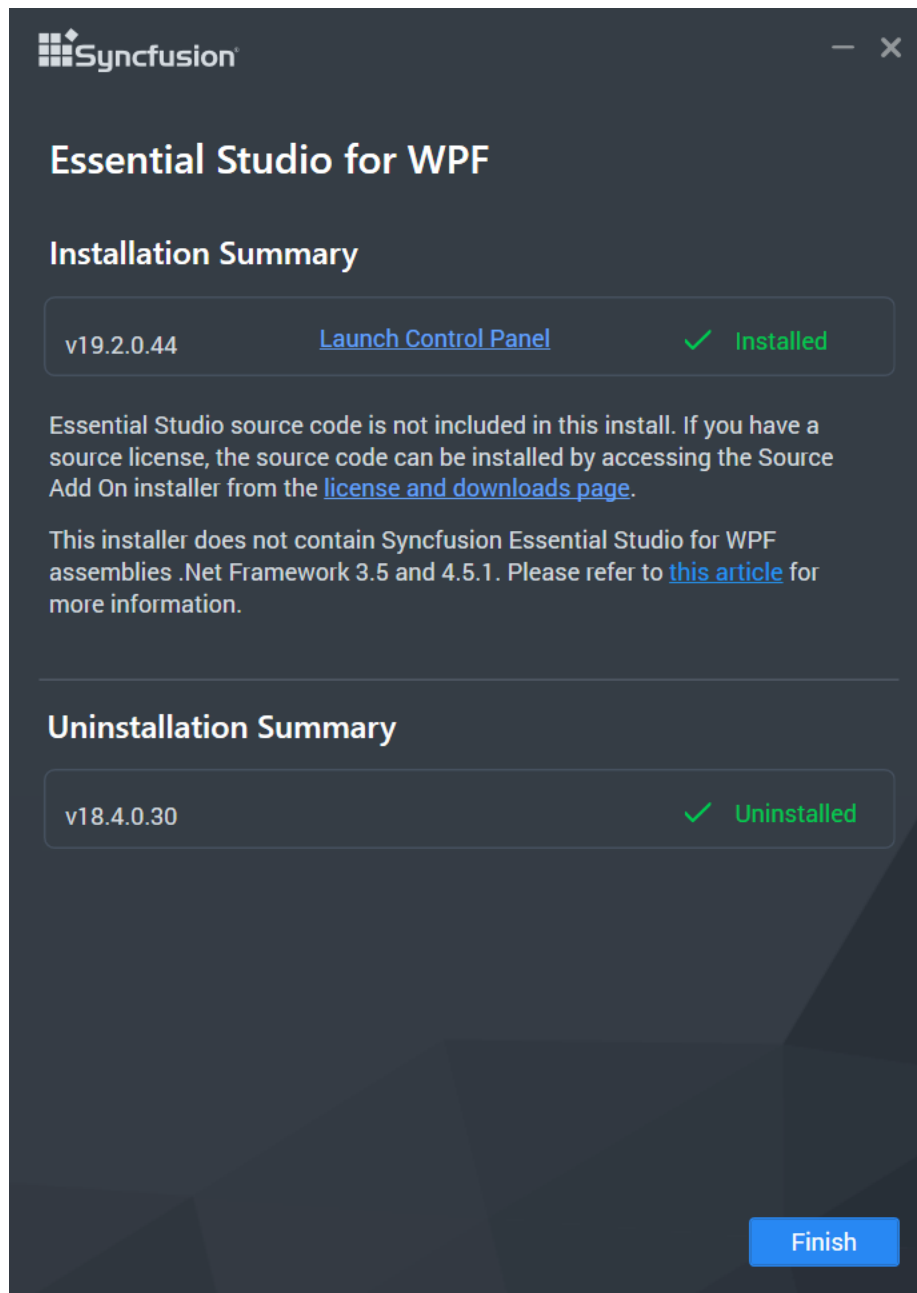
#### Uninstall Progress:



**Install Progress**



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



7. After installing, click the **Launch Control Panel** link to open the Syncfusion Control Panel.
8. Click the Finish button. Your system has been installed with the Syncfusion Essential Studio WPF product.

#### Installing in silent mode

The Syncfusion Essential Studio WPF Installer supports installation and uninstallation via the command line.

#### Command Line Installation

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

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

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

---

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

---

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

7. Essential Studio for WPF is installed.

---

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

---

#### Command Line Uninstallation

Syncfusion Essential WPF can be uninstalled silently using the Command Line.

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

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

**Example:** "D:\Temp\syncfusionessentialwpf\_x.x.x.x.exe" /uninstall silent

7. Essential Studio for WPF is uninstalled.

#### Common Installation Errors

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

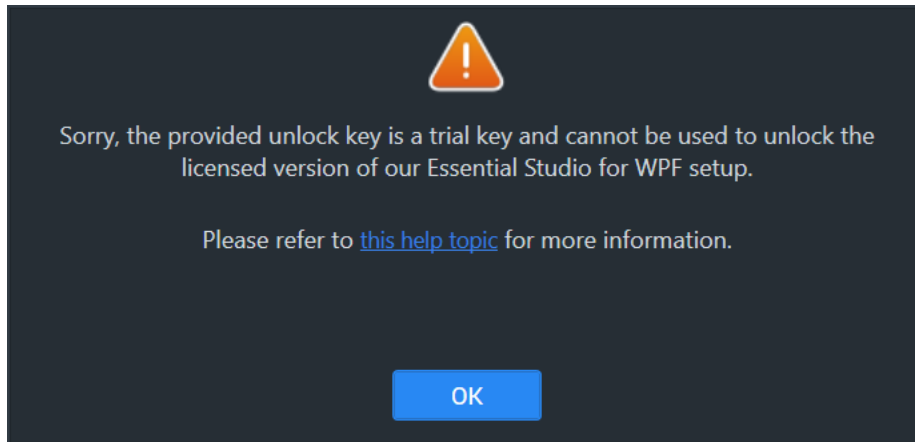
- [Unlocking the license installer using the trial key](#)
- [License has expired](#)
- [Unable to find a valid license or trial](#)
- [Unable to install because of another installation](#)
- [Unable to install due to controlled folder access](#)



Unlocking the license installer using the trial key

*Problem*

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



*Reason*

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

*Suggested solution*

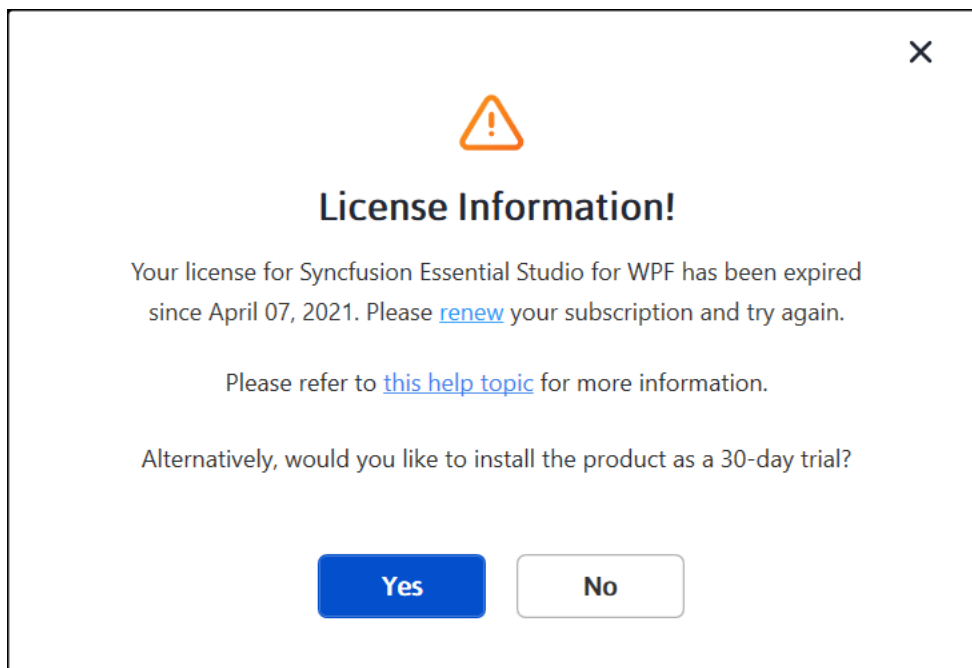
Only a licensed unlock key can unlock a licensed installer. So, to unlock the Licensed installer, use the Licensed unlock key. To generate the licensed unlock key, refer to [this](#) article.

License has expired

*Problem*

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

**Online Installer**



*Reason*

This error message will appear if your license has expired.

*Suggested solution*

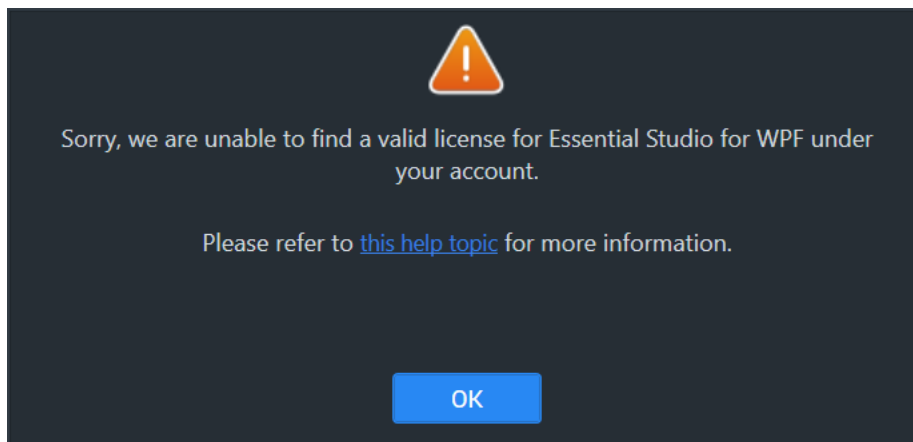
You can choose from the options listed below.

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

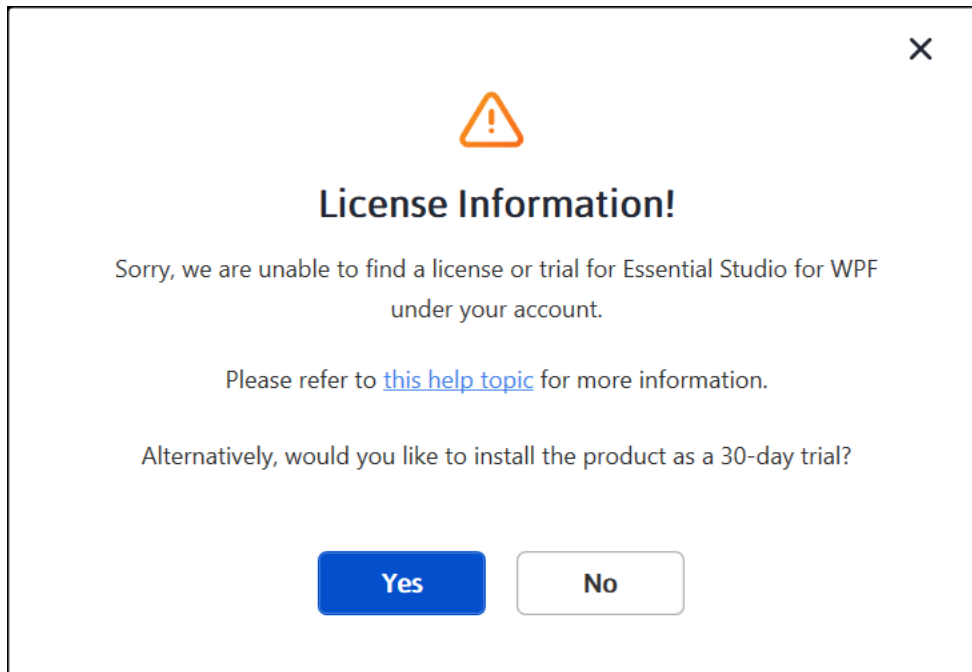
*Unable to find a valid license or trial**Problem*

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

<em>Offline installer</em>



<em>Online installer</em>



#### Reason

The following are possible causes of this error:

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

#### Suggested solution

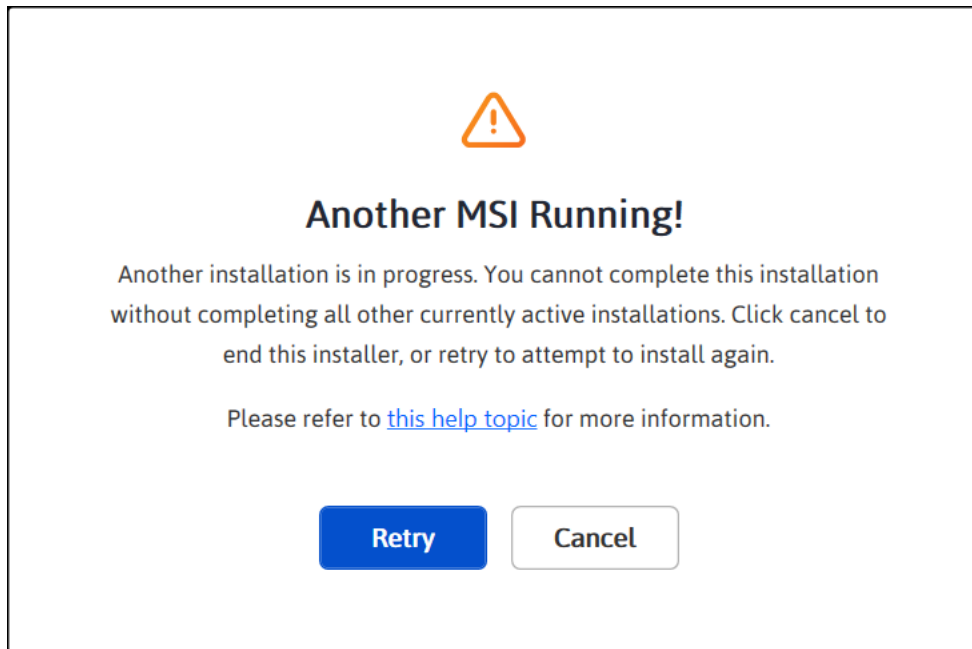
You can choose from the options listed below.

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

#### Unable to install because of another installation

##### Problem

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



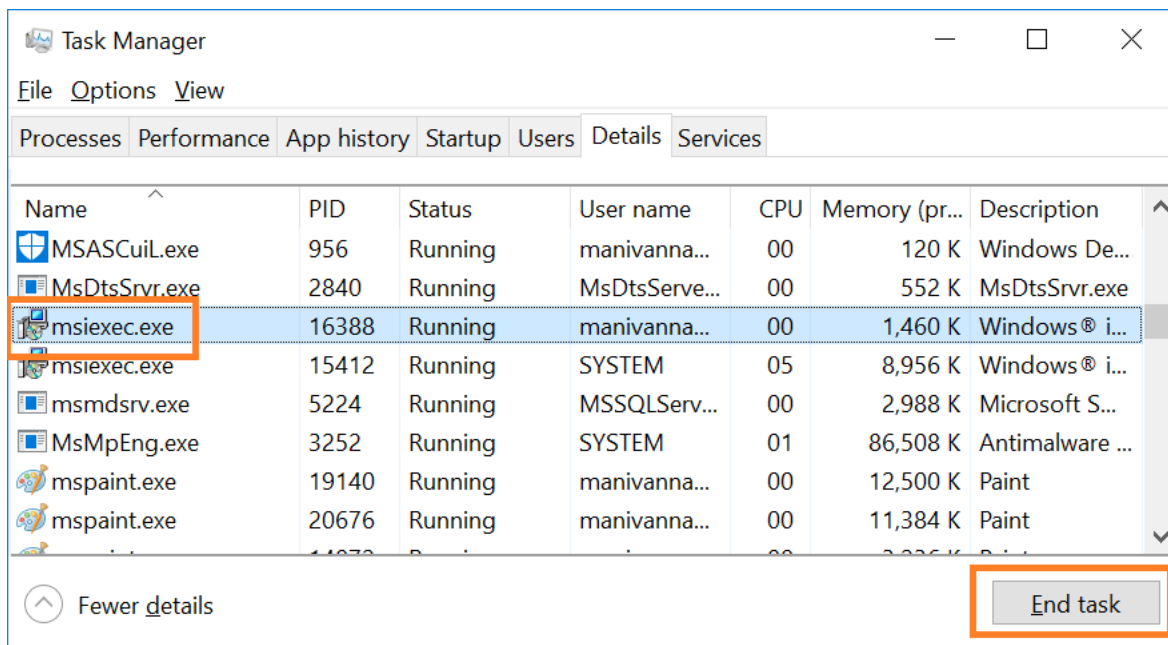
#### Reason

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

#### Suggested solution

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

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

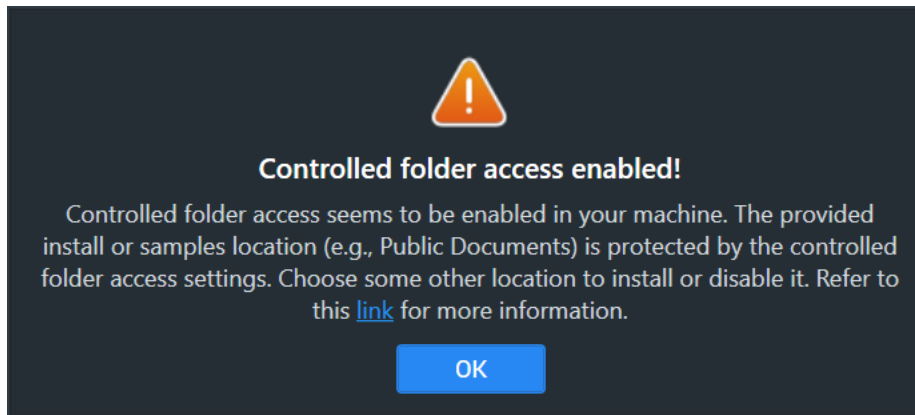


Unable to install due to controlled folder access

*Problem*

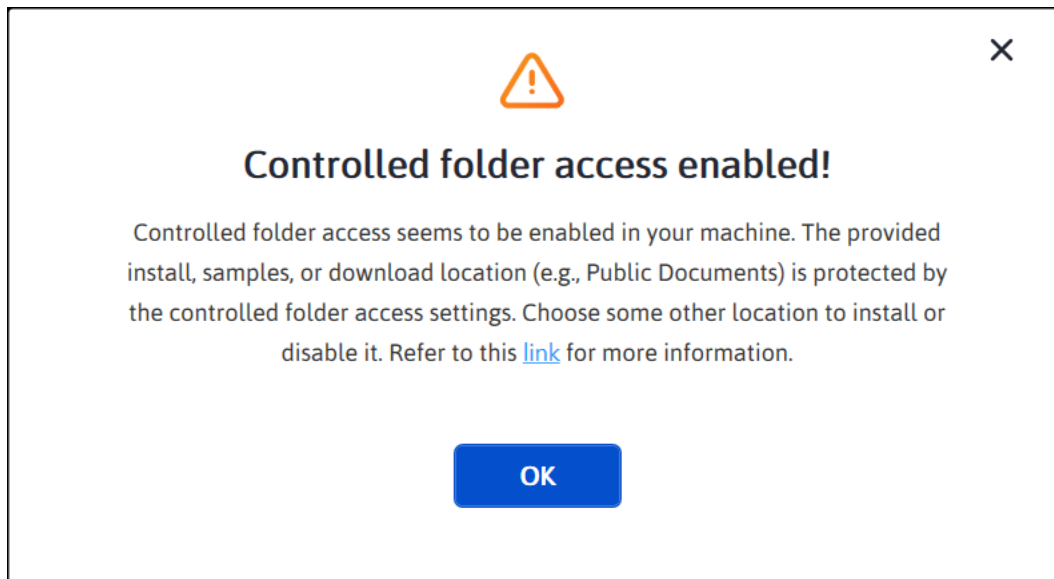
*Offline:*

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



*Online:*

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



*Reason*

You have enabled controlled folder access settings on your computer.

*Suggested solution*

Select a different location to install or deactivate your machine's controlled folder access settings, and then try installing.

## Upgrading Syncfusion WPF

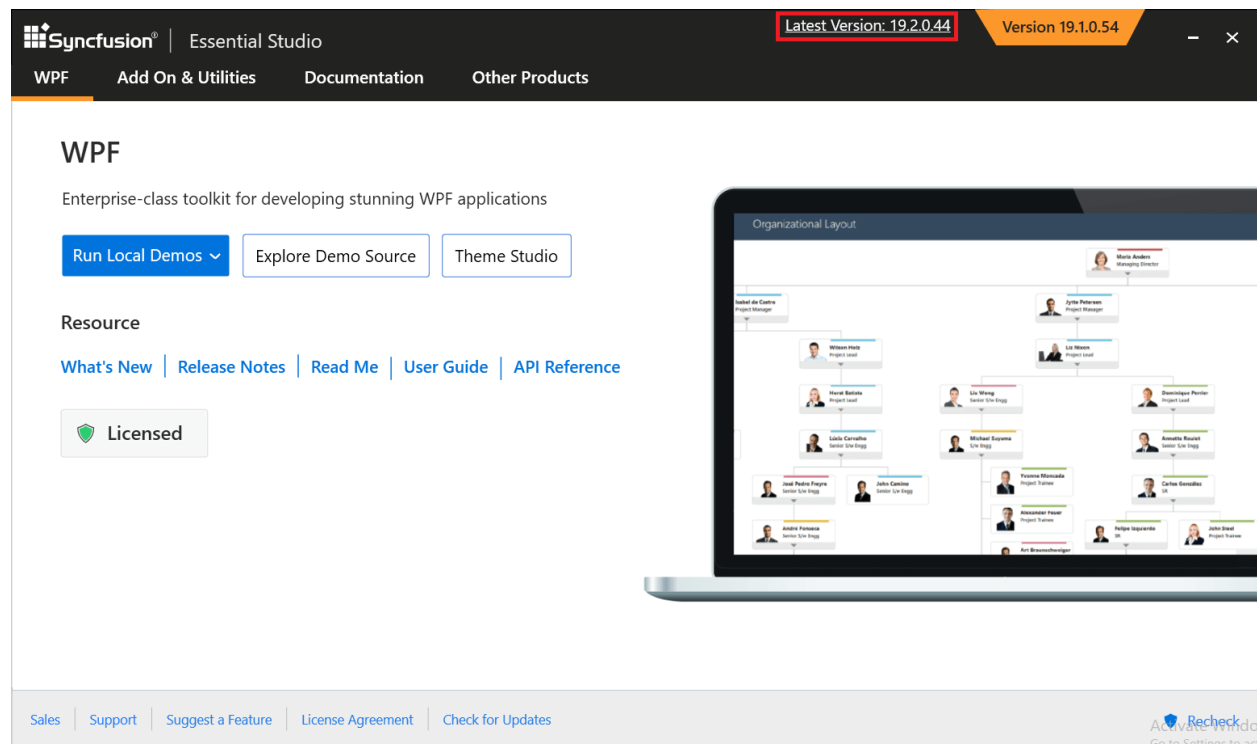
Syncfusion releases new volumes once every three months, with exciting new features. There will be one Service Pack release for this volume releases. Service Pack releases are provided to address major bug fixes in the volume releases.

You can upgrade to our latest version from any installed Syncfusion version.

See our "[Upgrade Guide](#)" for WPF to learn more about the "Breaking Changes, Bug Fixes, Features and Known Issues" between your current version and the latest version you are trying to upgrade.

## Upgrading to the latest version

The most recent version of Syncfusion WPF can be downloaded and installed by clicking on the "Latest Version: {Version}" link at the top of the Syncfusion WPF Control Panel.



You can also upgrade to the latest version just by downloading and installing the products you require from [this](#) link. The existing installed versions are not required to be uninstalled.

It is not required to install the Volume release before installing the Service Pack release. As releases for Volume and Service Packs work independently, you can install the latest version with major bug fixes directly.

## Upgrade from trial version to license version

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

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

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

## Licensing

<style>

license {

font-size: .88em!important;

margin-top: 1.5em; margin-bottom: 1.5em;

background-color: #fbefca;

padding: 10px 17px 14px;

}

</style>

### Syncfusion Licensing in WPF

We have introduced a new licensing system starting with version 16.2.0.x release of Essential Studio. These modifications apply to all evaluators and only to paid customers who use NuGet packages from [nuget.org](https://nuget.org). Starting with v16.2.0.x, if you use the evaluation installer or the NuGet feed to reference Syncfusion assemblies, you must also include the corresponding platform and version license key in your projects.

#### *Difference between unlock key and license key*

Please note that this license key is different from the installer unlock key that you might have used in the past and needs to be separately generated from Syncfusion website. Refer [this](#) KB article to know more about difference between the Syncfusion Unlock Key and the Syncfusion License Key.

Following licensing error will be shown if the license key is not registered in your projects, while using assemblies from evaluation installer or from the nuget.org.

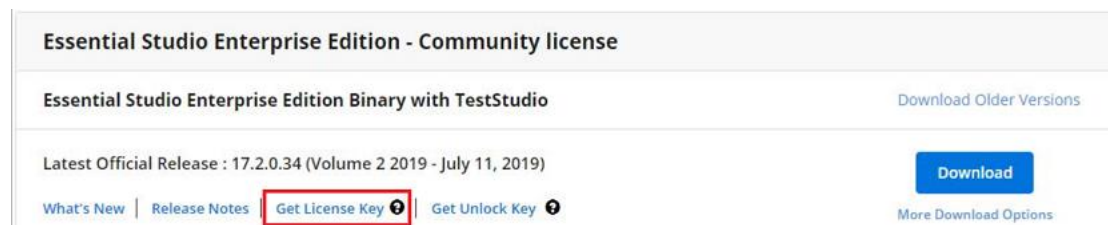
<div id="license">

This application was built using a trial version of Syncfusion Essential Studio. Please include a valid license to permanently remove this license validation message. You can also obtain a free 30 day evaluation license to temporarily remove this message during the evaluation period. Please refer to this [a href="/common/essential-studio/licensing/license-key"](/common/essential-studio/licensing/license-key)>help topic</a> for more information

</div>

### License Key Generation

License keys for WPF can be generated from the [License & Downloads](#) or [Trail & Downloads](#) section from your Syncfusion account.



---

**Information:** \* Syncfusion license keys are **version and platform specific**, refer to the [KB](#) to generate the license key for the required version and platform.

---

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

### License Key Registration

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

#### C#

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

---

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

---

#### WPF

You can register the license key in App constructor of **App.xaml.cs** in C#. If App constructor not available in **App.xaml.cs**, create the "App()" constructor in **App.xaml.cs** and register the license key inside the constructor. In Visual Basic, register the license code in **App.xaml.vb**.

#### C#

```
public partial class App : Application
{
    public App()
    {
        //Register Syncfusion license
        Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
    }
}
```

#### VB.NET

```
Private Sub New()
    'Register Syncfusion License
    Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("YOUR LICENSE KEY")
End Sub
```

### Common Licensing Errors

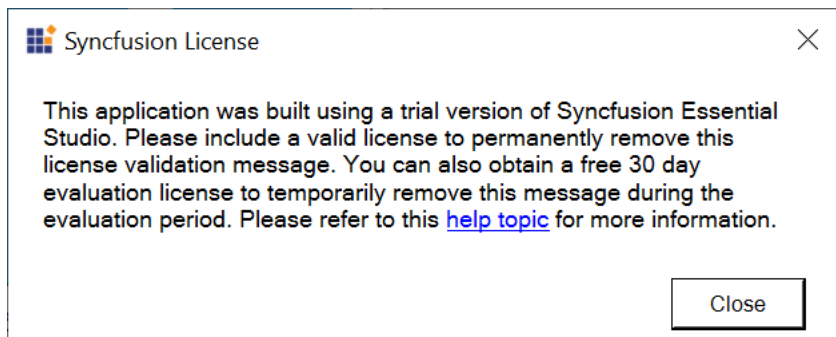
Licensing error popup is displayed with various messages under different circumstances. Here are some ways to resolve different issues.

#### License key not registered

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



**Error message:** This application was built using a trial version of Syncfusion Essential Studio. Please include a valid license to permanently remove this license validation message. You can also obtain a free 30 day evaluation license to temporarily remove this message during the evaluation period. Please refer to this help topic(<https://help.syncfusion.com/es/licensing/>) for more information.

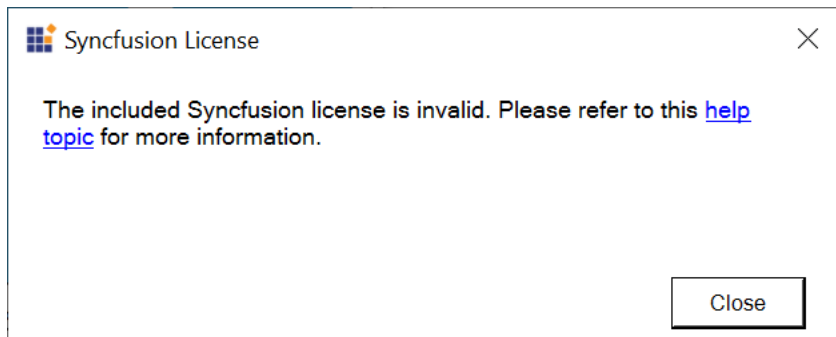


**Solution:** Generate a valid license key from here [Licensed users](#) or [Trial users](#) for a specific version and platform.

#### Invalid key

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

**Error Message:** The included Syncfusion license is invalid. Please refer to this help topic(<https://help.syncfusion.com/es/licensing/invalid/>) for more information.

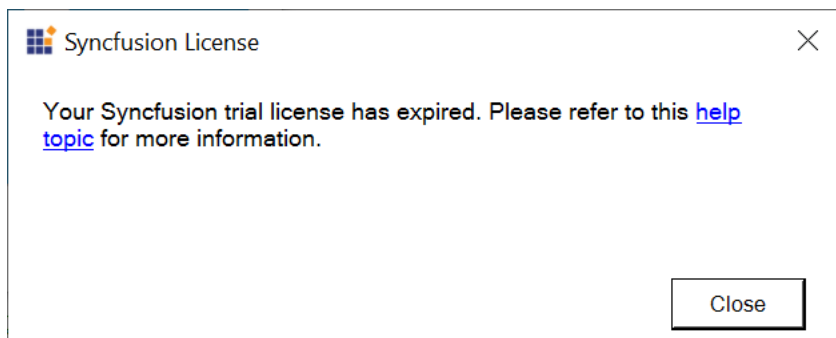


**Solution:** Generate a valid license key from here [Licensed users](#) or [Trial users](#) for a specific version and platform.

#### Trial Expired

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

**Error Message:** Your Syncfusion trial license has expired. Please refer to this help topic(<https://help.syncfusion.com/es/licensing/expired>) for more information.

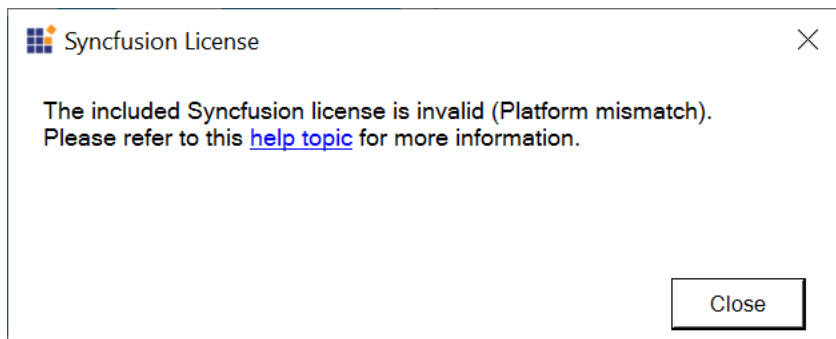


**Solution:** Purchase from [here](#) to get a valid Syncfusion license.

#### Platform Mismatch

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

**Error Message:**The included Syncfusion license is invalid (Platform mismatch). Please refer to this help topic(<https://help.syncfusion.com/es/licensing/platform-mismatch/>) for more information.

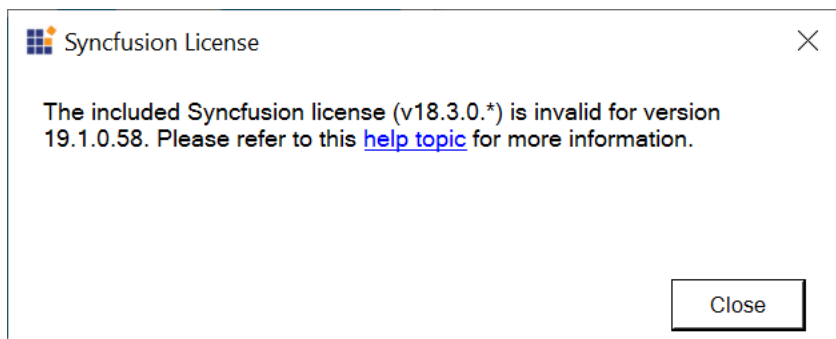


**Solution:** Generate a valid license key from here [Licensed users](#) or [Trial users](#) for a specific version and platform.

#### Version Mismatch

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

**Error Message:**The included Syncfusion license ({Registered Version}) is invalid for version {Required version}. Please refer to this help topic(<https://help.syncfusion.com/es/licensing/version-mismatch/>) for more information.



**Solution:** Generate a valid license key from here [Licensed users](#) or [Trial users](#) for a specific version and platform. Kindly follow the [KB](#) to generate license key.

Could not load Syncfusion.Licensing.dll assembly version...?

Make sure that all the referenced Syncfusion assemblies are of the same version. Try cleaning and rebuilding the application to resolve assembly conflict issues.

## Licensing FAQ

How to upgrade from Trial version after purchasing a license?

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

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

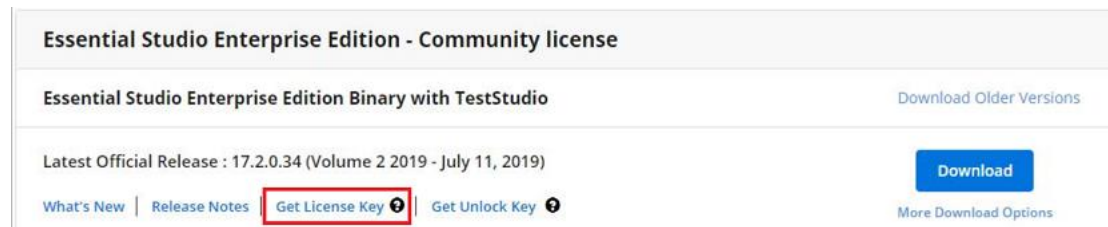
---

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

---

Where can I get a license key?

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



---

**Information:** \* Syncfusion license keys are **version and platform specific**, refer to the [KB](#) to generate the license key for the required version and platform.

---

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

Registering Syncfusion account for direct NuGet.org user

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

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

## Applying the Patches

Syncfusion provides patch installer for major version or service pack version, either to add new features or to fix issues. You have to install the patches in the order you have received.

### Installing the Patch installer

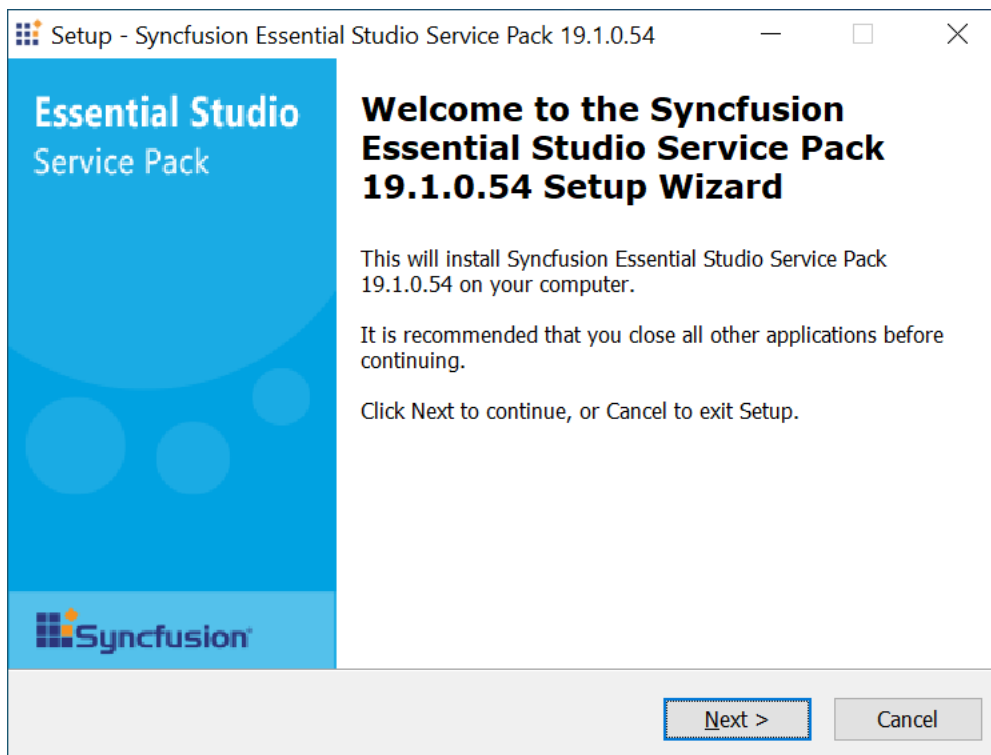
The steps below show how to install a patch.

---

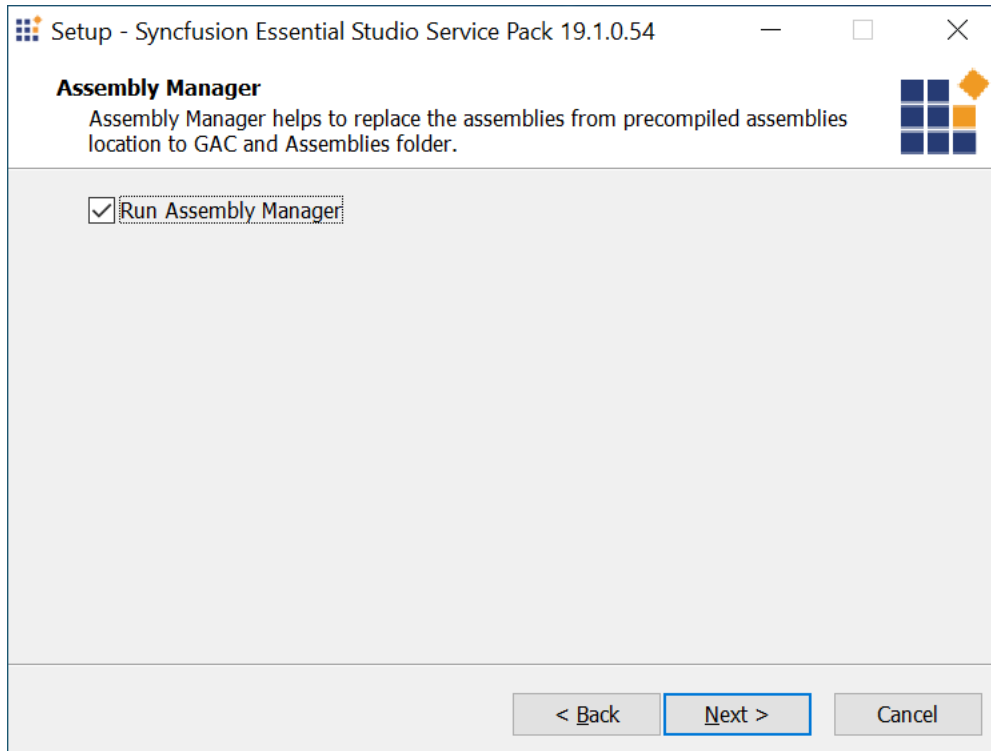
**Information:** Before installing the patch, ensure that corresponding Essential Studio version platform has been installed in your machine.

---

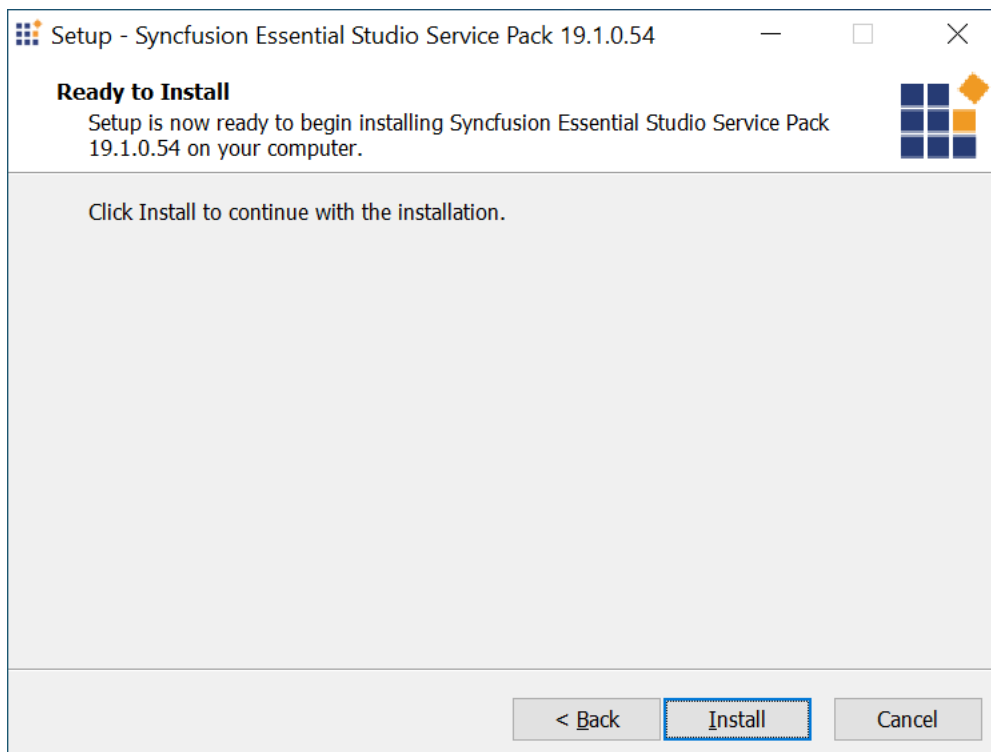
1. Double-click the Syncfusion Essential Studio patch installer. The Syncfusion Essential Studio Service Pack opens.



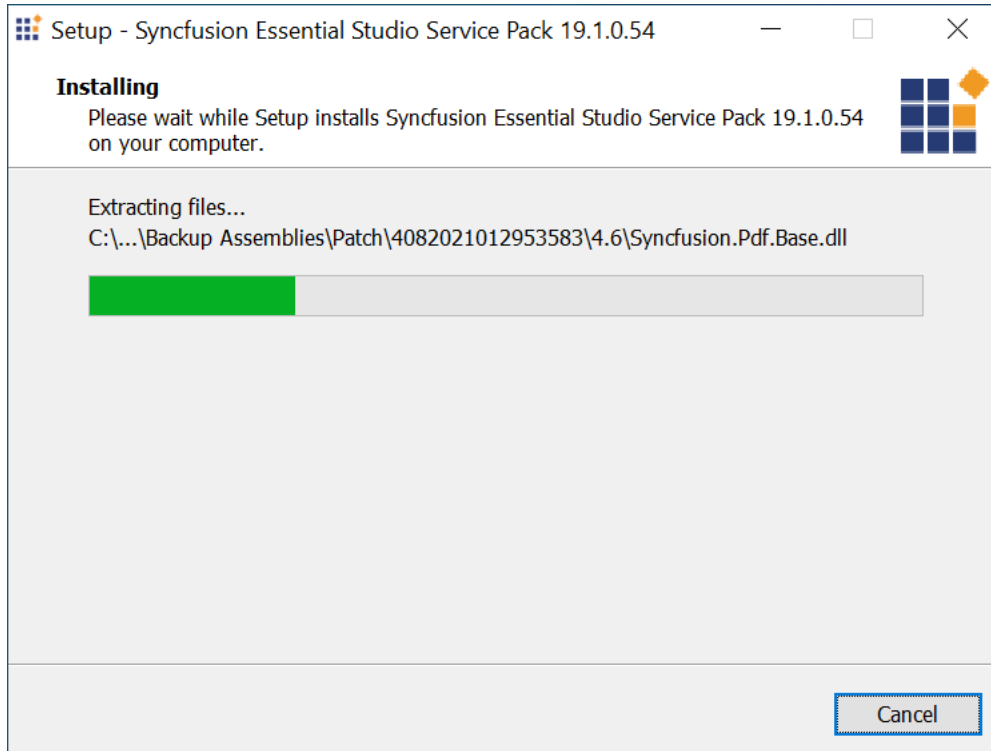
2. Click Next. The Assembly Manager screen opens.



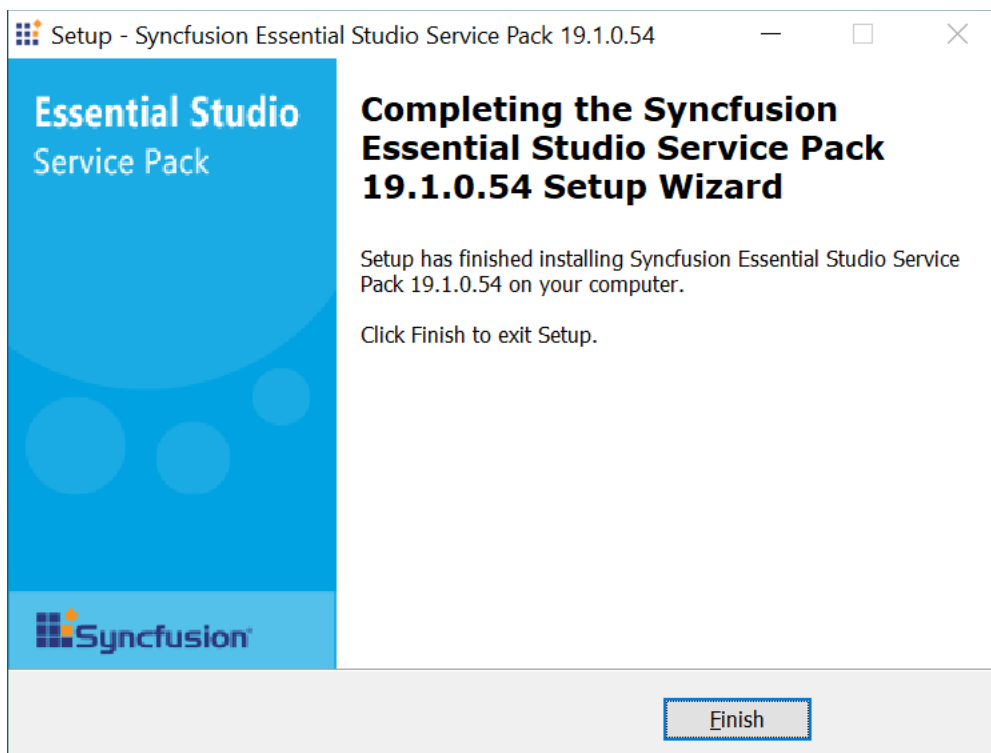
3. Select the Run Assembly Manager check box to install the assemblies in GAC.
4. Click Next. The Ready To Install screen opens.



5. Click Install to continue installing.



**Note:** The patch is installed on your computer, and a dialog box appears when the installation is complete.



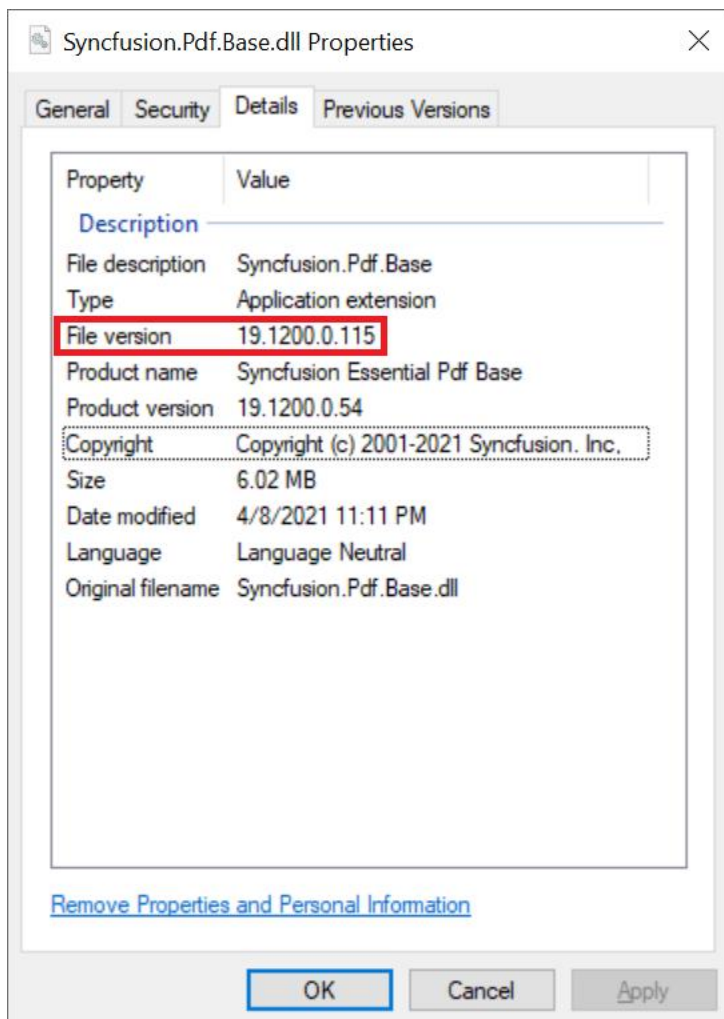
6. Click Finish.

The new assemblies are placed in the Pre-Compiled Assemblies folder. These new assemblies can be referenced in your project.

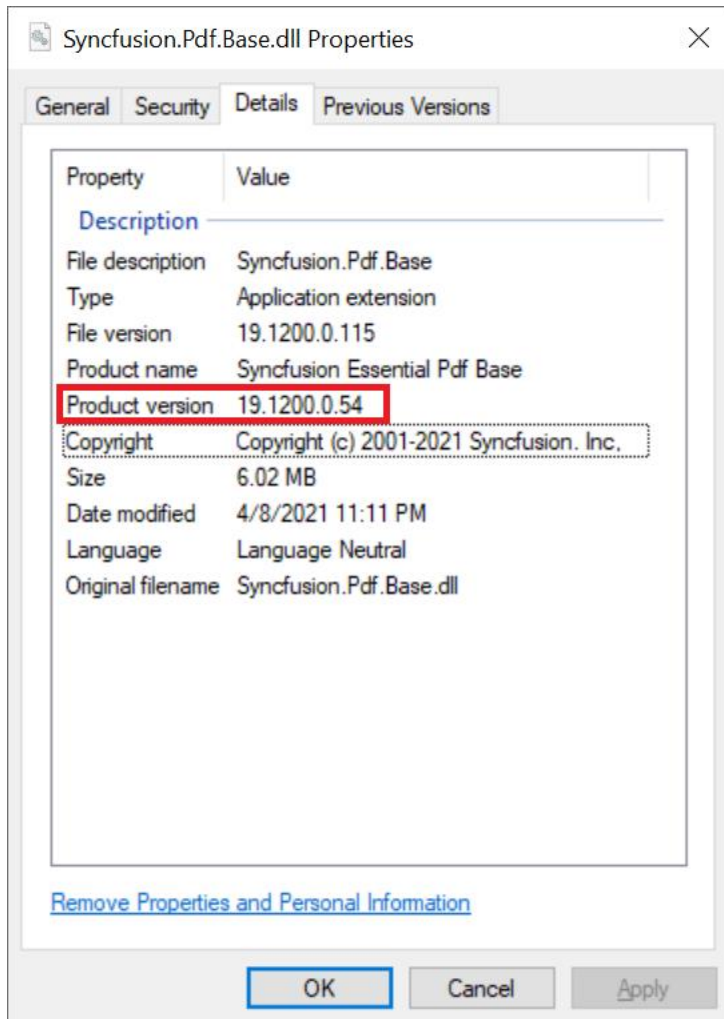
### Patch Assembly Version Format

In the patch assembly, the **File Version** and **Product Version** will be different. Product Version will be the release version and File Version will be the increment of the release version's **revision** number. For each patch, the File Version will be a different one. You can differentiate between the build and patch assemblies by File Version.

#### File Version of the assembly shipped in build:



#### Product Version of the assembly shipped in patch:



## Visual Studio Integration

### Syncfusion WPF Extension

The Syncfusion WPF Studio Extensions can be accessed through the Syncfusion Menu to create and configure the project with Syncfusion references in Visual Studio. The Syncfusion WPF Extensions supports Microsoft Visual Studio 2013 or higher.

---

**Note:** Syncfusion Extension is published in Visual Studio Marketplace. You can download WPF Extensions from [Visual Studio Marketplace](#).

---

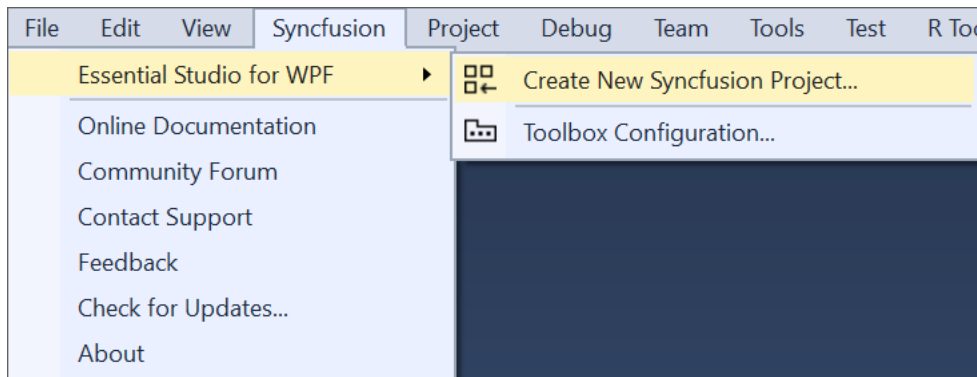
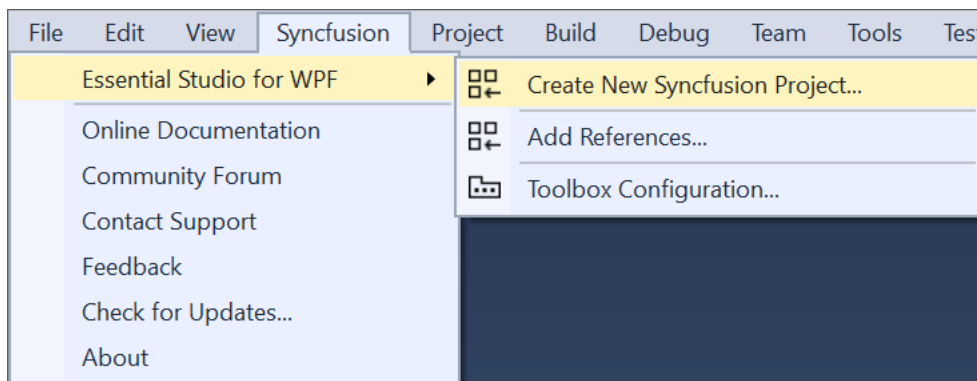
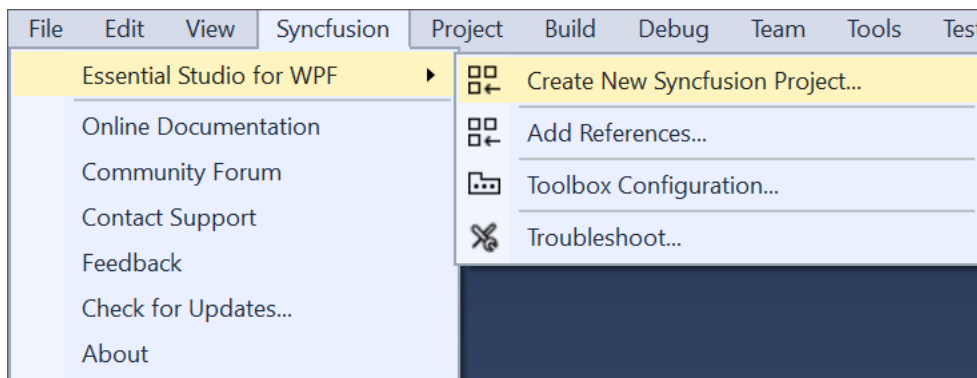
**Information:** The Syncfusion WPF menu option is available from v17.1.0.32.

---

The Syncfusion provides the following extension supports in Visual Studio:

1. [Create Project](#): Creates the Syncfusion WPF application by adding the required Syncfusion assemblies and XAML.
2. [Add Item](#): Add Syncfusion WPF Window into the WPF application with add Syncfusion WPF assemblies/NuGet packages
3. [Add References](#): Add the required Syncfusion assembly to WPF project reference based on the selected control(s).
4. [Toolbox Configuration](#): Configure the Syncfusion controls into the Visual Studio .NET toolbox.
5. [Troubleshooter](#): Troubleshoots the project with the Syncfusion configuration and apply the fix like, wrong Framework Syncfusion assembly added to the project or missing any Syncfusion dependent assembly of a referred assembly.



**No project selected in Visual Studio****Selected Microsoft WPF application in Visual Studio****Selected Syncfusion WPF application in Visual Studio**

---

**Note:** In Visual Studio 2019, Syncfusion menu is available under Extensions in Visual Studio menu.

---

### Toolbox Configuration

The Syncfusion Toolbox Installer utility adds the Syncfusion WPF controls into the Visual Studio .NET toolbox.

---

**Note:** Toolbox configuration support is not available for the Visual Studio Express Edition. However, you can manually configure the Syncfusion controls into the Visual Studio Express Toolbox. To do so, refer the [Manual Toolbox Configuration](#).

---

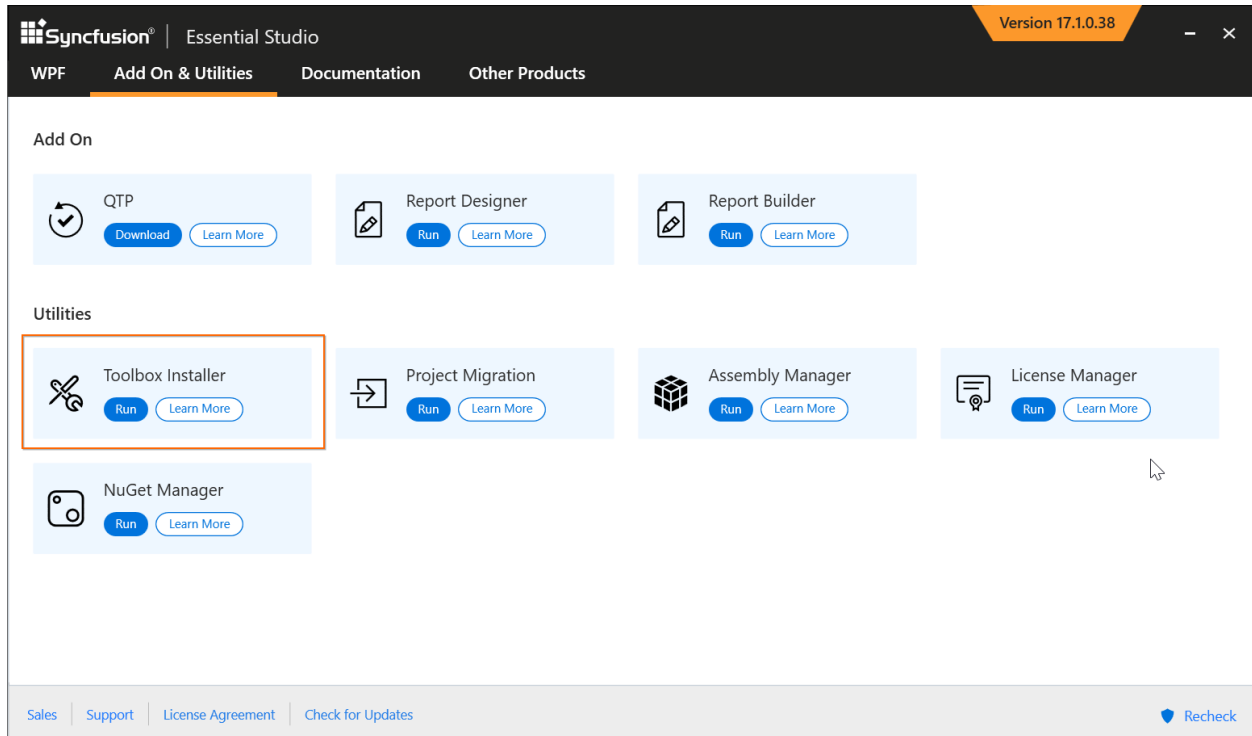
Syncfusion controls will be automatically configured in the Visual Studio toolbox, while installing the Syncfusion WPF installer, if the <b>“Configure Syncfusion Controls in Visual Studio”</b> checkbox is selected from installer UI.

Use the following steps to add the Syncfusion WPF controls through the Syncfusion Toolbox Installer:

1. To launch Toolbox configuration utility, follow either one of the options below:

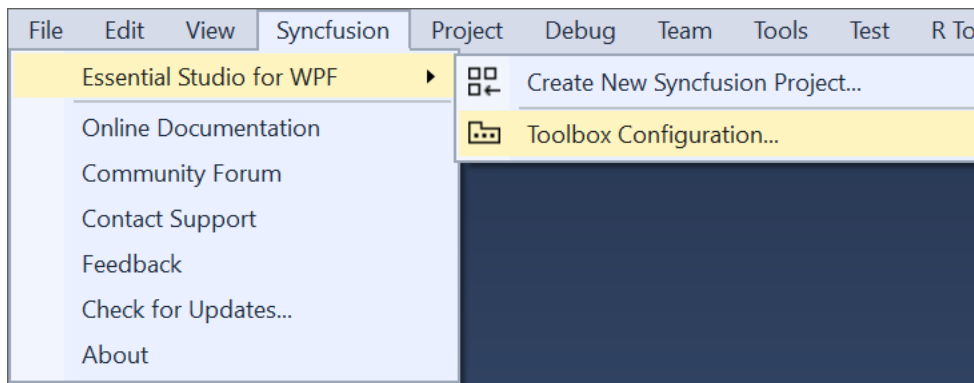
#### Option 1:

To open the Syncfusion Control Panel, click **Add On and Utilities > Toolbox Installer**.



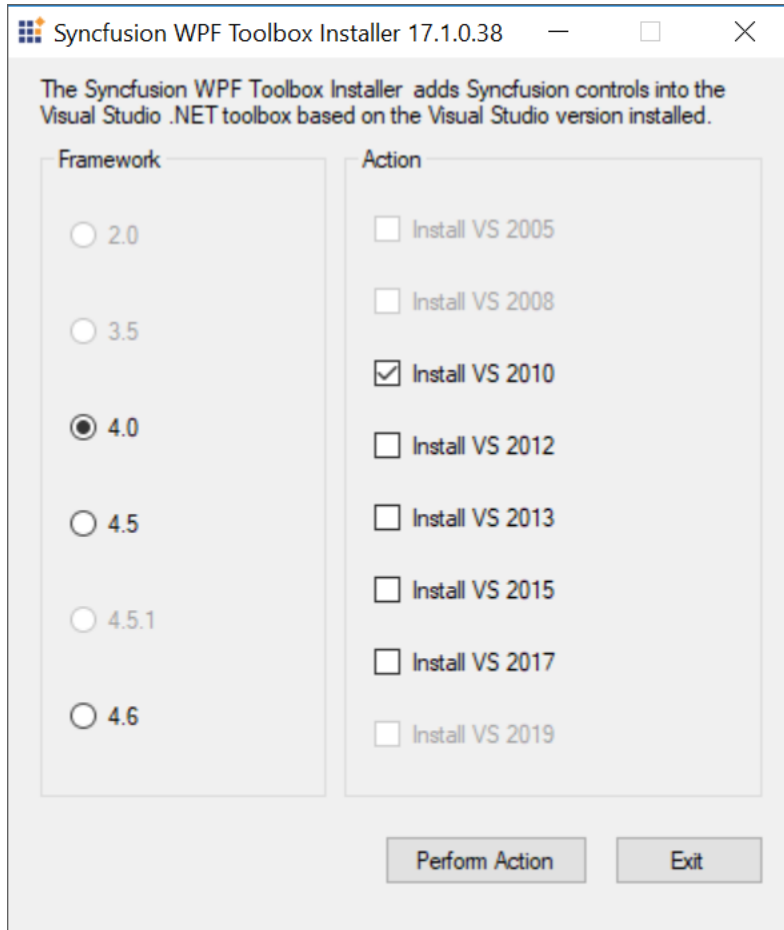
#### Option 2:

Click **Syncfusion menu** and choose **Essential Studio for WPF > Toolbox Configuration...** in **Visual Studio**



**Note:** In Visual Studio 2019, Syncfusion menu is available under Extensions in Visual Studio menu.

2. Toolbox Installer will be opened.



The following options are available in Toolbox Configuration:

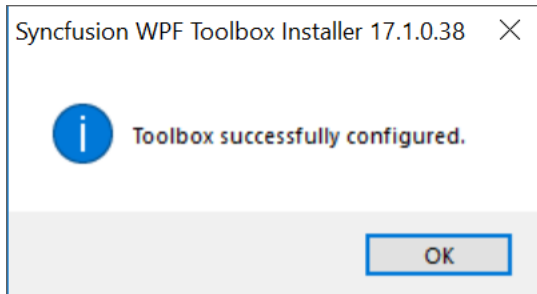
- Install VS2005 – Configures Framework 2.0 Syncfusion controls in VS 2005 toolbox.
- Install VS2008 – Configures Framework 3.5 Syncfusion controls in VS 2008 toolbox.
- Install VS2010 – Configures Framework 4.0 Syncfusion controls in VS 2010 toolbox.
- Install VS2012 – Configures Framework 4.5 Syncfusion controls in VS 2012 toolbox.
- Install VS2013 – Configures Framework 4.5.1 Syncfusion controls in VS 2013 toolbox.
- Install VS2015 – Configures Framework 4.6 Syncfusion controls in VS 2015 toolbox.
- Install VS2017 – Configures Framework 4.6 Syncfusion controls in VS 2017 toolbox.
- Install VS2019 – Configures Framework 4.6 Syncfusion controls in VS 2019 toolbox

---

**Note:** You can also configure Syncfusion controls from a lower version Framework assembly to higher version of Visual Studio.

---

3. An Information message is displayed indicating the successful configuration of Toolbox. Click OK.



**Note:** \* You must reset the toolbox, when the installed controls are not reflected properly in the Toolbox.

- This tool configures only the controls that are located under {Installed Location}\Assemblies\{Framework version}.

#### Configuring toolbox for WPF .NET 5.0 projects

From 2021 Volume 1, Syncfusion started providing toolbox support for WPF .NET 5.0 framework in Visual Studio 2019. Syncfusion controls will be automatically configured in the Visual Studio 2019 toolbox for WPF .NET 5.0 project, after installing the Syncfusion WPF installer.

**Note:** \* Syncfusion included this toolbox support for .NET 5.0 WPF platform from 2021 Volume 1 release version v19.1.0.54 only.

- After installing the WPF setup, if the project created with TargetFramework .NET Core 3.1 and changed to .NET 5.0, you need to restart Visual Studio to get the Syncfusion controls in Visual Studio Toolbox.
- Visual Studio 2019 16.7 Preview 2 and later is required

#### Upgrading the Syncfusion WPF toolbox .NET 5.0 controls without installing the build

You can upgrade the Syncfusion WPF toolbox for .NET 5.0 control with NuGet packages downloaded from [nuget.org](https://nuget.org). Download "[Syncfusion.UI.WPF.NET](https://nuget.org/packages/Syncfusion.UI.WPF.NET)" package from nuget.org in your machine.

Use the following steps to add the Syncfusion WPF controls through Syncfusion NuGet packages:

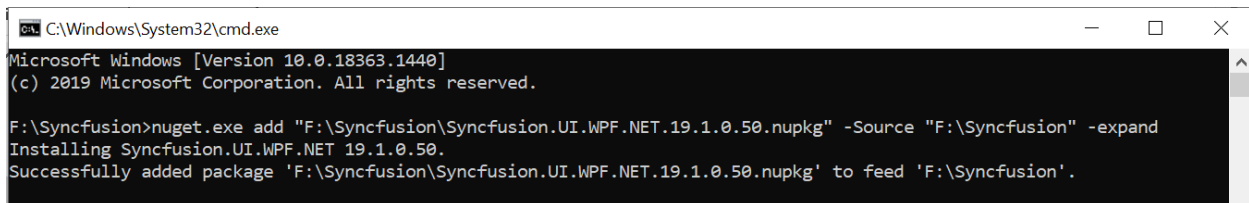
##### step 1:

Extract "**Syncfusion.UI.WPF.NET**" package by using the below commands.

Open Command prompt from nuget.exe path and run the following commands

**Command:** {nuget.exe path} add "F:\Syncfusion\Syncfusion.UI.WPF.NET.{version}.nupkg" -Source "F:\Syncfusion\Expand" -expand

**Example:** F:\Syncfusion>nuget.exe add "F:\Syncfusion\Syncfusion.UI.WPF.NET.19.1.0.50.nupkg" -Source "F:\Syncfusion" -expand



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1440]
(c) 2019 Microsoft Corporation. All rights reserved.




F:\Syncfusion>nuget.exe add "F:\Syncfusion\Syncfusion.UI.WPF.NET.19.1.0.50.nupkg" -Source "F:\Syncfusion" -expand
Installing Syncfusion.UI.WPF.NET 19.1.0.50.
Successfully added package 'F:\Syncfusion\Syncfusion.UI.WPF.NET.19.1.0.50.nupkg' to feed 'F:\Syncfusion'.
```

**step 2:**

Open “**Syncfusion Toolbox for WPF.config**” file from the following location.

**Location:** "C:\Program Files (x86)\NuGet\Config\Syncfusion Toolbox for WPF.config"

PC > Local Disk (C:) > Program Files (x86) > NuGet > Config

<input type="checkbox"/> Name	Date modified	Type	Size
 Microsoft.VisualStudio.Offline	7/2/2020 12:20 AM	XML Configuration File	1 KB
<input checked="" type="checkbox"/>  Syncfusion Toolbox for WPF	3/11/2021 4:51 PM	XML Configuration File	1 KB
 Xamarin.Offline	8/28/2020 1:24 PM	XML Configuration File	1 KB

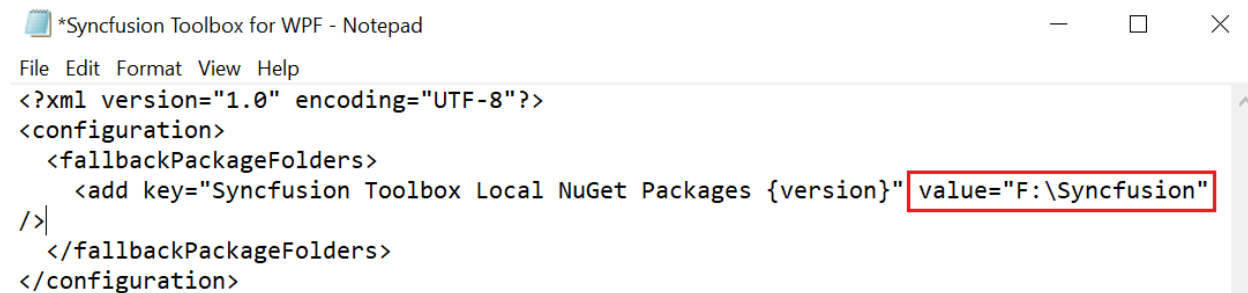
Or you can create this file in the same location by using the XML format given below

**XML**

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <fallbackPackageFolders>
    <add key="Syncfusion Toolbox Local NuGet Packages {version}"
value="F:\Syncfusion" />
  </fallbackPackageFolders>
</configuration>
```

**step 3:**

Update extracted Syncfusion NuGet package path in **value** attribute.

**Example:****step 4:**

Now restart the Visual Studio 2019 to get populate the latest Syncfusion controls in Toolbox.

**Configuring toolbox for .NET Core 3.1 projects**

To configure the Syncfusion toolbox in the WPF .NET Core application, the Syncfusion NuGet packages should be installed in the .NET Core application. After installing the Syncfusion NuGet packages in .NET Core application, the corresponding NuGet packages Syncfusion components will be configured in Visual Studio toolbox.

Please refer the documentation [link](#), to learn more about how to use the Syncfusion components using the Syncfusion NuGet packages in .NET Core application.

## Visual Studio Extensions

### Download and Installation

Syncfusion publishing the WPF Visual Studio extension in [Visual Studio marketplace](#). You can either install it from Visual Studio or download and install it from the Visual Studio marketplace.

#### *Install through the Visual Studio Manage Extensions*

The following steps explain how to install the extensions for Syncfusion WPF from Visual Studio **Manage Extensions**.

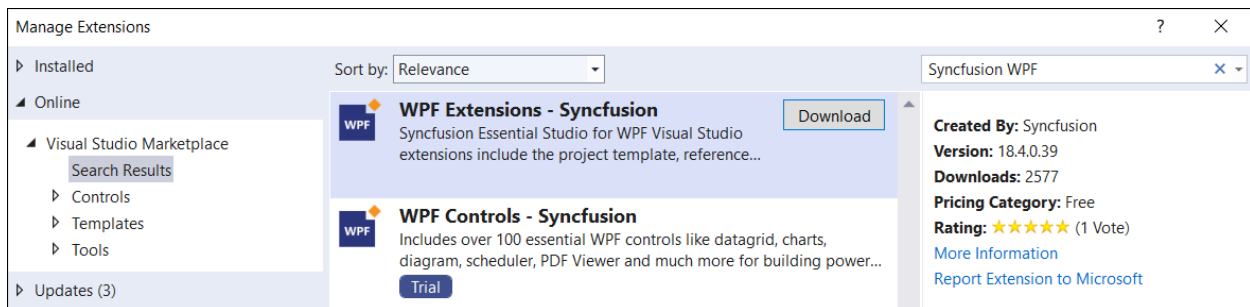
1. Open the Visual Studio.
2. Open Manage Extensions by navigating the menu, Extension ->Manage Extensions.

---

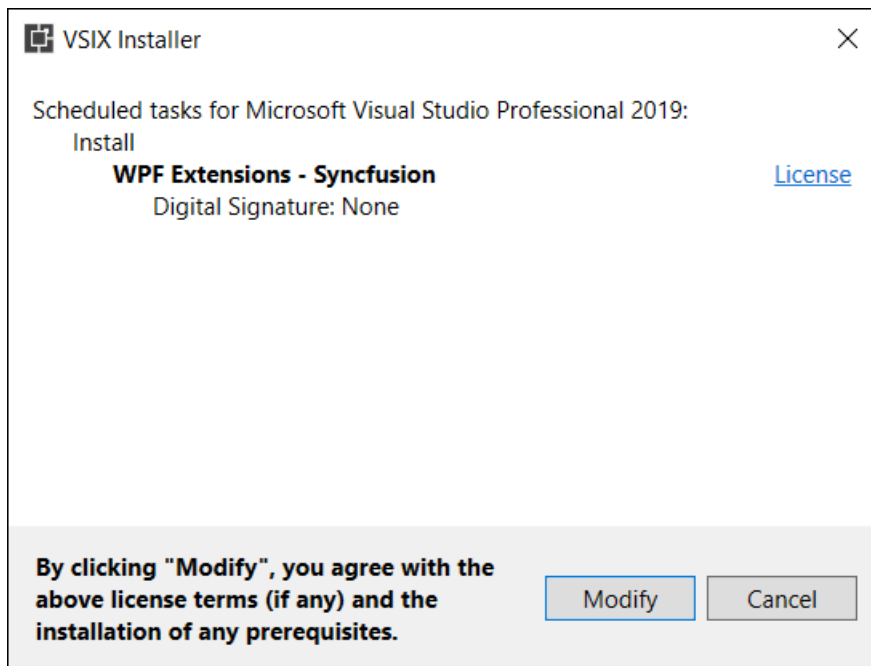
In Visual Studio 2017 or lower, goto Tools -> Extensions and Updates

---

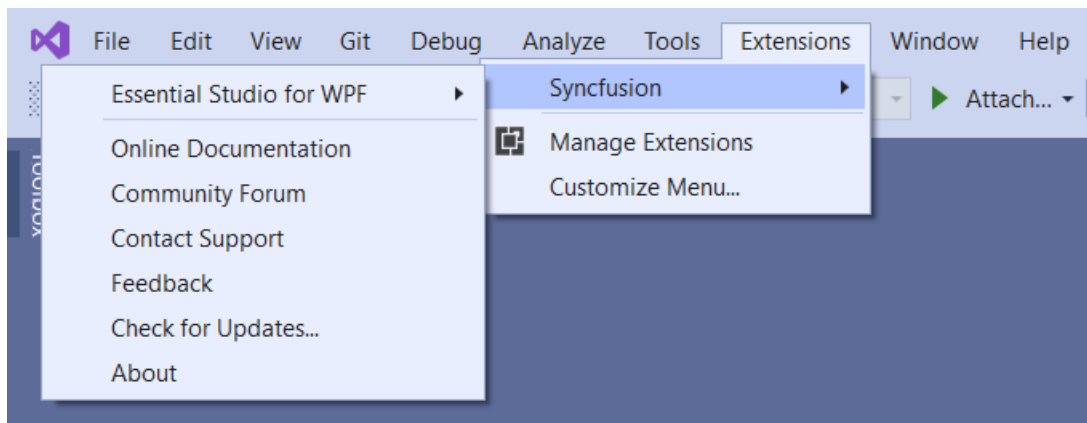
3. Select the Online tab on the left and type “Syncfusion WPF” in the search box.



4. In the “WPF Extensions - Syncfusion” extensions, click the Download button.
5. Once the extensions have been downloaded, close all Visual Studio instances to start the installation process. You will see the following VSIX installation prompt.



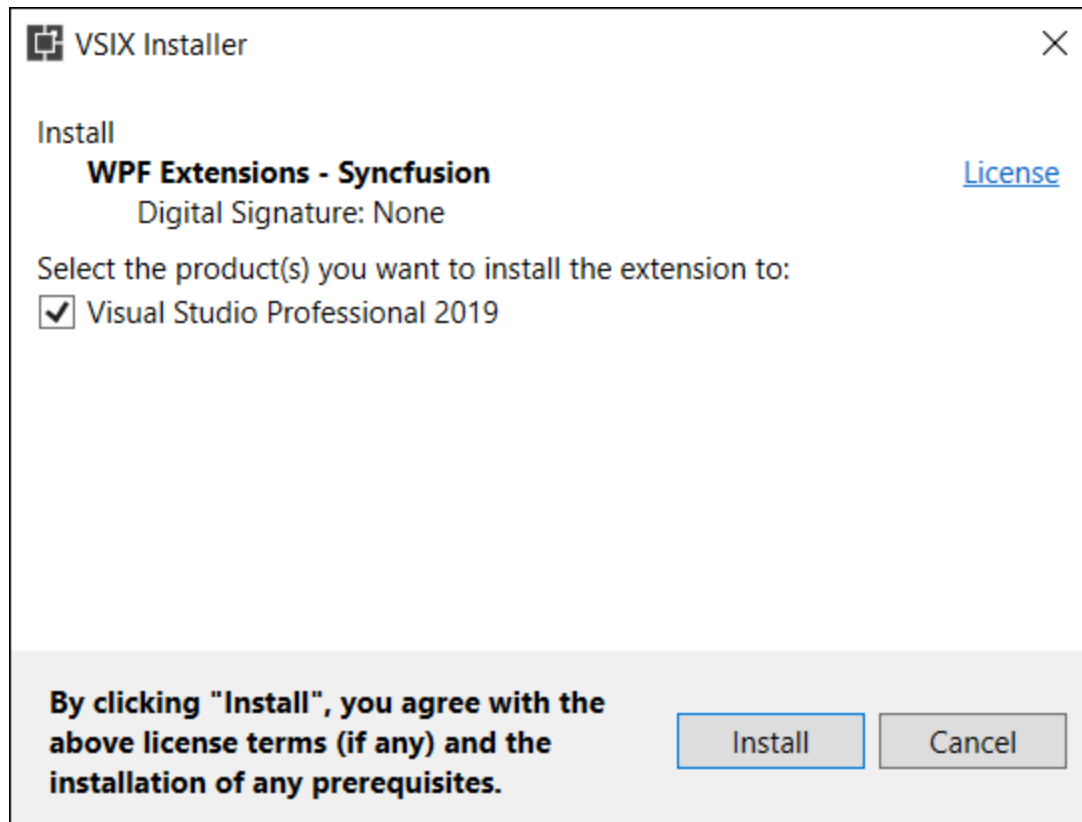
6. Click the Modify button.
7. After the installation is complete, open Visual Studio.
8. Now, under the menu Extensions, you can use the Syncfusion extensions from the Visual Studio application.



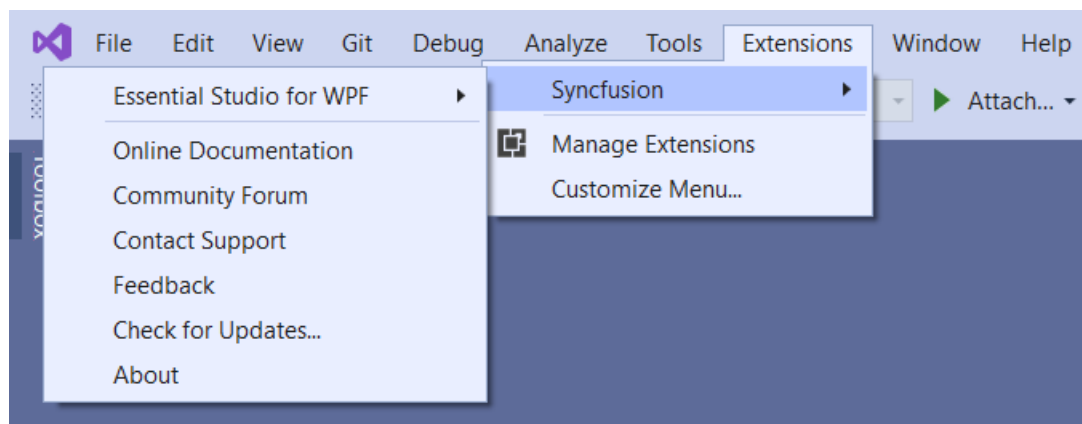
#### *Install from the Visual Studio Marketplace*

The following steps explain how to download and install the Syncfusion WPF extension from the Visual Studio Marketplace.

1. Download the [Syncfusion WPF Extensions](#) from the Visual Studio Marketplace.
2. Close all Visual Studio instances running, if any.
3. Double-click to install the downloaded VSIX file. You will see the VSIX installation prompts. If Visual Studio is not installed, it will not be possible to install the extension.



4. Click the Install button.
5. After the installation is complete, open Visual Studio 2019. You can now use Syncfusion extensions from the Visual Studio under the Extensions menu.



### Create WPF application

Syncfusion provides the Visual Studio Project Templates for the Syncfusion WPF platform to create the Syncfusion WPF Application by adding the required Syncfusion assemblies and XMAL.

---

**Information:** The Syncfusion WPF templates are available from v16.1.0.24.

---

Use the following steps to create the Syncfusion WPF project through the Visual Studio Project Template.



---

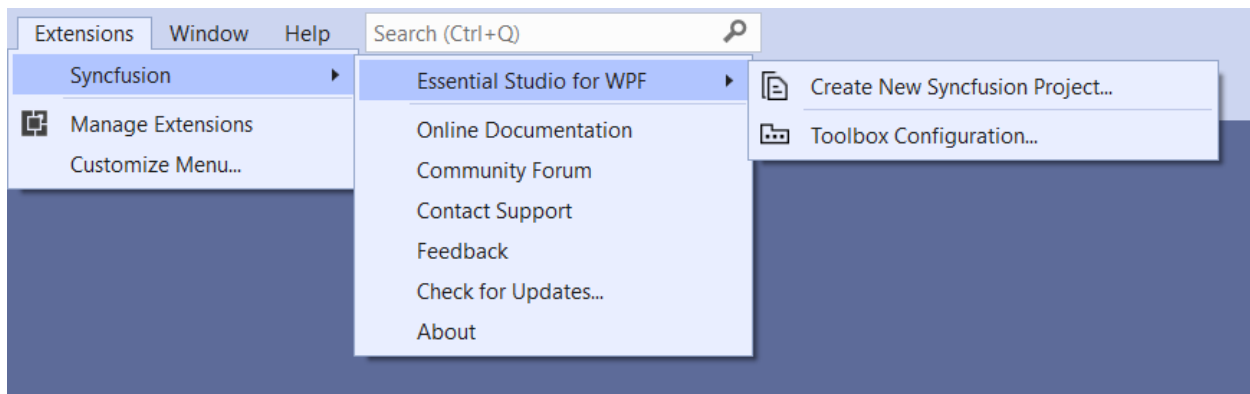
Before use the Syncfusion WPF Project Template, check whether the **WPF Extensions - Syncfusion** installed or not in Visual Studio Extension Manager by clicking on the Tools -> Extensions and Updates -> Installed for Visual Studio 2017 or lower and for Visual Studio 2019 by clicking on the Extensions -> Manage Extensions -> Installed.

---

1. To create a Syncfusion WPF project, follow either one of the options below:

**Option 1:**

Click **Extensions > Syncfusion Menu** and choose **Essential Studio for WPF > Create New Syncfusion Project...** in **Visual Studio**.



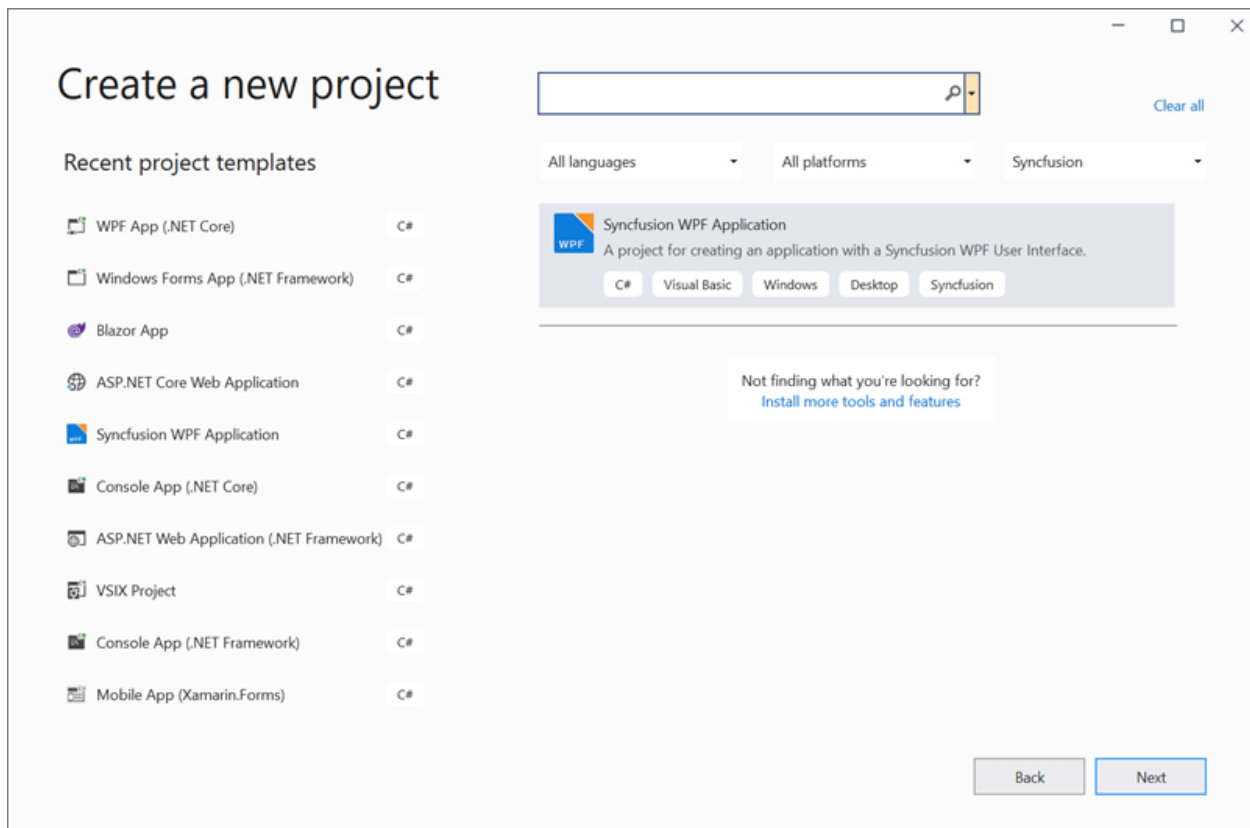
---

**Note:** In Visual Studio 2017 or lower, the Syncfusion menu is available in the Visual Studio menu.

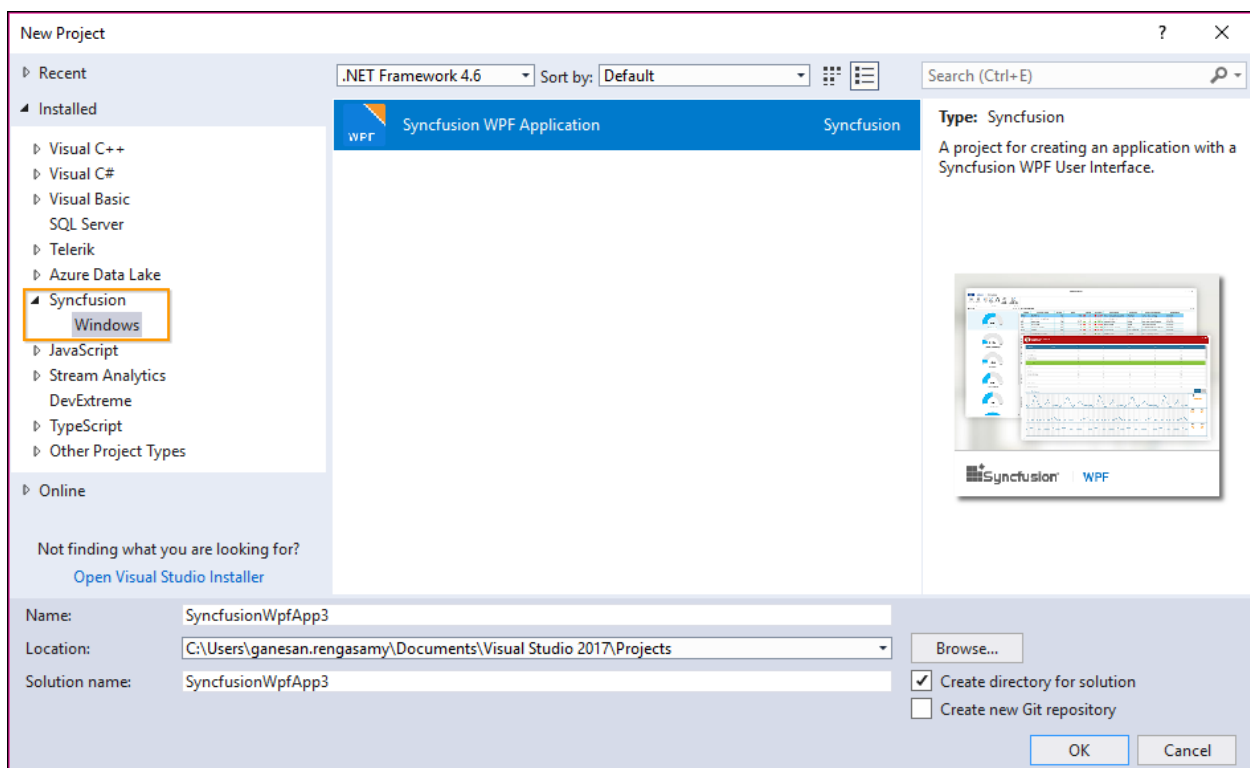
---

**Option 2:**

Choose **File -> New -> Project**. Opens a new dialog to create a new project. You can obtain the templates provided by Syncfusion for WPF by filtering the project type with Syncfusion or by using the Syncfusion keyword in the search option.



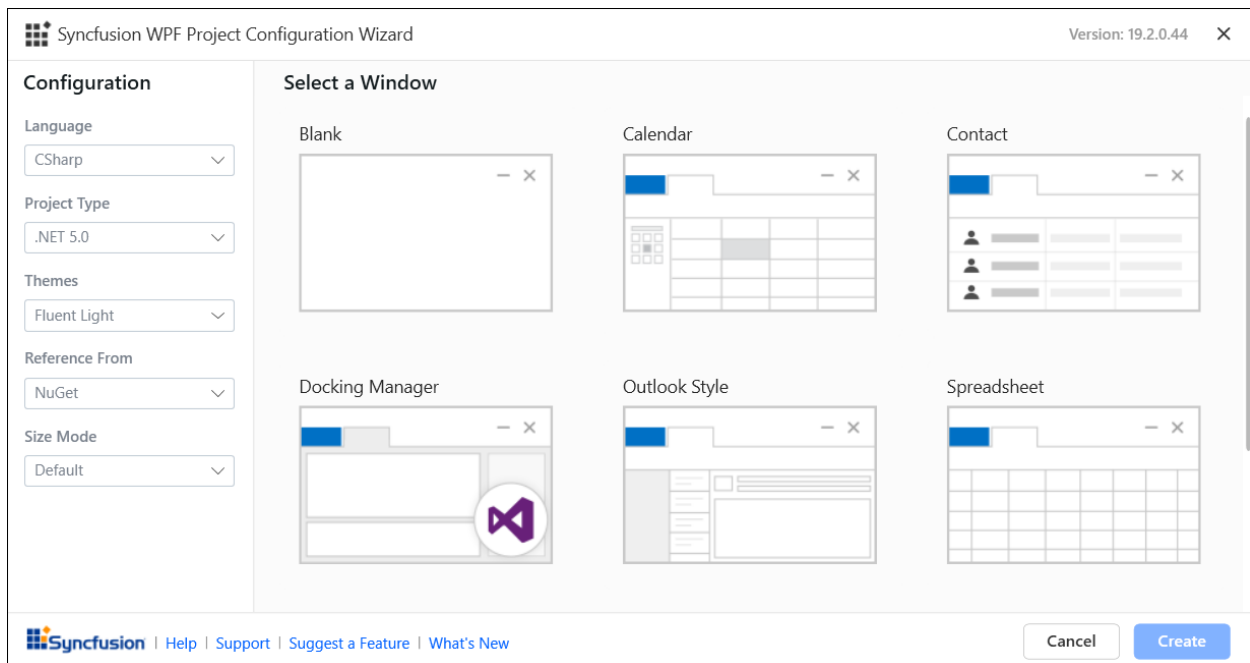
In Visual Studio 2017 or lower, Select File > New > Project and navigate to Syncfusion > Windows > Syncfusion WPF Application in Visual Studio.



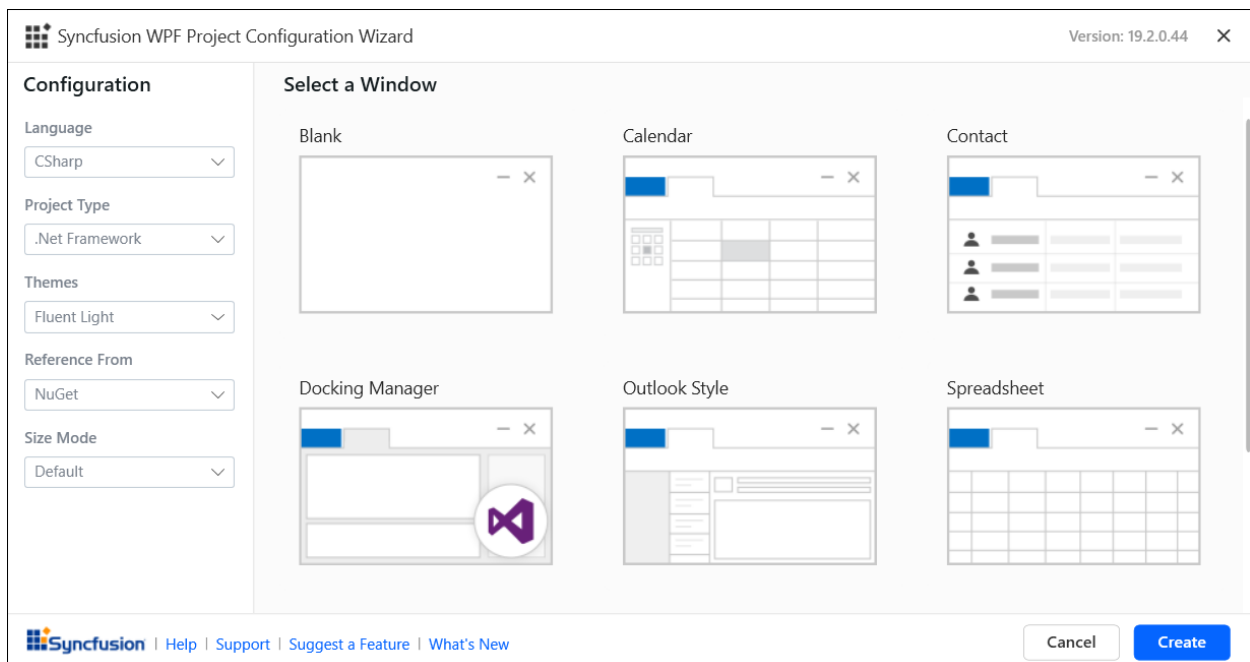
2. Name the **Project**, choose the destination location when required, and set the Framework of the project, then click **OK**.

**Note:** Minimum target Framework is 4.0 for Syncfusion WPF project templates.

3. Choose the options to configure the Syncfusion WPF Application by using the following Project Configuration Wizard.



In Visual Studio 2017 or lower, Syncfusion WPF Application project configuration wizard



## Project Configurations

**Project Type:** Select the project type, either .Net 5.0, .Net Core 3.1 or .Net Framework.

**Note:** Project type selection option will be available only in Visual Studio 2019 Syncfusion WPF Project template configuration. The .Net Core 3.1 and .Net 5.0 option will be listed in project type only when the .Net Core 3.0 and .Net 5.0 setup has been installed.

**Language:** Select the language, either CSharp or VB.

**Note:** C# language is available only when you choose .NET Core 3.1 or .Net 5.0 from project type option in Visual Studio 2019.

**Choose Theme:** Choose the required theme.

**Reference From:** Choose the assembly location, from where the assembly is added to the project.

**Note:** Installed location and GAC option will be available only when the Syncfusion Essential WPF setup has been installed. You can use NuGet option without installing the Syncfusion Essential WPF setup. Also, the GAC option will not be available when you choose .Net Core 3.1 and .Net 5.0 from project type option in Visual Studio 2019.

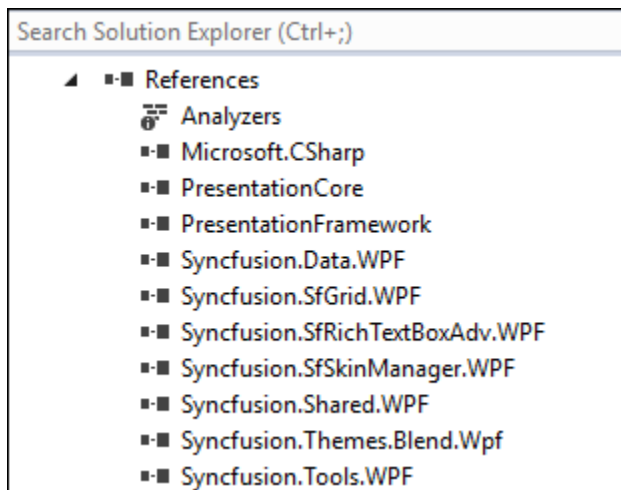
**Installed ES Build Version:** Choose the build version to add the corresponding version assemblies to the project.

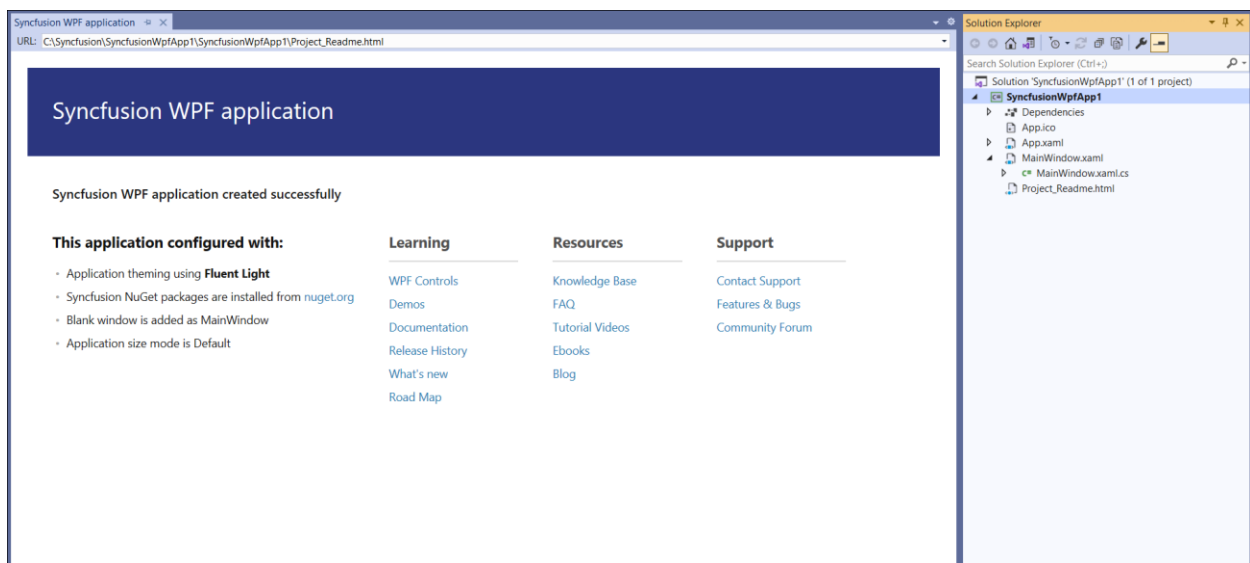
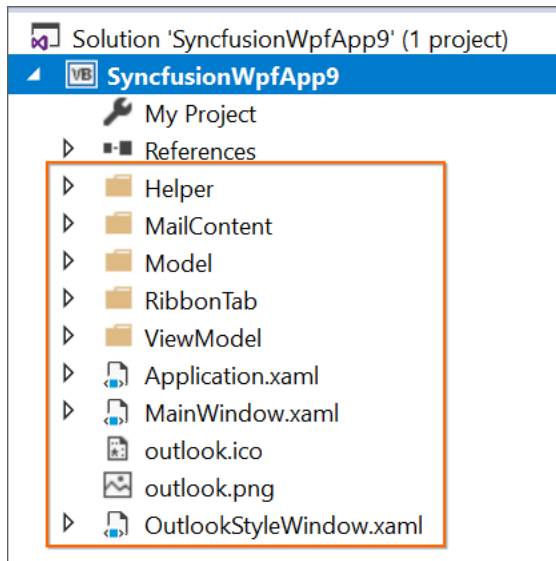
**Note:** Installed ES build version option will be available only when you install the Syncfusion Essential WPF setup and choose the assembly location as Installed Location or GAC.

**Size Mode:** Choose the Size Mode either Default or Touch.

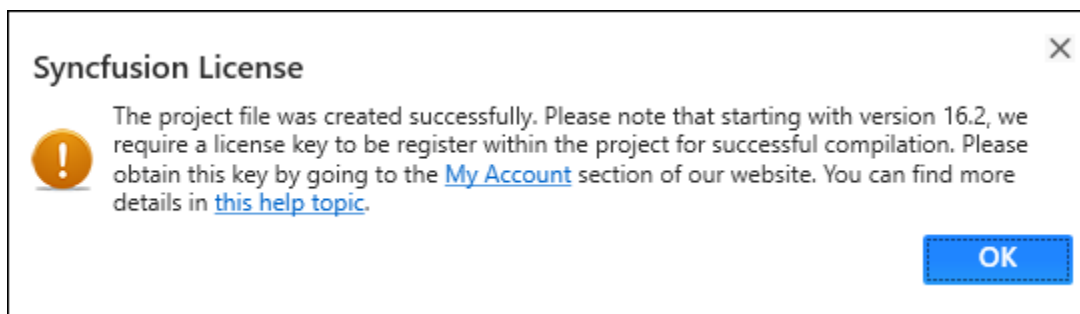
**Select Window:** Choose the window as required for application.

4. After the Project Configuration Wizard is done, the Syncfusion WPF project is created with required references and XAML.





- Then, the Syncfusion licensing registration required message box will be shown, if you installed the trial setup or NuGet packages since Syncfusion introduced the licensing system from 2018 Volume 2 (v16.2.0.41) Essential Studio release. Navigate to the [help topic](#), which is shown in the licensing message box to generate and register the Syncfusion license key to your project. Refer to this [blog](#) post to learn more about the licensing changes introduced in Essential Studio.



### Add Syncfusion window to WPF application

Syncfusion provides the Visual Studio Item Templates support to add Syncfusion WPF Window into the WPF application with Syncfusion WPF references. Syncfusion WPF window provides an option to customize the window title bar quickly and comes with various built-in themes to present an appealing User Interface.

---

**Information:** The Syncfusion WPF item templates are available from v19.1.0.54.

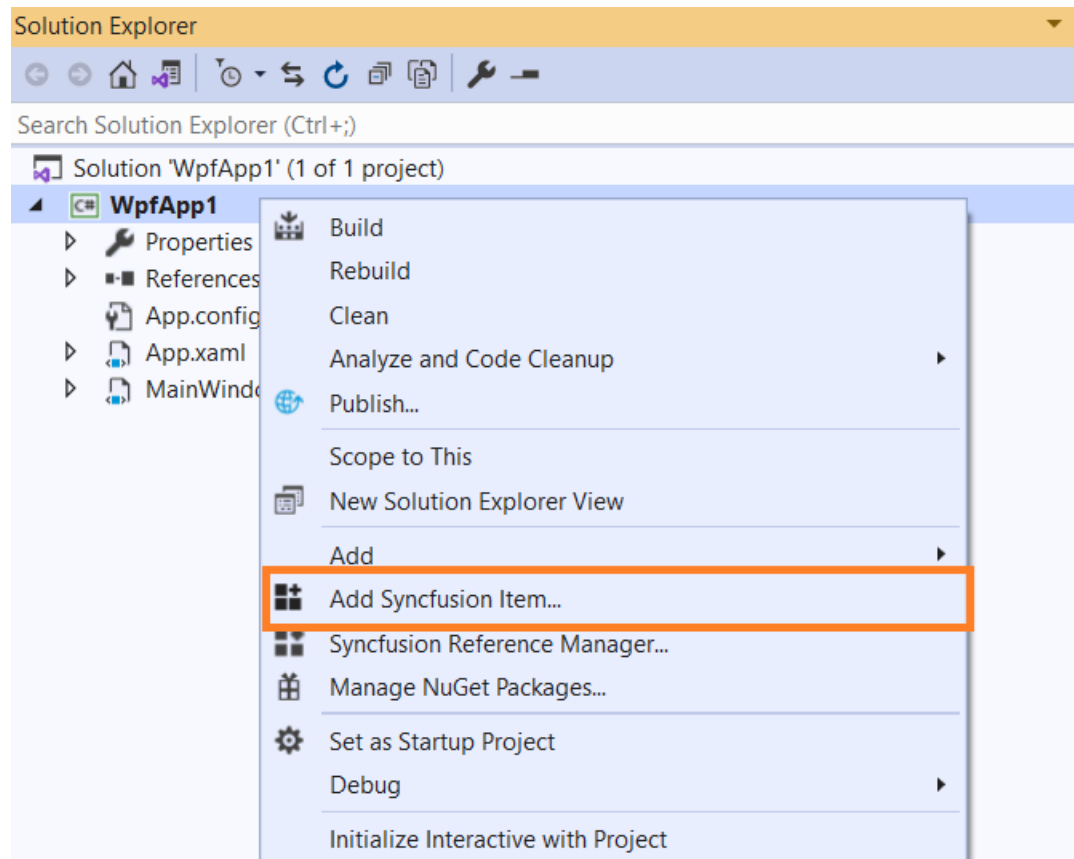
---

The following steps help you to add the Syncfusion WPF window in the Visual Studio WPF application.

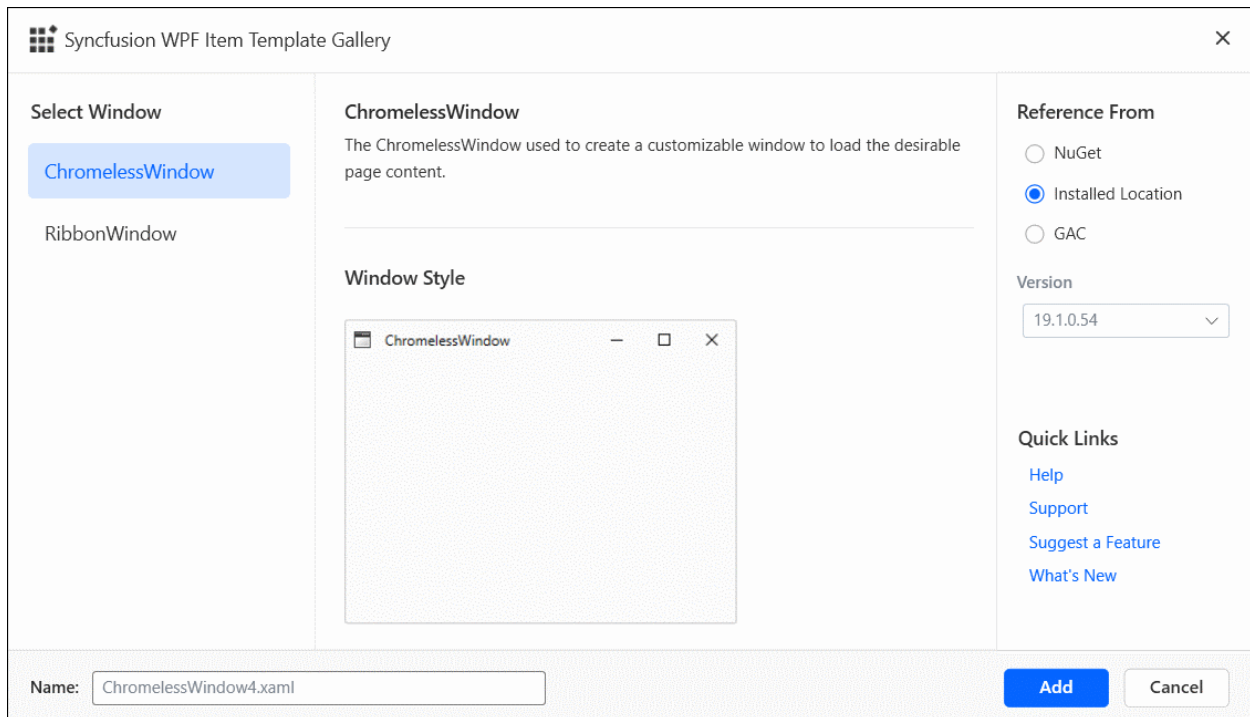
Before using the Syncfusion WPF Item Template, check whether the **WPF Extensions - Syncfusion** installed or not in Visual Studio Extension Manager by clicking on the Tools > Extensions and Updates > Installed for Visual Studio 2017 or lower and for Visual Studio 2019 by clicking on the Extensions > Manage Extensions > Installed. If it is not installed, then install the [Syncfusion WPF Extensions](#).

---

1. Open a new or existing WPF application.
2. Right-click on the WPF application from the Solution Explorer. Select the Add Syncfusion Item... option.



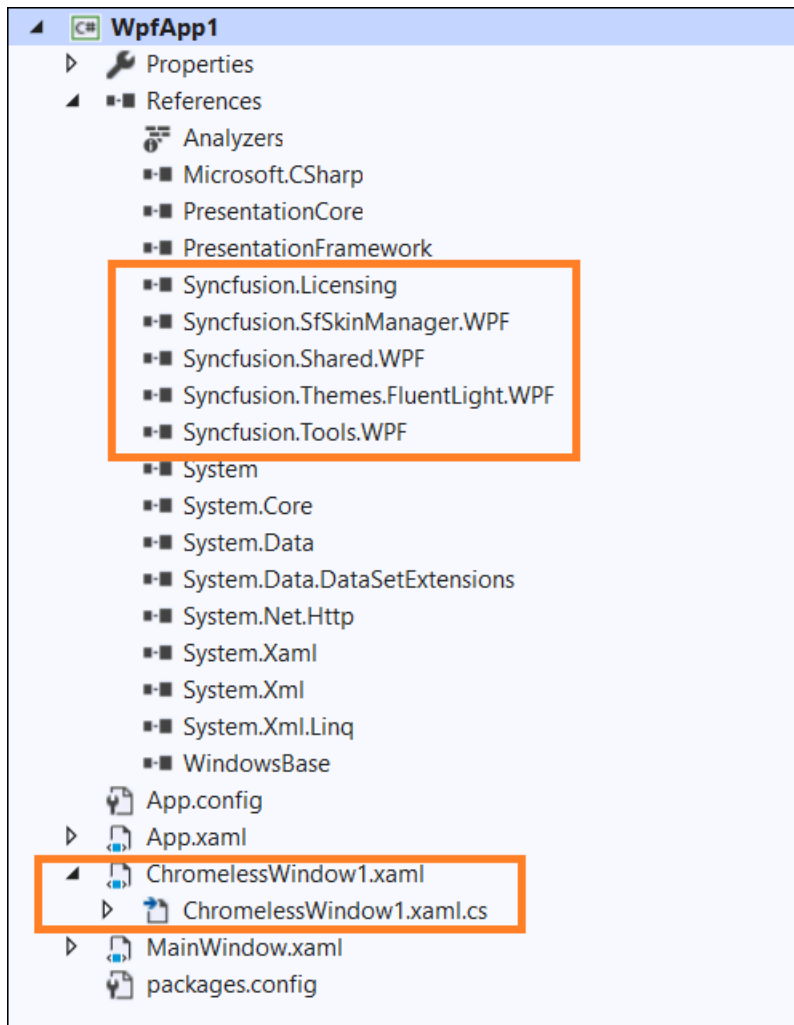
3. The Syncfusion WPF Item Template Gallery wizard will open.



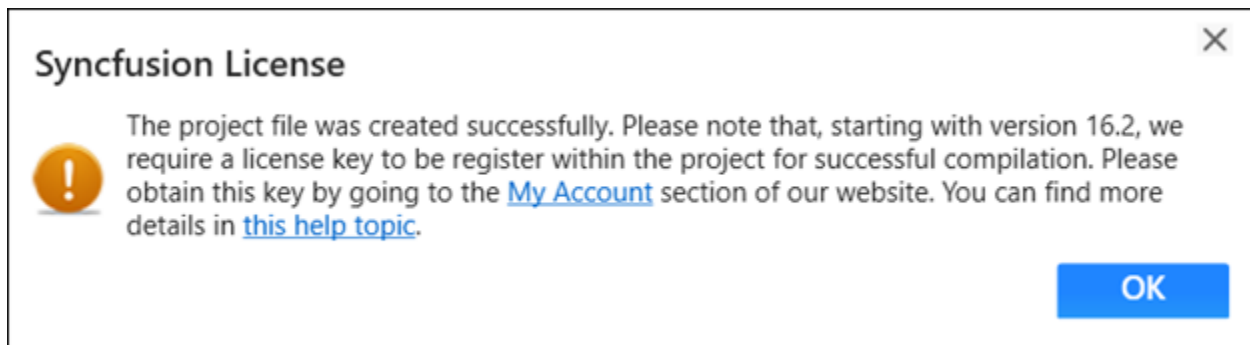
4. Syncfusion provided the Chromeless Window and Ribbon Window. Select the required window in the Select Window section. 5. Choose the assembly reference option for wherefrom the required Syncfusion assemblies are added to the project.

The Installed location and GAC option will enable if Syncfusion Essential WPF build are installed. You can use the NuGet option without installing the Syncfusion Essential WPF setup. The GAC option will not be listed if adding the Syncfusion WPF window in the .NET Core application. The WPF installed versions are listed in the Version dropdown.

6. Provide the name for the selected window. 7. Click Add button to add the selected window into the WPF application along with required Syncfusion assemblies.



8. The Syncfusion licensing registration required message box will be shown if you installed the trial setup or NuGet packages since Syncfusion introduced the licensing system from the 2018 Volume 2 (v16.2.0.41) Essential Studio release. Navigate to the help topic, which is shown in the licensing message box to generate and register the Syncfusion license key to your project. Refer to this blog post for understanding the licensing changes introduced in Essential Studio.



*Add window using the Visual Studio Add New Item dialog*

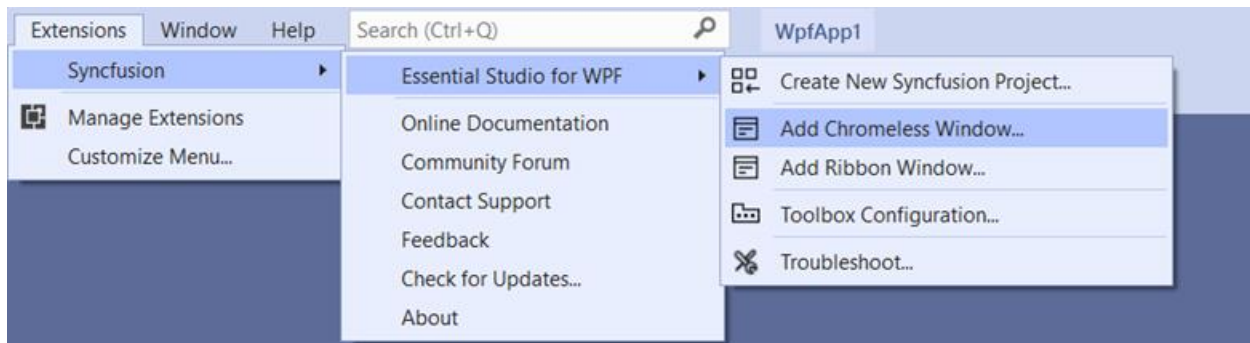
The Syncfusion Item Template can also be added from the Visual Studio Add New Item dialog.

1. To add a Syncfusion WPF window, follow either one of the following options:



**Option 1:**

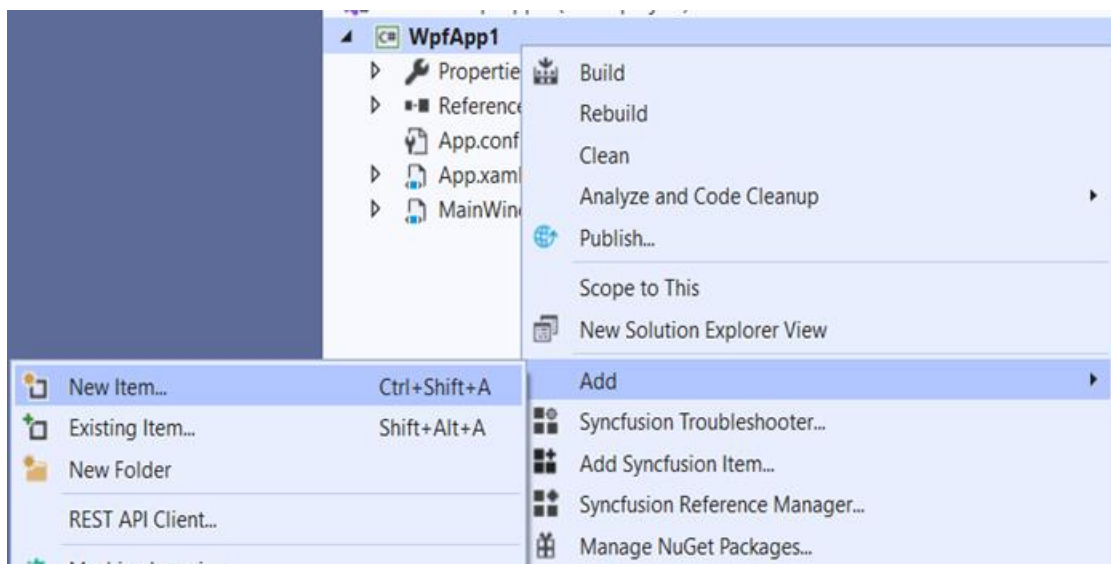
Click Extensions > Syncfusion Menu and choose Essential Studio for WPF > Add ChromelessWindow... or any other window in Visual Studio.



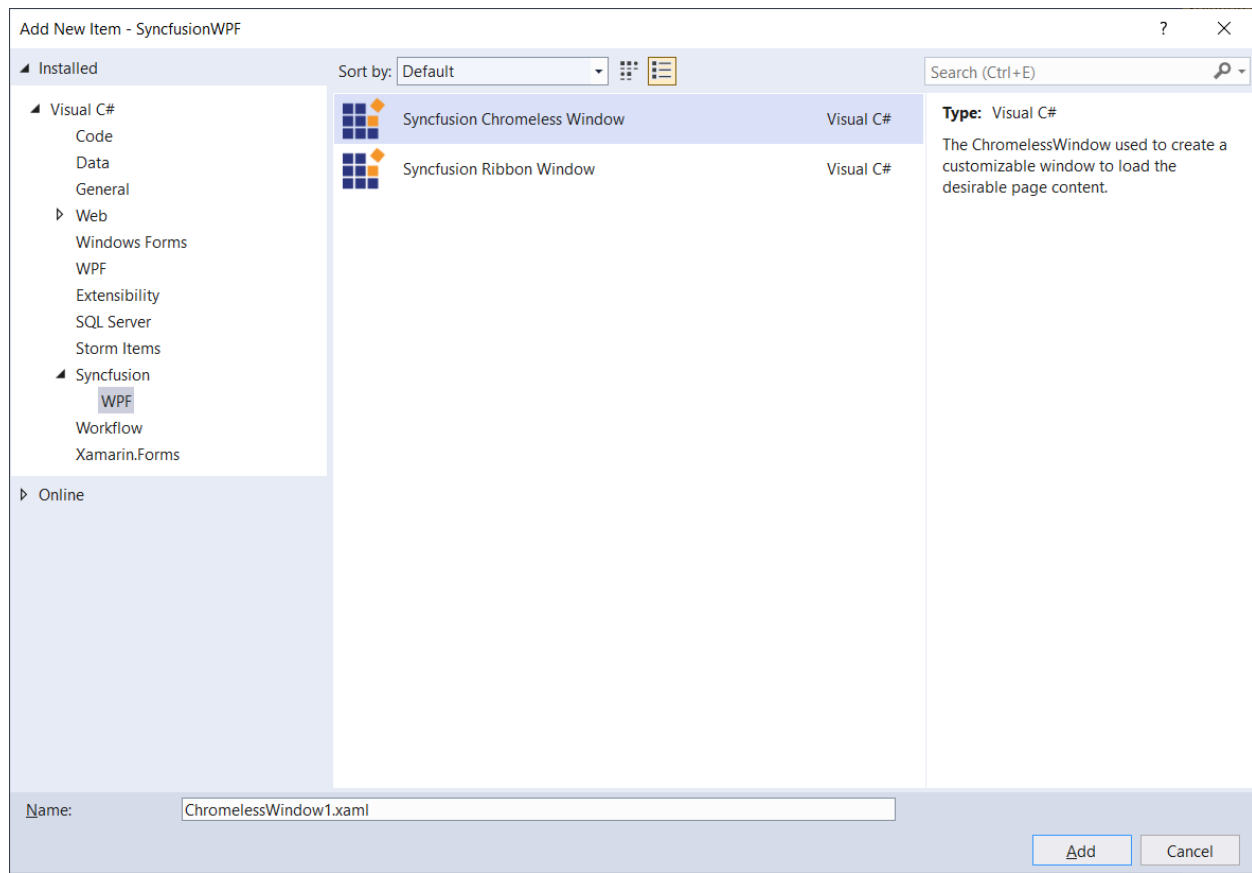
In Visual Studio 2017 or lower, the Syncfusion menu is available in the Visual Studio menu.

**Option 2:**

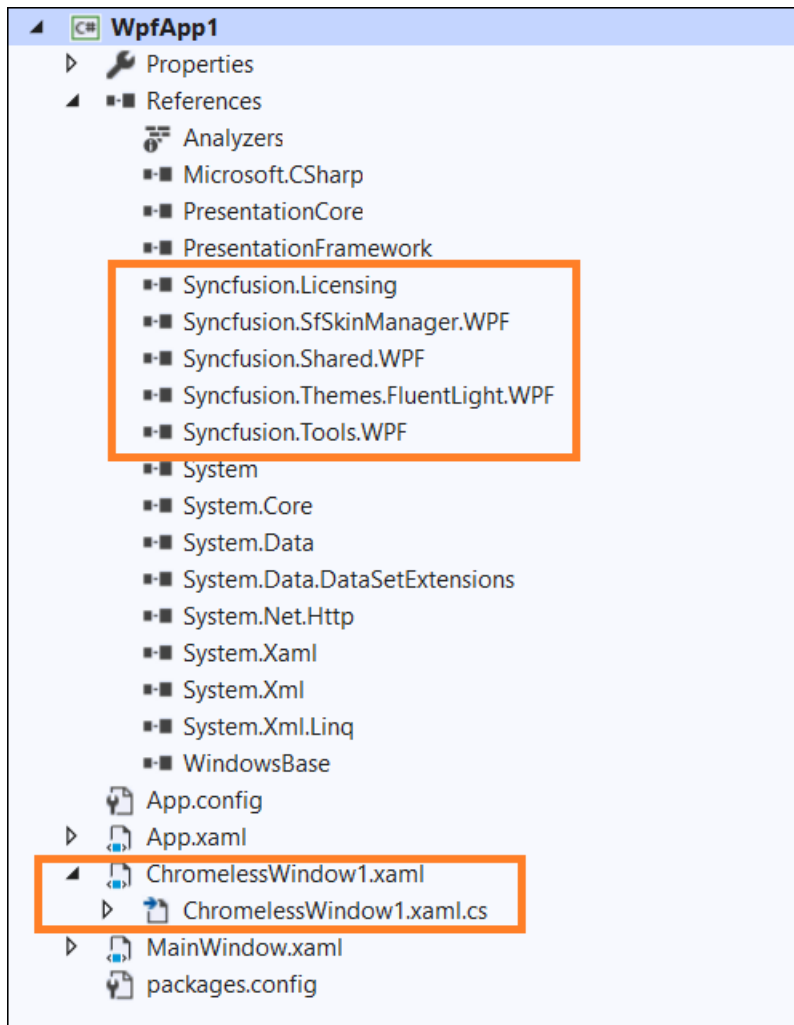
Right-click the WPF application in Solution Explorer, select Add > New Item, and then navigate to Visual C# Items or VB Items. Refer to the following screenshot for more information.



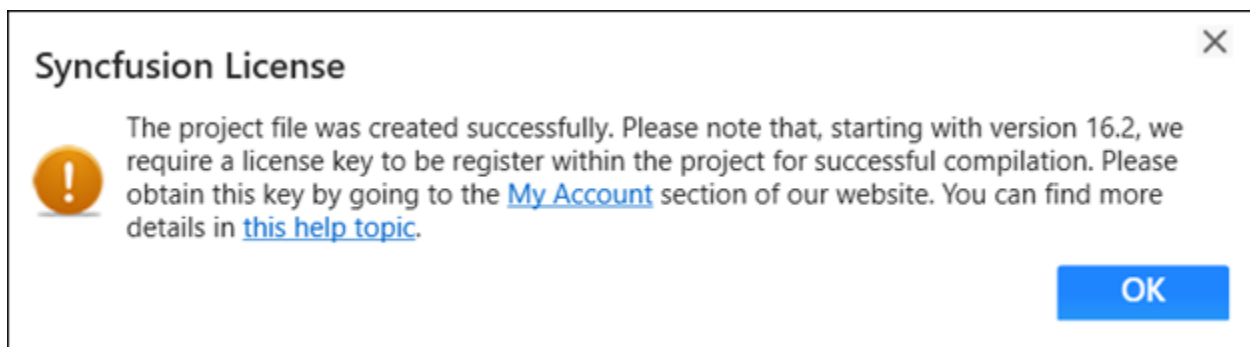
The Syncfusion WPF Item Templates are available under the Syncfusion > WPF tab. It is available for both C# Items and VB Items.



2. Click Add button and now the selected window is added to the project along with the Syncfusion NuGet reference.



3. The Syncfusion licensing registration required message box will be shown if you installed the trial setup or NuGet packages since Syncfusion introduced the licensing system from the 2018 Volume 2 (v16.2.0.41) Essential Studio release. Navigate to the help topic, which is shown in the licensing message box to generate and register the Syncfusion license key to your project. Refer to this blog post for understanding the licensing changes introduced in Essential Studio.



### Troubleshoot the project

Troubleshoot the project with the Syncfusion configuration and apply the fix like, wrong .NET Framework version of added Syncfusion assembly to the project or missing any Syncfusion dependent assembly of a referred assembly. The Syncfusion Troubleshooter can do the following:

- Report the Configuration issues.
- Apply the solution

### Report the Configuration issues

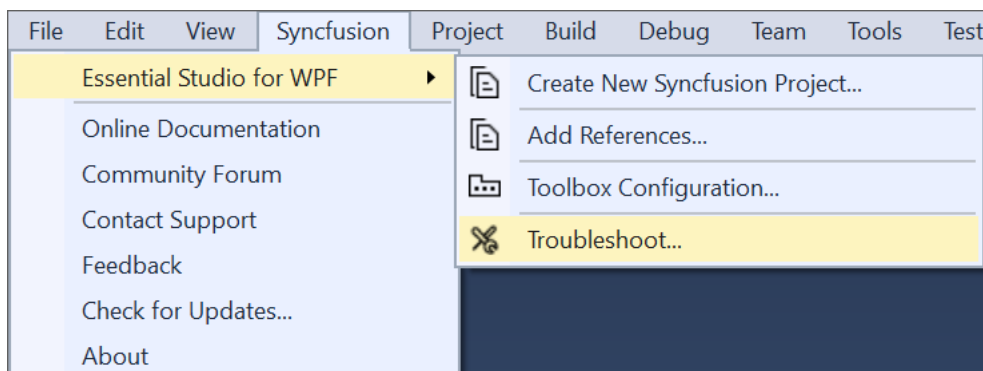
The following steps help you to utilize the Syncfusion Troubleshooter by Visual Studio.

Before use the Syncfusion Troubleshooter for WPF, check whether the **WPF Extensions - Syncfusion** installed or not in Visual Studio Extension Manager by clicking on the Tools -> Extensions and Updates -> Installed for Visual Studio 2017 or lower and for Visual Studio 2019 by clicking on the Extensions -> Manage Extensions -> Installed.

1. To open Syncfusion Troubleshooter Wizard, follow either one of the options below:

#### Option 1

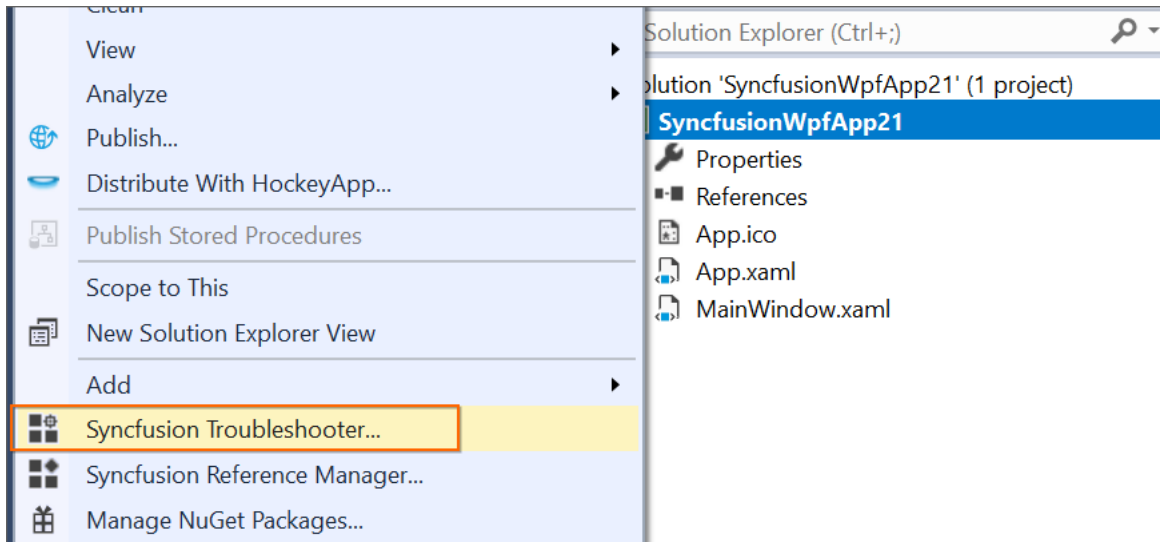
Open an existing Syncfusion WPF Application, Click **Syncfusion Menu** and choose **Essential Studio for WPF > Troubleshoot...** in Visual Studio.



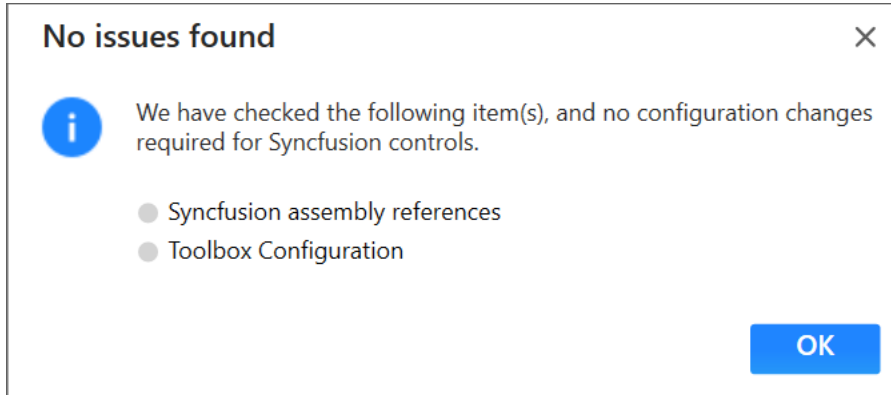
**Note:** In Visual Studio 2019, Syncfusion menu is available under Extensions in Visual Studio menu.

#### Option 2

Right-click the Project file in Solution Explorer, then select the command Syncfusion Troubleshooter...



2. Now, analyze the project and it will report the project configuration issues of Syncfusion controls in the Troubleshooter dialog if any issues found. If the project does not have any configuration issues, the dialog box will show there is no configuration changes required in following areas:
  - Syncfusion assembly references.
  - Syncfusion NuGet Packages.
  - Syncfusion Toolbox Configuration.



**Information:** The Syncfusion Troubleshooter command will be visible only for Syncfusion projects that means the project should contain Syncfusion assemblies or Syncfusion NuGet packages referred.

The Syncfusion Troubleshooter handles the following project configuration issues:

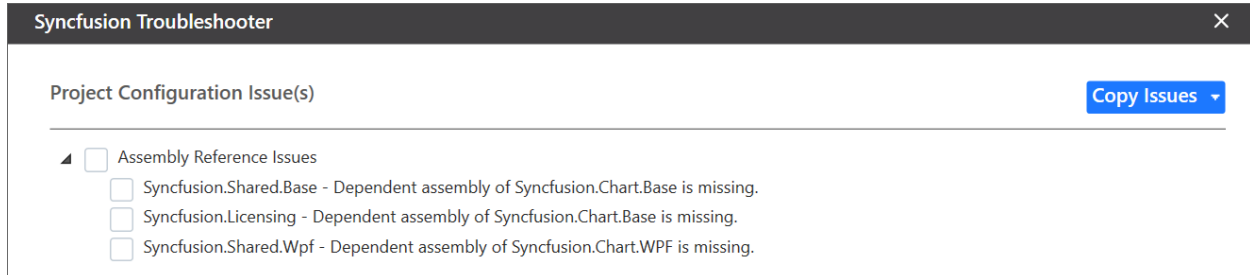
1. Assembly Reference Issues.
2. NuGet related Issues.
3. Toolbox Configuration Issues.

#### Assembly Reference Issues

The Syncfusion Troubleshooter deals with the following assembly reference issues in Syncfusion Projects.

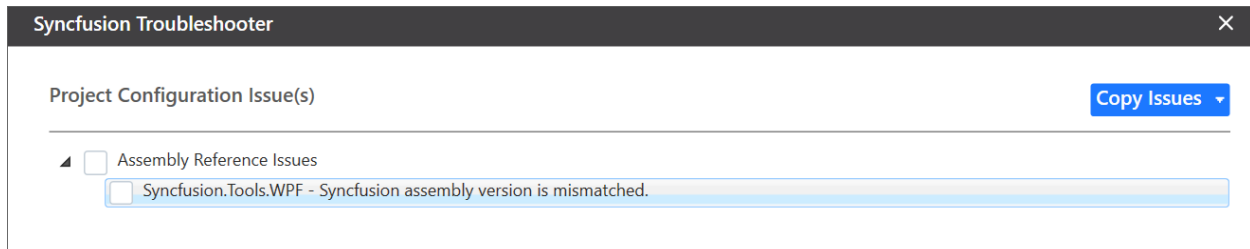
1. Dependent assemblies are missing for referred assemblies from project.

**For Instance:** : If “Syncfusion.Chart.WPF” assembly referred in project and “Syncfusion.Shared.WPF” (dependent of Syncfusion.Chart.Base) not referred in project, the Syncfusion Troubleshooter will show dependent assembly missing.



2. Syncfusion assembly version mismatched. Compare to all Syncfusion assembly's versions in the same project. If found any Syncfusion assembly version inconsistency, the Syncfusion Troubleshooter will show Syncfusion assemblies version mismatched.

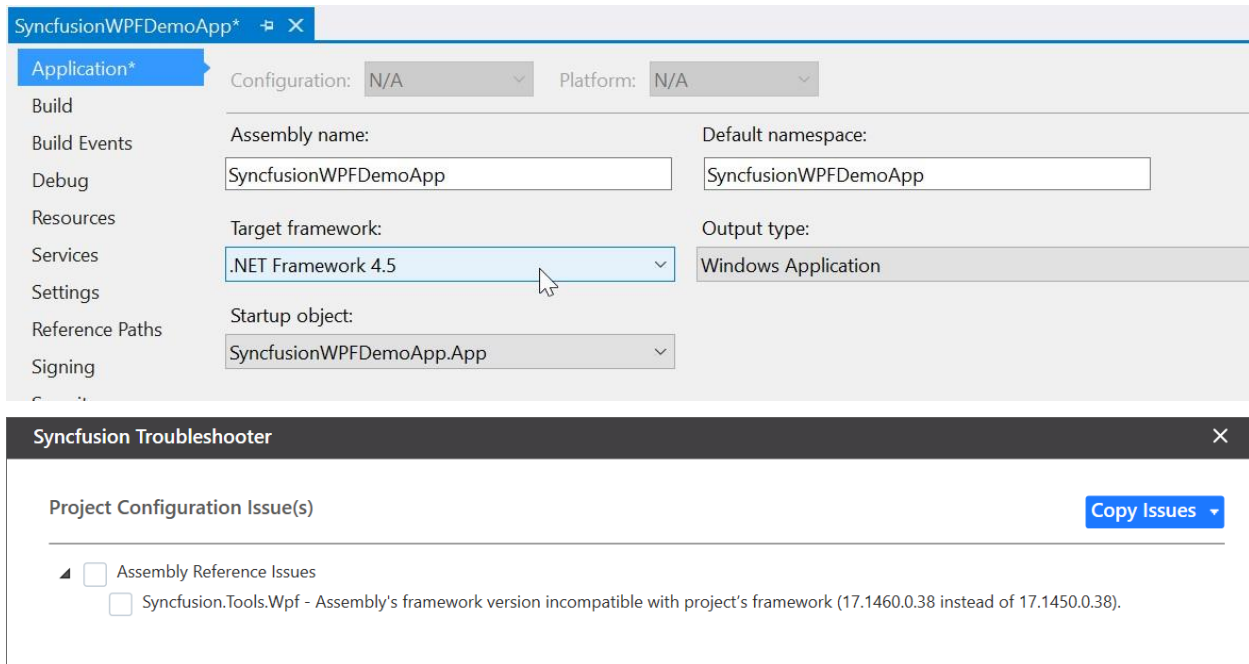
**For Instance:** If “Syncfusion.Tools.WPF” assembly (v17.1450.0.32) referred in project, but other Syncfusion assemblies referred assembly version is v17.1450.0.38. The Syncfusion Troubleshooter will show Syncfusion assembly version mismatched.



3. Framework version mismatching (Syncfusion Assemblies) with project's .NET Framework version. Find the supported .NET Framework details for Syncfusion assemblies in the following link,

<https://help.syncfusion.com/common/essential-studio/assembly-information#supported-framework-version-for-essential-studio-assemblies>

**For Instance:** The .NET Framework of the application is v4.5 and “Syncfusion.Tools.WPF” assembly (v17.1460.0.38 & .NET Framework version 4.6) referred in same application. The Syncfusion Troubleshooter will show Syncfusion assembly .NET Framework version is incompatible with project's .NET Framework version.



### NuGet Issues

The Syncfusion Troubleshooter deals with the following NuGet package related issues in Syncfusion projects.

1. Multiple versions of Syncfusion NuGet Packages are installed. If Syncfusion NuGet Package version is differ from other Syncfusion NuGet Package version, the Syncfusion Troubleshooter will show Syncfusion NuGet package version is mismatched.

**For Instance:** Syncfusion WPF platform packages installed multiple version (v16.4.0.54 & v17.1.0.38), Syncfusion Troubleshooter will be shown Syncfusion package version mismatched.



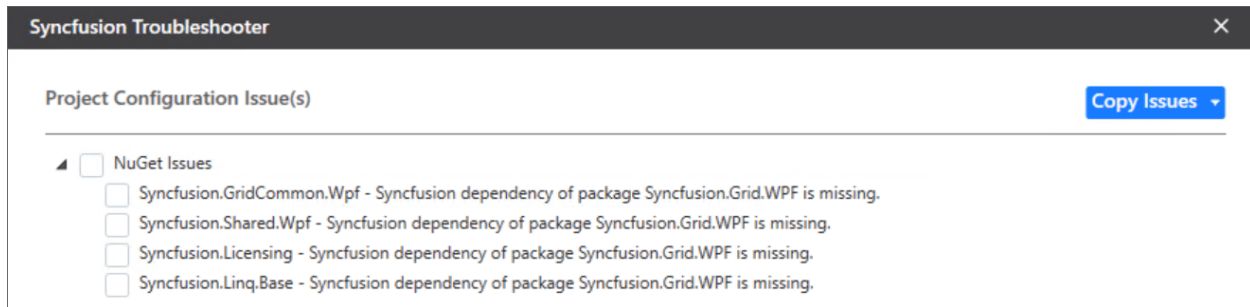
2. Installed Syncfusion NuGet package's Framework version is differing from the project's .NET Framework version.

**For Instance:** If "Syncfusion.SfBulletGraph.WPF40" NuGet package version(v15.4.0.17 with 4.0 Framework) installed in project, But the project .NET Framework version is 4.5. So, the Syncfusion Troubleshooter will show Syncfusion package Framework version is mismatched.



3. Dependent NuGet package of the installed Syncfusion NuGet packages is missing.

**For Instance:** If install Syncfusion.Chart.WPF NuGet package alone in project, Syncfusion Troubleshooter will show the Syncfusion.Chart.Base and other dependent NuGet package missing.



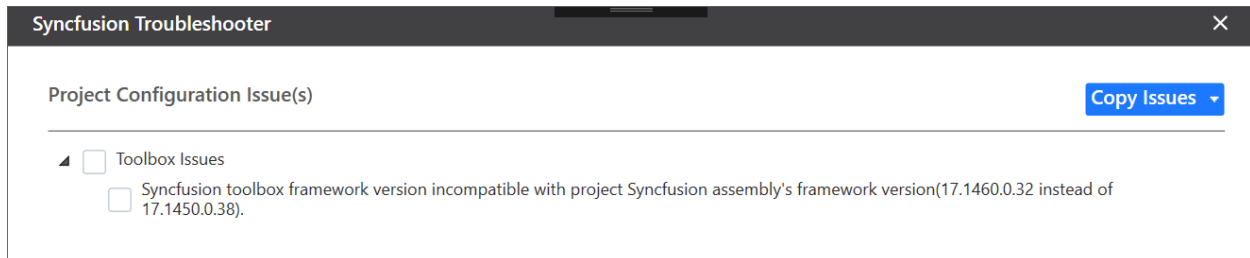
**Information:** Internet connection is required to restore the missing dependent packages. If internet is not available, the dependent packages will not be restored.

#### Toolbox Configuration Issues

The Syncfusion Troubleshooter deals with the following Toolbox Configuration related issues in Syncfusion projects.

1. If the project .NET Framework version is not installed/configured Syncfusion Toolbox, the Syncfusion Troubleshooter will show Syncfusion Toolbox .NET Framework version is mismatched.

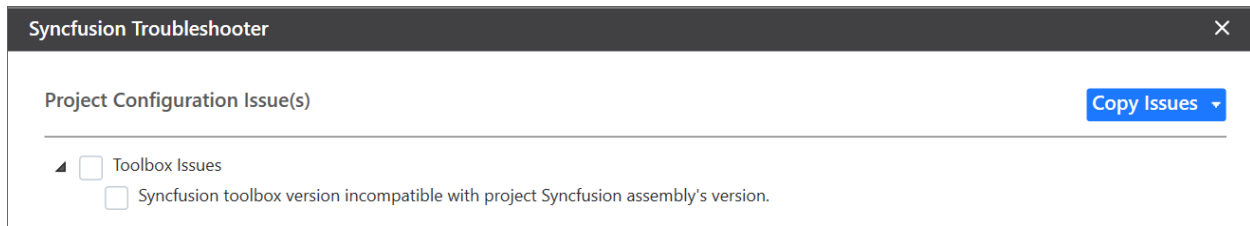
**For Instance:** The project .NET Framework version is 4.5 and Syncfusion Toolbox is not configured 4.6 framework assemblies only in corresponding Visual Studio, the Syncfusion Troubleshooter will show Syncfusion Toolbox framework version mismatched.



2. If Syncfusion Toolbox configured version is differed from latest Syncfusion assembly reference version or NuGet package version in same project, the Syncfusion Troubleshooter will show Syncfusion Toolbox version is mismatched.

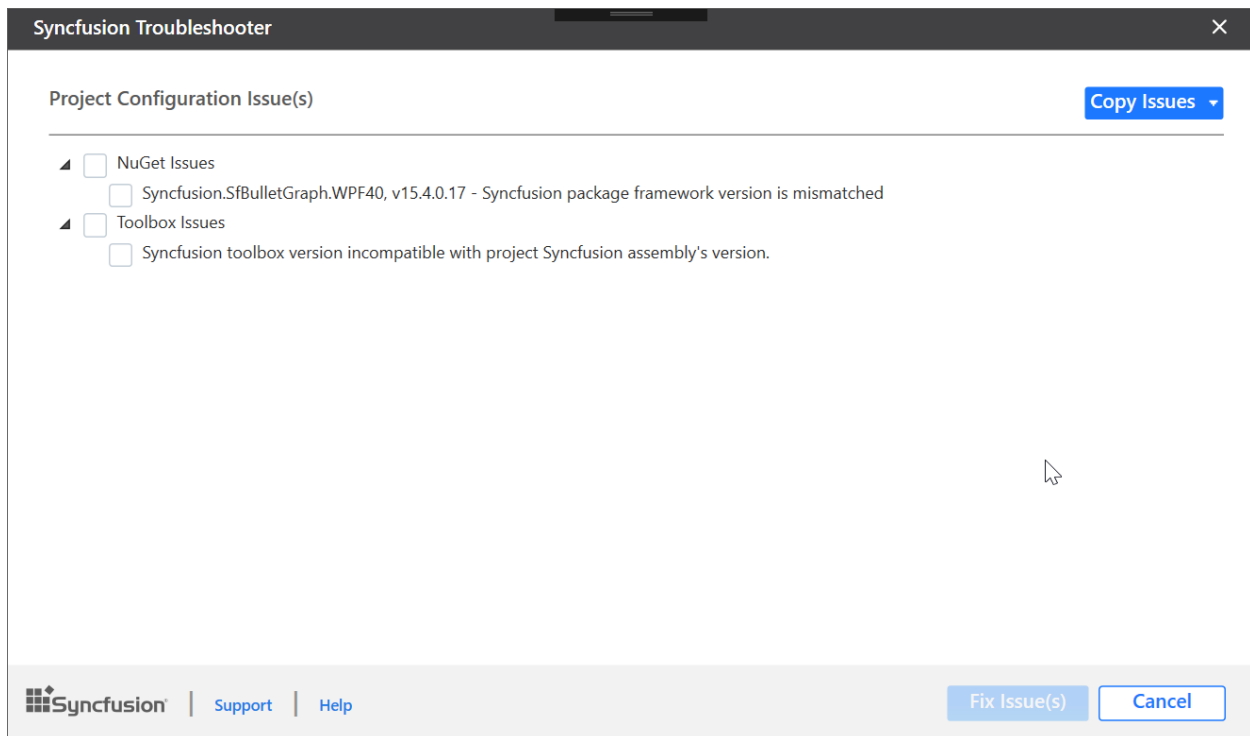


**For Instance:** If latest Syncfusion assembly reference version is v17.1.0.38 but Toolbox assemblies configured v17.1.0.32, the Syncfusion Troubleshooter will show Syncfusion Toolbox version mismatched.

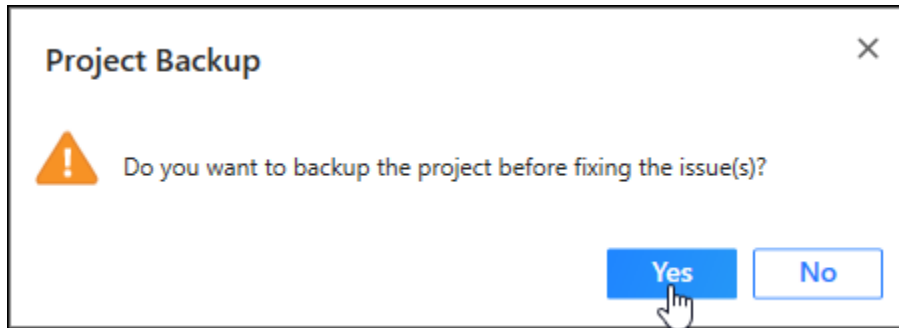


### *Apply the solution*

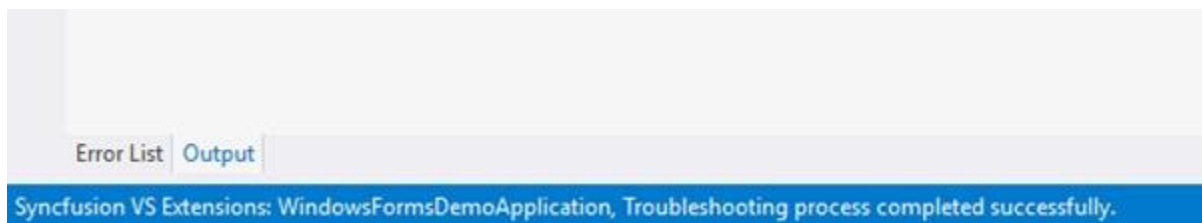
1. After loading the Syncfusion Troubleshooter dialog, check the corresponding check box of the issue to be resolved. Then click the “Fix Issue(s)” button.



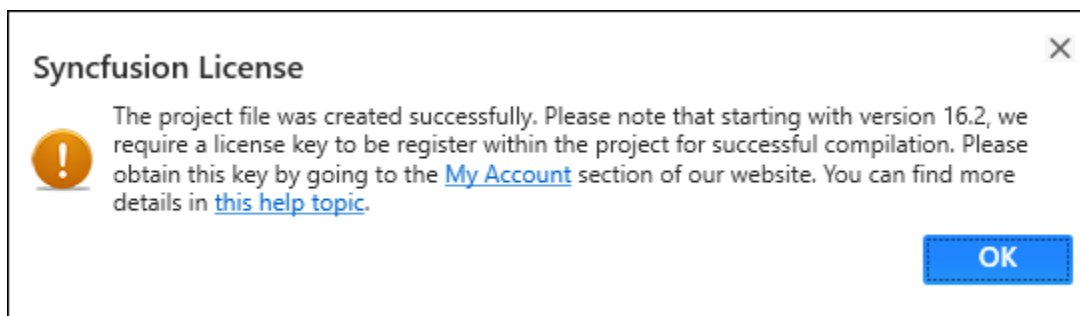
2. A dialog appears, which will ask to take a backup of the project before performing the troubleshooting process. If you need to backup the project before troubleshooting, click “Yes” button.



3. Wait for a while, the Syncfusion Troubleshooter is resolving the selected issues. After the troubleshooting process completed, there will be a status message in the Visual Studio status bar as "Troubleshooting process completed successfully".



4. Then, Syncfusion licensing registration required message box will be shown, if you installed the trial setup or NuGet packages since Syncfusion introduced the licensing system from 2018 Volume 2 (v16.2.0.41) Essential Studio release. Navigate to the [help topic](#), which is shown in the licensing message box to generate and register the Syncfusion license key to your project. Refer to this [blog](#) post for understanding the licensing changes introduced in Essential Studio.



#### Add Reference for WPF

Syncfusion Reference Manager is the Visual Studio Add-In for WPF platform. It adds the Syncfusion assembly reference to the project, either from the GAC location or from where Essential Studio is installed. It can also migrate the projects that contain the old versions of the Syncfusion assembly reference to newer or specific versions of the Syncfusion assembly reference. It supports Microsoft Visual Studio 2013 or higher. This Visual Studio extension is included from Essential Studio 2013 Volume 3 release.

**Note:** This Reference Manager can be applied to a project for Syncfusion assembly versions 10.4.0.71 and later.

Follow the given steps to add the Syncfusion references in Visual Studio:

---

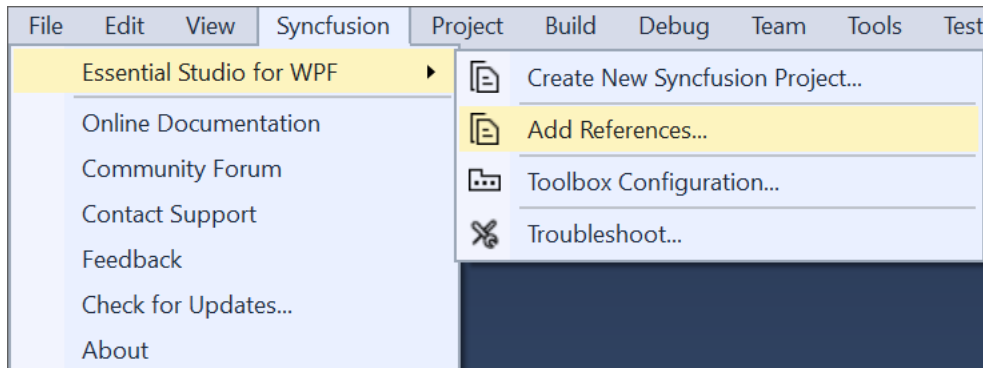
Before use the Syncfusion WPF Reference Manager, check whether the **WPF Extensions - Syncfusion** installed or not in Visual Studio Extension Manager by clicking on the Tools -> Extensions and Updates -> Installed for Visual Studio 2017 or lower and for Visual Studio 2019 by clicking on the Extensions -> Manage Extensions -> Installed.

---

1. Open a new or existing **WPF** application.
2. To open Syncfusion Reference Manager Wizard, follow either one of the options below:

**Option 1:**

Click **Syncfusion Menu** and choose **Essential Studio for WPF > Add References...** or any other Form in **Visual Studio**.



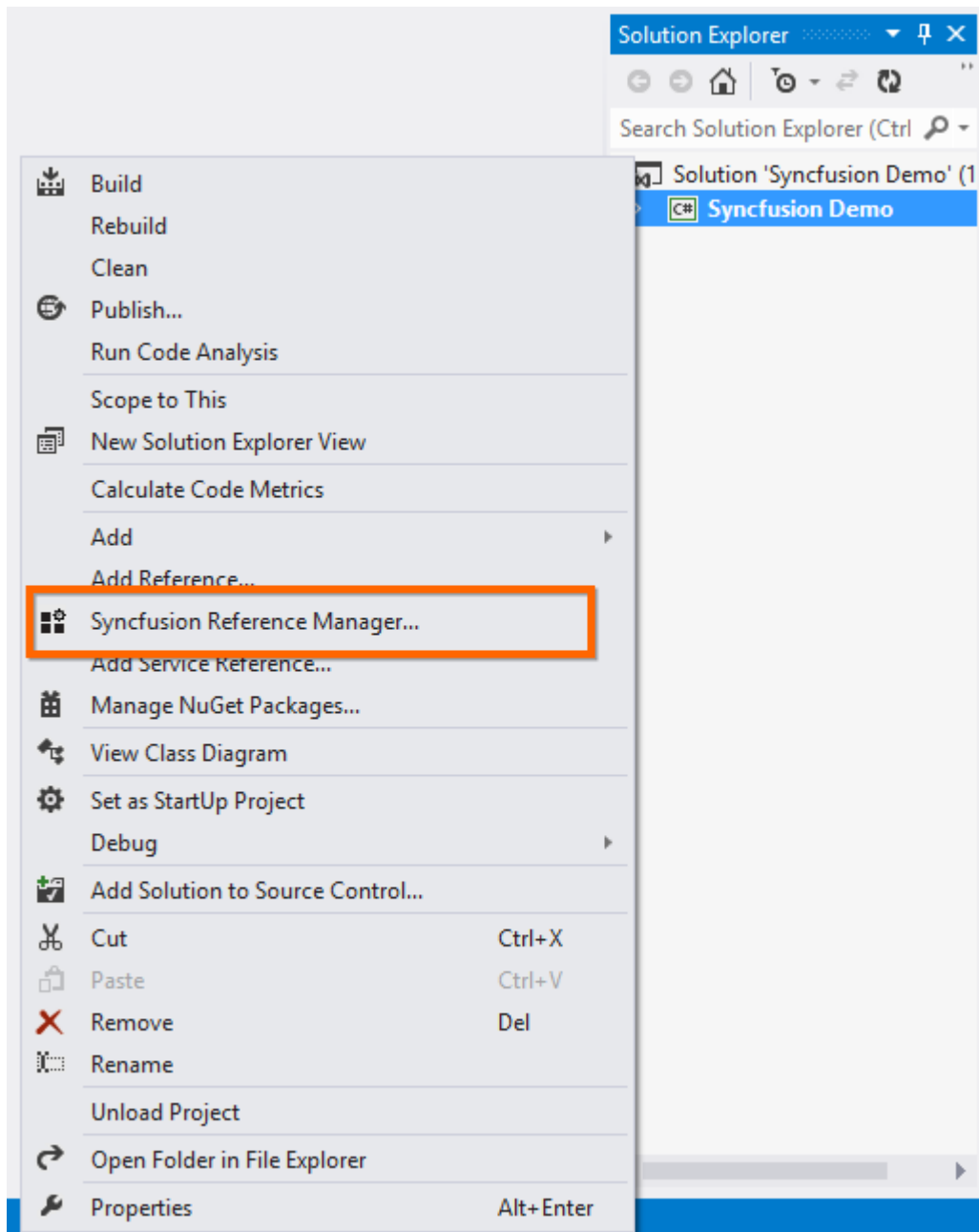
---

**Note:** In Visual Studio 2019, Syncfusion menu is available under Extensions in Visual Studio menu.

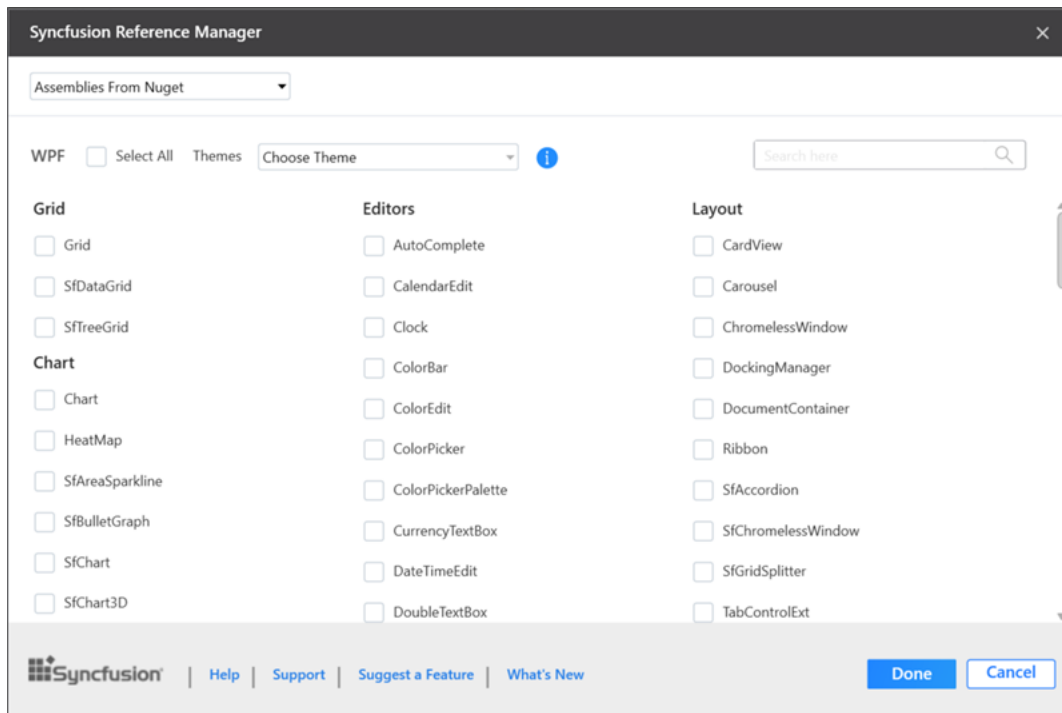
---

**Option 2:**

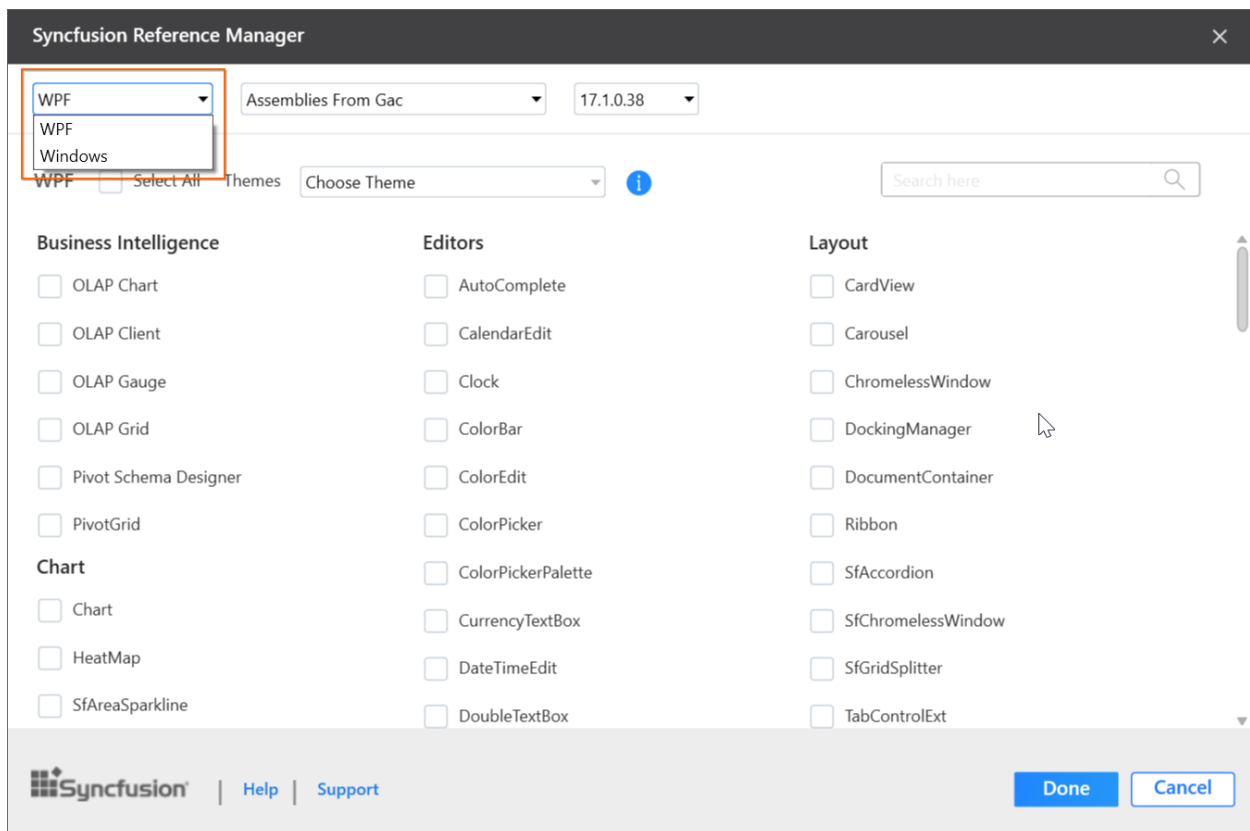
Right-click the selected project file from Solution Explorer, then select **Syncfusion Reference Manager...** from **Context Menu**. The following screenshot shows this option in Visual Studio.



3. The Syncfusion Reference Manager Wizard that contains the list of Syncfusion WPF controls that are loaded.



**Platform Selection:** If launched the Syncfusion Reference Manager from Console/Class Library project, Platform selection option will be appeared as option in Syncfusion Reference Manager. Choose the required platform.

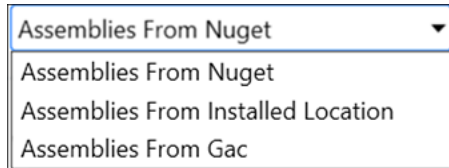


---

**Note:** Platform selection option will be appeared only if Essential Studio for Enterprise Edition with the platforms WPF and Windows Forms has been installed or both Essential Studio for WPF and WinForms has been installed.

---

**Assembly From:** Choose the assembly location from, where the assembly is added to the project.

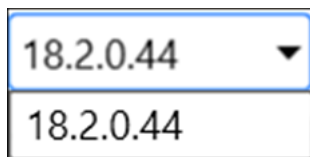


---

**Note:** The GAC option will not be available when you selected WPF (.NET Core 3.1 and .Net 5.0) application in Visual Studio 2019.

---

**Version:** Choose the build version to add the corresponding version assemblies to the project.



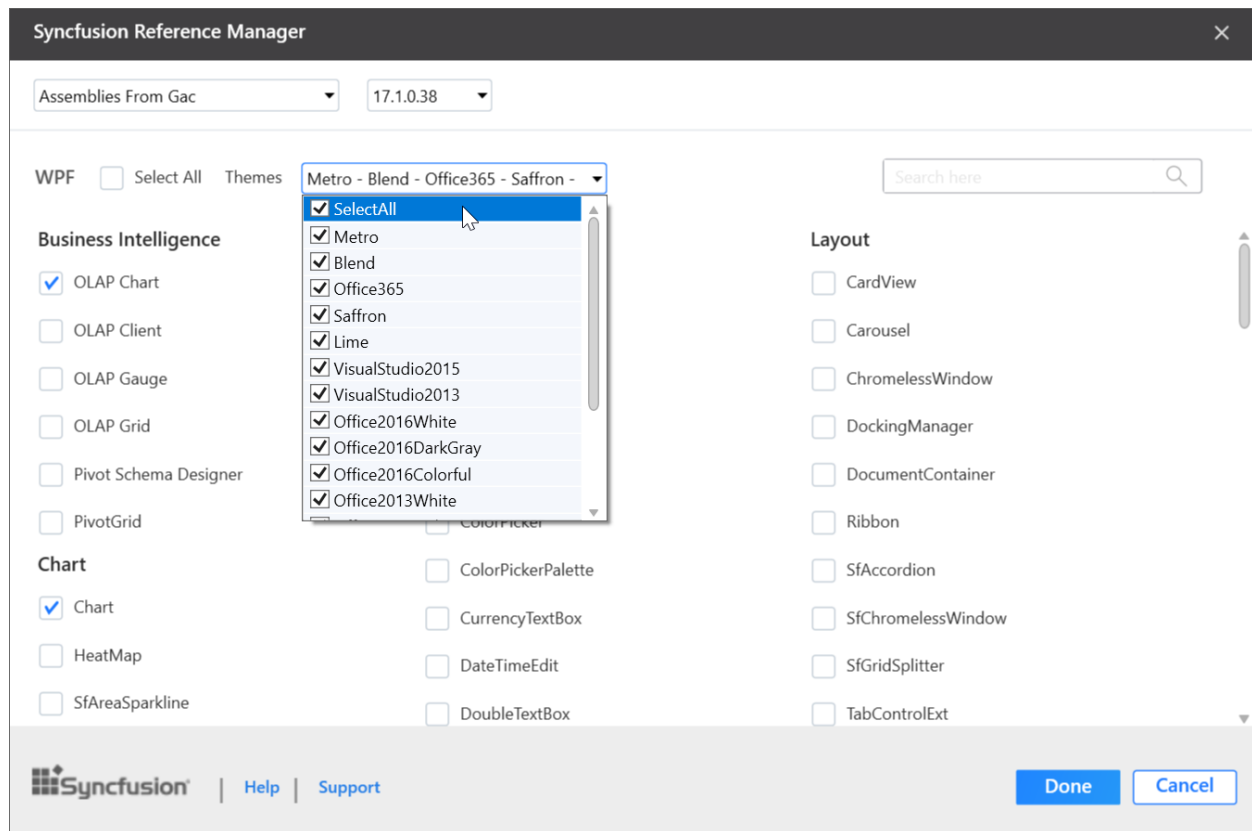
---

**Note:** WPF (.NET Core 3.1 and .Net 5.0) application in Visual Studio 2019 is supported from 18.2.0.44 version and Version combobox is not visible for NuGet option.

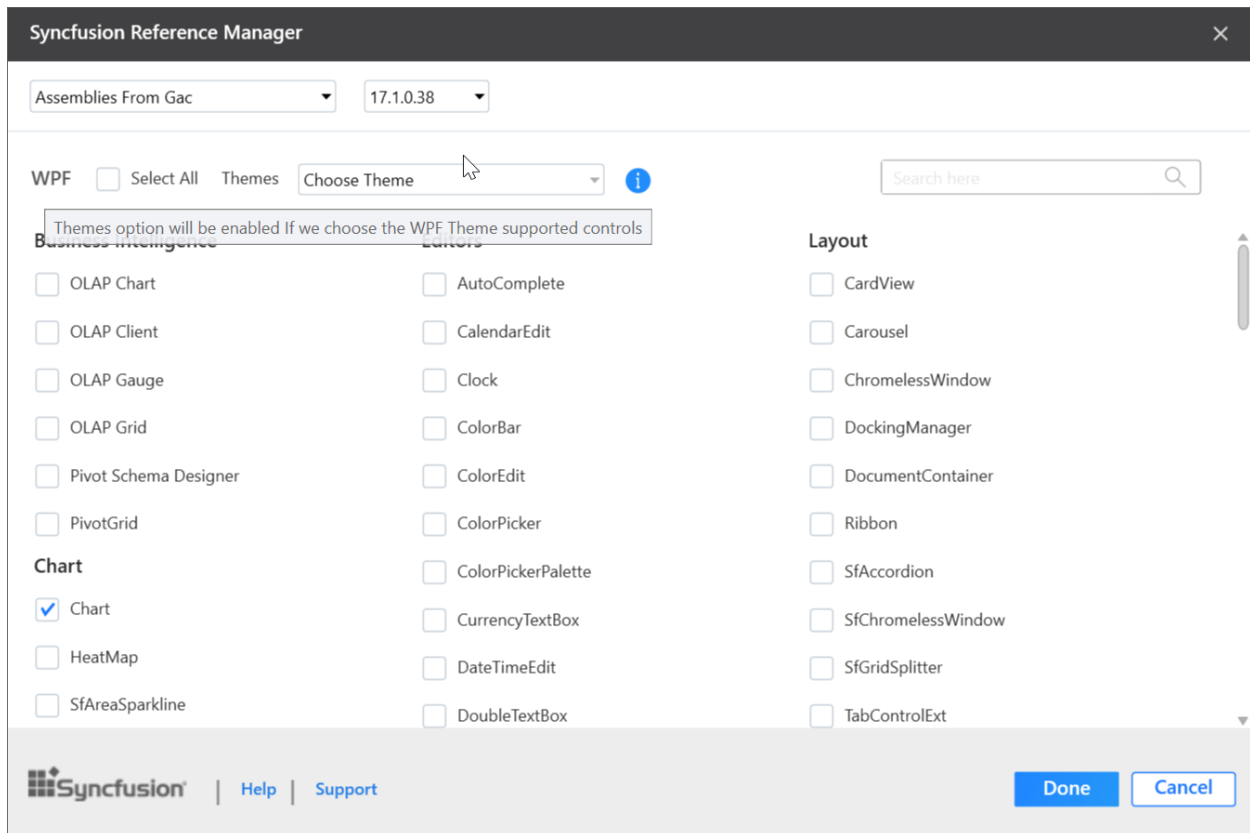
---

**Themes Option:** Choose the required themes based on your need. Refer the below link to know more about built in themes and its available assemblies.

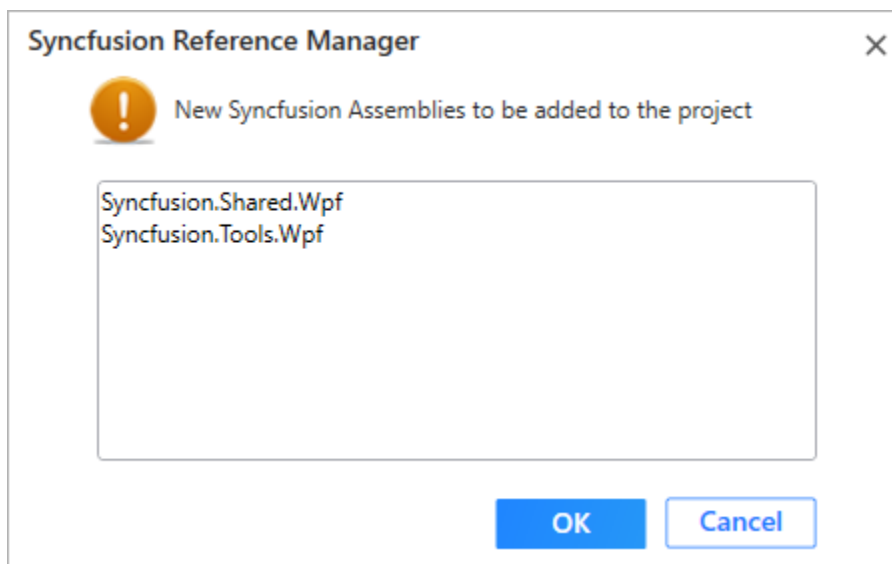
<https://help.syncfusion.com/wpf/themes/>



**Note:** Themes option will be enabled only if selected SfSkinManager supported controls.



4. Choose the required controls that you want to include in the project. Then, click **Done** to add the required assemblies for the selected controls into the project. The following screenshot shows the list of required assemblies for the selected controls to be added.

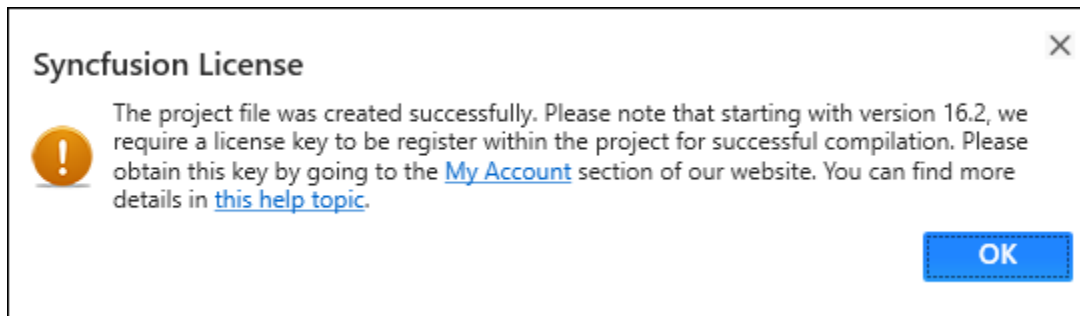


5. Click **OK**. The listed Syncfusion assemblies are added to project. Then it notifies "Syncfusion assemblies have been added successfully" in Visual Studio status bar.



Syncfusion assemblies have been added successfully.

- Then, Syncfusion licensing registration required message box will be shown, if you installed the trial setup or NuGet packages since Syncfusion introduced the licensing system from 2018 Volume 2 (v16.2.0.41) Essential Studio release. Navigate to the [help topic](#), which is shown in the licensing message box to generate and register the Syncfusion license key to your project. Refer to this [blog](#) post for understanding the licensing changes introduced in Essential Studio.



**Note:** Syncfusion provides Reference Manager support for specific .NET Framework, which is shipped (assemblies) in Syncfusion Essential Studio setup. So, if you try to add Syncfusion assemblies in the project and project framework is not supported with selected Syncfusion version assemblies, the dialog appears along with “**Current build v{version} is not supported this framework v{Framework Version}**” message.

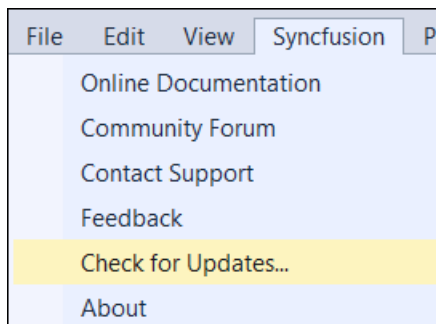
#### Check for Updates in Syncfusion Essential WPF

Syncfusion provides the Extensions to update most recent version of the Essential Studio release. So that, you always get the latest features, fixes, and improvements by installing the latest version.

**Information:** The Syncfusion Check for updates is available from v17.1.0.32.

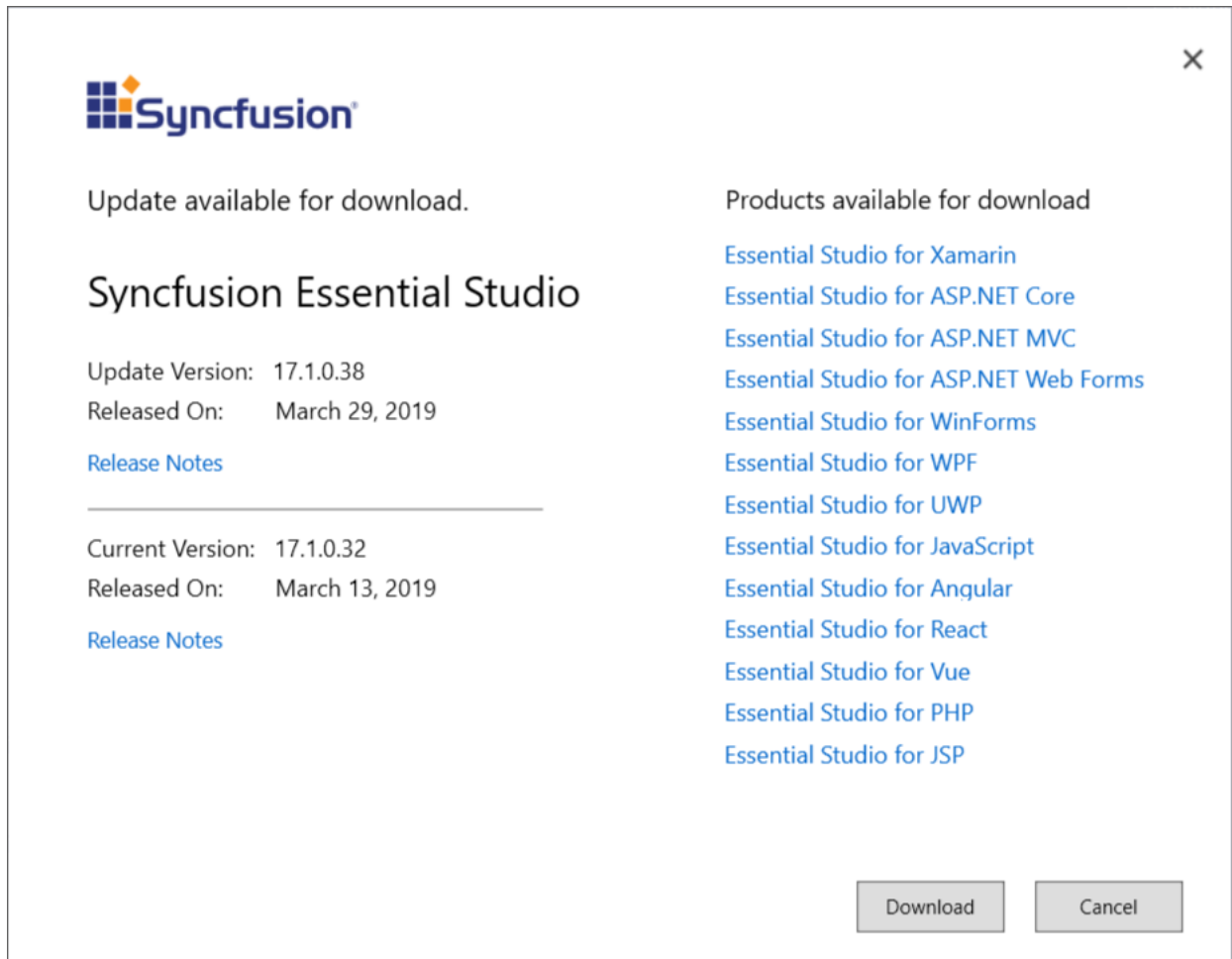
You can check updates availability in Visual Studio, and then install the update version if required.

- Choose **Syncfusion -> Check for Updates...** in the Visual Studio menu



**Note:** In Visual Studio 2019, Syncfusion menu is available under Extensions in Visual Studio menu.

- When there is an update, **Update** dialog box opens.



3. You can download the Syncfusion Essential Studio from the Syncfusion website by selecting **Download**.

### NuGet Packages in Syncfusion Essential WPF

[NuGet](#) can be used to automatically add files and references to your Visual Studio projects. You can use the Syncfusion WPF NuGet packages without installing the Essential Studio or WPF platform installation to development with the Syncfusion WPF controls.

From v16.2.0.46 (2018 Volume 2 Service Pack 1) onwards, all the Syncfusion components are available as NuGet packages at [nuget.org](https://nuget.org).

Starting with v16.2.0.x, if you reference Syncfusion assemblies from trial setup or from the NuGet package, you must include a license key in your projects. Refer to this [link](#) to learn about registering Syncfusion license key in your WPF application to use Syncfusion controls.

---

**Note:** Starting from v17.1.0.32 (2018 Volume 1), Syncfusion will no longer publish NuGet packages at [nuget.syncfusion.com](https://nuget.syncfusion.com).

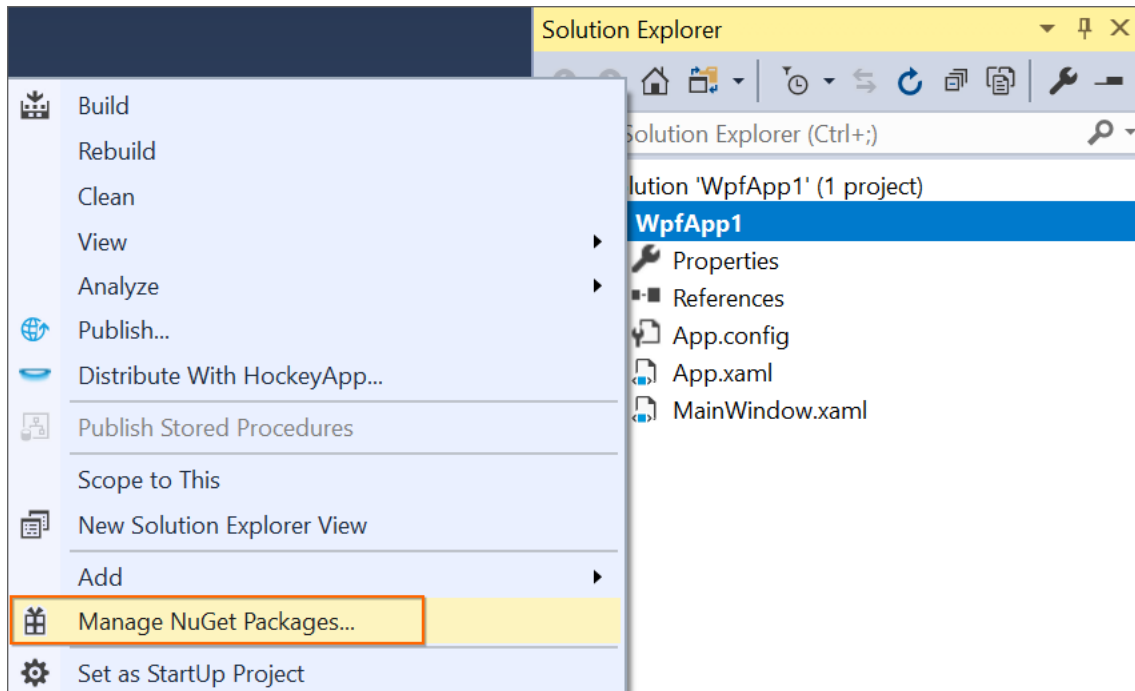
---

## Installing NuGet Packages

### *Using NuGet Package Manager*

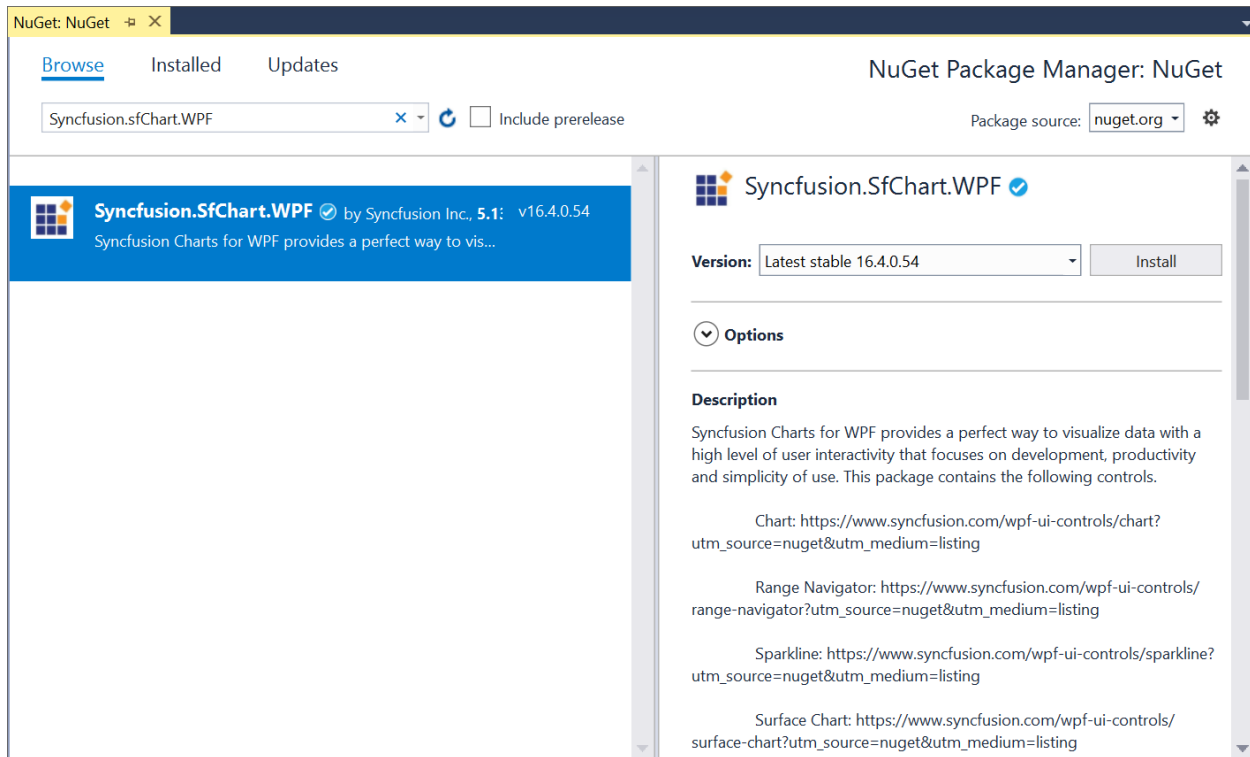
The NuGet Package Manager can be used to search and install NuGet packages in the Visual Studio solution or project:

1. Right-click the project or solution in the Solution Explorer tab, and choose **Manage NuGet Packages...**



Alternatively, click **Tools** menu, **NuGet Package Manager | Manage NuGet Packages for Solution...**

2. Select the NuGet.org from the Package source drop-down.



3. All the Syncfusion NuGet Packages are listed and available. Search and install the required packages in your application, by clicking the **Install** button.

**Note:** The Syncfusion NuGet packages are published in public [NuGet.org](https://www.nuget.org) from v16.2.0.46. To Install earlier version of 16.2.0.46 Syncfusion NuGet packages, [configure Syncfusion private feed URL](#).

#### Using Package Manager Console

To reference the Syncfusion WPF component using the Package Manager Console as NuGet packages, follow the below steps:

1. On the **Tools** menu, select **NuGet Package Manager** and then **Package Manager Console**. 2. Run the following NuGet installation commands.

#### XML

```
<Window x:Class="WpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp1"
xmlns:sync="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid Name="ROOT_Grid">
<sync:ButtonAdv Label="Hello World" Height = "35" Width = "150"/>
</Grid>
</Window>
```

b. Following code explains how to create the **ButtonAdv** control through code behind.

**C#**

```
using Syncfusion.Windows.Tools.Controls;
ButtonAdv button = new ButtonAdv();
button.Height = 35;
button.Width = 150;
button.Label = "Hello World!";
ROOT_Grid.Children.Add(button);
```



## Common

## Control Dependencies in Syncfusion's WPF Controls

This section lists needed assembly or NuGet references to use any control in the application. You can refer to the [installation and deployment section](#) to know assembly installation location and [NuGet packages](#) section to know how to add NuGet reference.

You can refer to the [syncfusion controls section](#) to learn how to add syncfusion control.

**Note:** Starting with version 16.2(2018 Vol 2), the `Syncfusion.Licensing.dll` will be added as reference for all the Syncfusion WPF controls. Please refer to this [help topic](#) for more information.

## AutoComplete

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## BusyIndicator

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## ButtonAdv

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## Calculate

Assembly references	NuGet package
Syncfusion.Calculate.Base Syncfusion.Calculate.WPF	Syncfusion.Calculate.WPF

## CalendarEdit

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## CardView

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## Carousel

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## CheckListBox

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## ChromelessWindow

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## Clock

Assembly references	NuGet package

Syncfusion.Shared.WPF	Syncfusion.Shared.WPF
-----------------------	-----------------------

## ColorPicker

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## ColorPickerPalette

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## ComboBoxAdv

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## CurrencyTextBox

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## DateTimeEdit

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## DockingManager

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## DocumentContainer

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## DoubleTextBox

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## DropDownButtonAdv

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## Edit Control

Assembly references	NuGet package
Syncfusion.Edit.WPF Syncfusion.GridCommon.WPF Syncfusion.Shared.WPF Syncfusion.Tools.WPF	Syncfusion.Edit.WPF

## Gantt

Assembly references	NuGet package
Syncfusion.Gantt.WPF Syncfusion.Grid.WPF Syncfusion.GridCommon.WPF Syncfusion.Shared.WPF Syncfusion.ProjIO.Base	Syncfusion.Gantt.WPF

## Grid Control

Assembly references	NuGet package
Syncfusion.Core Syncfusion.Grid.WPF Syncfusion.GridCommon.WPF Syncfusion.Shared.WPF	Syncfusion.Grid.WPF

## GroupBar

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF



## HierarchyNavigator

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## IntegerTextBox

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## ImageEditor

Assembly references	NuGet package
Syncfusion.SfImageEditor.WPF Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.SfImageEditor.WPF

## MaskedTextBox

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## MenuAdv

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## NotifyIcon

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## OlapChart

Assembly references	NuGet package
Syncfusion.Chart.WPF Syncfusion.Linq.Base Syncfusion.Olap.Base Syncfusion.Olap.Base Syncfusion.OlapShared.WPF	Syncfusion.OlapChart.WPF

Syncfusion.Shared.WPF Syncfusion.Tools.WPF	
---	--

#### Exporting OLAP Chart to PDF, Word and Image

For exporting OLAP chart contents to PDF, word and image, the following reference needs to be added in addition to the above assemblies.

Assembly references	NuGet package
Syncfusion.OlapChartConverter.WPF	Syncfusion.ExportOlapChart.Wpf

#### OlapClient

Assembly references	NuGet package
Syncfusion.Chart.WPF Syncfusion.Grid.WPF Syncfusion.GridCommon.WPF Syncfusion.Linq.Base Syncfusion.Olap.Base Syncfusion.OlapChart.WPF Syncfusion.OlapChartConverter.WPF Syncfusion.OlapGrid.WPF Syncfusion.OlapGridCommon.WPF Syncfusion.OlapGridConverter.WPF Syncfusion.OlapShared.WPF Syncfusion.OlapTools.WPF Syncfusion.Shared.WPF Syncfusion.Tools.WPF Syncfusion.Tools.WPF.Resources	Syncfusion.OlapClient.WPF

#### OlapGauge

Assembly references	NuGet package
Syncfusion.Gauge.WPF Syncfusion.Olap.Base Syncfusion.OlapShared.WPF Syncfusion.Shared.WPF	Syncfusion.OlapGauge.WPF

#### OlapGrid

Assembly references	NuGet package
Syncfusion.Grid.WPF Syncfusion.GridCommon.WPF Syncfusion.Linq.Base Syncfusion.Olap.Base	Syncfusion.OlapGrid.WPF

Syncfusion.OlapGridCommon.WPF Syncfusion.OlapShared.WPF Syncfusion.Shared.WPF Syncfusion.Tools.WPF	
---	--

#### *Exporting OLAP Grid to Excel, Word, PDF and CSV*

For exporting OLAP grid contents to excel, word, PDF and CSV, the following reference needs to be added in addition to the above assemblies.

Assembly references	NuGet package
Syncfusion.OlapGridConverter.WPF	Syncfusion.ExportOlapGrid.Wpf

#### PDFViewer

Assembly references	NuGet package
Syncfusion.Compression.Base Syncfusion.Pdf.Base Syncfusion.PdfViewer.WPF Syncfusion.Shared.WPF	Syncfusion.PdfViewer.WPF

#### PercentTextBox

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

#### PinnableListBox

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

#### PivotGrid

Assembly references	NuGet package
Syncfusion.Grid.WPF Syncfusion.GridCommon.WPF Syncfusion.Linq.Base Syncfusion.PivotAnalysis.Base Syncfusion.Shared.WPF	Syncfusion.PivotTable.Wpf

#### *Exporting Pivot Grid to Excel, Word, PDF and CSV*

For exporting pivot grid contents to excel, word, PDF and CSV, the following references need to be added in addition to the above assemblies.

Assembly references	NuGet package
Syncfusion.PivotGridConverter.WPF Syncfusion.XlsIO.Base	Syncfusion.PivotTableExport.Wpf

## PropertyGrid

Assembly references	NuGet package
Syncfusion.PropertyGrid.WPF Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.PropertyGrid.WPF

## ReportDesigner

Assembly references	NuGet package
Syncfusion.Chart.Wpf Syncfusion.Gauge.Wpf Syncfusion.Grid.Wpf Syncfusion.Linq.Base Syncfusion.PropertyGrid.Wpf Syncfusion.ReportControls.Wpf Syncfusion.ReportDesigner.Wpf Syncfusion.ReportViewer.Wpf Syncfusion.SfMaps.Wpf Syncfusion.SfSkinManager.Wpf Syncfusion.Shared.Wpf Syncfusion.Tools.Wpf	Syncfusion.ReportDesigner.WPF

## ReportViewer

Assembly references	NuGet package
Syncfusion.Chart.Wpf Syncfusion.Compression.Base Syncfusion.DocIO.Base Syncfusion.Gauge.Wpf Syncfusion.Grid.Wpf Syncfusion.Linq.Base Syncfusion.Pdf.Base Syncfusion.PropertyGrid.Wpf Syncfusion.ReportControls.Wpf Syncfusion.ReportViewer.Wpf Syncfusion.SfMaps.Wpf Syncfusion.SfSkinManager.Wpf Syncfusion.Shared.Wpf	Syncfusion.ReportViewer.WPF

Syncfusion.Tools.Wpf Syncfusion.XlsIO.Base	
---	--

## ReportWriter

Assembly references	NuGet package
Syncfusion.Chart.Wpf Syncfusion.Compression.Base Syncfusion.DocIO.Base Syncfusion.Gauge.Wpf Syncfusion.Linq.Base Syncfusion.Pdf.Base Syncfusion.ReportControls.Wpf Syncfusion.ReportWriter.Base Syncfusion.SfMaps.Wpf Syncfusion.Shared.Wpf Syncfusion.XlsIO.Base	No separate nuget

## Ribbon

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## RichTextBoxAdv

Assembly references	NuGet package
Syncfusion.RichTextBoxAdv.WPF Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## RichTextDocIOParser

Assembly references	NuGet package
Syncfusion.RichTextDocIOParser.WPF Syncfusion.Compression.Base Syncfusion.OfficeChart.Base Syncfusion.RichTextBoxAdv.WPF Syncfusion.Shared.WPF For 3.5 and 4.0 frameworks: Syncfusion.DocIO.ClientProfile  For 4.5 and higher frameworks: Syncfusion.DocIO.Base	Syncfusion.RichTextDocIOParser.WPF

## RichTextRibbon

Assembly references	NuGet package
Syncfusion.RichTextRibbon.WPF Syncfusion.RichTextBoxAdv.WPF Syncfusion.Shared.WPF Syncfusion.Tools.WPF	Syncfusion.RichTextRibbon.WPF

## SfAccordion

Assembly references	NuGet package
Syncfusion.SfAccordion.WPF	Syncfusion.SfAccordion.WPF

## SfAreaSparkline

Assembly references	NuGet package
Syncfusion.SfChart.WPF	Syncfusion.SfChart.WPF

## SfBadge

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## SfBarcode

Assembly references	NuGet package
Syncfusion.SfBarcode.WPF	Syncfusion.SfBarcode.WPF

## SfBulletGraph

Assembly references	NuGet package
Syncfusion.SfBulletGraph.WPF	Syncfusion.SfBulletGraph.WPF

## SfBusyIndicator

Assembly references	NuGet package
Syncfusion.SfBusyIndicator.WPF Syncfusion.SfShared.WPF	Syncfusion.SfBusyIndicator.WPF

## SfCalculator

Assembly references	NuGet package
Syncfusion.SfInput.WPF Syncfusion.SfShared.WPF	Syncfusion.SfInput.WPF

## SfChart

Assembly references	NuGet package
Syncfusion.SfChart.WPF	Syncfusion.SfChart.WPF

## SfChromelessWindow

Assembly references	NuGet package
Syncfusion.SfChromelessWindow.WPF Syncfusion.Shared.WPF	Syncfusion.SfChromelessWindow.WPF

## SfCircularProgressBar

Assembly references	NuGet package
Syncfusion.SfProgressBar.WPF	Syncfusion.SfProgressBar.WPF

## SfColorPalette

Assembly references	NuGet package
Syncfusion.SfColorPalette.WPF Syncfusion.SfShared.WPF	Syncfusion.SfColorPalette.WPF

## SfColumnSparkline

Assembly references	NuGet package
Syncfusion.SfChart.WPF	Syncfusion.SfChart.WPF

## SfDataGrid

Assembly references	NuGet package
Syncfusion.SfGrid.WPF Syncfusion.Data.WPF Syncfusion.Shared.WPF	Syncfusion.SfGrid.WPF

*Exporting DataGrid to Excel, PDF and CSV*

For exporting DataGrid to excel, pdf or csv, the following references need to be added in your application.

Assembly references	NuGet package
Syncfusion.SfGridConverter.WPF Syncfusion.XlsIO.Base Syncfusion.Pdf.Base Syncfusion.Compression.Base	Syncfusion.DataGridExcelExport.Wpf

Refer [themes section](#) for theming assembly references.

*SfDataPager*

Assembly references	NuGet package
Syncfusion.SfGrid.WPF Syncfusion.Data.WPF Syncfusion.Shared.WPF	Syncfusion.SfGrid.WPF

*SfDatePicker*

Assembly references	NuGet package
Syncfusion.SfInput.WPF Syncfusion.SfShared.WPF	Syncfusion.SfInput.WPF

*SfDateTimeRangeNavigator*

Assembly references	NuGet package
Syncfusion.SfChart.WPF	Syncfusion.SfChart.WPF

*SfDiagram*

Assembly references	NuGet package
Syncfusion.SfDiagram.WPF	Syncfusion.SfDiagram.WPF

*Printing and Print Preview*

For printing the Diagram with new enhanced print preview, the below assembly need to be added in your application.

Assembly references	NuGet package
Syncfusion.SfShared.WPF	Syncfusion.SfShared.WPF



*Diagram Ribbon*

When your application already having SfDiagram and you need the enhanced ribbon to work with the most important settings and features of the Diagram, then the following assemblies need to be added in your application.

Assembly references	NuGet package
Syncfusion.SfDiagramRibbon.WPF Syncfusion.Compression.Base Syncfusion.Pdf.Base Syncfusion.Shared.WPF Syncfusion.Tools.WPF	Syncfusion.SfDiagramRibbon.WPF

*SfDomainUpDown*

Assembly references	NuGet package
Syncfusion.SfInput.WPF Syncfusion.SfShared.WPF	Syncfusion.SfInput.WPF

*SfGauge*

Assembly references	NuGet package
Syncfusion.SfGauge.WPF	Syncfusion.SfGauge.WPF

*SfGridSplitter*

Assembly references	NuGet package
Syncfusion.SfInput.WPF Syncfusion.SfShared.WPF	Syncfusion.SfInput.WPF

*SfHeatMap*

Assembly references	NuGet package
Syncfusion.SfHeatMap.WPF	Syncfusion.SfHeatMap.WPF

*SfHubTile*

Assembly references	NuGet package
Syncfusion.SfHubTile.WPF Syncfusion.SfShared.WPF	Syncfusion.SfHubTile.WPF

*SfKanban*

Assembly references	NuGet package

Syncfusion.SfKanban.WPF	Syncfusion.SfKanban.WPF
-------------------------	-------------------------

### SfLinearProgressBar

Assembly references	NuGet package
Syncfusion.SfProgressBar.WPF	Syncfusion.SfProgressBar.WPF

### SfLineSparkline

Assembly references	NuGet package
Syncfusion.SfChart.WPF	Syncfusion.SfChart.WPF

### SfMaps

Assembly references	NuGet package
Syncfusion.SfMaps.WPF Syncfusion.Shared.WPF	Syncfusion.SfMaps.WPF

### SfMaskedEdit

Assembly references	NuGet package
Syncfusion.SfInput.WPF Syncfusion.SfShared.WPF	Syncfusion.SfInput.WPF

### SfNavigationDrawer

Assembly references	NuGet package
Syncfusion.SfNavigationDrawer.WPF	Syncfusion.SfNavigationDrawer.WPF

### SfPulsingTile

Assembly references	NuGet package
Syncfusion.SfHubTile.WPF Syncfusion.SfShared.WPF	Syncfusion.SfHubTile.WPF

### SfRadialMenu

Assembly references	NuGet package
Syncfusion.SfRadialMenu.WPF Syncfusion.Shared.WPF	Syncfusion.SfRadialMenu.WPF

## SfRadialSlider

Assembly references	NuGet package
Syncfusion.SfRadialMenu.WPF Syncfusion.SfShared.WPF	Syncfusion.SfRadialMenu.WPF

## SfRangeSlider

Assembly references	NuGet package
Syncfusion.SfInput.WPF Syncfusion.SfShared.WPF	Syncfusion.SfInput.WPF

## SfRating

Assembly references	NuGet package
Syncfusion.SfInput.WPF Syncfusion.SfShared.WPF	Syncfusion.SfInput.WPF

## SfRichTextBoxAdv

Assembly references	NuGet package
Syncfusion.SfRichTextBoxAdv.WPF Syncfusion.Compression.Base Syncfusion.OfficeChart.Base Syncfusion.Shared.WPF For 3.5 and 4.0 frameworks: Syncfusion.DocIO.ClientProfile For 4.5 and higher frameworks: Syncfusion.DocIO.Base	Syncfusion.SfRichTextBoxAdv.WPF

## SfRichTextRibbon

Assembly references	NuGet package
Syncfusion.SfRichTextRibbon.WPF Syncfusion.Compression.Base Syncfusion.OfficeChart.Base Syncfusion.SfRichTextBoxAdv.WPF Syncfusion.Shared.WPF Syncfusion.Tools.WPF For 3.5 and 4.0 frameworks: Syncfusion.DocIO.ClientProfile For 4.5 and higher frameworks: Syncfusion.DocIO.Base	Syncfusion.SfRichTextRibbon.WPF

[SfScheduler](#)

Assembly references	NuGet package
Syncfusion.SfScheduler.WPF Syncfusion.SfInput.WPF Syncfusion.SfBusyIndicator.WPF Syncfusion.SfSkinManager.WPF Syncfusion.SfShared.WPF Syncfusion.Shared.WPF Syncfusion.Themes.MaterialLight.WPF	Syncfusion.SfScheduler.WPF

[SfSmithChart](#)

Assembly references	NuGet package
Syncfusion.SfSmithChart.WPF	Syncfusion.SfSmithChart.WPF

[SfSpellChecker](#)

Assembly references	NuGet package
Syncfusion.SpellChecker.Base Syncfusion.SfSpellChecker.WPF	Syncfusion.SfSpellChecker.WPF

[SfSpreadsheet](#)

Assembly references	NuGet package
Syncfusion.SfCellGrid.WPF Syncfusion.SfGridCommon.WPF Syncfusion.SfSpreadsheet.WPF Syncfusion.Shared.WPF Syncfusion.Tools.WPF Syncfusion.XlsIO.Base	Syncfusion.SfSpreadsheet.WPF

[Import Chart, Sparklines from Excel to SfSpreadsheet](#)

For Importing Chart or Sparklines in SfSpreadsheet, the following references need to be added in your application.

Assembly references	NuGet package
Syncfusion.SfSpreadsheetHelper.WPF Syncfusion.ExcelChartToImageConverter.WPF Syncfusion.SfChart.WPF	Syncfusion.SfSpreadsheetHelper.WPF

[Exporting SfSpreadsheet to PDF](#)

For exporting SfSpreadsheet to pdf, the following references need to be added in your application.

Assembly references	NuGet package
Syncfusion.ExcelToPDFConverter.Base Syncfusion.Pdf.Base	Syncfusion.SfSpreadsheet.WPF

[SfSunburstChart](#)

Assembly references	NuGet package
Syncfusion.SfSunburstChart.WPF	Syncfusion.SfSunburstChart.WPF

[SfTextboxExt](#)

Assembly references	NuGet package
Syncfusion.SfInput.WPF Syncfusion.SfShared.WPF	Syncfusion.SfInput.WPF

[SfTextInputLayout](#)

Assembly references	NuGet package
Syncfusion.SfTextInputLayout.WPF	Syncfusion.SfTextInputLayout.WPF

[SfTimePicker](#)

Assembly references	NuGet package
Syncfusion.SfInput.WPF Syncfusion.SfShared.WPF	Syncfusion.SfInput.WPF

[SfTreeGrid](#)

Assembly references	NuGet package
Syncfusion.SfGrid.WPF Syncfusion.Data.WPF Syncfusion.Shared.WPF	Syncfusion.SfGrid.WPF

[Exporting TreeGrid to Excel, PDF and CSV](#)

For exporting TreeGrid to excel, pdf or csv, the following references need to be added in your application.

Assembly references	NuGet package
Syncfusion.SfGridConverter.WPF Syncfusion.XlsIO.Base	Syncfusion.DataGridExcelExport.Wpf

Syncfusion.Pdf.Base Syncfusion.Compression.Base	
--	--

**SfTreeView**

Assembly references	NuGet package
Syncfusion.SfTreeView.WPF Syncfusion.SfBusyIndicator.WPF Syncfusion.SfGridCommon.WPF	Syncfusion.SfTreeView.WPF

**SfTreeMap**

Assembly references	NuGet package
Syncfusion.SfTreeMap.WPF	Syncfusion.SfTreeMap.WPF

**SfTreeNavigator**

Assembly references	NuGet package
Syncfusion.SfTreeNavigator.WPF Syncfusion.SfShared.WPF	Syncfusion.SfTreeNavigator.WPF

**SfWinLossSparkline**

Assembly references	NuGet package
Syncfusion.SfChart.WPF	Syncfusion.SfChart.WPF

**SkinManager**

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

**SplitButton**

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

**TabControlExt**

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## TabNavigation

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## TabSplitter

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## TaskBar

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## TileView

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## TimespanEdit

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## ToolBarAdv

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## TreeViewAdv

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Tools.WPF

## UpDown

Assembly references	NuGet package
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## Wizard

Assembly references	NuGet package
Syncfusion.Tools.WPF Syncfusion.Shared.WPF	Syncfusion.Shared.WPF

## SfSkinManager Dependencies

## Blend

Assembly references	NuGet package
Syncfusion.Themes.Blend.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Blend.WPF

## Lime

Assembly references	NuGet package
Syncfusion.Themes.Lime.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Lime.WPF

## Metro

Assembly references	NuGet package
Syncfusion.Themes.Metro.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Metro.WPF

## Office365

Assembly references	NuGet package
Syncfusion.Themes.Office365.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Office365.WPF

## Office2010Black

Assembly references	NuGet package
Syncfusion.Themes.Office2010Black.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Office2010Black.WPF



*Office2010Blue*

Assembly references	NuGet package
Syncfusion.Themes.Office2010Blue.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Office2010Blue.WPF

*Office2010Silver*

Assembly references	NuGet package
Syncfusion.Themes.Office2010Silver.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Office2010Silver.WPF

*Office2013DarkGray*

Assembly references	NuGet package
Syncfusion.Themes.Office2013DarkGray.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Office2013DarkGray.WPF

*Office2013LightGray*

Assembly references	NuGet package
Syncfusion.Themes.Office2013LightGray.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Office2013LightGray.WPF

*Office2013White*

Assembly references	NuGet package
Syncfusion.Themes.Office2013White.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Office2013White.WPF

*Office2016Colorful*

Assembly references	NuGet package
Syncfusion.Themes.Office2016Colorful.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Office2016Colorful.WPF

*Office2016DarkGray*

Assembly references	NuGet package
Syncfusion.Themes.Office2016DarkGray.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Office2016DarkGray.WPF

*Office2016White*

Assembly references	NuGet package
Syncfusion.Themes.Office2016White.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Office2016White.WPF

*Saffron*

Assembly references	NuGet package
Syncfusion.Themes.Saffron.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Saffron.WPF

*SystemTheme*

Assembly references	NuGet package
Syncfusion.Themes.SystemTheme.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.SystemTheme.WPF

*VisualStudio2013*

Assembly references	NuGet package
Syncfusion.Themes.VisualStudio2013.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.VisualStudio2013.WPF

*VisualStudio2015*

Assembly references	NuGet package
Syncfusion.Themes.VisualStudio2015.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.VisualStudio2015.WPF

*MaterialLight*

Assembly references	NuGet package
Syncfusion.Themes.MaterialLight.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.MaterialLight.WPF

*MaterialDark*

Assembly references	NuGet package
Syncfusion.Themes.MaterialDark.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.MaterialDark.WPF

*MaterialLightBlue*

Assembly references	NuGet package
Syncfusion.Themes.MaterialLightBlue.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.MaterialLightBlue.WPF

*MaterialDarkBlue*

Assembly references	NuGet package
Syncfusion.Themes.MaterialDarkBlue.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.MaterialDarkBlue.WPF

*Office2019Colorful*

Assembly references	NuGet package
Syncfusion.Themes.Office2019Colorful.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Office2019Colorful.WPF

*Office2019Black*

Assembly references	NuGet package
Syncfusion.Themes.Office2019Black.WPF Syncfusion.SfSkinManager.WPF	Syncfusion.Themes.Office2019Black.WPF

## Add Syncfusion WPF Controls

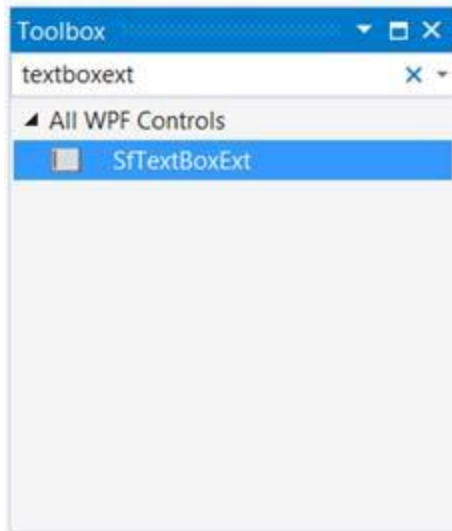
The Syncfusion WPF controls can be added in a Visual Studio projects by using either of the following ways,

- Through Designer
- Through Code-Behind
- Through Project Template

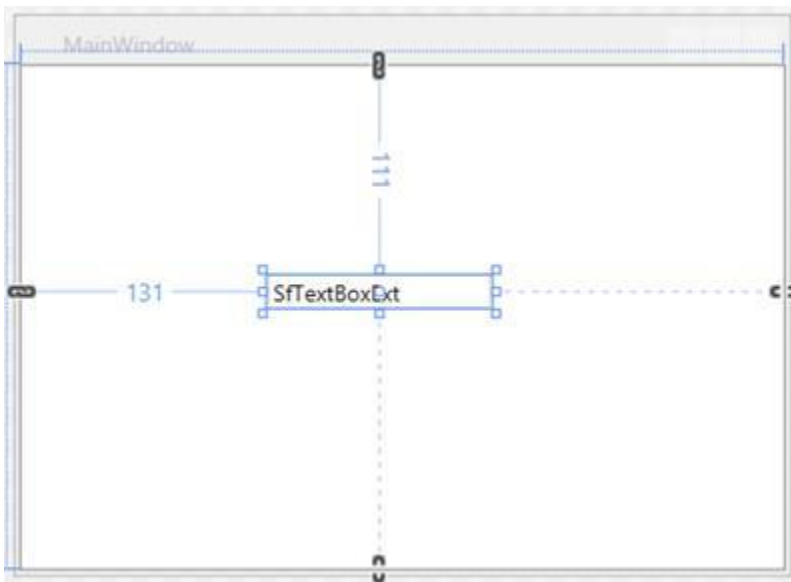
## Through Designer

Syncfusion UI for WPF are added automatically to the Visual Studio Toolbox during installation. The following steps helps to add required Essential WPF control through drag and drop from Toolbox. For example: **SfTextBoxExt**

1) Create a WPF project in Visual Studio. 2) Find **SfTextBoxExt** by typing the name of the "SfTextBoxExt" in the search box.



3) Drag **SfTextBoxExt** and drop it in the designer.



#### Through XAML

The following steps help to add a required Essential WPF Control through XAML Code, for example: **SfTextBoxExt**.

1) Create a WPF project in Visual Studio and refer the following assemblies. *Syncfusion.SfInput.WPF.dll* *Syncfusion.SfShared.WPF.dll* 2) Include an XML namespace for the above assemblies to the Main window.

#### XML

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:syncfusion="http://schemas.syncfusion.com/wpf" />
```

3) Now, Add the SfTextBoxExt control with a required optimal name, using the included namespace.

## XML

```
<syncfusion:SfTextBoxExt HorizontalAlignment="Center" Name="textBoxExt1"
Margin="10" Height ="20" Width="120" Text="SfTextBoxExt"
VerticalAlignment="Center"/>
```

## Through Code-Behind

Syncfusion UI for WPF can added at runtime using C# / VB. The following steps helps to add required Essential WPF control through code. For example: **SfTextBoxExt**.

1) Create a WPF project in Visual Studio and refer to the following assemblies.

*Syncfusion.SfInput.WPF.dll* *Syncfusion.SfShared.WPF.dll* 2) Create an instance of **SfTextBoxExt** using it namespace

## C#

```
Syncfusion.Windows.Controls.Input.SfTextBoxExt textBoxExt1 = new
Syncfusion.Windows.Controls.Input.SfTextBoxExt();
```

## VB.NET

```
Dim textBoxExt1 As New Syncfusion.Windows.Controls.Input.SfTextBoxExt()
```

3) Set Size and Alignment of the control with require value.

## C#

```
textBoxExt1.Height = 20;
textBoxExt1.Width = 120;
textBoxExt1.Margin = new Thickness(10, 10, 10, 10);
textBoxExt1.VerticalAlignment = VerticalAlignment.Center;
textBoxExt1.HorizontalAlignment = HorizontalAlignment.Center;
```

## VB.NET

```
textBoxExt1.Height = 20
textBoxExt1.Width = 120
textBoxExt1.Margin = New Thickness(10, 10, 10, 10)
textBoxExt1.VerticalAlignment = VerticalAlignment.Center
textBoxExt1.HorizontalAlignment = HorizontalAlignment.Center
```

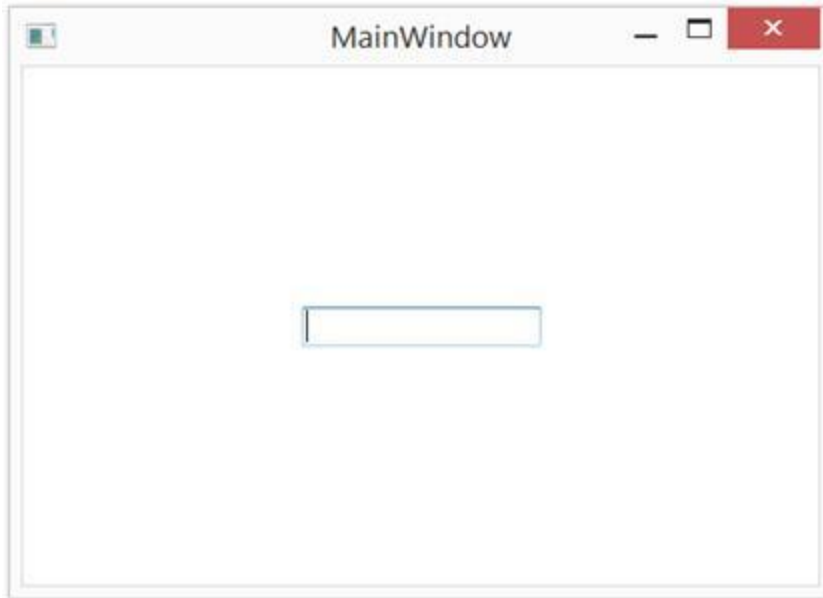
4) Add the created instance to the parent window (or the needed layout panels).

## C#

```
// Here this denotes parent Window
this.Content = textBoxExt1;
```

## VB.NET

```
' Here this denotes parent Window
Me.Content = textBoxExt1
```



#### Through Project Template

Syncfusion provides the Visual Studio Project Templates for the Syncfusion WPF platform to create Syncfusion WPF Application.

---

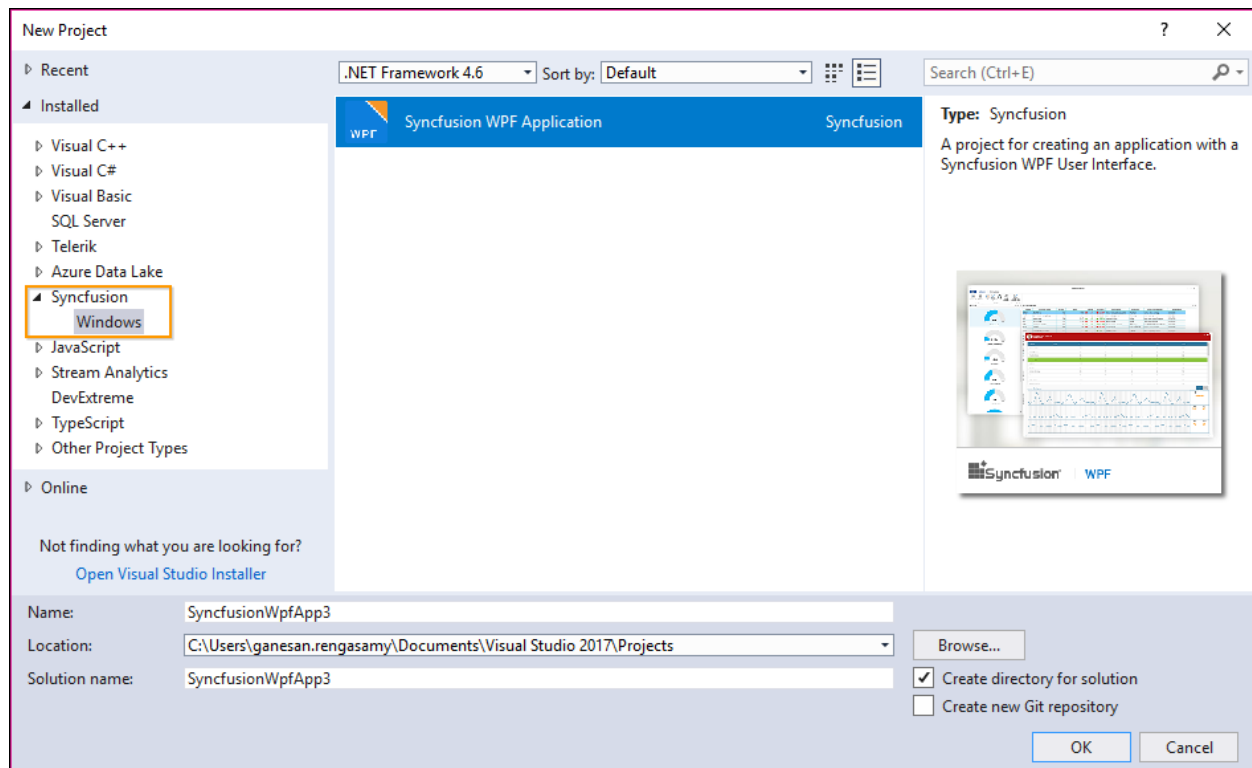
**Information:** The Syncfusion WPF templates are available from v16.1.0.24.

---

#### Create Syncfusion WPF Project

The following steps direct you to create the Syncfusion WPF project through the Visual Studio Project Template.

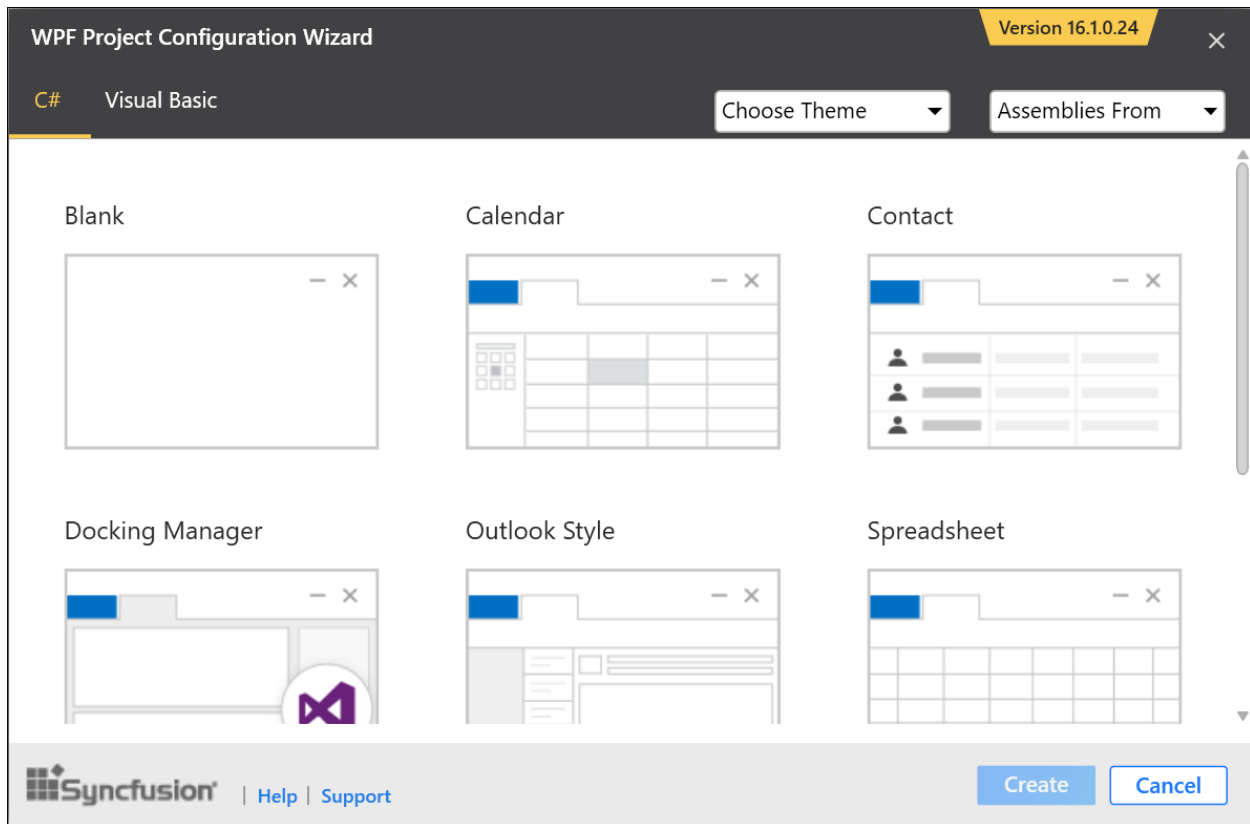
1) To create a Syncfusion WPF project, choose New Project-> Syncfusion->Windows->Syncfusion WPF Application from Visual Studio



2) Name the Project, choose the destination location when required and set the Framework of the project, then click OK.

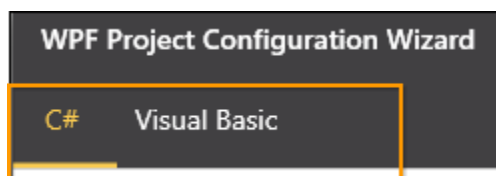
**Note:** Minimum target Framework is 4.0 for Syncfusion WPF project templates.

3) Choose the options to configure the Syncfusion WPF Application by using the following Project Configuration Wizard.



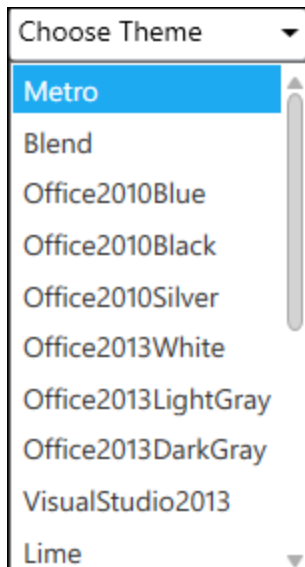
*Project configurations:*

**Language:** Select the language, either C# or VB.

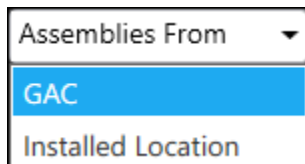


**Choose Theme:** Choose the required theme.

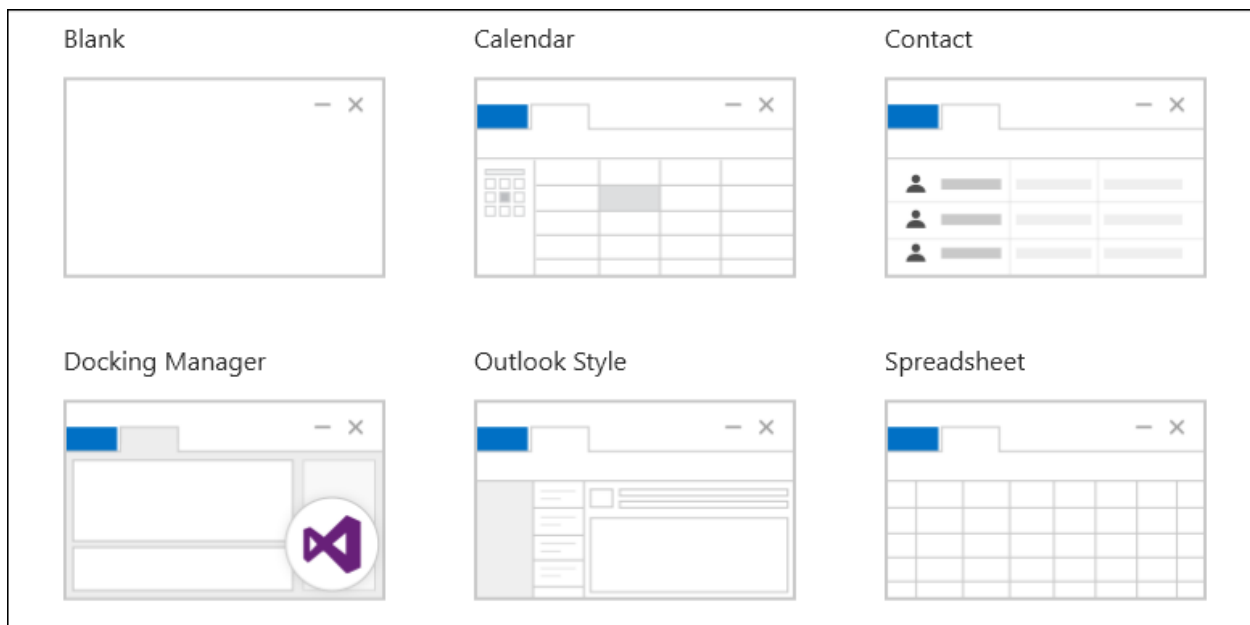




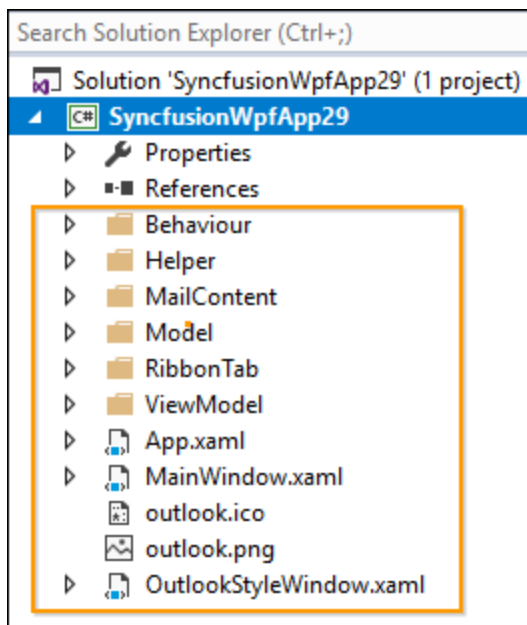
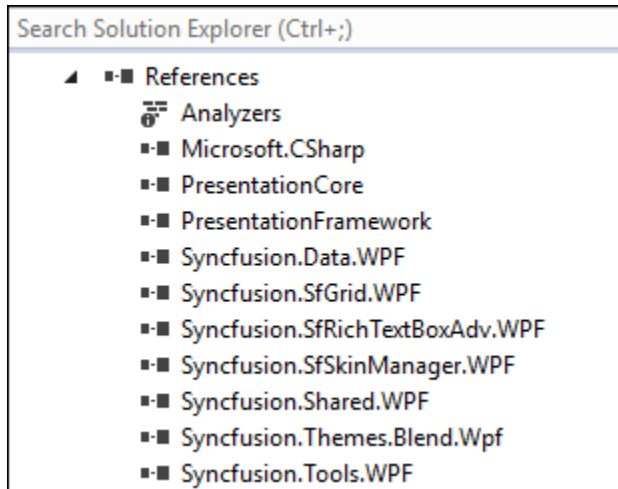
**Assemblies From:** Choose the assembly location from where it is going to be added to the project.



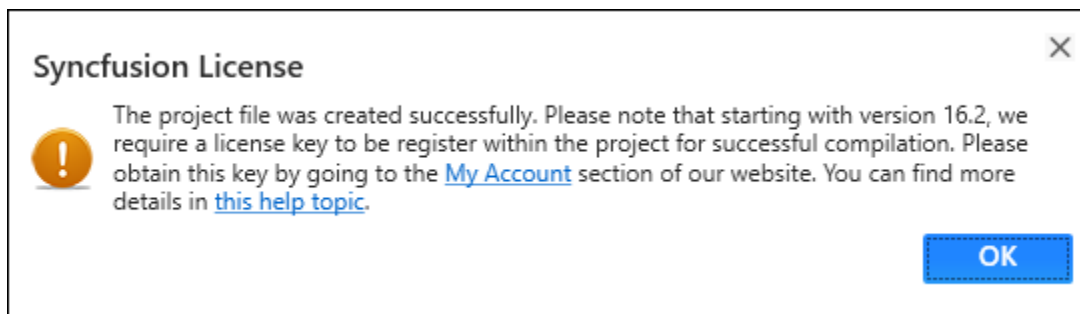
**Select Control:** Choose the control based on your need.



4) Once the Project Configuration Wizard is done, the Syncfusion WPF project is created with required references and XAML.



5) Then, Syncfusion licensing registration required message box will be shown as follow, if you are installed the trial setup or NuGet packages since Syncfusion introduced the licensing system from 2018 Volume 2 (v16.2.0.41) Essential Studio release. Please navigate to the [help topic](#) which is shown in the licensing message box to generate and register the Syncfusion license key to your project. Refer to this [blog](#) post for understanding the licensing changes introduced in Essential Studio.



## Localization of Syncfusion WPF Controls

Localization is the process of translating the application resources into different language for the specific cultures. You can localize the syncfusion WPF controls by adding resource file for each language.

### Changing application culture

When you are changing the application culture, then you can localize the application based on application culture by creating .resx file.

#### C#

```
public partial class MainWindow
{
    public MainWindow()
    {
        InitializeComponent();
        System.Threading.Thread.CurrentThread.CurrentUICulture = new
        System.Globalization.CultureInfo("de");
    }
}
```

#### VB.NET

```
Partial Public Class MainWindow
Public Sub New()
    InitializeComponent()
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("de")
End Sub
End Class
```

### Creating .resx files

You can create .resx files for any language by following below steps,

---

**Note:** You can get the default resource files of all Syncfusion WPF libraries from [GitHub](#).

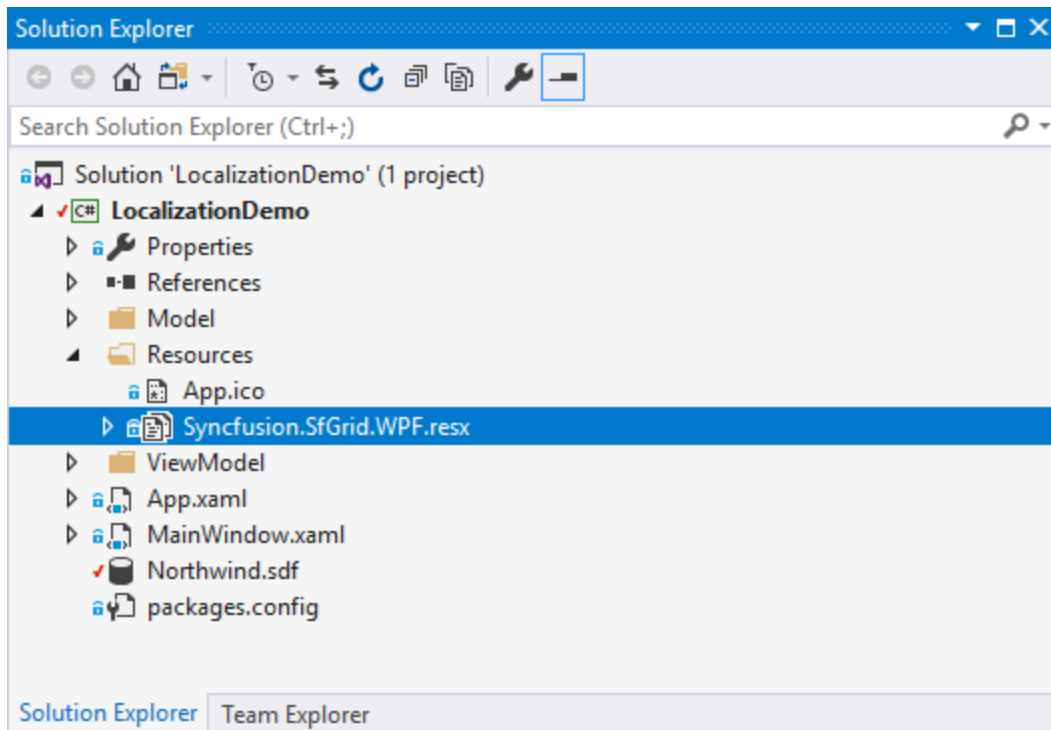
---

1) Right click your project and add new folder named Resources. 2) Add [default resource files](#) of libraries you are using into Resources folder and ensure AccessModifier specified as Public.

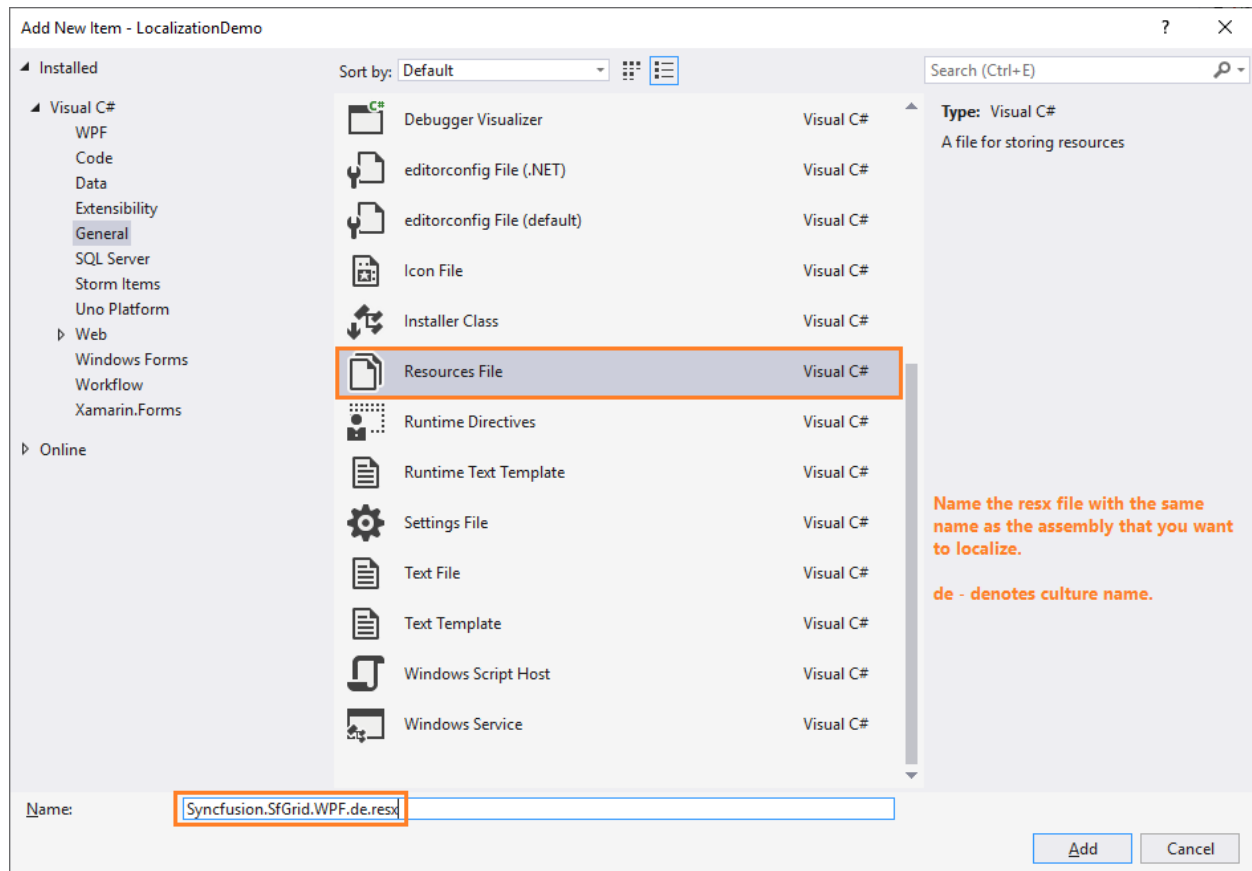
---

**Note:** Consider you are using SfDataGrid and Ribbon in your application. Then you need to copy and include Syncfusion.SfGrid.WPF.resx (since SfDataGrid present in Syncfusion.SfGrid.WPF library) and Syncfusion.Tools.Wpf.resx (since Ribbon present in Syncfusion.Tools.WPF library) files in your application under Resources folder. So, now you can know the key names and values of default stings used in Syncfusion.Tools.WPF.dll and Syncfusion.SfGrid.WPF.dll libraries.

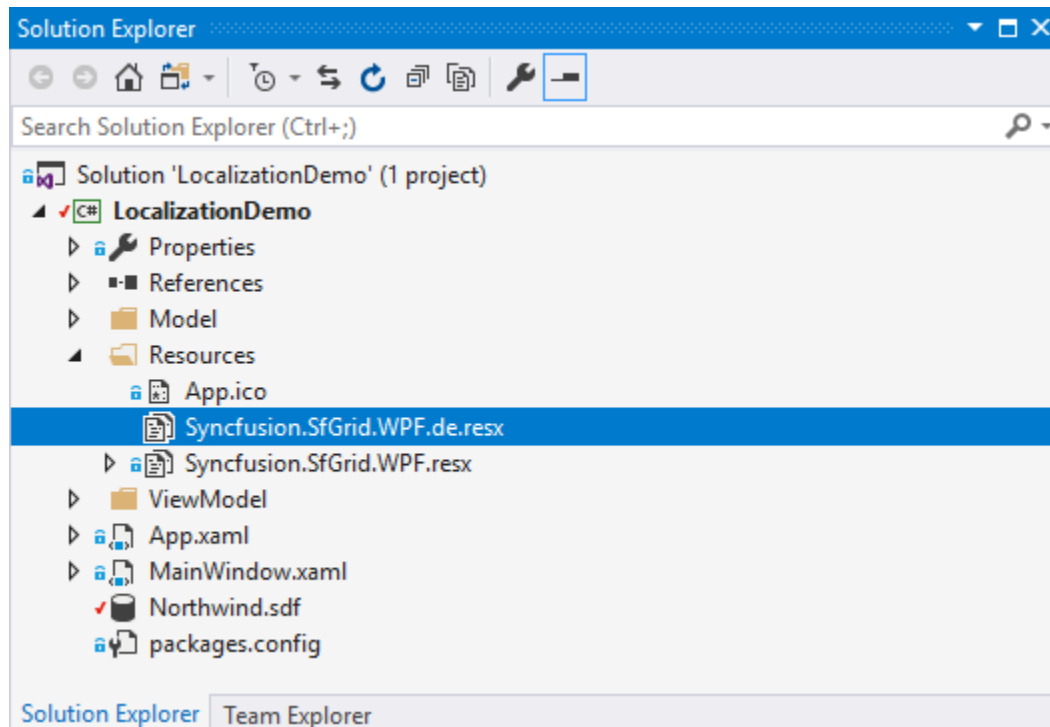
---



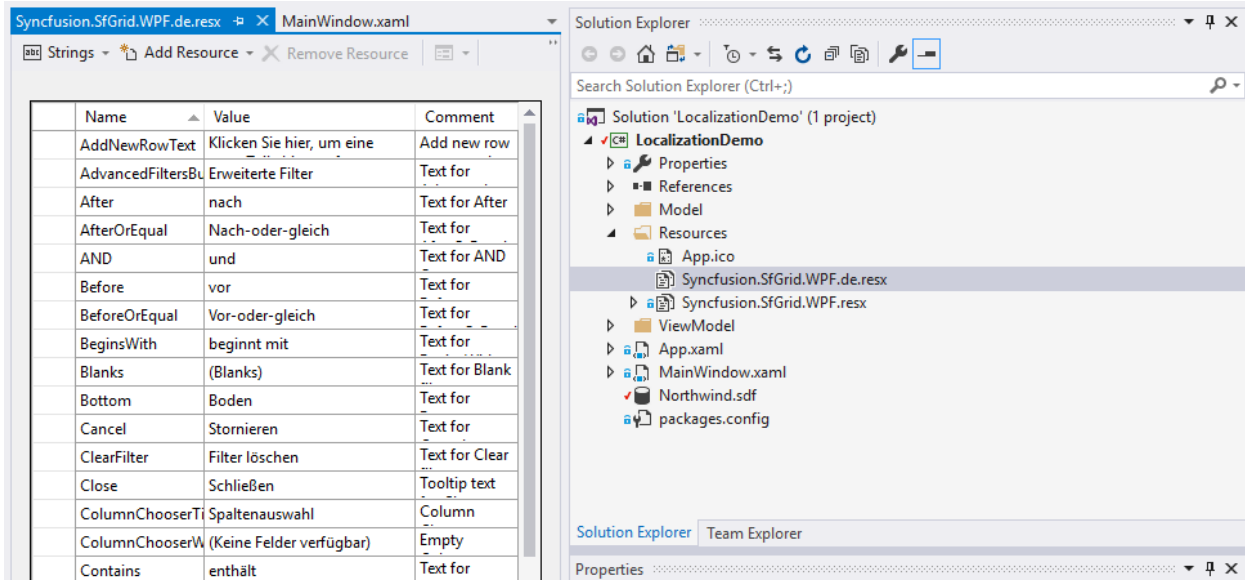
3) Now, right click **Resources** folder and select **Add** and then **NewItem**. In the **Add New Item** wizard, select the **Resource File** option and name the filename as **Syncfusion.SfGrid.WPF..resx**. For example, you have to give name as **Syncfusion.SfGrid.WPF.de.resx** for **German** culture. In the same way, add new resource files for other libraries used in your application.



4) Now, select **Add** and add resource file for German culture in **Resources** folder and set **AccessModifier** property to **No code generation**.



5) Now, you can copy the key names from default resource files and assign value based on the culture, the resource files targets.



**Note:** Download demo from [GitHub](#)

### Editing default culture strings

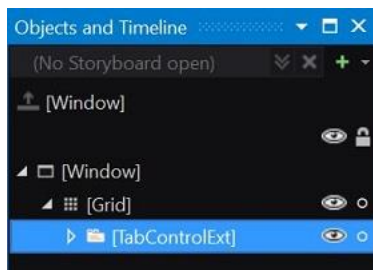
You can change default string of any control by adding the default .resx files ([from GitHub](#)) to **Resources** folder of your application. Syncfusion WPF controls reads the default string from the .resx files of application if its added.

### Getting Started for Blend Support

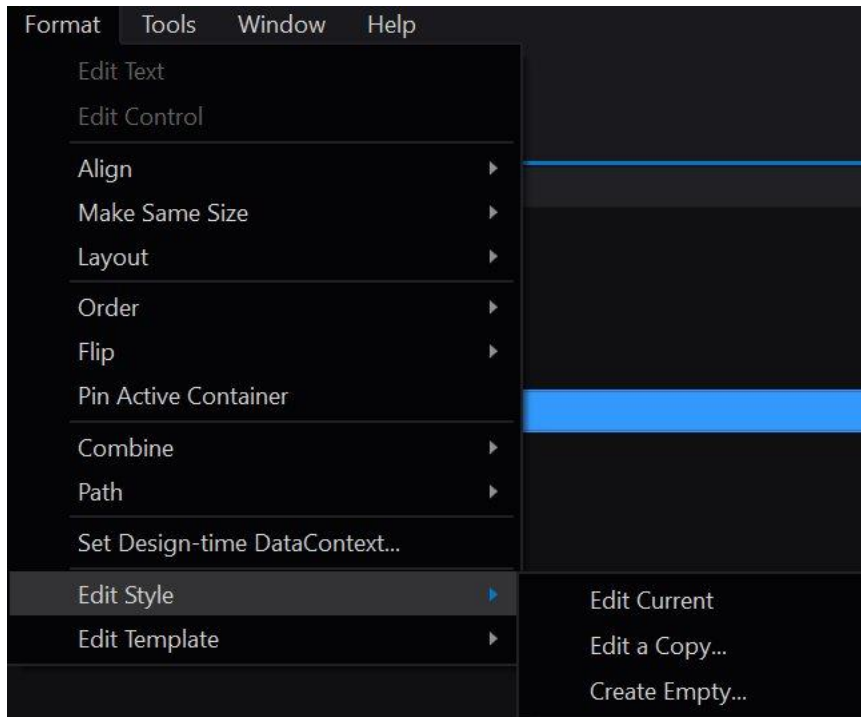
Essential WPF control's Style and Template can be editable in Expression Blend and this section explains the same.

#### Edit Control Style in Expression Blend

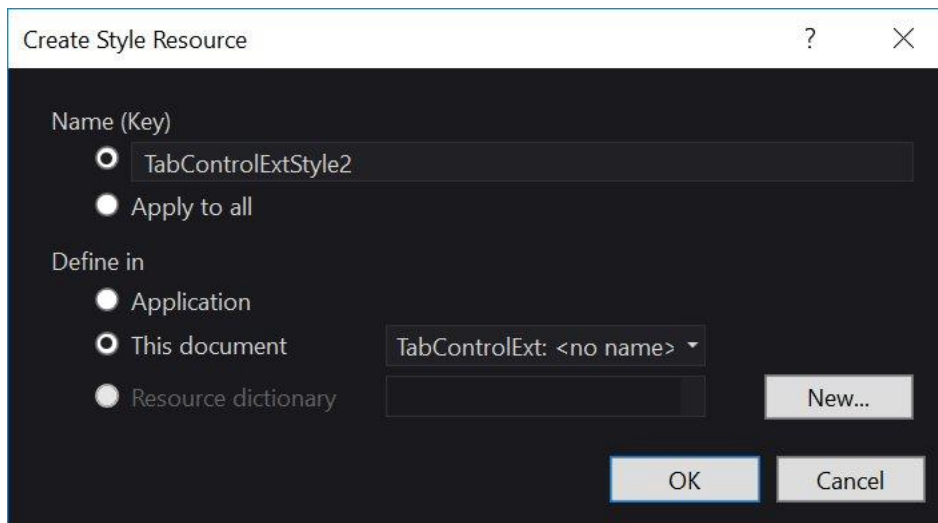
1.Open an application in Expression Blend. 2.Expand the Views, choose Other Windows and then choose Objects and Timeline. 3.In the “Objects and Timeline” pane, select a Syncfusion control to modify a style.



4.Expand the Format, then choose Edit Style menu.



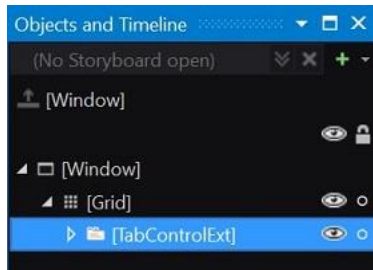
5. It provides the following options. *Edit Current* – To edit current applied style. This option is disabled when control doesn't have a style. *Edit a Copy* – Helps to edit a copy of default style. When it is selected, the Create Style Resource dialog box is opened; this dialog box is used to select the name for the style, as well as choose the location for where the file is defined in.



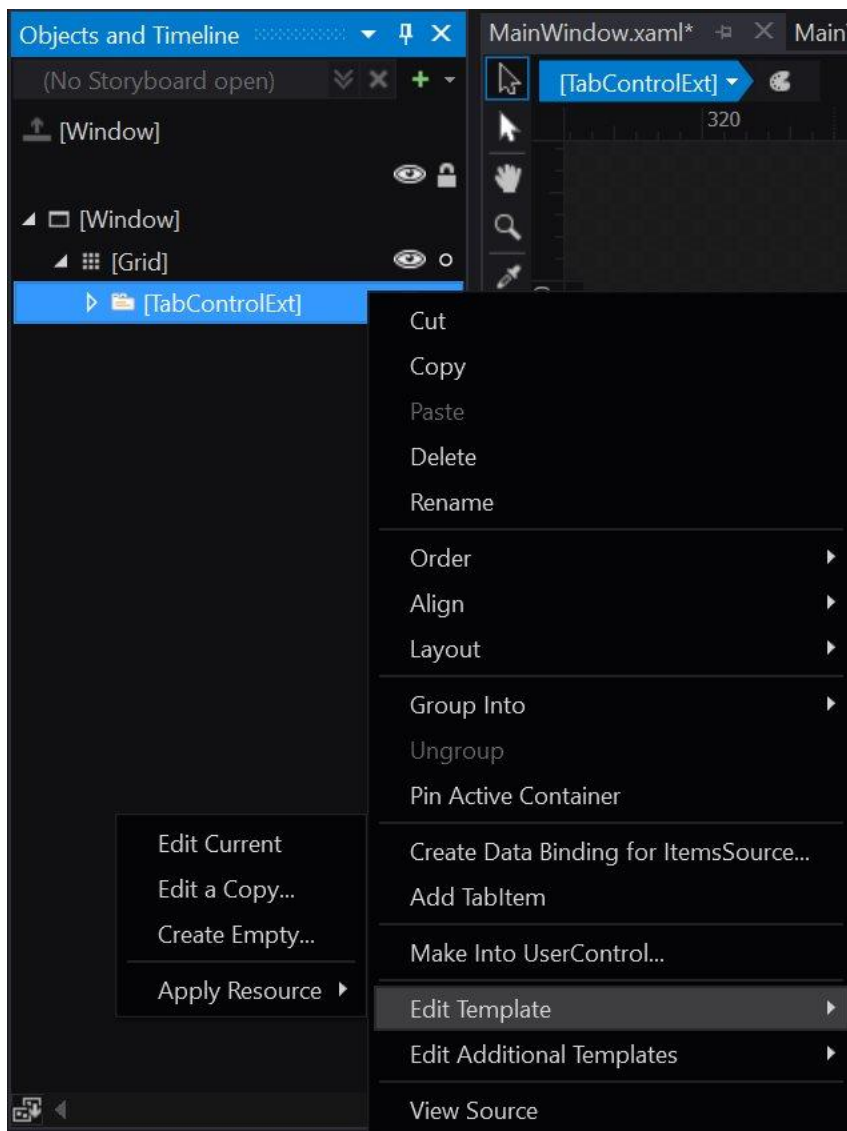
- *Create Empty* – this option helps to create an empty style for the selected control. When it is selected, the same Create Style Resource dialog box is opened and it is used to select the name for the style, as well as choose the location.

### Edit ControlTemplate in Expression Blend

1.Open an application in Expression Blend. 2.Expand the Views, choose Other Windows and then choose Objects and Timeline. 3.In the “Objects and Timeline” pane, select a Syncfusion control to modify a Template.



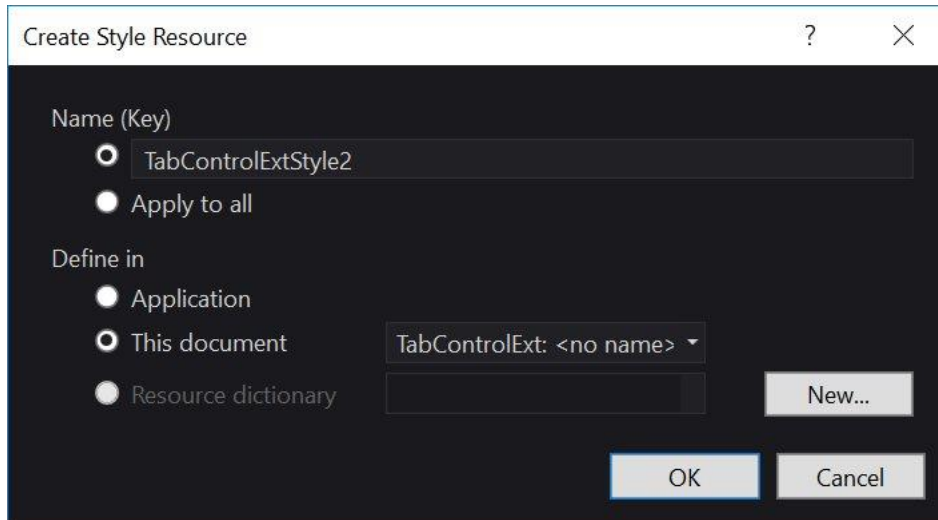
4.Right click on the Syncfusion control and choose Edit Template



5.It provide the following options. *Edit Current* – To edit the current Template of the control.This option is disabled when control don't have a Template. *Edit a Copy* – Helps to edit the default Template. When



it select, Create ControlTemplate Resource dialog box is opened, this dialog box is used to select the name for Template, as well as choose the location for where file is defined in.



- Create Empty – This option helps to create an empty template for selected control. When it selected, the same dialog box opened. It is used to select the name for Template, as well as the location.

### Right to Left support in Syncfusion WPF Controls

Right to Left(RTL) support displays the content from right-to-left direction by setting the [FlowDirection](#) property to `RightToLeft`. This is helpful to support the Right-to-Left scripted languages like Arabic, Hebrew, Urdu, etc.,

All WPF Syncfusion controls supports Right-to-Left (RTL) based on FlowDirection property. In addition to that, all controls provides [localization](#) support to change language of strings used in control specific to any culture.

### Pattern and Practices

#### Getting Started with MVVM

Essential WPF Controls are suitable for MVVM Pattern. Since the controls are provided with built-in commands.

DataContext property specifies the default source for Data Binding in MVVM pattern.

#### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
```

#### C#

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }
}
```

```
this.DataContext = new ViewModel();  
}  
}
```

This section explains how to perform TabControlExt's `selectionChanged` event, by create a ViewModel and define a Model collection that is used to bind with the TabControl's `ItemSource` property.

### XML

```
<Syncfusion:TabControlExt ItemsSource="{Binding tabcollection}">  
  <Syncfusion:TabControlExt.ItemTemplate>  
    <DataTemplate>  
      <TextBlock Text="{Binding HeaderName}"></TextBlock>  
    </DataTemplate>  
  </Syncfusion:TabControlExt.ItemTemplate>  
</Syncfusion:TabControlExt>
```

### C#

```
public class ViewModel:NotificationObject  
{  
  private ObservableCollection<model> _tabcollection;  
  public ObservableCollection<model> tabcollection  
  {  
    get  
    {  
      return _tabcollection;  
    }  
    set  
    {  
      _tabcollection = value;  
    }  
  }  
  private void Collection()  
  {  
    model model = new model()  
    {  
      HeaderName = "item1"  
    };  
    model model1 = new model()  
    {  
      HeaderName = "item2"  
    };  
    model model2 = new model()  
    {  
      HeaderName = "item3"  
    };  
    tabcollection.Add(model);  
    tabcollection.Add(model1);  
    tabcollection.Add(model2);  
  }  
  public ViewModel()  
  {  
    tabcollection=new ObservableCollection<model>();  
    Collection();  
  }  
}
```

```
}  
}  
public class model:NotificationObject  
{  
    public model() {}  
    private string _headername;  
    public string HeaderName  
    {  
        get  
        {  
            return _headername;  
        }  
        set  
        {  
            _headername = value;  
            this.RaisePropertyChanged("HeaderName");  
        }  
    }  
}
```

## VB.NET

```
Public Class ViewModel  
    Inherits NotificationObject  
    Private _tabcollection As ObservableCollection(Of model)  
    Public Property tabcollection() As ObservableCollection(Of model)  
    Get  
        Return _tabcollection  
    End Get  
    Set(ByVal value As ObservableCollection(Of model))  
        _tabcollection = value  
    End Set  
End Property  
Private Sub Collection()  
    Dim model As New model() With {.HeaderName = "item1"}  
    Dim model1 As New model() With {.HeaderName = "item2"}  
    Dim model2 As New model() With {.HeaderName = "item3"}  
    tabcollection.Add(model)  
    tabcollection.Add(model1)  
    tabcollection.Add(model2)  
End Sub  
Public Sub New()  
    tabcollection = New ObservableCollection(Of model)()  
End Sub  
End Class  
Public Class model  
    Inherits NotificationObject  
    Public Sub New()  
    End Sub  
    Private _headername As String  
    Public Property HeaderName() As String  
    Get  
        Return _headername  
    End Get  
    Set(ByVal value As String)
```

```

_headername = value
Me.RaisePropertyChanged("HeaderName")
End Set
End Property
End Class

```

To handle the `SelectionChanged` event of the `TabControlExt` in ViewModel, use `TabControlExtSelectionChangedCommand` and define a `SelectionChanged` command using `ICommand`.

The `TabControlExtSelectionChangedCommand` command is available in `Syncfusion.Tools.MVVM.WPF` assembly. And it requires `Syncfusion.Shared.MVVM.WPF` as dependency assembly.

### XML

```

<Syncfusion:TabControlExt ItemsSource="{Binding tabcollection}"
Syncfusion:TabControlExtSelectionChangedCommand.Command="{Binding
SelectionChanged}">
<Syncfusion:TabControlExt.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding HeaderName}"/>
</DataTemplate>
</Syncfusion:TabControlExt.ItemTemplate>
</Syncfusion:TabControlExt>

```

### C#

```

private ICommand selectionchanged;
public ICommand SelectionChanged
{
    get
    {
        return selectionchanged;
    }
}
public ViewModel()
{
    selectionchanged = new DelegateCommand<object>(PropertyChangedHandled);
}
private void PropertyChangedHandled (object obj)
{
    MessageBox.Show("Command Executed");
}

```

### VB.NET

```

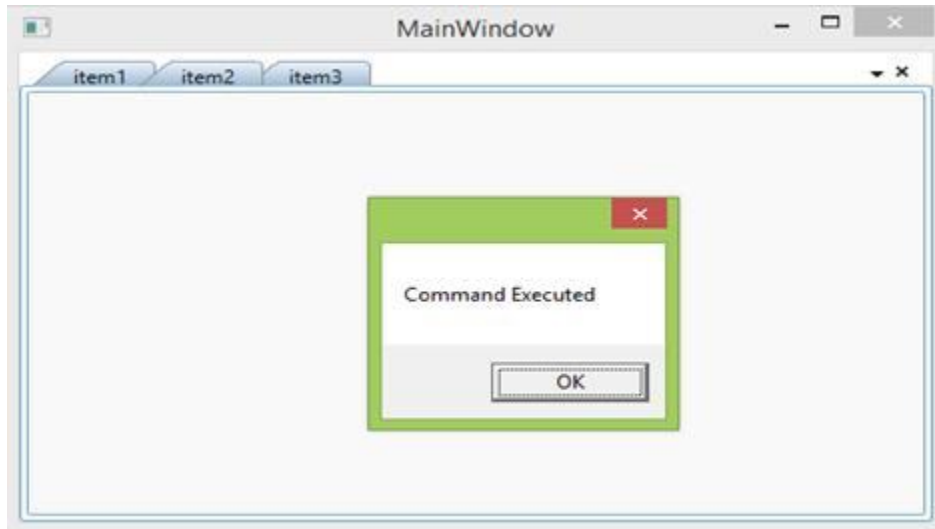
Private selectionchanged_Renamed As ICommand
Public ReadOnly Property SelectionChanged() As ICommand
Get
Return selectionchanged_Renamed
End Get
End Property
Public Sub New()

```

```

selectionchanged_Renamed = New DelegateCommand(Of Object) (AddressOf
PropertyChangedHandled)
End Sub
Private Sub PropertyChangedHandled(ByVal obj As Object)
    MessageBox.Show("Command Executed")
End Sub

```



### MVVM Commands

The following section covers usage of commands in code-behind with the help of ViewModel.

#### CommandParameter:

CommandParameter can easily pass an object or bind it to a property of another control. The following code example passes a string through CommandParameter.

#### XML

```

<Syncfusion:TabControlExt ItemsSource="{Binding tabcollection}"
Syncfusion:TabControlExtSelectionChangedCommand.Command="{Binding
SelectionChanged}"
Syncfusion:TabControlExtSelectionChangedCommand.CommandParameter="SelectedItem Command Parameter">

```

#### C#

```

private void PropertyChangedHandled (object obj)
{
    MessageBox.Show(obj.ToString());
}

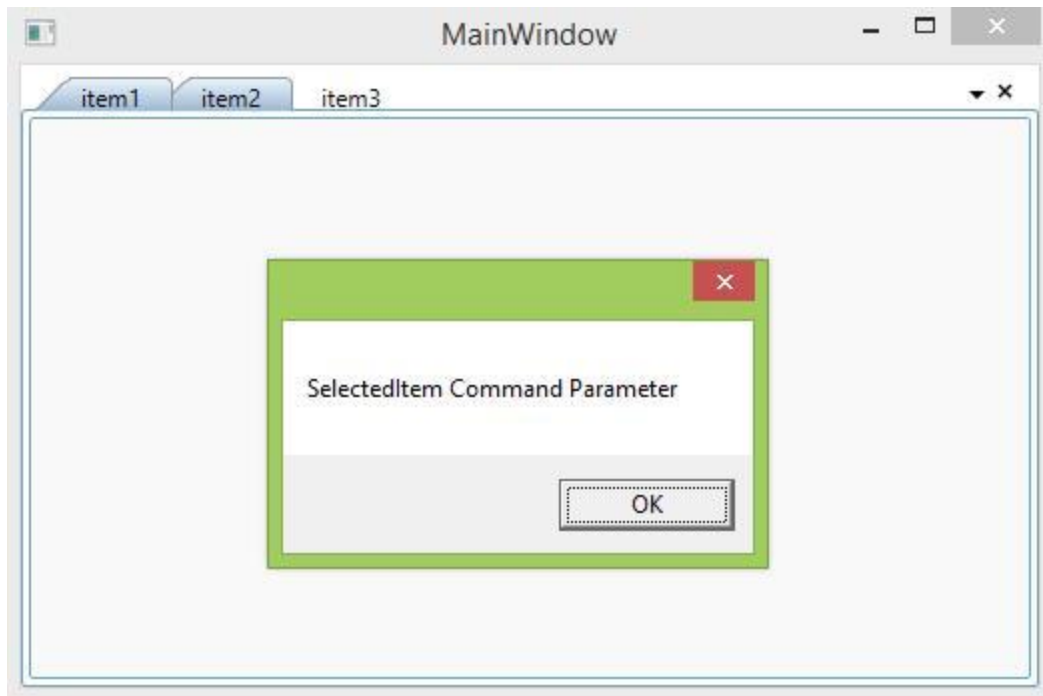
```

#### VB.NET

```

Private Sub PropertyChangedHandled(ByVal obj As Object)
    MessageBox.Show(obj.ToString())
End Sub

```



Pass a property value through Command Parameter:

Any property can bind with the **CommandParameter** to pass it as command in ViewModel.

#### XML

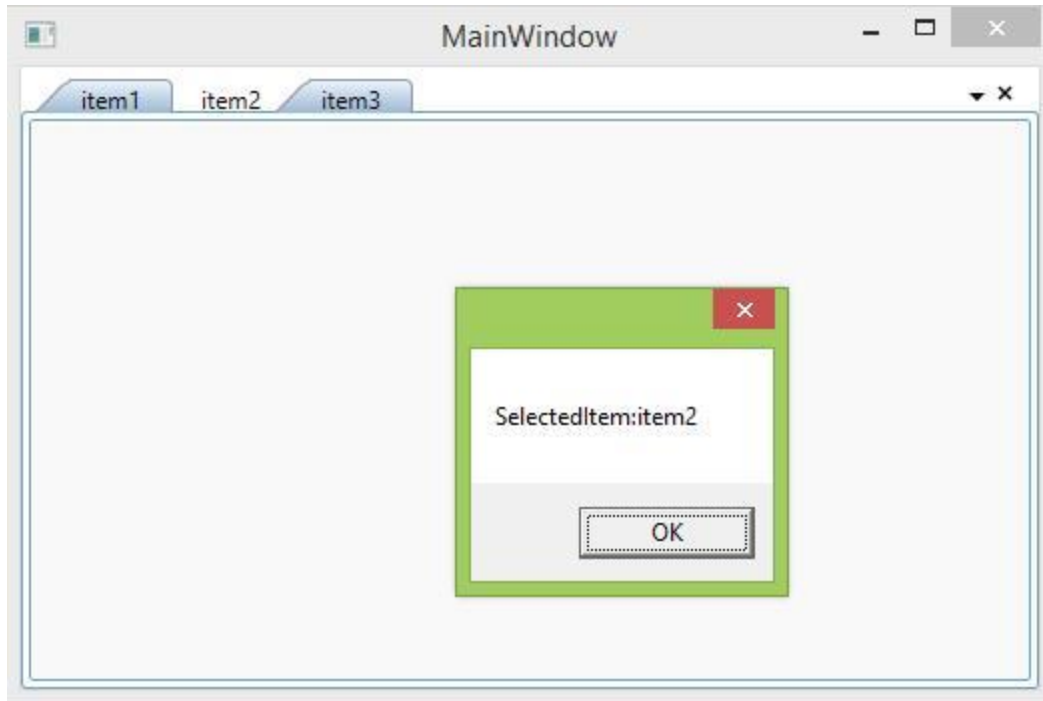
```
<Syncfusion:TabControlExt ItemsSource="{Binding tabcollection}"
Syncfusion:TabControlExtSelectionChangedCommand.Command="{Binding
SelectionChanged}"
Syncfusion:TabControlExtSelectionChangedCommand.CommandParameter="{Binding
Path=SelectedItem.HeaderName,
RelativeSource={RelativeSource Self}}">
```

#### C#

```
private void PropertyChangedHandled (object obj)
{
    MessageBox.Show("SelectedItem" +obj.ToString());
}
```

#### VB.NET

```
Private Sub PropertyChangedHandled(ByVal obj As Object)
    MessageBox.Show("SelectedItem" & obj.ToString())
End Sub
```



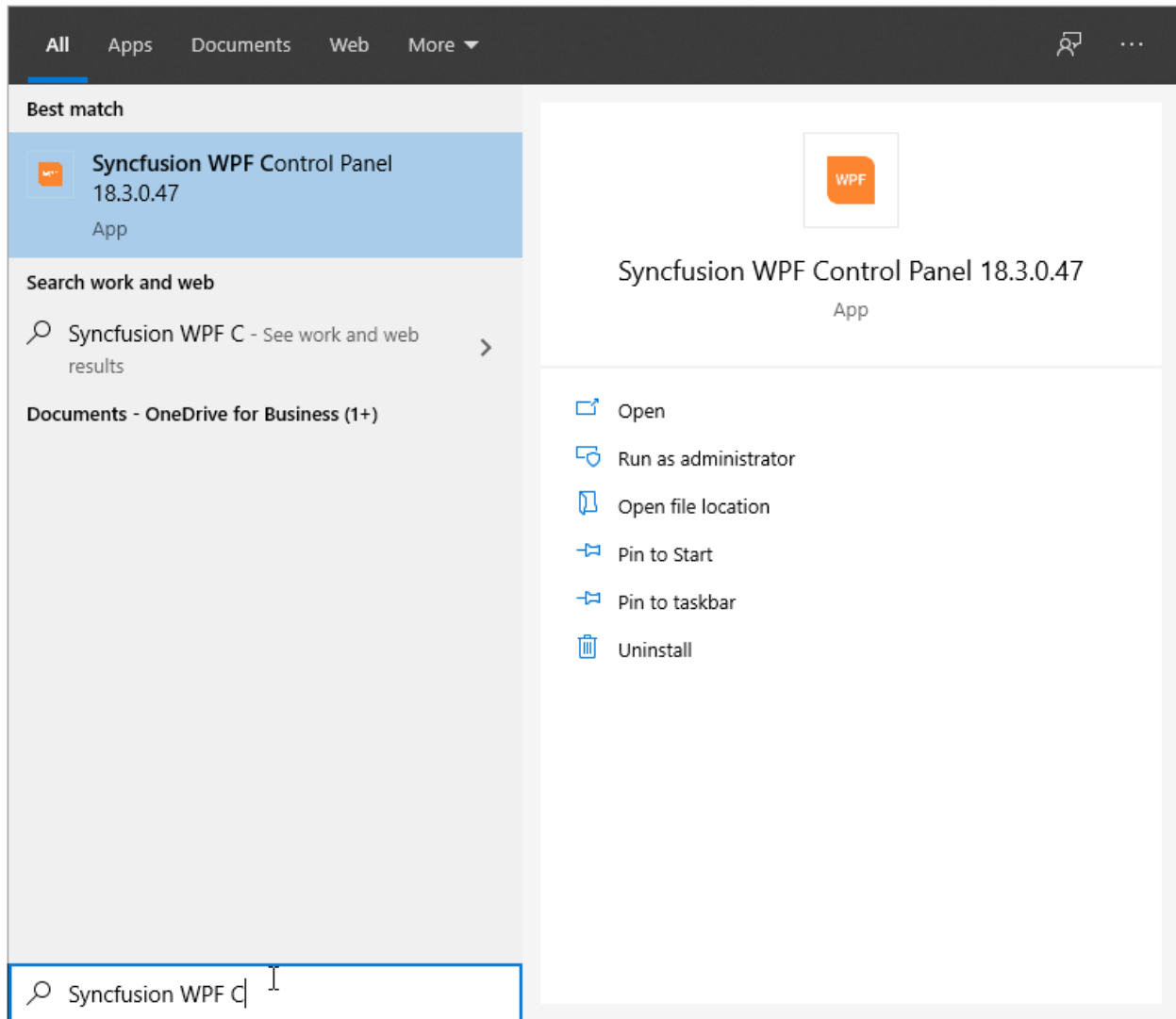
#### Command Target

The element where the command exists can be determined through EventBinding's **CommandTarget** property. Refer to [CommandTarget Property](#) for reference.

#### Featured Samples for Syncfusion WPF Controls

##### Syncfusion WPF Controls Panel

To explore Syncfusion WPF controls and components, Open **Syncfusion WPF Controls Panel** by searching it from start and open.



In another way, Open the Control Panel from the following installed location,

C:\Program Files (x86)\Syncfusion\Essential Studio\WPF\{{ site.releaseversion }}\Infrastructure\Launcher\Syncfusion WPF Control Panel.exe

**Note:** In above section, Latest Essential Studio version details has been provided. You can refer installed Essential Studio version instead of mentioned version.



**Syncfusion® | Essential Studio** Version 18.3.0.47

**WPF**

Enterprise-class toolkit for developing stunning WPF applications

**Run Local Demos** (dropdown menu: .NET Core, .NET Framework)

[Explore Demo Source](#) [Theme Studio](#)

[Notes](#) | [Read Me](#) | [User Guide](#) | [API Reference](#)

**Trial** You have 5 days left of your trial [Buy Now](#)

[Sales](#) | [Support](#) | [Suggest a Feature](#) | [License Agreement](#) | [Check for Updates](#) [Recheck](#)

## WPF Sample Browser

To explore the locally installed demos, click **Run Local Demos** and select either **.NET Core** or **.NET Framework** which will open **WPF Sample Browser**.

**WPF DEMOS 2020 Vol 3** Search Component

**Showcase Application**

- Patient History Demo
- Expense Analysis
- Car Dashboard
- Succinctly Series

**WPF Components**

GRIDS	CHARTS	DATA VISUALIZATION	LAYOUT	NAVIGATION
DataGrid	Charts	Diagram	Docking Manager	TabControl
TreeGrid	3D-Charts	Barcode	Document Container	TreeView <b>Now</b>
Grid Control	Range Navigator	Bullet Graph	Chromeless Window	Accordion
Property Grid <b>Updated</b>	Sparkline	Gauge <b>Updated</b>	Carousel	Hierarchical Navigator
	Smith Chart	Gantt	Card View	Navigation Drawer
	Sunburst Chart	Heat Map	Tile View	Tree Navigator
	Surface Chart	Kanban	Grid Splitter	Wizard Control

Documentation | Open source code in GitHub  
Copyright © 2001 - 2020 Syncfusion Inc.

## Offline Samples

The Offline samples are available in the following installed location where you can make changes and further exploration of controls.

C:\Users\Public\Documents\Syncfusion\WPF\{{ site.releaseversion }}

The offline samples can also be explored directly from Syncfusion WPF Sample Browser and opened in Visual Studio by run the required sample and click Open source code in visual studio button.

**DataGrid Demos** Office2019Colorful

← All Controls

**Getting Started**

This sample showcases the basic features in SfDataGrid by simple ObservableCollection binding.

Open source code in visual studio

Employee	Designator	Mail	Location	Status	Trust Worth	Rating	Salary	Address	Software Pr
Martin	Developer	martin@sample	UK	Active	Insufficient	★★★★☆	\$136,212.00	Kirchgasse 6	86.00%
Dodsworth	Developer	dodsworth@sam	Germany	Inactive	Sufficient	★★★★☆	\$309,510.00	Torikatu 38	90.00%
Michael	Designer	michael@rpy.co	France	Active	Insufficient	★★★★☆	\$134,055.00	Åkergatan 24	75.00%
Edward	Developer	edward@jourrap	Canada	Inactive	Insufficient	★★★★☆	\$155,377.00	Sierras de Gran	31.00%
Jack	System Analyst	jack@sample.co	Canada	Active	Insufficient	★★★★☆	\$113,365.00	Rua da Panificac	41.00%
Kathryn	Manager	kathryn@arpy.cc	Sweden	Inactive	Insufficient	★★★★☆	\$168,137.00	Torikatu 38	94.00%
Edward	Designer	edward@sample	France	Inactive	Perfect	★★★★☆	\$155,294.00	24, place Kléber	96.00%
Anne	Designer	anne@rpy.com	USA	Active	Insufficient	★★★★☆	\$305,558.00	P.O. Box 555	40.00%
Dodsworth	Manager	dodsworth@rpy	Canada	Active	Perfect	★★★★☆	\$356,812.00	2 rue du Comm	76.00%
Janet	Developer	janet@jourrapid	USA	Inactive	Perfect	★★★★☆	\$102,840.00	2817 Milton Dr.	96.00%
Leverling	CFO	leverling@jourra	Sweden	Inactive	Insufficient	★★★★☆	\$262,951.00	Mehrheimerstr. 3	94.00%
Dodsworth	System Analyst	dodsworth@arpy	Sweden	Inactive	Sufficient	★★★★☆	\$336,378.00	24, place Kléber	85.00%
Bergs	System Analyst	bergs@arpy.com	UK	Active	Sufficient	★★★★☆	\$295,926.00	Carrera 22 con A	64.00%
Dodsworth	Manager	dodsworth@jour	UK	Active	Insufficient	★★★★☆	\$303,879.00	Rua do Paço 67	54.00%
Fuller	Designer	fuller@arpy.com	Sweden	Active	Perfect	★★★★☆	\$384,353.00	Kirchgasse 6	82.00%
Andrew	Program Directo	andrew@arpy.cc	Sweden	Active	Insufficient	★★★★☆	\$103,353.00	Berliner Platz 43	86.00%
Tamer	Manager	tamer@sample.c	USA	Inactive	Insufficient	★★★★☆	\$276,804.00	P.O. Box 555	48.00%
Vinet	Program Directo	vinet@arpy.com	Argentina	Inactive	Perfect	★★★★☆	\$268,349.00	Berliner Platz 43	84.00%
Anne	Program Directo	anne@sample.c	UK	Inactive	Insufficient	★★★★☆	\$152,785.00	Carrera 22 con A	83.00%
Kathryn	Designer	kathryn@arpy.cc	Austria	Active	Insufficient	★★★★☆	\$235,254.00	P.O. Box 555	91.00%

### Offline Showcase Samples

To explore any individual showcase sample from Syncfusion WPF Sample Browser, Click Explore Demo Source and navigate to showcase folder.

**Syncfusion®** | Essential Studio

Version 18.3.0.47

WPF | Add On & Utilities | Documentation | Other Products

**WPF**

Enterprise-class toolkit for developing stunning WPF applications

Run Local Demos | **Explore Demo Source** | Theme Studio

**Resource**

What's New | Release Notes | Read Me | User Guide | API Reference

**Trial**

You have 9 days left of your trial

Buy Now

**Organizational Layout**

Sales | Support | Suggest a Feature | License Agreement | Check for Updates

Recheck

**Note:** To run the individual control demos, please refer the instruction from [Running Individual Control Demos](#).

### Online Samples

- Download and install .NetCore demos from [App Center](#).
- Download and install .Net Framework demos from [Microsoft Store](#).

Download demos from online (Clone from github repository)

You can explore Syncfusion WPF controls using [GitHub WPF demos](#) where all wpf demos are configured using **NuGet** to run without installing Syncfusion WPF Studio.

*Download showcase demos from online*

You can explore showcase demos from [GitHub WPF demos](#).

## Themes and Appearance

### Getting Started with WPF Skin Manager

The [SfSkinManager](#) helps you to apply the themes for both Syncfusion and Framework controls. There are 27 built-in themes that can be applied using the [SfSkinManager](#) for a rich user interface experience. Some of the built-in themes color derivations can be customized using [WPF Theme Studio](#).

**Note:** Theme Studio-based themes provide improved consistency and uniqueness among various controls when compared to other themes. It is preferable to use Theme Studio-based themes in the application over other themes.

### Themes list

The following table lists the available themes as well as the assembly or NuGet reference to be used in the application.

Styles	Assembly	NuGet package	Supported in Theme Studio	Alternative theme suggestion to use
FluentLight	Syncfusion.Themes.FluentLight.Wpf.dll	<a href="#">Syncfusion.Themes.FluentLight.WPF</a>	Yes	-
FluentDark	Syncfusion.Themes.FluentDark.Wpf.dll	<a href="#">Syncfusion.Themes.FluentDark.WPF</a>	Yes	-
MaterialLight	Syncfusion.Themes.MaterialLight.Wpf.dll	<a href="#">Syncfusion.Themes.MaterialLight.WPF</a>	Yes	-
MaterialDark	Syncfusion.Themes.MaterialDark.Wpf.dll	<a href="#">Syncfusion.Themes.MaterialDark.WPF</a>	Yes	-
MaterialLightBlue	Syncfusion.Themes.MaterialLightBlue.Wpf.dll	<a href="#">Syncfusion.Themes.MaterialLightBlue.WPF</a>	Yes	-
MaterialDarkBlue	Syncfusion.Themes.MaterialDarkBlue.Wpf.dll	<a href="#">Syncfusion.Themes.MaterialDarkBlue.WPF</a>	Yes	-

Office2019Colorful	Syncfusion.Themes.Office2019Colorful.Wpf.dll	<a href="#">Syncfusion.Themes.Office2019Colorful.WPF</a>	Yes	-
Office2019Black	Syncfusion.Themes.Office2019Black.Wpf.dll	<a href="#">Syncfusion.Themes.Office2019Black.WPF</a>	Yes	-
Office2019White	Syncfusion.Themes.Office2019White.Wpf.dll	<a href="#">Syncfusion.Themes.Office2019White.WPF</a>	Yes	-
Office2019DarkGray	Syncfusion.Themes.Office2019DarkGray.Wpf.dll	<a href="#">Syncfusion.Themes.Office2019DarkGray.WPF</a>	Yes	-
Office2019HighContrast	Syncfusion.Themes.Office2019HighContrast.Wpf.dll	<a href="#">Syncfusion.Themes.Office2019HighContrast.WPF</a>	Yes	-
Office2019HighContrastWhite	Syncfusion.Themes.Office2019HighContrastWhite.Wpf.dll	<a href="#">Syncfusion.Themes.Office2019HighContrastWhite.WPF</a>	Yes	-
SystemTheme	Syncfusion.Themes.SystemTheme.Wpf.dll	<a href="#">Syncfusion.Themes.SystemTheme.WPF</a>	Yes	-
Metro	Syncfusion.Themes.Metro.Wpf.dll	<a href="#">Syncfusion.Themes.Metro.WPF</a>	-	FluentLight , Office2019 Colorful
Lime	Syncfusion.Themes.Lime.Wpf.dll	<a href="#">Syncfusion.Themes.Lime.WPF</a>	-	FluentLight , Office2019 Colorful
Saffron	Syncfusion.Themes.Saffron.Wpf.dll	<a href="#">Syncfusion.Themes.Saffron.WPF</a>	-	FluentLight , Office2019 Colorful
Blend	Syncfusion.Themes.Blend.Wpf.dll	<a href="#">Syncfusion.Themes.Blend.WPF</a>	-	FluentDark, MaterialDark, Office2019 Black
Office2013White	Syncfusion.Themes.Office2013White.Wpf.dll	<a href="#">Syncfusion.Themes.Office2013White.WPF</a>	-	Office2019 White
Office2013LightGray	Syncfusion.Themes.Office2013LightGray.Wpf.dll	<a href="#">Syncfusion.Themes.Office2013LightGray.WPF</a>	-	Office2019 Colorful
Office2013DarkGray	Syncfusion.Themes.Office2013DarkGray.Wpf.dll	<a href="#">Syncfusion.Themes.Office2013DarkGray.WPF</a>	-	Office2019 DarkGray

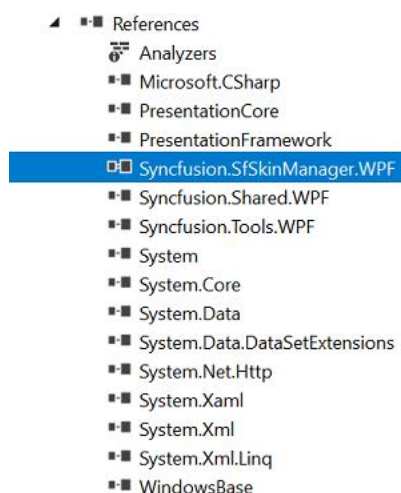
VisualStudio2013	Syncfusion.Themes.VisualStudio2013.Wpf.dll	<a href="#">Syncfusion.Themes.VisualStudio2013.WPF</a>	-	-
Office2010Black	Syncfusion.Themes.Office2010Black.Wpf.dll	<a href="#">Syncfusion.Themes.Office2010Black.WPF</a>	-	-
Office2010Blue	Syncfusion.Themes.Office2010Blue.Wpf.dll	<a href="#">Syncfusion.Themes.Office2010Blue.WPF</a>	-	-
Office2010Silver	Syncfusion.Themes.Office2010Silver.Wpf.dll	<a href="#">Syncfusion.Themes.Office2010Silver.WPF</a>	-	-
Office365	Syncfusion.Themes.Office365.Wpf.dll	<a href="#">Syncfusion.Themes.Office365.WPF</a>	-	Office2019 Colorful
Office2016Colorful	Syncfusion.Themes.Office2016Colorful.Wpf.dll	<a href="#">Syncfusion.Themes.Office2016Colorful.WPF</a>	-	Office2019 Colorful
Office2016White	Syncfusion.Themes.Office2016White.Wpf.dll	<a href="#">Syncfusion.Themes.Office2016White.WPF</a>	-	Office2019 White
Office2016DarkGray	Syncfusion.Themes.Office2016DarkGray.Wpf.dll	<a href="#">Syncfusion.Themes.Office2016DarkGray.WPF</a>	-	Office2019 DarkGray
VisualStudio2015	Syncfusion.Themes.VisualStudio2015.Wpf.dll	<a href="#">Syncfusion.Themes.VisualStudio2015.WPF</a>	-	-

Apply a theme to a control

[Add SkinManager reference](#)

There are several ways for including the Syncfusion [SfSkinManager](#) reference in the Visual Studio WPF project. The following steps will help you to add by XAML Code:

1) Add a reference to the `Syncfusion.SfSkinManager.WPF` assembly or [Syncfusion.SfSkinManager.WPF nuget package](#) to the project. 2) Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> or the assembly namespace `Syncfusion.SfSkinManager` into a XAML page.

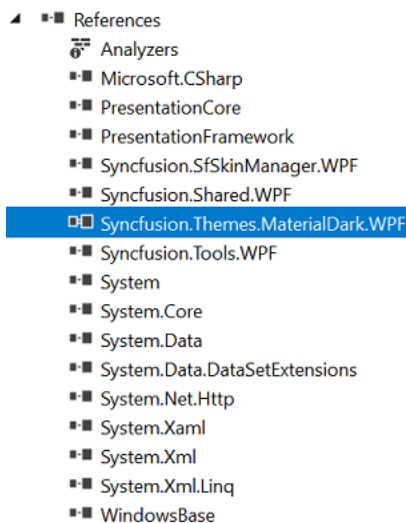


## XML

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf" />
```

### Add a theme assembly reference

The [SfSkinManager](#) supports to apply themes listed in [themes list](#). To use a theme in the application, add Reference to the corresponding theme assembly. For example, to apply **MaterialDark** theme, attach **Syncfusion.Themes.MaterialDark.Wpf** assembly or [NuGet](#) reference to the project. While applying a theme to a Window, SkinManager inherits the same theme to all the elements inside the Window.



### Set theme

Themes will be applied to both Syncfusion and Framework controls by using [Theme](#) attached property of the [SfSkinManager](#). The theme assemblies have resource dictionaries with styles of controls. Thus, when the **Theme** property is set, the skin manager merges the theme resource dictionaries of an element to which the theme is applied and its descendants into the resource dictionary of the element to which the theme is applied or **Application.Current.Resource**.

**Note:** While applying the theme to a Window or any element, **SkinManager** inherits the same theme to all its descendants.

## XML

```
<syncfusion:ChromelessWindow x:Class="DataGrid_Themes.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:DataGrid_Themes"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
Icon="App.ico"
Title="Getting Started"
WindowStartupLocation="CenterScreen"
```

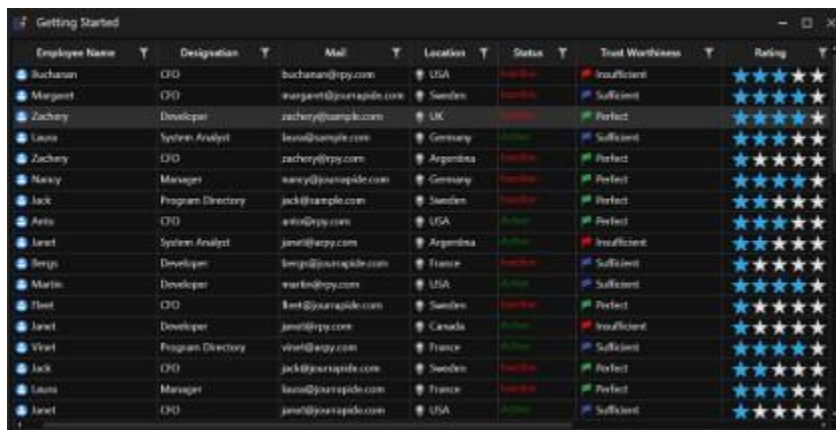
```

syncfusionskin:SfSkinManager.Theme="{syncfusionskin:SkinManagerExtension
ThemeName=MaterialDark}">
<Grid DataContext="{StaticResource viewmodel}">
<syncfusion:SfDataGrid Name="sfgrid" Margin="5"
AutoGenerateColumns="False"
AllowDraggingColumns="True"
AllowEditing="True"
LiveDataUpdateMode="AllowDataShaping"
AllowFiltering="True"
HeaderRowHeight="26"
SelectionMode="Extended"
ColumnSizer="Auto"
ItemsSource="{Binding EmployeeDetails}">
</syncfusion:SfDataGrid>
</Grid>
</syncfusion:ChromelessWindow>

```

**C#**

```
SfSkinManager.SetTheme(this, new Theme("MaterialDark"));
```



Employee Name	Designation	Mail	Location	Status	Trust Worthiness	Rating
Buchanan	CEO	buchanan@py.com	USA	Available	Insufficient	★★★★★
Margaret	CEO	margaret@py.com	Sweden	Available	Sufficient	★★★★★
Zachery	Developer	zachery@py.com	UK	Available	Perfect	★★★★★
Laura	System Analyst	laura@py.com	Germany	Available	Sufficient	★★★★★
Zachery	CEO	zachery@py.com	Argentina	Available	Perfect	★★★★★
Nancy	Manager	nancy@py.com	Germany	Available	Perfect	★★★★★
Jack	Program Director	jack@py.com	Sweden	Available	Perfect	★★★★★
Ante	CEO	ante@py.com	USA	Available	Perfect	★★★★★
Janet	System Analyst	janet@py.com	Argentina	Available	Insufficient	★★★★★
Berge	Developer	berge@py.com	France	Available	Sufficient	★★★★★
Martin	Developer	martin@py.com	USA	Available	Sufficient	★★★★★
Flot	CEO	flot@py.com	Sweden	Available	Perfect	★★★★★
Janet	Developer	janet@py.com	Canada	Available	Insufficient	★★★★★
Vincent	Program Director	vincent@py.com	France	Available	Sufficient	★★★★★
Jack	CEO	jack@py.com	Sweden	Available	Perfect	★★★★★
Laura	Manager	laura@py.com	France	Available	Perfect	★★★★★
Janet	CEO	janet@py.com	USA	Available	Sufficient	★★★★★

**Note:** [View sample in GitHub.](#)

Apply a theme globally in the application

By default, [SfSkinManager](#) merges the required resource files from the theme assembly to the element to which the theme is applied. To apply a theme globally in an application, set the `ApplyStylesOnApplication` property to `True`. It merges all the theme resource files to `Application.Current.Resources`.

**Note:** The `SfSkinManager.ApplyStylesOnApplication` static property should be set before `InitializeComponent` of the window or during application start up, when applying for multiple windows.

**C#**

```
SfSkinManager.ApplyStylesOnApplication = true;
```



### Customize theme colors and fonts in the application

To customize the theme colors and fonts in the application, call [RegisterThemeSettings](#) method and pass the theme name and respective theme setting instance as parameters.

Each theme supported by the theme studio has its own theme settings class, which begins with the prefix of the themes' name. For example, if the theme name is **MaterialDark**, then there will be theme settings class called **MaterialDarkThemeSettings**.

**Note:** Need to register theme settings before setting respective theme for window or control.

Please find the complete list of theme names, respective theme settings class, and supported palette.

Styles/Theme name	Respective theme settings class to customize	Supported palette
FluentLight	<a href="#">FluentLightThemeSettings</a>	<a href="#">FluentPalette</a>
FluentDark	<a href="#">FluentDarkThemeSettings</a>	<a href="#">FluentPalette</a>
MaterialLight	<a href="#">MaterialLightThemeSettings</a>	<a href="#">MaterialPalette</a>
MaterialDark	<a href="#">MaterialDarkThemeSettings</a>	<a href="#">MaterialPalette</a>
MaterialLightBlue	<a href="#">MaterialLightBlueThemeSettings</a>	<a href="#">MaterialPalette</a>
MaterialDarkBlue	<a href="#">MaterialDarkBlueThemeSettings</a>	<a href="#">MaterialPalette</a>
Office2019Colorful	<a href="#">Office2019ColorfulThemeSettings</a>	<a href="#">Office2019Palette</a>
Office2019Black	<a href="#">Office2019BlackThemeSettings</a>	<a href="#">Office2019Palette</a>
Office2019White	<a href="#">Office2019WhiteThemeSettings</a>	<a href="#">Office2019Palette</a>
Office2019DarkGray	<a href="#">Office2019DarkGrayThemeSettings</a>	<a href="#">Office2019Palette</a>
Office2019HighContrast	<a href="#">Office2019HighContrastThemeSettings</a>	<a href="#">HighContrastPalette</a>
Office2019HighContrastWhite	<a href="#">Office2019HighContrastWhiteThemeSettings</a>	<a href="#">HighContrastPalette</a>
SystemTheme	<a href="#">SystemThemeThemeSettings</a>	-

### Customize theme colors and fonts in the application

#### C#

```

FluentDarkThemeSettings themeSettings = new FluentDarkThemeSettings();
themeSettings.PrimaryBackground = new SolidColorBrush(Colors.Red);
themeSettings.PrimaryForeground = new SolidColorBrush(Colors.AntiqueWhite);
themeSettings.BodyFontSize = 15;
themeSettings.HeaderFontSize = 18;
themeSettings.SubHeaderFontSize = 17;
themeSettings.TitleFontSize = 17;
themeSettings.SubTitleFontSize = 16;
themeSettings.BodyAltFontSize = 15;
themeSettings.FontFamily = new FontFamily("Callibri");
SfsSkinManager.RegisterThemeSettings("FluentDark", themeSettings);

```



Getting Started											
Employee Name	Designation	Mail	Location	Status	Trust Worthiness	Rating	Salary	Address	Software Proficiency		
Edward	Manager	edward@rpy.com	Sweden	Active	Perfect	★★★★★	\$159,101.00	Carrera 22 con Ave.	5.00%		
Bergs	Manager	bergs@arpy.com	Germany	Active	Sufficient	★★★★★	\$358,806.00	Carrera 22 con Ave.	47.00%		
Tamer	Manager	tamer@jourapide.c	Sweden	Active	Sufficient	★★★★★	\$324,370.00	P.O. Box 555	72.00%		
Tamer	System Analyst	tamer@jourapide.c	France	Inactive	Perfect	★★★★★	\$143,381.00	Åkergatan 24	69.00%		
Martin	System Analyst	martin@jourapide.c	UK	Inactive	Perfect	★★★★★	\$284,356.00	Torikatu 38	73.00%		
Anto	Program Directory	anto@arpy.com	UK	Active	Sufficient	★★★★★	\$144,609.00	2 rue du Commerce	30.00%		
Tamer	Program Directory	tamer@sample.com	USA	Active	Sufficient	★★★★★	\$263,264.00	Berliner Platz 43	60.00%		
Van	Program Directory	van@sample.com	USA	Active	Sufficient	★★★★★	\$340,638.00	Luisenstr. 48	89.00%		
Vinet	Program Directory	vinet@arpy.com	Germany	Inactive	Sufficient	★★★★★	\$216,007.00	Rua da Panificadora 1	92.00%		
Kathryn	Designer	kathryn@rpy.com	UK	Inactive	Perfect	★★★★★	\$327,761.00	Torikatu 38	71.00%		
Jack	CFO	jack@arpy.com	Austria	Inactive	Insufficient	★★★★★	\$337,608.00	Torikatu 38	6.00%		
Laura	CFO	laura@rpy.com	UK	Inactive	Perfect	★★★★★	\$393,191.00	P.O. Box 555	31.00%		
Kathryn	CFO	kathryn@arpy.com	Germany	Active	Insufficient	★★★★★	\$271,132.00	Torikatu 38	30.00%		
Buchanan	Developer	buchanan@arpy.com	UK	Inactive	Sufficient	★★★★★	\$134,116.00	Hauptstr. 31	30.00%		
Zachery	Developer	zachery@arpy.com	France	Inactive	Insufficient	★★★★★	\$293,283.00	Rua do mailPaço 67	43.00%		
Zachery	Designer	zachery@rpy.com	Austria	Inactive	Insufficient	★★★★★	\$123,762.00	Sierras de Granada 95	21.00%		
Anto	CFO	anto@sample.com	UK	Active	Perfect	★★★★★	\$119,394.00	5ª Ave. Los Palos Gra	45.00%		
Van	CFO	van@arpy.com	Austria	Active	Sufficient	★★★★★	\$103,217.00	Torikatu 38	75.00%		
Margaret	Program Directory	margaret@rpy.com	Sweden	Inactive	Insufficient	★★★★★	\$203,173.00	Starenweg 5	37.00%		
Laura	CFO	laura@arpy.com	Argentina	Active	Perfect	★★★★★	\$394,943.00	1029 - 12th Ave. S.	12.00%		
Bergs	Developer	bergs@sample.com	Austria	Active	Sufficient	★★★★★	\$289,467.00	Rua do Mercado, 12	97.00%		
Bergs	CFO	bergs@arpy.com	USA	Inactive	Sufficient	★★★★★	\$133,832.00	Carlos Soubllette #8-	13.00%		
Leverling	Program Directory	leverling@arpy.com	Austria	Active	Sufficient	★★★★★	\$195,463.00	Sierras de Granada 95	57.00%		
Fuller	System Analyst	fuller@jourapide.c	UK	Active	Sufficient	★★★★★	\$118,856.00	Rua do Mercado, 12	84.00%		
Van	Program Directory	van@rpy.com	UK	Active	Perfect	★★★★★	\$258,298.00	Kirchgasse 6	7.00%		
Martin	Manager	martin@arpy.com	Canada	Active	Perfect	★★★★★	\$262,463.00	Åkergatan 24	57.00%		
Callahan	Developer	callahan@rpy.com	USA	Active	Sufficient	★★★★★	\$377,724.00	Starenweg 5	8.00%		
Laura	Program Directory	laura@arpy.com	Austria	Active	Perfect	★★★★★	\$170,591.00	1029 - 12th Ave. S.	7.00%		
Andrew	System Analyst	andrew@arpy.com	Austria	Inactive	Sufficient	★★★★★	\$255,357.00	Rua do Mercado, 12	20.00%		

Customize theme colors using the predefined palette

### C#

```
FFluentDarkThemeSettings themeSettings = new FluentDarkThemeSettings();
themeSettings.Palette = FluentPalette.PinkRed;
SfSkinManager.RegisterThemeSettings("FluentDark", themeSettings);
```

**Note:** [View sample in GitHub.](#)

Apply themes to the controls derived from Syncfusion controls

To apply themes to the derived control using `SfSkinManager`, call `SetResourceReference` method and, pass the `StyleProperty` and derived control type as parameters.

### XML

```
<local:SfDataGridExt x:Name="grid"
AllowGrouping="True"
AutoGenerateColumns="False"
ItemsSource="{Binding EmployeeDetails}"
ShowGroupDropArea="True">
<local:SfDataGridExt.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.SfGrid.WPF;component/Styles/Styles.xaml" />
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</local:SfDataGridExt.Resources>
<local:SfDataGridExt.Columns>
<syncfusion:GridNumericColumn MappingName="EmployeeAge" />
<syncfusion:GridTextColumn MappingName="EmployeeName" />
<syncfusion:GridTextColumn MappingName="EmployeeGender" />
<syncfusion:GridTextColumn AllowEditing="True" MappingName="Country" />
```

```
<syncfusion:GridNumericColumn MappingName="EmployeeSalary" />
</local:SfDataGridExt.Columns>
</local:SfDataGridExt>
```

### C#

```
public class SfDataGridExt : SfDataGrid
{
    public SfDataGridExt()
    {
        SetResourceReference(StyleProperty, typeof(SfDataGrid));
    }
}
```

### Clearing SkinManager instance in an application

The `SfSkinManager` will hold some instances to use it further when applying the theme. However, this can be cleared using the function named `Dispose(object)`, which must be called when the theme applied by `SfSkinManager` is to be cleared, as shown in the following code. **Object** refers to the element whose instance needs to be cleared.

### C#

```
private void Window_Closed(object sender, EventArgs e)
{
    SfSkinManager.Dispose(this);
}
```

### How to

#### *Apply custom theme in the application*

To apply a custom theme in the application, export the custom theme project from ThemeStudio using [this reference](#).

For demonstration purposes, the exported theme name has been used as `MaterialDarkYellow` and assembly name as `Syncfusion.Themes.MaterialDarkYellow.WPF`.

Now, for the control used in the application, set the `SfSkinManager` attached property `Theme` to `MaterialDarkYellow;MaterialDark`. Since, custom theme name should be updated in the following format: `CustomTheme1;BaseThemeName`, where `CustomTheme1` denotes the custom theme name and `BaseThemeName` denotes the theme name from which it is derived. For example, `MaterialDarkYellow;MaterialDark`.

### C#

```
SfSkinManager.SetTheme(this, new Theme("MaterialDarkYellow;MaterialDark"));
```

#### *Override syncfusion themes in the application*

All Syncfusion themes are [supported in theme studio](#). A common naming convention can be used to override control styles. A unique key is given to every style so that the styles can be overridden using the `BasedOn` property.

The naming convention of a control style will be like `Syncfusion-ControlName-Style`. For example, `MaterialDarkButtonAdvStyle`.

The following steps explain how to override the Syncfusion themes:

#### Step1:

Add respective resource dictionary from the themes assembly to the application.

The resource dictionary will be in this format:

`/Syncfusion.Themes.<ThemeName>.WPF;component/<ControlName>/<ControlName>.xml`, where `ThemeName` denotes the theme name for which overridden style is going to be applied and `ControlName` denotes the control name. For example, `/Syncfusion.Themes.MaterialDark.WPF;component/ButtonAdv/ButtonAdv.xml`.

#### XML

```
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.Themes.MaterialDark.WPF;component/ButtonAdv/ButtonAdv.xml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

#### Step2:

Define the new style using the `BasedOn` property.

The following code sample overrides the Syncfusion style for the `ButtonAdv` Control.

#### XML

```
<Grid>
<Grid.Resources>
<Style x:Key="CustomButtonAdvStyle" TargetType="syncfusion:ButtonAdv"
BasedOn="{StaticResource SyncfusionButtonAdvStyle}" >
<Setter Property="Foreground" Value="Red"/>
</Style>
</Grid.Resources>
<syncfusion:ButtonAdv Name="buttonAdv" Content="Testing" Width="150"
Height="30" SmallIcon="{x:Null}" Style="{StaticResource
CustomButtonAdvStyle}"></syncfusion:ButtonAdv>
</Grid>
```



#### *Change visual style at runtime*

Themes for application can be changed at runtime by changing `VisualStyle` property. Make sure that the new theme assembly is attached as reference in the application when applying theme.

- Syncfusion.Linq.Base
- Syncfusion.SfGrid.WPF
- Syncfusion.SfInput.WPF
- Syncfusion.SfShared.WPF
- Syncfusion.SfSkinManager.WPF
- Syncfusion.Shared.WPF
- Syncfusion.Themes.Blend.WPF
- Syncfusion.Themes.Lime.WPF
- Syncfusion.Themes.MaterialDark.WPF
- Syncfusion.Themes.MaterialDarkBlue.WPF
- Syncfusion.Themes.MaterialLight.WPF
- Syncfusion.Themes.MaterialLightBlue.WPF
- Syncfusion.Themes.Metro.WPF
- Syncfusion.Themes.Office2010Black.WPF
- Syncfusion.Themes.Office2010Blue.WPF
- Syncfusion.Themes.Office2010Silver.WPF
- Syncfusion.Themes.Office2013DarkGray.WPF
- Syncfusion.Themes.Office2013LightGray.WPF
- Syncfusion.Themes.Office2013White.WPF
- Syncfusion.Themes.Office2016Colorful.WPF
- Syncfusion.Themes.Office2016DarkGray.WPF
- Syncfusion.Themes.Office2016White.WPF
- Syncfusion.Themes.Office2019Black.WPF

## XML

```
<syncfusion:ChromelessWindow x:Class="DataGrid_Themes.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:DataGrid_Themes"
xmlns:system="clr-namespace:System;assembly=microsoftcorlib"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:syncfusionskin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStyle="{Binding
ElementName=comboVisualStyle, Path=SelectedValue, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">
<syncfusion:ChromelessWindow.Resources>
<ObjectDataProvider
x:Key="Themes"
MethodName="GetValues"
ObjectType="{x:Type system:Enum}">
<ObjectDataProvider.MethodParameters>
<x:Type TypeName="syncfusionskin:VisualStyles" />
</ObjectDataProvider.MethodParameters>
</ObjectDataProvider>
</syncfusion:ChromelessWindow.Resources>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="40"/>
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<ComboBox
x:Name="comboVisualStyle"
Grid.Row="0"
Width="250"
Margin="5"
ItemsSource="{Binding Source={StaticResource Themes}}"
SelectedIndex="18" />
```

```
<Grid Grid.Row="1" DataContext="{StaticResource viewmodel}">
<syncfusion:SfDataGrid Name="sfgrid" Margin="5"
AutoGenerateColumns="False"
AllowDraggingColumns="True"
AllowEditing="True"
LiveDataUpdateMode="AllowDataShaping"
AllowFiltering="True"
HeaderRowHeight="26"
SelectionMode="Extended"
ColumnSizer="Auto"
ItemsSource="{Binding EmployeeDetails}">
</syncfusion:SfDataGrid>
</Grid>
</Grid>
</syncfusion:ChromelessWindow>
```

**Note:** [View sample in GitHub.](#)

## Touch Support in Syncfusion WPF Controls

### Size mode

The [SfSkinManager](#) helps you to render in different size modes for both Syncfusion and Framework controls using the [SizeMode](#) property, which will be available only in [themes supported from theme studio](#).

The [SizeMode](#) enum contains the following values:

- **Default** - The default size will be updated for control and its inner elements.
- **Touch** - The touch interactive size mode will be applied for control. In this mode, more space is added around the inner elements, to make them easier to tap.

**Note:** The default value is `SizeMode.Default`.

### XML

```
<syncfusion:IntegerTextBox syncfusion:SfSkinManager.SizeMode="Touch">
</syncfusion:IntegerTextBox >
```

### Gesture

Gestures determine whether a finger or stylus has moved over a control or not. Syncfusion WPF controls support the following touch gestures: Tap, Swipe, Pinch, and Hold.<br/>.

The following table shows the gesture mappings for each control.

Category	Control	Swipe	Pinch	Tap	Hold
GRIDS	Cell Grid	-	-	-	-
	Data Grid	Selection, Scrolling, and Drag-Drop	-	Click and Select	Right-Click

	Tree Grid	Selection, Scrolling, and Drag-Drop	-	Click and Select	Right-Click
	Property Grid	-	-	Click	-
	Spreadsheet	Scrolling and Resizing	Zoom in & Zoom out	Click and Selection	Right-Click
DATA VISUALIZATION	Charts	Panning, Scrolling, Annotation resizing, dragging	Zooming	ToolTip, Series selection, Segment selection, Annotation text editing, Annotation selection	-
	Sunburst Chart	-	-	Click and Select	-
	Smith Chart	-	-	Click	-
	Kanban	Panning/Scrolling, Dragging	-	Column collapse/expand	Hold to select the card for dragging
	Diagramming	Diagram: Pan/Scroll, MultipleSelect, Drop, Zoom, Connect. Stencil: Drag & Drop. Node and Annotation: Drag, Resize, Rotate, Connect. Connector: Segment editing, Connect. Port: Drag, Connect.	Zoom	Select	Context Menu
	Maps	-	Pan	Click and Select	-
	Treemap	-	-	Click	-
	Scheduler	View swiping, Scrolling and Dragging	-	Tap support for the Cell selection, Appointment selection, and	By default, the cell context menu will be opened when you hold down any timeslot or month

				Double-tap support to the appointment editor.	cell. The appointment context menu will be opened by holding, only if the appointments' drag and drop is disabled using the <a href="#">AppointmentEditFlag</a> property.
	Linear Gauge	-	-	-	-
	Digital Gauge	-	-	-	-
	Radial Gauge	-	-	-	-
	Bullet Graph	-	-	Click	-
	Gantt	Selection, drag and drop	-	Click and Select	Select
	Range Navigator	Scroll	-	Range selection	-
	Sparkline	-	-	Click	-
	Surface Chart	-	-	Click	-
	Barcode	NA	NA	NA	NA
	Heat Map	Pan/Scroll	-	-	-
DATA SCIENCE	Predictive Analytics	NA	NA	NA	NA
LAYOUT	Ribbon	Swipe	-	Click	Context Menu
	Docking Manager	Swipe to rearrange Tab order in Tabbed and Document state windows.	-	Tap to select the active window of DockingManager. Double tap to float the active window	Hold any docking window header Context menu opened
	Document Container	Swipe to rearrange TDI mode Items	-	Tap to select the active document window	Hold any document window header Context menu opened
	Tile View	Swipe to reorder the tile items	-	Tap to select the item	-

	Carousel	To view the next or previous items	-	Click and select the item	-
	Tab Splitter	Swipe to move the splitter	-	Tap to select the item	Hold on TabBarSplitterItem to open the context menu
	Chromeless Window	Move	-	Select	-
	Card View	-	-	Click and Select	-
	Accordion	-	-	Tap to select the accordion item and expand or collapse item	-
	Grid Splitter	Swipe to move the splitter horizontally or vertically	-	-	-
EDITORS	Autocomplete	Scroll	-	Select	-
	Textbox	Move	-	double tap to select the word	Hold to open the default context menu
	Currency Textbox	Move	-	double tap to select the word	Hold to open the default context menu
	Double Textbox	Move	-	double tap to select the word	Hold to open the default context menu
	Integer Textbox	Move	-	double tap to select the word	Hold to open the default context menu
	Date Picker	Swipe to change the dates	-	click to select the date	-
	Time Picker	Swipe to change the dates	-	click to select the date	-
	Date Time Editor	Swipe	-	Select	-
	Button Adv	-	-	Click	-
	Masked Textbox	-	-	Click and Select	Context Menu



	Percent Textbox	-	-	double tap to select the word	Hold to open the default context menu
	RichTextBox	Swipe left/right to scroll the pages horizontally. Swipe up/down to scroll the pages vertically	Zoom in/out	Click and select	-
	Timespan Editor	-	-	-	-
	Numeric Updown	Selection	-	Tap up down button to change the values	Hold to open the default context menu
	Domain Updown	-	-	Tap up down button to change the values	-
	Combobox	Swipe	-	Click and Select	-
	Range Slider	Swipe horizontally or vertically to change the minimum and maximum range values	-	Tap to increase or decrease the values based on the step value.	-<
	Radial Slider	Swipe to change the slider positions	-	Tap to update the selected value	-
	Color Picker	-	-	Click	-
	Color Palette	Swipe	-	Select	-
	Color Picker Palette	Swipe	-	Click	-
	Calculator	-	-	Tap to enter the values in text area	-
	Rating	Swipe to change the rating values	-	Tap to increase or decrease the rating values	-

	Checked Listbox	Scrolling	-	Tap to select the item	-
	Dropdown Button	-	-	Dropdown open	-
	Split Button	Click	-	Select	-
	Multicolumn Drop-Down	Scrolling	-	Click and Select	Right Click
	Calendar	Swipe to change the month view	-	Tap to select the date	-
	Syntax Editor	-	-	tap to set the cursor index. double-tab to select the current word of the cursor-placed index	Hold to open Context menu
NAVIGATION	Tree View	Swipe	-	Click	Context Menu, Drag-drop
	Menu	-	-	Click	-
	Toolbar	-	-	Click	ToolTip
	Hierarchical Navigator	-	-	Tab to navigate the next item	-
	Groupbar	-	-	Tap to select the groupbar item. Expand or collapse the item.	Hold to open the Context menu
	Tabs	Change the tab index positions	-	Tap to select the item	Hold to open the Context menu
	Tab Navigation	Scroll	-	Tap to navigate the next or previous view	-
	Taskbar	-	-	Tap to expand or collapse the task bar item	-
	Radial Menu	Swipe to rotate the radial menu items	-	Tap to expand or collapse the radial menu item	Indicates the selected item

	Tree Navigator	-	-	Tap to navigate the item	-
	Navigation Drawer	To open the primary or secondary drawers	-	-	-
NOTIFICATION	Busy Indicator	-	-	-	-
	Notify Icon	-	-	Tap to close the notification icon pop-up	-
	Hub Tile	-	-	Tap to perform the scale animation based on tile pressed direction	-
FILE FORMAT LIBRARY	Excel	NA	NA	NA	NA
	PDF	NA	NA	NA	NA
	Word	NA(Non UI Component)	NA(Non UI Component)	NA(Non UI Component)	NA(Non UI Component)
	PDF Viewer	Scroll	Zoom	Button click	-
	PowerPoint	NA	NA	NA	NA
REPORTING	Report Viewer	Scrolling	-	Click and Select	-
	Report Writer	NA (Non UI component)	NA (Non UI component)	NA (Non UI component)	NA (Non UI component)
	Report Designer	Scrolling, Resizing	-	Click and select	Right click
BUSINESS INTELLIGENCE	Olap Grid	Drag and drop, expand and collapse, and resize	-	Click, hyperlink, and selection	Right click and tooltip
	Olap Chart	Drag and drop, expand and collapse, and resize	Zooming and panning	-	Right-click, tooltip, and context menu

	Olap Client	Drag and drop, expand and collapse, and resize	Zooming and panning	Click, hyperlink, and selection	Right-click, tooltip, and context menu
	Olap Gauge	-	-	-	Tooltip
	Pivot Grid	Drag and drop, expand and collapse, and resize	-	Click, hyperlink, and selection	Right-click, context menu, and tooltip
MISCELLANEOUS	SpellChecker	-	-	Tap to perform the spell check operations in spell check dialog window	Hold to open the suggestion context menu
	Wizard	-	-	Click	-
	Calculate	NA (Non UI component)	NA (Non UI component)	NA (Non UI component)	NA (Non UI component)
	Diagram(classic)	Diagram: Pan/Scroll, MultipleSelect, Drop, Zoom, Connect. Stencil: Drag & Drop. Node and Annotation: Drag, Resize, Rotate, Connect Connector: Segment editing, Connect Port: Drag, Connect	Zoom	Select	Context Menu
	GridTreeControl	Scrolling, Drag and drop, and resize	-	Click and Selection	Right click
	Spreadsheet(classic)	-	-	-	-
	GridDataControl	Scrolling, Drag and drop, and resize	-	Click and Selection	Right click

	Chart(classic)	Panning, Scrolling, Dragging	Zooming	ToolTip, Series selection, Segment selection	-
	TabControlExt	Swipe to move the tabitem	-	Tab to change the SelectedItem	Hold to open TabItemExt ContextMenu
	SfMaskEdit	Select	-	Tap to place the cursor	Hold to open the context menu
	ColorEdit	Select the color	-	Tap to select the color, Tap on ColorCode Editor box to place the cursor	Hold on ColorCode Editor to open the ContextMenu
	SfCircularGauge	-	-	-	-
	PinnableListBox	-	-	Click and selection	Right click

### Getting Started with Windows 10 Compact ScrollBar (Touch ScrollBars)

The [SfSkinManager](#) allows you to apply various scrollbar styles like Windows 10 compact scrollbar, for both Syncfusion and Framework controls using the [ScrollBarMode](#) property, which will be available only in [themes supported by theme studio](#).

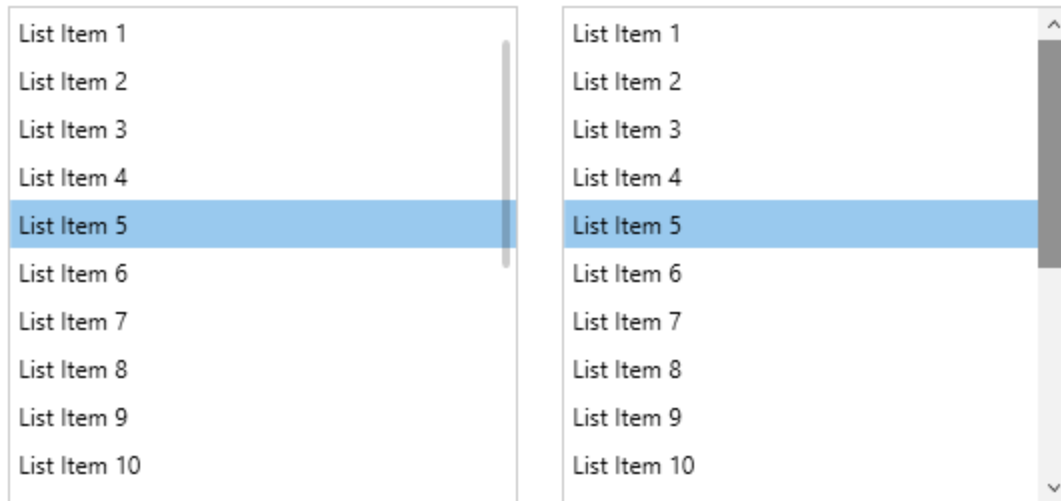
The [ScrollBarMode](#) enum contains the following values:

- **Default** - The ScrollBar will be displayed with the default look of a thumb along with up and down arrow to scroll.
- **Compact** - The Windows 10 like compact scrollbar will have the look of a thin layer of thumb until the user hovers over the scrollbar area.

**Note:** The **Compact** scrollbar mode is enabled by default in [WPF Fluent Theme](#).

### C#

```
public partial class MainWindow : ChromelessWindow
{
    public MainWindow()
    {
        SfSkinManager.SetTheme(this, new Theme() { ThemeName = "MaterialDark",
        ScrollBarMode = ScrollBarMode.Compact });
        InitializeComponent();
    }
}
```



### DataGrid

Name	QS1	QS2	QS3
Maciej Dusza	7	5	18
Shelley Dyck	9	7	20
Linda Ecoffey	11	9	3
Carla Eldridge	13	11	5
Carol Elliott	5	13	7
Shannon Elliott	7	15	9
Jauna Elson	9	17	11
Michael Emanuel	11	19	13
Terry Eminhizer	12	21	15

**Note:** [View sample in GitHub.](#)

### Getting Started with Keyboard Focus Visual for WPF Controls

The [WPF Skin Manager](#) allows you to apply various keyboard focus visual styles for both Syncfusion and Framework controls using the [FocusVisualKind](#) property.

The [FocusVisualKind](#) enum contains the following values:

- **Default** - The default keyboard focus visual style will be applied.
- **HighVisibility** - High visibility keyboard visual feedback is an effect that shows a border for focusable elements when the user moves the keyboard focus to that element.

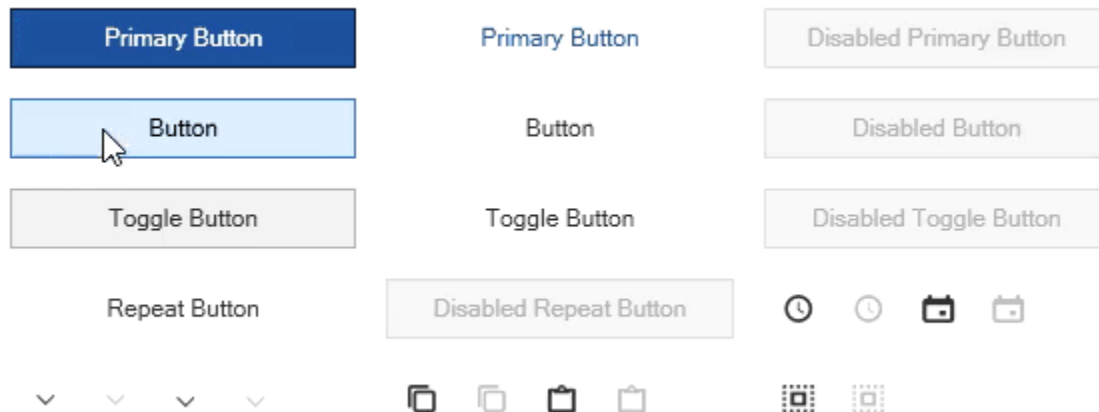
**Note:** The **HighVisibility** keyboard focus visual is enabled by default in the Fluent theme.

### C#

```
public partial class MainWindow : ChromelessWindow
```

```
{
public MainWindow()
{
SfSkinManager.SetTheme(this, new Theme() { ThemeName = "MaterialDark",
FocusVisualKind = FocusVisualKind.HighVisibility });
InitializeComponent();
}
}
```

### Buttons



**Note:** [View sample in GitHub.](#)

## Getting Started with WPF Theme Studio

The Theme Studio helps users to transform their visual presentation into a new theme in minutes. End-users can select an appropriate base theme, change its primary color, preview and export it as a theme project, and utilize it in their applications for a rich user interface experience.

### Supported themes

The WPF Theme Studio comes with the following set of themes:

- Fluent Light
- Fluent Dark
- Material Light
- Material Dark
- Material Light Blue
- Material Dark Blue
- Office 2019 Colorful
- Office 2019 Black
- Office 2019 White
- Office 2019 Dark Gray
- Office 2019 High Contrast
- Office 2019 High Contrast White
- System Theme

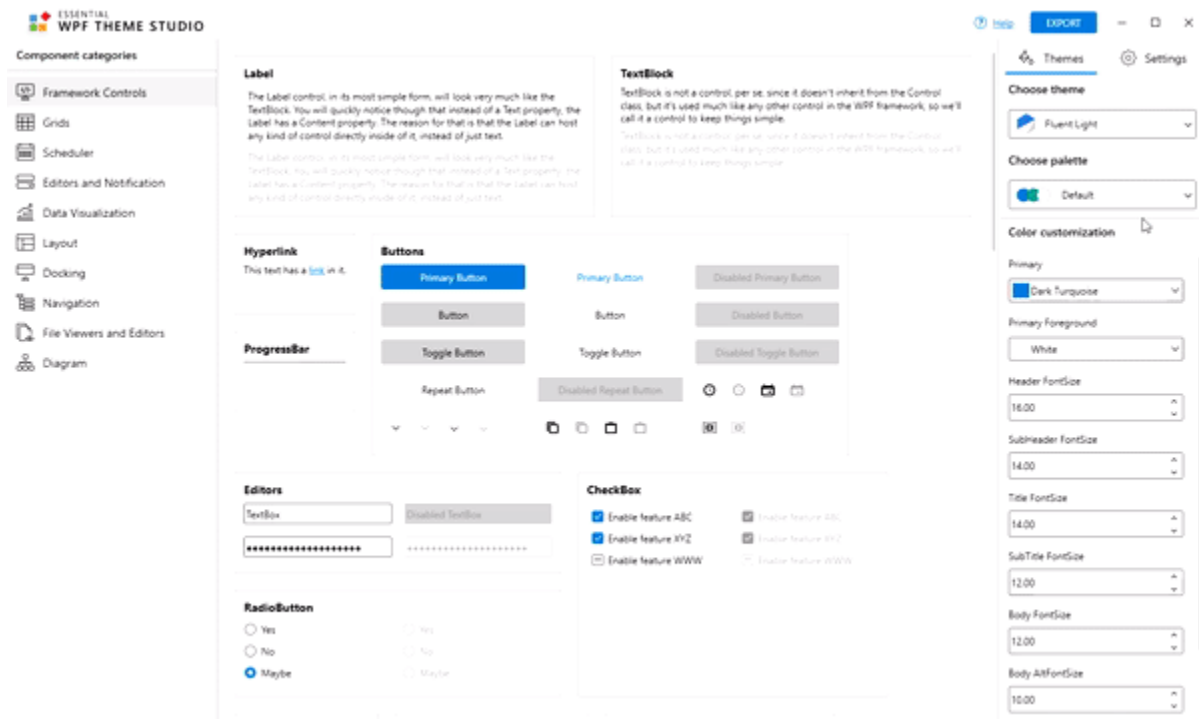
### Supported palettes

The WPF Theme Studio comes with various set of palettes for each theme variant.

Theme variant	Palette list	Supported themes
Fluent	<ul style="list-style-type: none"> <li>• PinkRed</li> <li>• Red</li> <li>• RedOrange</li> <li>• Orange</li> <li>• Green</li> <li>• GreenCyan</li> <li>• Cyan</li> <li>• CyanBlue</li> <li>• Blue</li> <li>• BlueMagenta</li> <li>• Magenta</li> <li>• MagentaPink</li> </ul>	<ul style="list-style-type: none"> <li>• FluentLight</li> <li>• FluentDark</li> </ul>
Office 2019	<ul style="list-style-type: none"> <li>• Red</li> <li>• Blue</li> <li>• Green</li> <li>• RedOrange</li> <li>• Violet</li> <li>• DeepGreen</li> <li>• DeepViolet</li> <li>• RedBrown</li> <li>• GreenCyan</li> <li>• DeepGreenCyan</li> <li>• LightViolet</li> <li>• MarinerBlue</li> <li>• DeepBlue</li> <li>• LightBlue</li> </ul>	<ul style="list-style-type: none"> <li>• Office2019Colorful</li> <li>• Office2019Black</li> <li>• Office2019White</li> <li>• Office2019DarkGray</li> </ul>
High Contrast	<ul style="list-style-type: none"> <li>• Red</li> <li>• Blue</li> <li>• Green</li> <li>• RedOrange</li> <li>• Violet</li> <li>• DeepGreen</li> <li>• DeepViolet</li> <li>• RedBrown</li> <li>• GreenCyan</li> <li>• DeepGreenCyan</li> <li>• LightViolet</li> <li>• MarinerBlue</li> <li>• DeepBlue</li> <li>• LightBlue</li> </ul>	<ul style="list-style-type: none"> <li>• Office2019HighContrast</li> <li>• Office2019HighContrastWhite</li> </ul>
Material	<ul style="list-style-type: none"> <li>• Red</li> <li>• Pink</li> </ul>	<ul style="list-style-type: none"> <li>• MaterialLight</li> <li>• MaterialDark</li> </ul>



	<ul style="list-style-type: none"> <li>Purple</li> <li>DeepPurple</li> <li>Indigo</li> <li>Blue</li> <li>LightBlue</li> <li>Cyan</li> <li>Green</li> <li>Orange</li> </ul>	<ul style="list-style-type: none"> <li>MaterialLightBlue</li> <li>MaterialDarkBlue</li> </ul>
--	--	---



## Creating custom theme

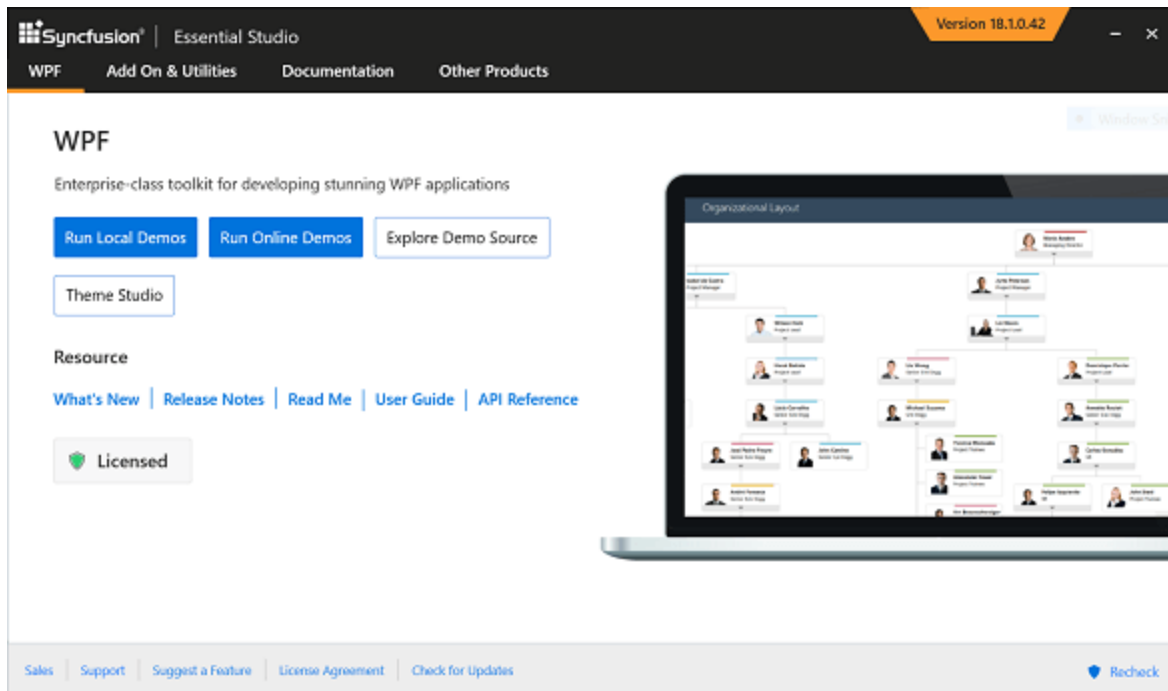
### *Customizing theme color from theme studio*

In theme studio utility, each theme has a unique common variable list. When user changes the common variable color code value, it will be reflected in all the Syncfusion WPF controls. All Syncfusion WPF control styles are derived from these theme-based common variables. This common variable list is handled inside the theme studio application for customizing theme-based colors.

Let us see the step-by-step procedure to launch and work with the theme studio utility as follows.

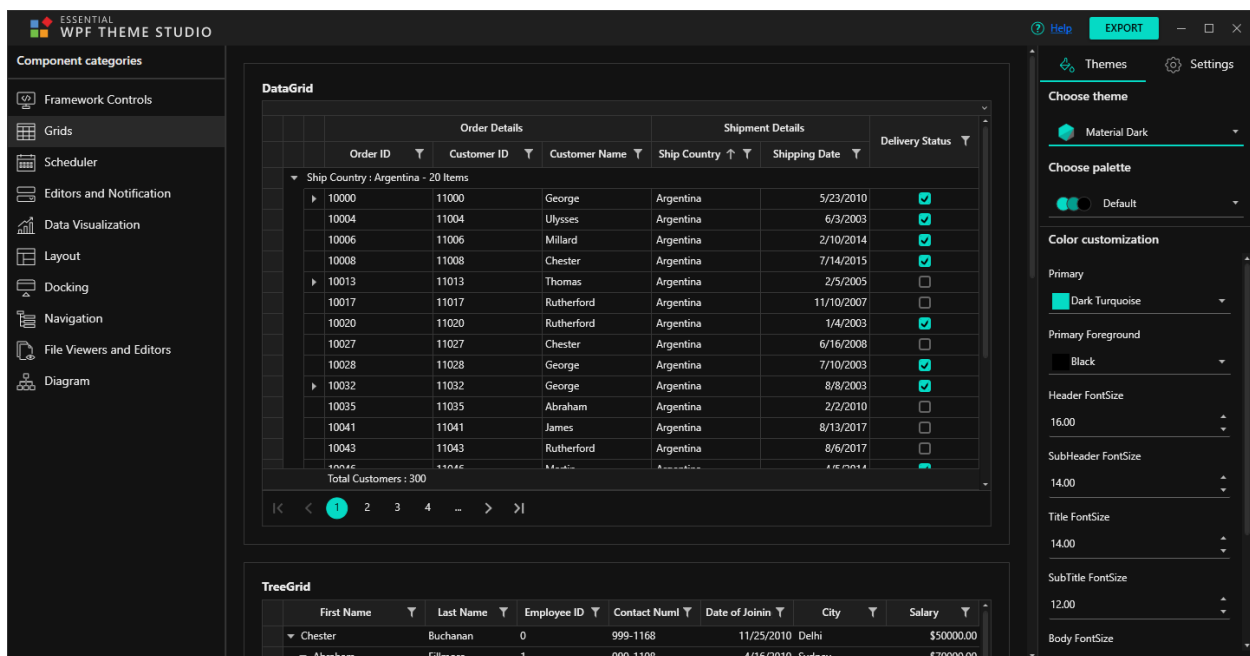
### Step 1

After installing the Syncfusion WPF suite, launch it and select Theme Studio from the start-up panel.



## Step 2

The theme studio application has been divided into two sections: the controls preview section on the left, and the theme customization section on the right.



## Step 3

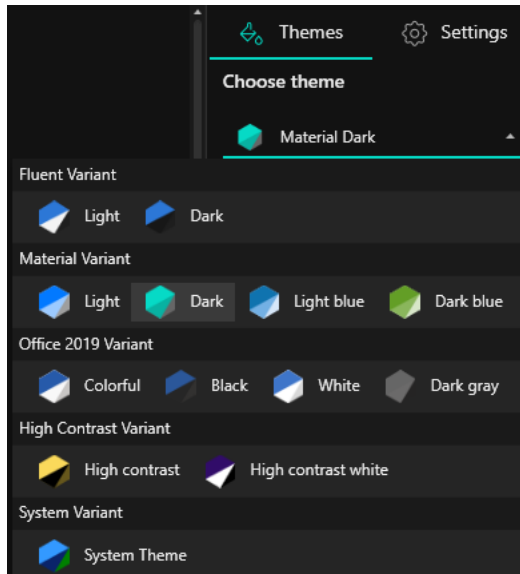
<<<<<<<< HEAD

To apply predefined themes, select the appropriate themes from Themes List Drop-down available in the top left corner.

=====

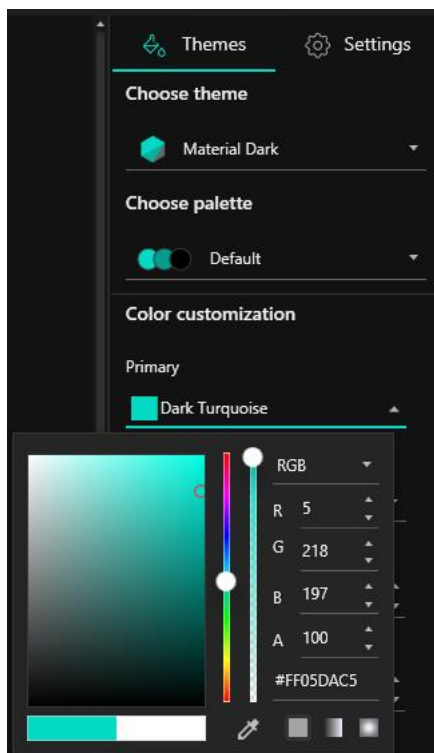
To apply predefined themes, select the appropriate themes from Themes List Dropdown.

>>>>> efb6053a56299c2bbbcef47fa9083b496356d1bf



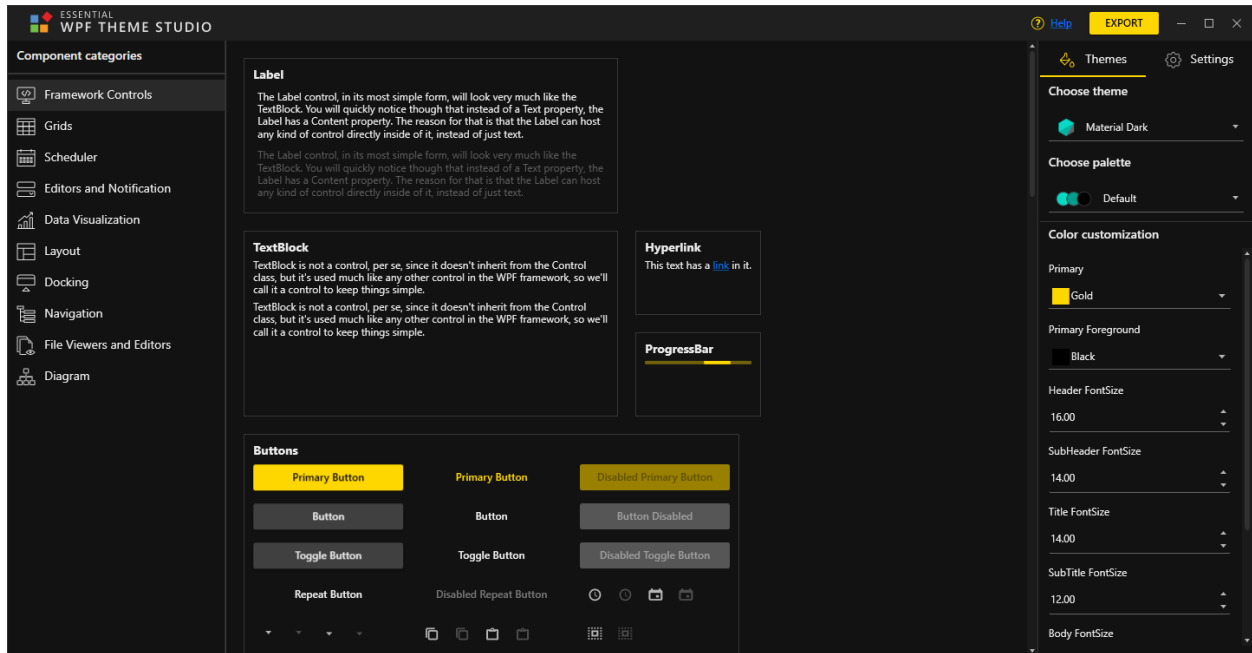
#### Step 4

Use the built-in **color picker** in the theme customization section to select the desired solid or gradient color.



#### Step 5

The Syncfusion WPF controls will be rendered with the newly selected colors in the preview section, after selecting the desired color.



**Note:** The WPF theme studio groups both Syncfusion and Framework controls under different tabs for a quick preview of the UI when the color changes.

### Exporting theme project

Let us see the step-by-step procedure for exporting theme project from theme studio.

#### Step 1

<<<<<<< HEAD

Click **Export** in the top right corner, below the exit of the theme studio application.

=====

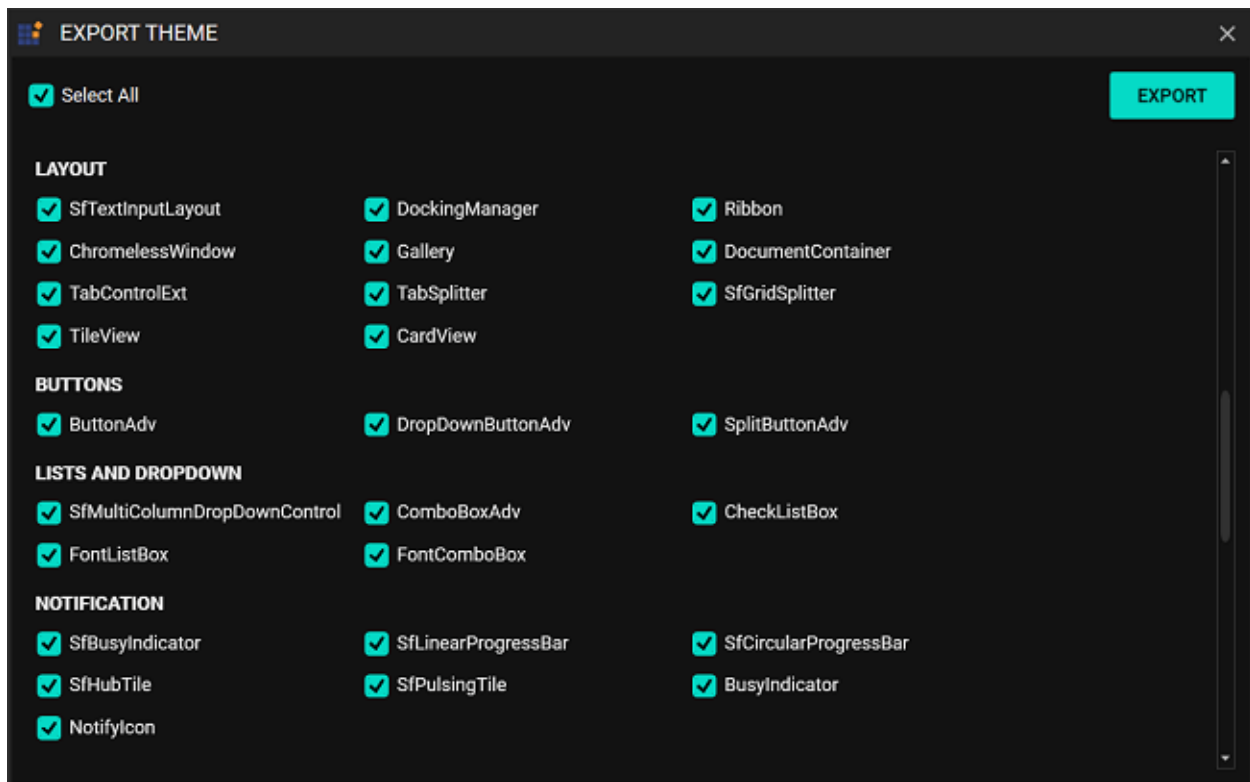
Click the **Export** button in the top right corner, beside the exit of the theme studio application.

>>>>>> efb6053a56299c2bbbcef47fa9083b496356d1bf



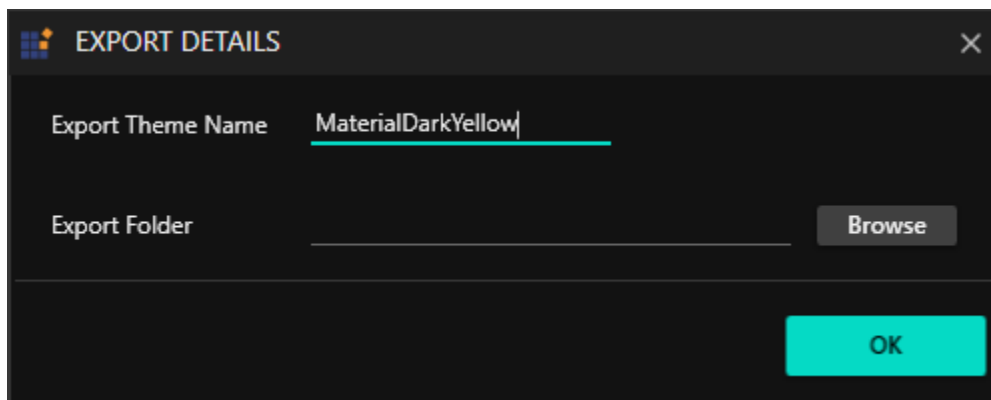
#### Step 2

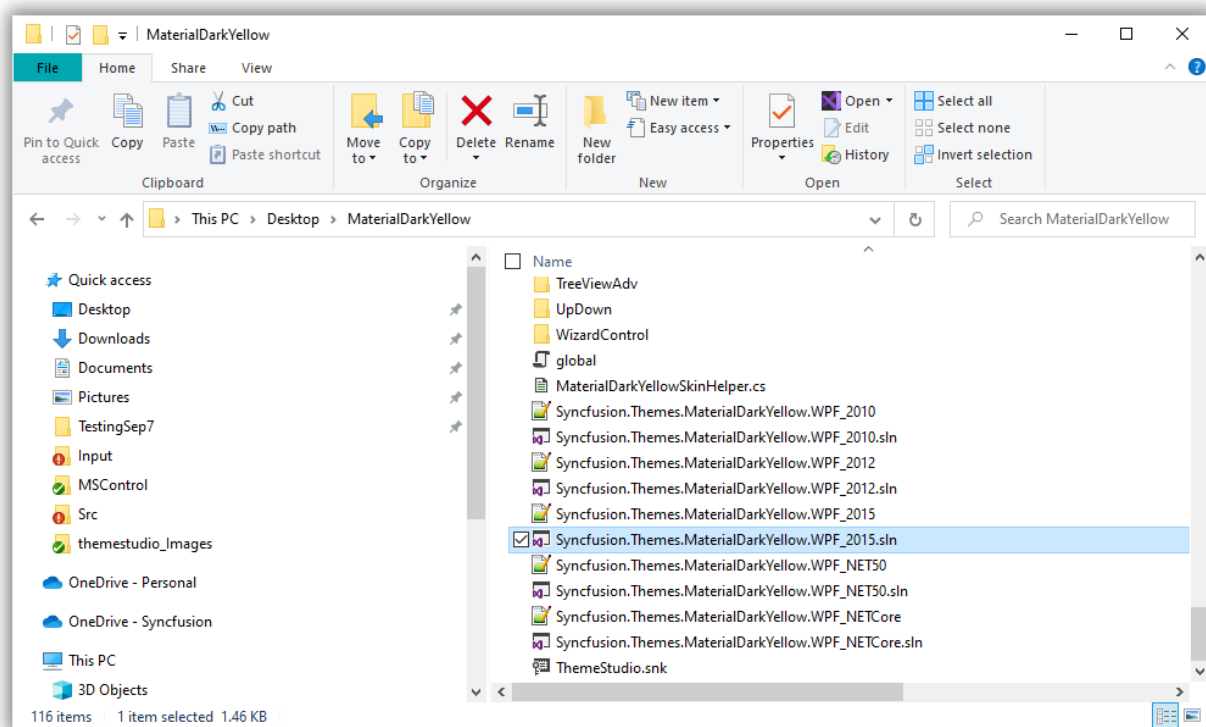
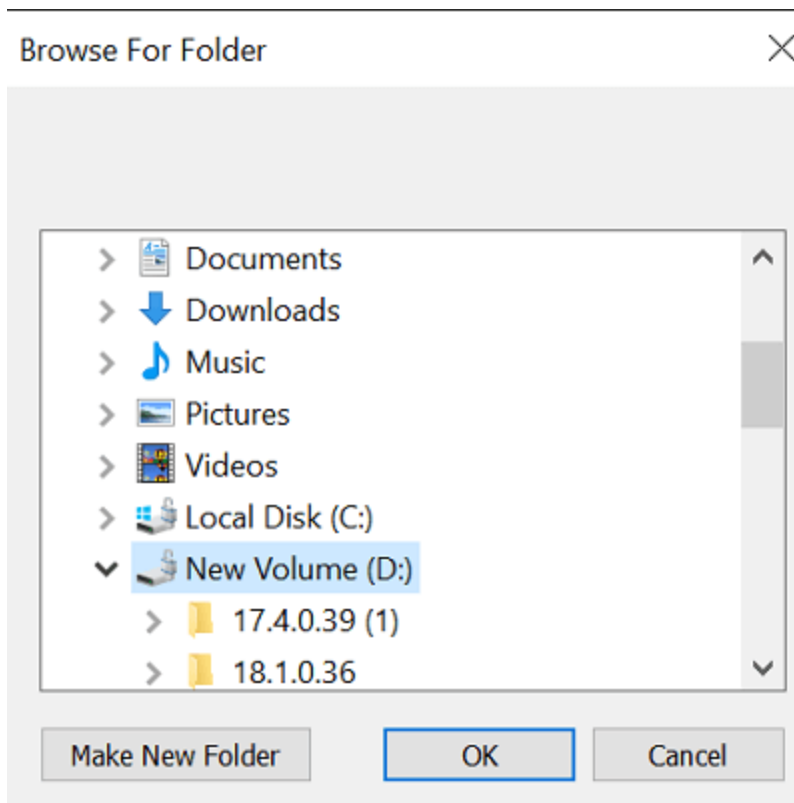
Now, the Export dialog box appears with the option to select either the entire controls or just the desired control(s). This option is useful only for selective list of Syncfusion WPF controls in the application. The theme studio will filter only the selected controls and customize the final output for those controls alone.



### Step 3

Provide the theme name, in which the theme should be exported and select the required folder for Theme Export to be selected. When you export the download theme, it will come as a theme project with color codes for the selected Syncfusion WPF controls.





### Generating theme assembly

Let us see the step-by-step procedure for ensuring theme assembly generation for exported theme project.

## Step 1

The following exported theme project should be attached for corresponding target frameworks used in the WPF application.

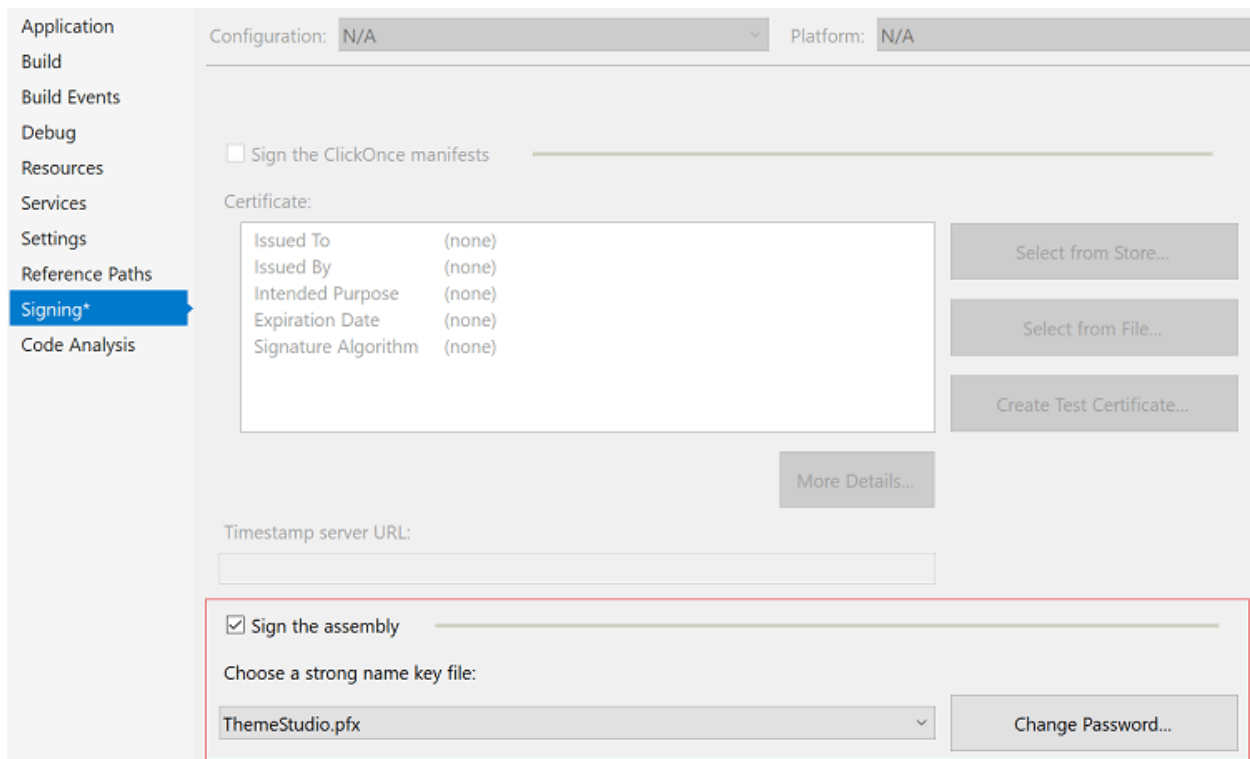
Target Framework	Solution Project
.Net 5.0	Syncfusion.Themes.MaterialDarkYellow.WPF_NET50.csproj
.Net Framework 4.6	Syncfusion.Themes.MaterialDarkYellow.WPF_2015.csproj
.Net Framework 4.5.1	Syncfusion.Themes.MaterialDarkYellow.WPF_2013.csproj
.Net Framework 4.5	Syncfusion.Themes.MaterialDarkYellow.WPF_2012.csproj
.Net Framework 4.0	Syncfusion.Themes.MaterialDarkYellow.WPF_2010.csproj
.Net Core 3	Syncfusion.Themes.MaterialDarkYellow.WPF_NETCore.csproj

## Step 2

The exported theme project should be rebuild in **Release** mode to generate theme assembly.

The export theme project has default **ThemeStudio.snk** key pair. If it is not required, use the already created private key pair by referring to the export theme project inside the application properties or [Create a new key pair](#) using Visual Studio if the private key pair was not created externally.

Ensure whether the **Sign the assembly** checkbox is clicked or not, to use the private key pair for generating theme assembly.



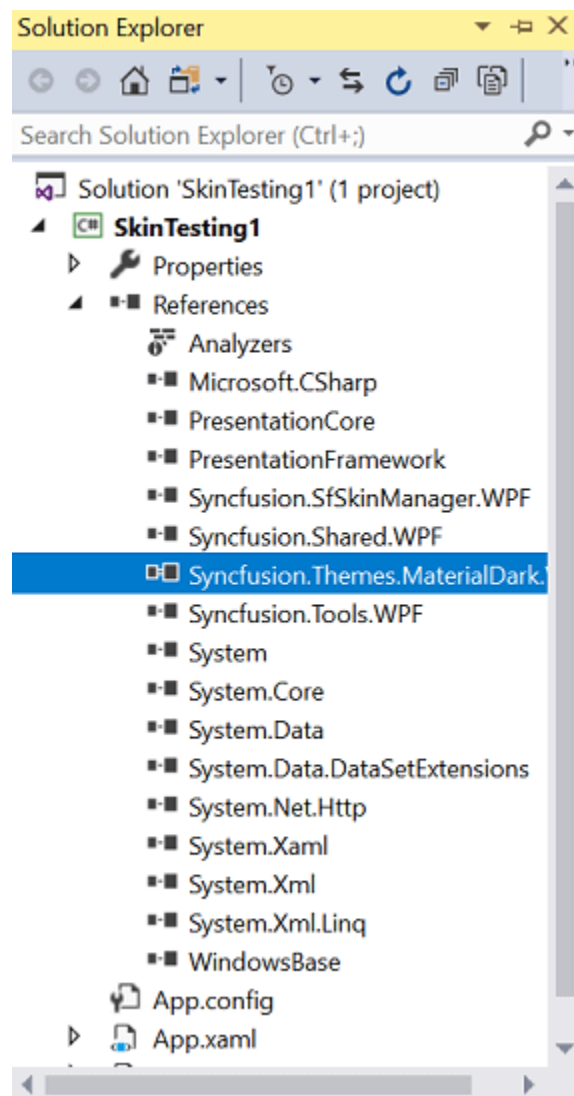
### *Integrating custom theme to the application*

The [SfSkinManager](#) control helps to apply the built-in themes to the Syncfusion UI controls for WPF.

Let us see the step-by-step procedure for adding exported theme project as assembly(.dll) and witness the custom theme set for Docking Manager.

#### Step 1

Now, add the exported theme project as an assembly (.dll) from the **Release** folder of the export theme project into the WPF application.



#### Step 2

Add reference of **Syncfusion.SfSkinManager.Wpf.dll** to the WPF application and import **SfSkinManager** namespace in Main window.

#### XML

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```



```
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"/>
```

### Step 3

The appearance of any Syncfusion UI controls for WPF can be customized by using the [Theme](#) attached property of the [SfSkinManager](#).

Now, use the [RegisterTheme](#) method to register the instance of generated MaterialDarkYellow (Syncfusion.Themes.MaterialDarkYellow.WPF) assembly from the exported theme project for demonstration purposes, passing the exported custom theme name and respective theme assembly instance as parameters.

### C#

```
string style = "MaterialDarkYellow";
SkinHelper styleInstance = null;
var skinHelperStr = "Syncfusion.Themes." + style + ".WPF." + style +
"SkinHelper, Syncfusion.Themes." + style + ".WPF";
Type skinHelperType = Type.GetType(skinHelperStr);
if (skinHelperType != null)
styleInstance = Activator.CreateInstance(skinHelperType) as SkinHelper;
if (styleInstance != null)
{
SfSkinManager.RegisterTheme("MaterialDarkYellow", styleInstance);
}
```

### Step 4

Now, set the [SfSkinManager](#) attached property [Theme](#) as [MaterialDarkYellow;MaterialDark](#) for the [DockingManager](#) control since the [MaterialDarkYellow](#) (Syncfusion.Themes.MaterialDarkYellow.WPF) assembly has been generated from the exported theme project for demonstration purposes.

**Note:** Custom theme name is provided in this format: [CustomTheme1;BaseThemeName](#), where [CustomTheme1](#) denotes the custom theme name and [BaseThemeName](#) denotes the theme name from which it is derived. For example, [MaterialDarkYellow;MaterialDark](#).

### XML

```
<syncfusion:DockingManager x:Name="SyncDockingManager"
UseDocumentContainer="True"
PersistState="True">
<ContentControl x:Name="SolutionExplorer"
syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.SideInDockedMode="Right"/>
<ContentControl x:Name="ToolBox" syncfusion:DockingManager.Header="Toolbox"
syncfusion:DockingManager.State="AutoHidden" />
<ContentControl x:Name="Output" syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="SolutionExplorer"/>
<ContentControl x:Name="EndPage" syncfusion:DockingManager.Header="End Page"
syncfusion:DockingManager.State="Document" >
```

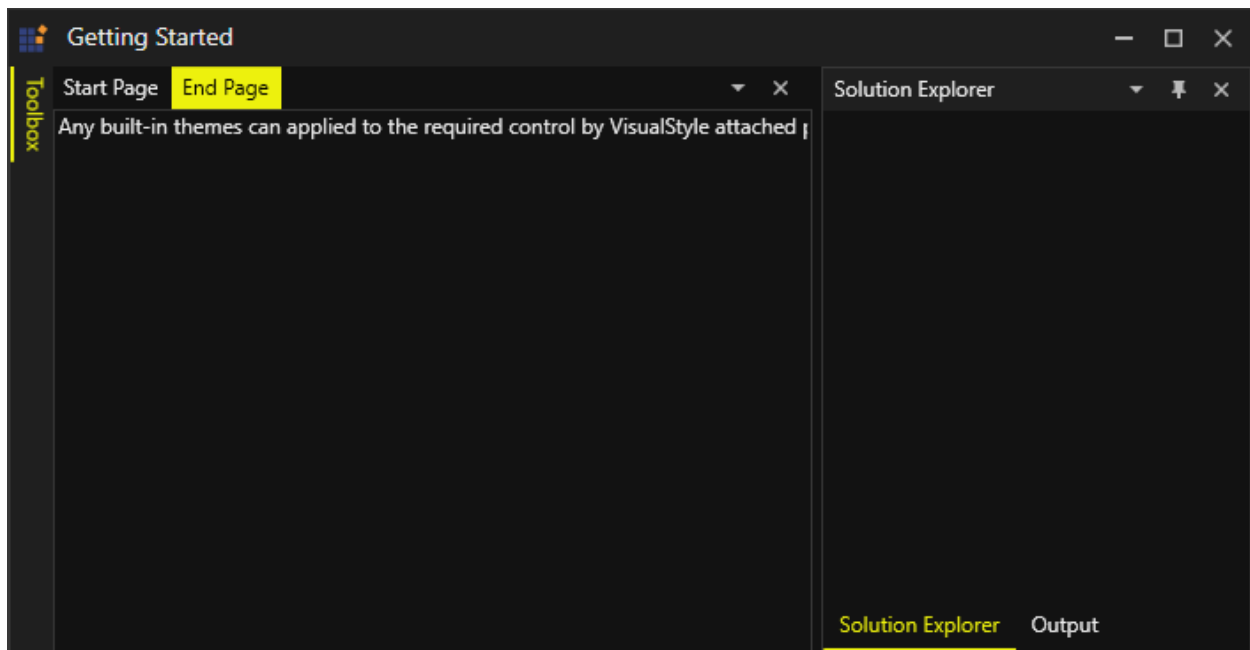
```
<TextBlock Text="Any built-in themes can applied to the required control by
VisualStyle attached property of the SfSkinManager." />
</ContentControl>
<ContentControl x:Name="StartPage" syncfusion:DockingManager.Header="Start
Page"
syncfusion:DockingManager.State="Document" >
<TextBlock Text="Any built-in themes can applied to the required control by
VisualStyle attached property of the SfSkinManager." />
</ContentControl>
</syncfusion:DockingManager>
```

## C#

```
string style = "MaterialDarkYellow";
SkinHelper styleInstance = null;
var skinHelperStr = "Syncfusion.Themes." + style + ".WPF." + style +
"SkinHelper, Syncfusion.Themes." + style + ".WPF";
Type skinHelperType = Type.GetType(skinHelperStr);
if (skinHelperType != null)
styleInstance = Activator.CreateInstance(skinHelperType) as SkinHelper;
if (styleInstance != null)
{
SfSkinManager.RegisterTheme("MaterialDarkYellow", styleInstance);
}
SfSkinManager.SetTheme(this, new Theme("MaterialDarkYellow;MaterialDark"));
```

## Step 4

Compile and run the WPF application and witness the custom theme being applied to Docking Manager control at run-time.



## Testing

### Coded UI Testing in WPF

Automated tests that drive your application through its user interface (UI) are known as Coded UI Tests (CUITs). These tests include functional testing of the UI controls. Our controls support CUITs Coded UI automation that helps you to create automated tests for inner elements and records the sequence of actions. When dragging the crosshair shown in Coded UI Test Builder on UI elements, it shows the properties of the respective UI elements. You can also add assertion for each of the properties.

Provided here are the three levels of support in Coded UI Test automation.

Levels	Description
Level 1	Record and detect the UI elements when performing actions in the control.
Level 2	Provide custom properties for UI elements when dragging the crosshair to any UI element.
Level 3	Coded UI Test Builder generates code from the recorded session and custom class is implemented to access custom properties, so the generated code is simplified.

### Prerequisites and compatibility

#### Prerequisites

The prerequisites are tabulated in the following table.

.Net Framework	4.5,4.5.1,4.6
Other Requirement	Extension dll and Syncfusion build installed dll should be referred as same version.

#### Compatibility

Visual Studio	Coded UI provides support only in Visual Studio 2010, 2012 and 2013 Ultimate and Premium platforms for Visual Studio 2015 and 2017 supported only in Enterprise platform. For more information about the platforms and configurations supported by Coded UI tests, refer to this <a href="#">link</a> .
---------------	---

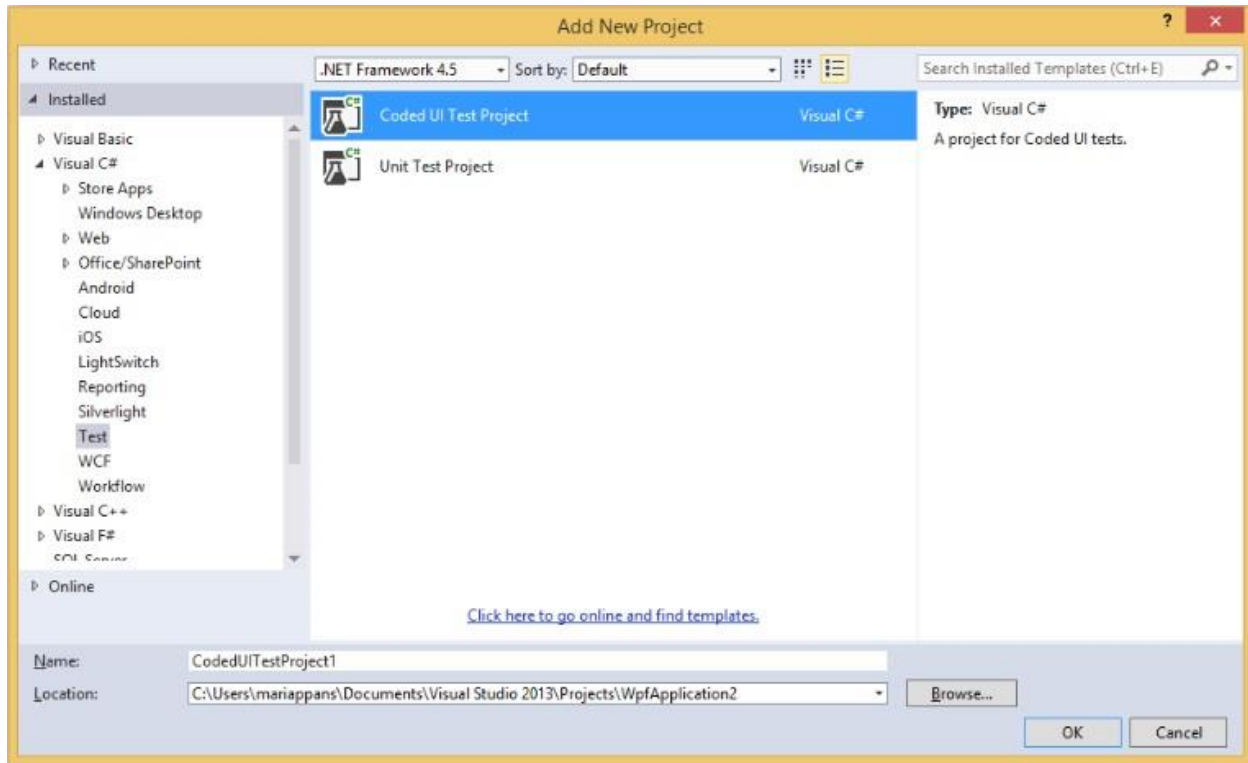
### Create, record, and run the tests

#### Creating Coded UI Project

CUIT contains the Coded UI test project. When application does not contain the CUIT project, create a new project. In the Solutions Explorer, on the shortcut menu of the solution is as follows:

- Choose Add New Project.
- Select either Visual Basic or Visual C#.
- Right-click the test project, choose Add, and select Coded UI Test.

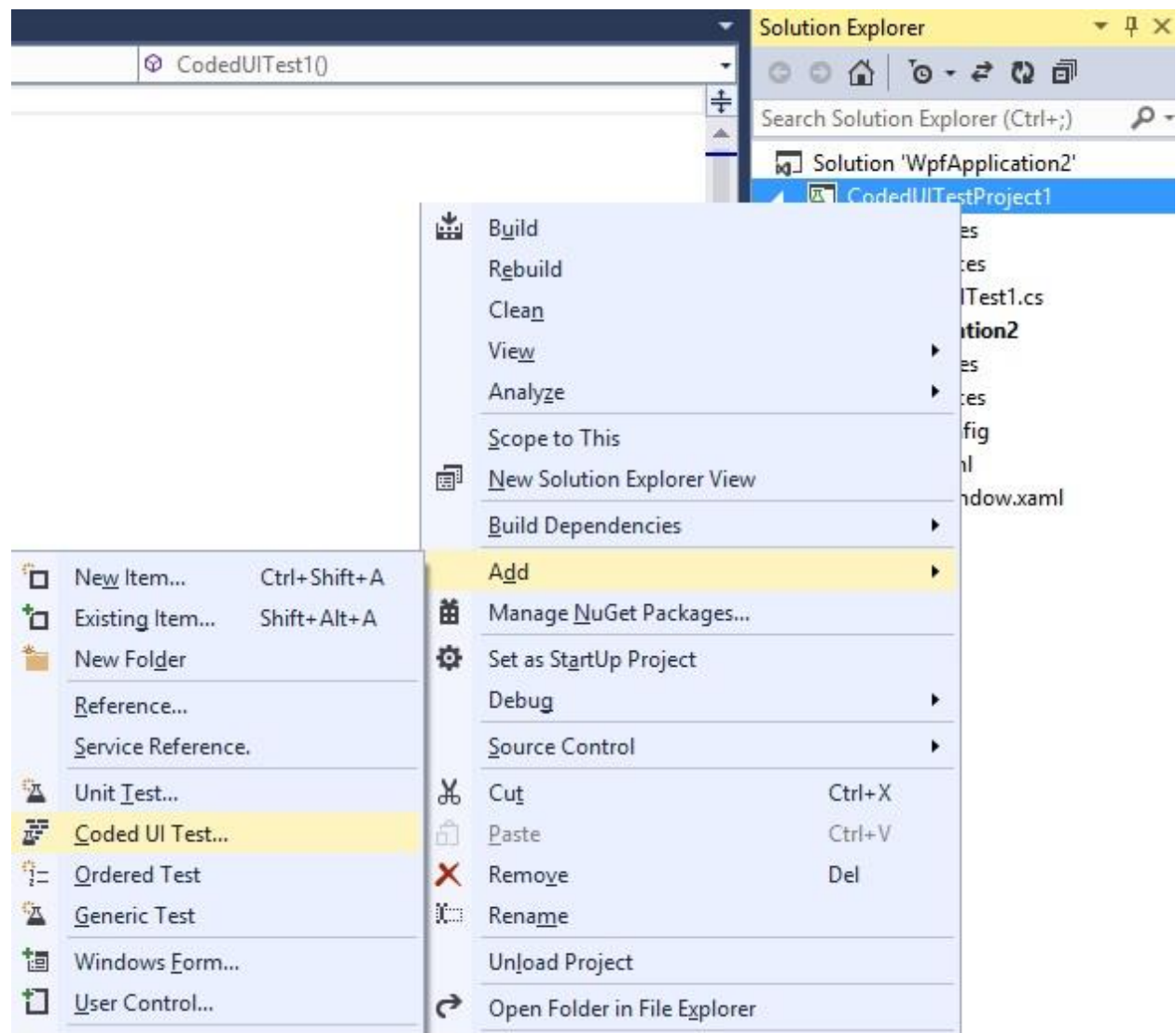
Coded UI test project does not appear on Visual Studio, if Visual Studio Enterprise edition does not installed in the system. To create CUIT, install Visual Studio Enterprise edition in system.



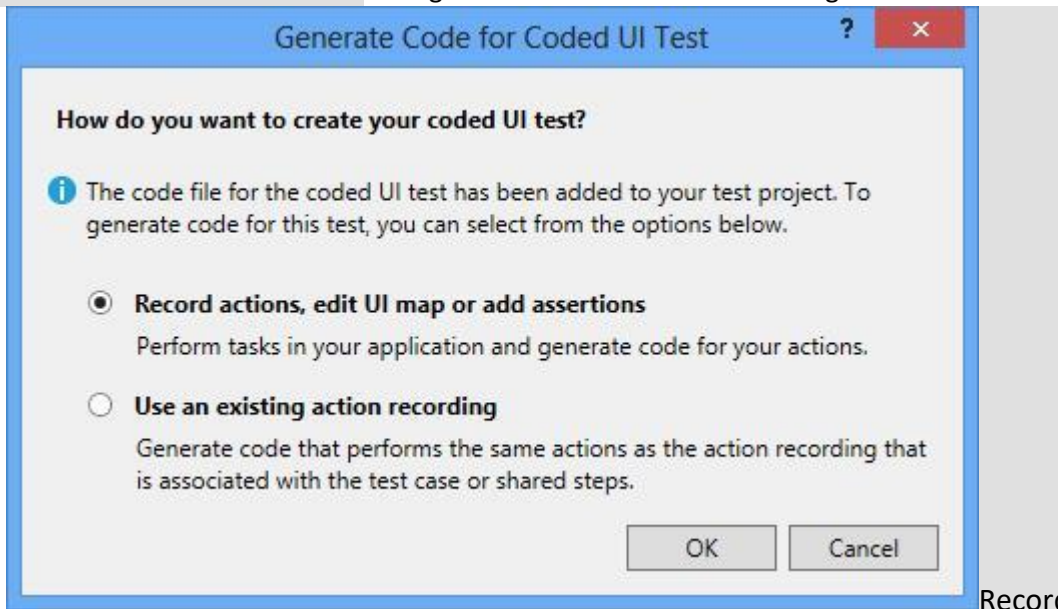
#### *Adding Coded UI Test file*

To add the CUIT file, follow the steps:

1. Create the CUIT project, the CUIT file will be automatically generated. To add another test file, right-click the test project, choose Add, and select Coded UI Test.

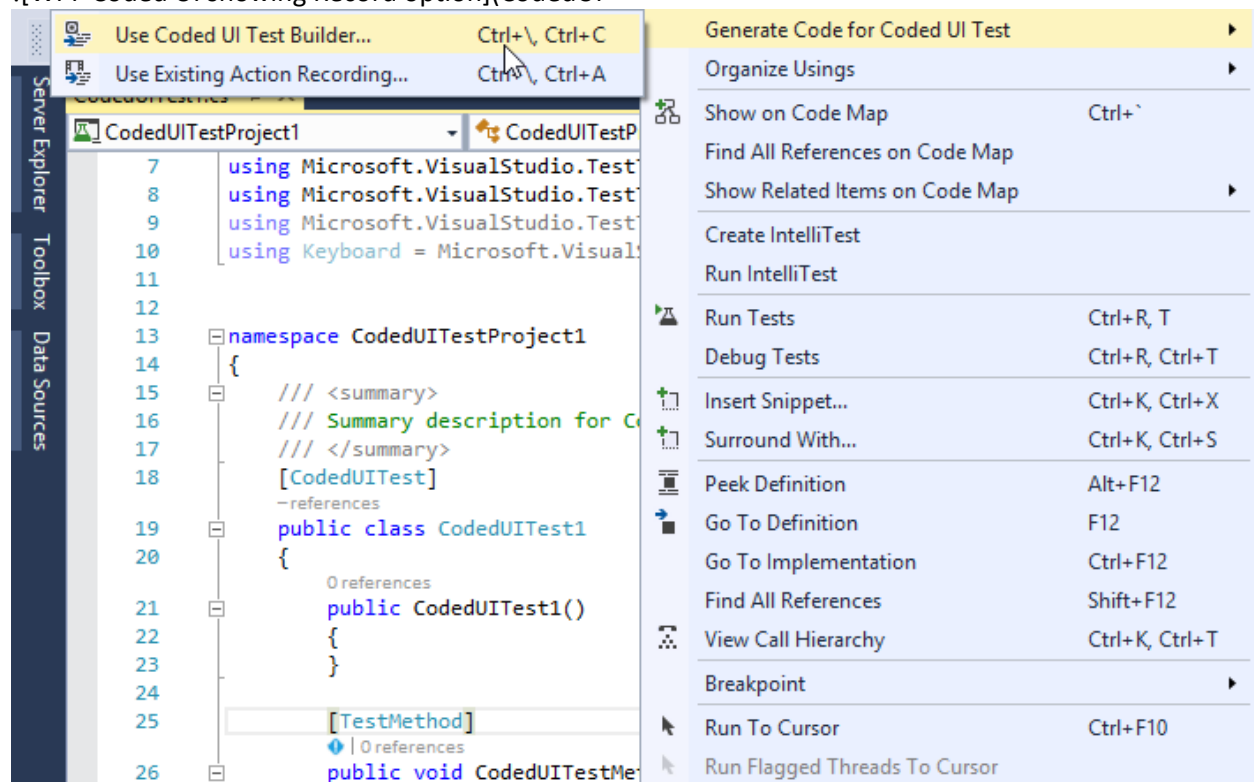


2. Generate Code for Coded UI Test dialog box will be shown after selecting the Coded UI test file.



Choose **Record** to record the test actions.

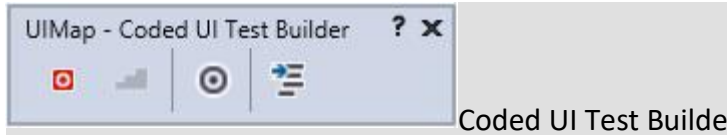
![WPF Coded UI showing Record option](CodedUIimg3.jpeg)



images/CodedUIimg3.jpeg)

3. Then, right-click the body of the Coded UI Test file.

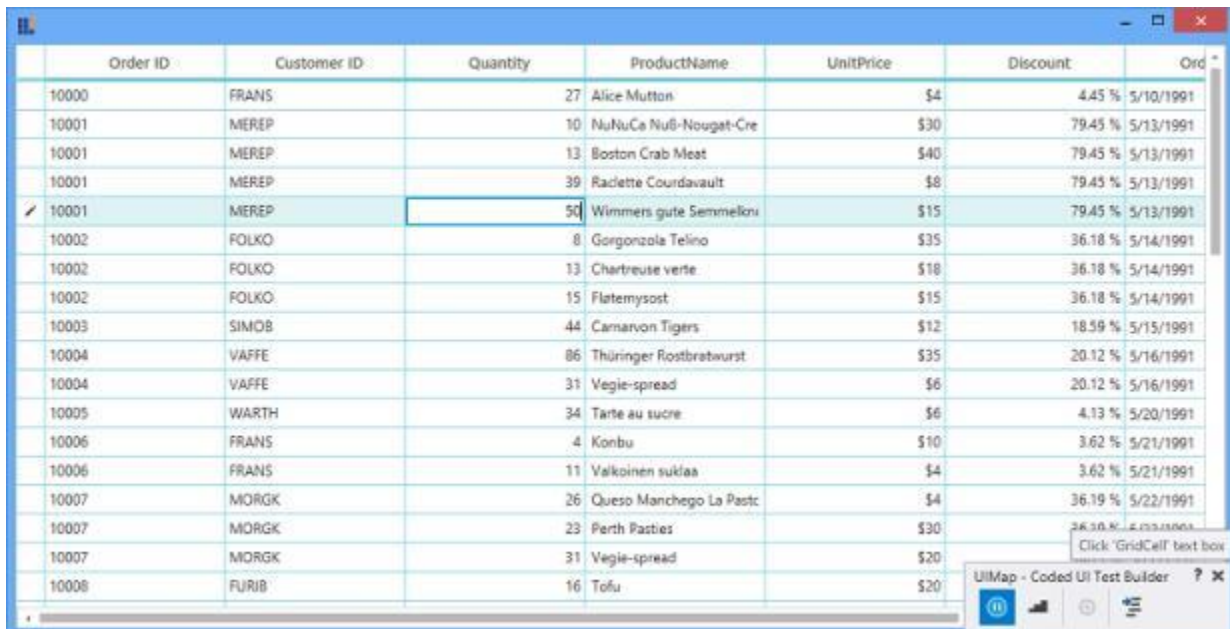
![WPF Coded UI showing test file content](CodedUIimg4.png)



4. Choose Coded UI Test Builder (UI Map) at the bottom right side of the window.

![WPF Coded UI showing UI Map](CodedUIimages/CodedUIimg5.jpeg)

5. Start the recording option from UI Map to record the test actions and recording option can be paused when testing the project. 6. Run the application, on which you will be performing UI testing. 7. Drag the crosshairs on to the UI elements of your WPF grid control application. It shows the available properties of the inner UI elements in the grid controls. 8. The actions made on UI elements can be recorded by clicking **Record** on the Coded UI Test builder. For example, you can record the action of changing the cell value in SfDataGrid. Click



Pause to finish the record.

![WPF Coded UI showing pause option](CodedUIimages/Featuresimg1.jpeg)

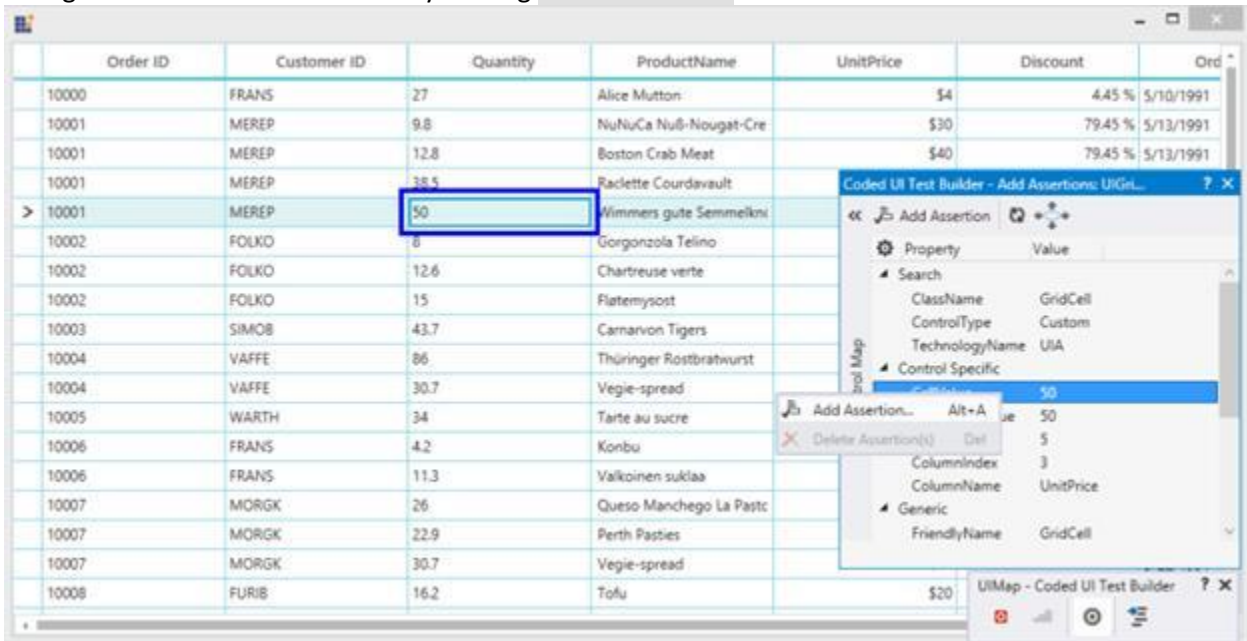
9. Then, click the **Generate Code** icon in Coded UI Test Builder to generate a test method. Close the Coded UI Test Builder. You can see the generated code for the cell value changed action.

**C#**

```
public void RecordedMethod1 ()
{
    #region Variable Declarations
    WpfText uIMEREPTText =
    this.UIWpfWindow.UISfDataGridCustom.UIGridCellCustom4.UIMEREPTText;
    WpfEdit uIGridCellEdit = this.UIWpfWindow.UISfDataGridCustom.UIGridCellEdit;
    WpfSfGridCell uIGridCellCustom11 =
    this.UIWpfWindow.UISfDataGridCustom.UIGridCellCustom11;
    WpfSfGridCell uIGridCellCustom12 =
    this.UIWpfWindow.UISfDataGridCustom.UIGridCellCustom12;
    #endregion
}
```



10. An assertion can also be created to check the modified cell value. Drag the crosshair to the modified cell, the Assertion window appears. The properties for the control (Cell) will be listed in the Assertion dialog box. You can add assertion by clicking **Generate Code** in



Coded UI Test Builder.

![WPF Coded UI showing Add Assertion](CodedUIimages/Featuresimg2.jpeg)

### Testing recorded steps

Application can be tested with the generated CUIT method. Follow the procedure to test the recorded steps:

1. Add a test method called CodedUITestProject1.

### C#

```
public void CodedUITestMethod1 ()
{
    // Generates code for this test. Select "Generate Code for Coded UI Test"
    // from the shortcut menu and select one of the menu items.
    this.UIMap.RecordedMethod1 ();
}
```

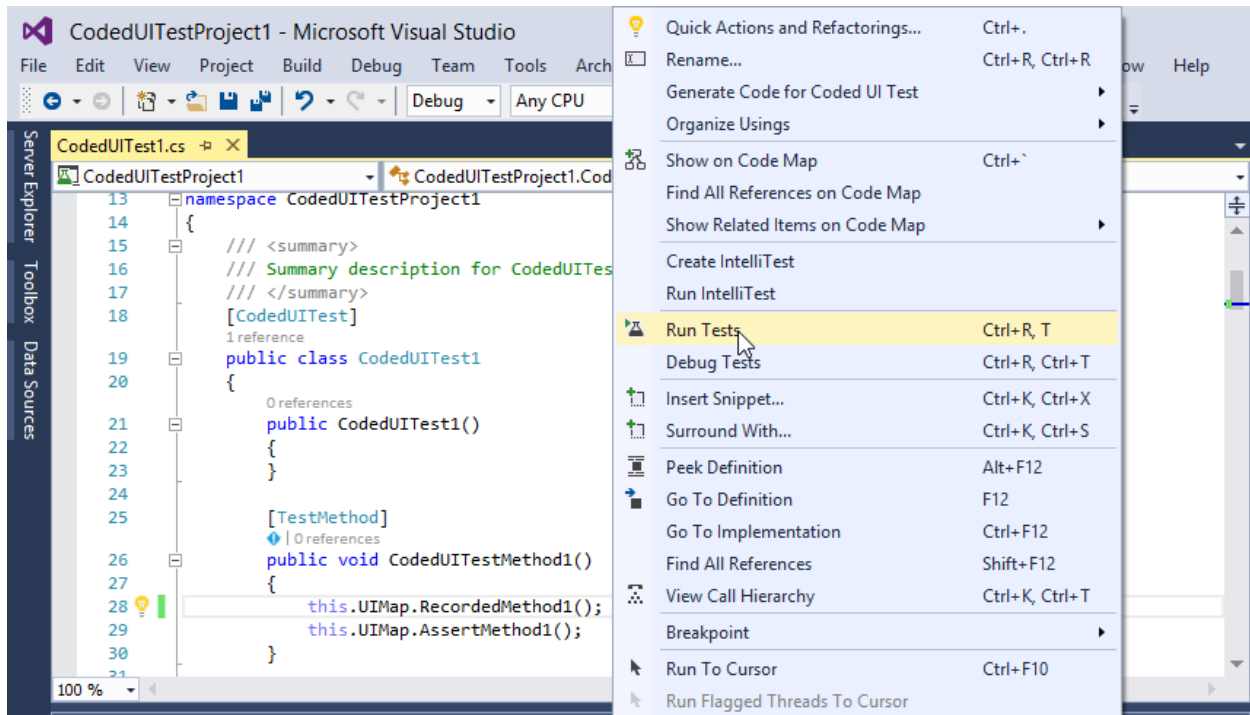
### VB.NET

```
Public Sub CodedUITestMethod1 ()
    ' Generates code for this test. Select "Generate Code for Coded UI Test"
    ' from the shortcut menu and select one of the menu items.
    Me.UIMap.RecordedMethod2 ()
End Sub
```

2. Build and run the application that has already been configured. CUIT builder can be opened with the following steps. 3. Right-click the body of the **CodedUITestMethod** and



select



Run Tests.

![WPF Coded UI showing Run Tests](CodedUIimages/CodedUIRunTests.png)

4. Before running tests, ensure that the test application is launched and the initial states of controls are same as at the beginning of the test recording.

#### Supported controls

The following controls are featured in Coded UI testing support.

Supported Controls	Levels
DataGrid	3 Levels
GridControl	3 Levels
SfDataPager	3 Levels
TreeGrid	3 Levels
ComboBoxAdv	2 Levels
DockingManager	2 Levels
DocumentContainer	2 Levels
DateTimeEdit	2 Levels

GridData	2 Levels
GridTree	2 Levels
PivotGrid	2 Levels
PropertyGrid	2 Levels
Ribbon	2 Levels
SfChart	2 Levels
SfMultiColumnDropDown	2 Levels
TabControlExt	2 Levels
TileView	2 Levels
TreeView	2 Levels

**Note:** The three level supported control can only have Coded UI Test extension project.

#### Coded UI Test extension support

The Coded UI Test(CUIT) extension is a technology manager that tells Coded UI Test to use UI automation for Syncfusion WPF controls.

The following controls have extension project.

- SfDataGrid
- SfTreeGrid
- GridControl

To test the Coded UI supported controls with CUITs, build the Extension project and place in the following mentioned location.

Controls	Compile assemblies	Adding Extension assembly	Extension Project
SfDataGrid	SfGrid.WPF	Syncfusion.VisualStudio.TestTools.UITest.SfGridExtension.dll	Get the Extension project from [this]( <a href="https://github.com/SyncfusionExamples/coded-ui-test-external-plugin-source-project-for-wpf-datagrid-and-treegrid">https://github.com/SyncfusionExamples/coded-ui-test-external-plugin-source-project-for-wpf-datagrid-and-treegrid</a> ) location.
SfTreeGrid	SfGrid.WPF	Syncfusion.VisualStudio.TestTools.UITest.SfGridExtension.dll	Get the Extension project from <code>{{'this' markdownify}}</code> location.

GridControl	Grid.WPF	Syncfusion.VisualStudio.TestTools.UITest.GridExtension.dll	Get the Extension project from <a href="#">this</a> location.
-------------	----------	--	---

To run the CUITs, follow the steps:

1. Run the Extension project and build it.
2. You can get the following tabulated assembly from the bin folder.

The above assembly must be placed in the following directory based on your Visual Studio version.

Visual Studio Version	Relative Path
2010	C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\10.0\UITestExtensionPackages
2012	C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\11.0\UITestExtensionPackages
2013	C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\13.0\UITestExtensionPackages
2015	C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\14.0\UITestExtensionPackages
2017	C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\15.0\UITestExtensionPackages

**Note:** The Extension package should be installed(For example, **Syncfusion.VisualStudio.TestTools.UITest.SfGridExtension.dll**) in GAC location. Refer to the MSDN link for [GAC](#) installation.

#### *UI Elements for supported controls*

The following UI Elements for Coded UI supported controls,

Controls Name	UI Elements
ComboBoxAdv	ComboBoxAdv and ComboBoxItemAdv
TreeViewAdv	TreeViewAdv and TreeViewItemAdv
PropertyGrid	PropertyGrid, ClearButton, DescriptionBorder, DescriptionDetailsTextBlock, DescriptionNameTextBlock, PropertyCategoryViewItem, PropertyGridScrollViewer, PropertyGridSplitter, PropertyItemTextBlock, PropertyView, PropertyViewItem, SearchTextBox, ToggleSortButton, and ToggleGroupButton

Ribbon	Ribbon, RibbonTab, RibbonBar, RibbonButton, RibbonCheckBox, RibbonComboBox, RibbonComboBoxItem, RibbonGallery, RibbonGalleryPopup, RibbonGalleryItem, RibbonMenuGroup, RibbonMenuItem, RibbonMenuItem, DropDownButton, SplitButton, ApplicationMenu, MenuButton, SimpleMenuButton, SplitMenuButton, BackStage, BackStageCommandButton, BackStageTabItem, RibbonToggleButton, QATListBox, QuickAccessToolBar, and QuickAccessToolBarDropDown
SfMultiColumnDropDown	SfMultiColumnDropDownControl
SfDataPager	SfDataPager
SfDataGrid	GridStackedHeaderCellControl, GridTableSummaryCell, GridGroupSummaryCell, GridCaptionSummaryCell, GridDetailsViewExpanderCellControl, and GridIndentCell
SfTreeGrid	TreeGridStackedHeaderCell and TreeGridExpanderCell
PivotGrid	PivotGrid

### UI Elements and Properties

The following Coded UI supported controls have UI Elements and other supported properties apart from default properties are listed below.

Control Name	UI Elements	Properties
SfDataGrid	DataGrid	RowCount, ColumnCount, SelectionMode, SelectionUnit, SelectedIndex, and SelectedItemCount.
	GridCell	CellValue, FormattedValue, RowIndex, ColumnIndex, and ColumnName.
	GridHeaderCellControl	MappingName, IsFilterApplied, FilterIconVisibility, SortDirection, and SortNumberVisibility.
	GridRowHeaderCell	RowErrorMessage and RowIndexState
	GroupDropArea	IsExpanded and GroupDropAreaText
	DetailsViewDataGrid	RowCount, ColumnCount, SelectionMode, SelectionUnit, SelectedIndex, and SelectedItemCount.
	GroupDropAreaItem	GroupNameSortDirection
TreeGrid	TreeGrid	RowCount, ColumnCount, SelectionMode, SelectionUnit, SelectionIndex, and SelectedItemCount.
	TreeGridCell	CellValue, FormattedValue, RowIndex, ColumnIndex, and ColumnName.

	TreeGridHeaderCell	MappingName, SortDirection, and SortNumberVisibility.
	TreeGridRowHeaderCell	RowErrorMessage, RowIndex, and State.
GridControl	GridCell	Text, CellValue, ColumnHeader, Format, Description, FormulaTag, CellWidth, CellHeight, FormattedText.
	Grid	SelectedRanges, GridName, RowCount, ColumnCount.
SfMultiColumnDrop Down	SfMultiColumnDropDown Control	AllowAutoComplete, AllowNullInput, AllowImmediatePopup, AllowIncrementalFiltering, AllowCaseSensitiveFiltering, AllowSpinOnMouseWheel, DisplayMember, IsDropDownOpenSelectedIndex, ValueMember.
SfDataPager	SfDataPager	AccentBackground, AccentForeground, AutoEllipsisMode, AutoEllipsisText, DisplayMode, EnableGridPaging, NumericButtonCount, Orientation, PageCount, PageSize, UseOnDemandPaging.

## Steps to work with Coded UI

### *SfDataGrid*

Follow the steps to create a **Coded UI Test** project and test the SfDataGrid application.

1. Create a new WPF application or open an existing WPF application with SfDataGrid. Enable the **Coded UI Test** by setting AutomationPeerHelper.EnableCodedUI to true and access the AutomationPeerHelper class from Syncfusion.UI.Xaml.Grid namespace.

### **C#**

```
using Syncfusion.UI.Xaml.Grid;
public MainWindow()
{
    InitializeComponent();
    AutomationPeerHelper.EnableCodedUI = true;
}
```

2. Build the application and launch the .exe file from bin folder. 3. Create, record, and run the tests for Coded UI Test project by referring [Create, Record, and Run the tests](#) section.

### *Hand code for CUIT operation*

This session demonstrates the example codes to perform the operation with hand code.

### *Hand code for changing the cell value*

### **C#**

```
//Change Cell Value
[TestMethod]
public void CodedUITestMethod1()
{
    //this.UIMap.AssertMethod1();
}
```

```

var cell
= this.UIMap.UIMainWindowWindow.UISfDataGridCustom.UIVirtualizingCellsConCus
tom.UIGridCellCustom;
cell.DrawHighlight();
Mouse.DoubleClick(cell);
Keyboard.SendKeys("Raj");
Keyboard.SendKeys("{Enter}");
Task.Delay(10);
cell.DrawHighlight();
}

```

Hand code for find GridCell

**C#**

```

[TestMethod]
public void FindGridCell()
{
    WpfSfDataGrid dataGrid = this.UIMap.UIMainWindowWindow.UISfDataGridCustom;
    WpfSfGridCell cell = new WpfSfGridCell(dataGrid);
    cell.SearchProperties[WpfSfGridCell.PropertyNames.RowIndex] = "1";
    cell.SearchProperties[WpfSfGridCell.PropertyNames.ColumnIndex] = "1";
    Assert.IsTrue(cell.TryFind());
}

```

**Note:** In the previous code snippet, at least two unique properties of GridCell (row index and column index / row index and cell value / column index and cell value) should be provided to search in SfDataGrid.

Hand code for sort click operation

**C#**

```

[TestMethod]
public void Mouse_Click_on_GridHeaderCell()
{
    WpfSfDataGrid dataGrid = this.UIMap.UIMainWindowWindow.UISfDataGridCustom;
    WpfSfHeaderCellControl cell = new WpfSfHeaderCellControl(dataGrid);
    cell.SearchProperties[WpfSfHeaderCellControl.PropertyNames.ColumnName]
= "OrderID";
    if (cell.TryFind())
    {
        cell.DrawHighlight();
        Mouse.DoubleClick(cell);
    }
}

```

### *SfTreeGrid*

Follow the steps to create a **Coded UI Test** project and test the tree grid application.

1. Create a new WPF application or open an existing WPF application with tree grid. Enable Coded UI Test in tree grid by setting **AutomationPeerHelper.EnableCodedUI** to true and access the AutomationPeerHelper class from Syncfusion.UI.Xaml.Grid namespace.

**C#**

```

using Syncfusion.UI.Xaml.Grid;

```

```
public MainWindow()
{
    InitializeComponent();
    AutomationPeerHelper.EnableCodedUI = true;
}
```

2. Build the application and launch the .exe file from the bin folder. 3. Create, record and run the tests for Coded UI Test project by referring [Create, Record, and Run the tests](#) section.

Automation peer class name	Control name in code generation	Property provider class name
SfTreeGridAutomationPeer	WpfSfTreeGrid	SfTreeGridPropertyProvider
TreeGridCellAutomationPeer	WpfSfTreeGridCell	SfTreeGridCellPropertyProvider
TreeGridHeaderCellAutomationPeer	WpfSfTreeGridHeaderCell	SfTreeGridHeaderCellPropertyProvider
TreeGridRowHeaderCellAutomationPeer	WpfSfTreeGridRowHeaderCell	SfTreeGridRowHeaderCellPropertyProvider
TreeGridStackedHeaderCellAutomationPeer	WpfSfTreeGridStackedHeaderCell	SfTreeGridStackedHeaderCellPropertyProvider
TreeGridExpanderCellAutomationPeer	WpfTreeGridExpanderCell	SfTreeGridExpanderCellPropertyProvider

### GridControl

Follow the steps to prepare the grid application.

1. Syncfusion.VisualStudio.TestTools.UITest.GridExtensions.dll contains implementation to easily change an existing application to the test application, plugin is required. Add a reference to this assembly.
2. Open App.xaml file.

### XML

```
<Application x:Class="WpfApplication3.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
StartupUri="Window1.xaml">
<Application.Resources>
</Application.Resources>
</Application>
```

3. Change the application to Syncfusion:GridControlTestApplication.

### XML

```
<syncfusion:GridControlTestApplication x:Class="WpfApplication3.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
StartupUri="Window1.xaml">
```

```
<syncfusion:GridControlTestApplication.Resources>
</syncfusion:GridControlTestApplication.Resources>
</syncfusion:GridControlTestApplication>
```

4. For the code behind file (App.xaml.cs), make sure to inherit from **GridControlTestApplication**.

#### XML

```
namespace WpfApplication3
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : GridControlTestApplication
    {
    }
}
```

5. Build the application for testing. 6. Add the following reference in your sample.

*Syncfusion.VisualStudio.TestTools.UITest.GridCommunication.dll*

*Syncfusion.VisualStudio.TestTools.UITest.GridExtension.dll* 7. To create, record and run the tests for Coded UI Test project, you can refer this section [Create, Record, and Run the tests](#).

#### Hand code for CUIT operation

This session demonstrates the example codes to perform the operation with hand code.

#### Hand code to find GridCell

##### C#

```
[TestMethod]
public void FindGridCell()
{
    var datagrid
= this.UIMap.UISyncfusionGridDataCoWindow.UISyncfusionGridDataCoWindow1;
    UITestControl cell = new UITestControl(datagrid);
    cell.SearchProperties["ControlType"] = "Cell";
    cell.SearchProperties["RowIndex"] = "16";
    cell.SearchProperties["ColumnIndex"] = "3";
    Assert.IsTrue(cell.TryFind());
}
```

#### Hand code for changing the cell value

##### C#

```
[TestMethod]
public void CodedUITestMethod1()
{
    var datagrid
= this.UIMap.UISyncfusionGridDataCoWindow.UISyncfusionGridDataCoWindow1;
    var cell
= this.UIMap.UISyncfusionGridDataCoWindow.UISyncfusionGridDataCoWindow1.UITtem24Grid1Cell;
    cell.DrawHighlight();
    var rowindex = Convert.ToInt16(cell.GetProperty("RowIndex").ToString());
}
```



```

var columnIndex =
Convert.ToInt32(cell.GetProperty("ColumnIndex").ToString());
Playback.Wait(100);
GridCellInfo info = new GridCellInfo(rowindex, columnIndex, "Grid1");
Playback.Wait(50);
//click the cell for enter into edit mode.
Playback.Wait(50);
Keyboard.SendKeys("^ (A)"); //select all the text
Playback.Wait(150);
Keyboard.SendKeys("{DELETE}"); //Delete the previous value
Playback.Wait(150);
Keyboard.SendKeys("23"); //insert new value
Playback.Wait(50);
Keyboard.SendKeys("{TAB}"); //Commit value using tab key navigation
Playback.Wait(50);
var value = cell.GetProperty("CellValue");
Playback.Wait(200);
if (value.ToString() == "23")
{
Assert.IsTrue(true);
}
else
Assert.IsFalse(true);
}

```

## Overview in UFT Testing

UFT (formerly known as HP Quick Test Professional - QTP) is an automated testing software designed for testing various software applications and environments. Syncfusion provides QTP add-in that contains custom libraries, that help UFT or QTP to recognize Syncfusion controls. These custom libraries are built with the help of .NET add-in extensibility.

### Prerequisites and Compatibility

#### Prerequisites

The prerequisites are tabulated in the following table.

Testing Environments	QuickTest Professional of version 9.5 and above or UFTQuickTest Professional .NET add-in or UFT
.NET Framework	.NET Framework of version 3.5 , 4.0, 4.5 , 4.5.1 or 4.6
Other Requirement	Essential Studio (User Interface edition “ WPF) of the same version as the Essential QTP Add-on.

#### Compatibility

Operating Systems	Microsoft Windows XPMicrosoft Windows VistaMicrosoft Windows 7Microsoft Windows 8.1Microsoft Windows 10
-------------------	---

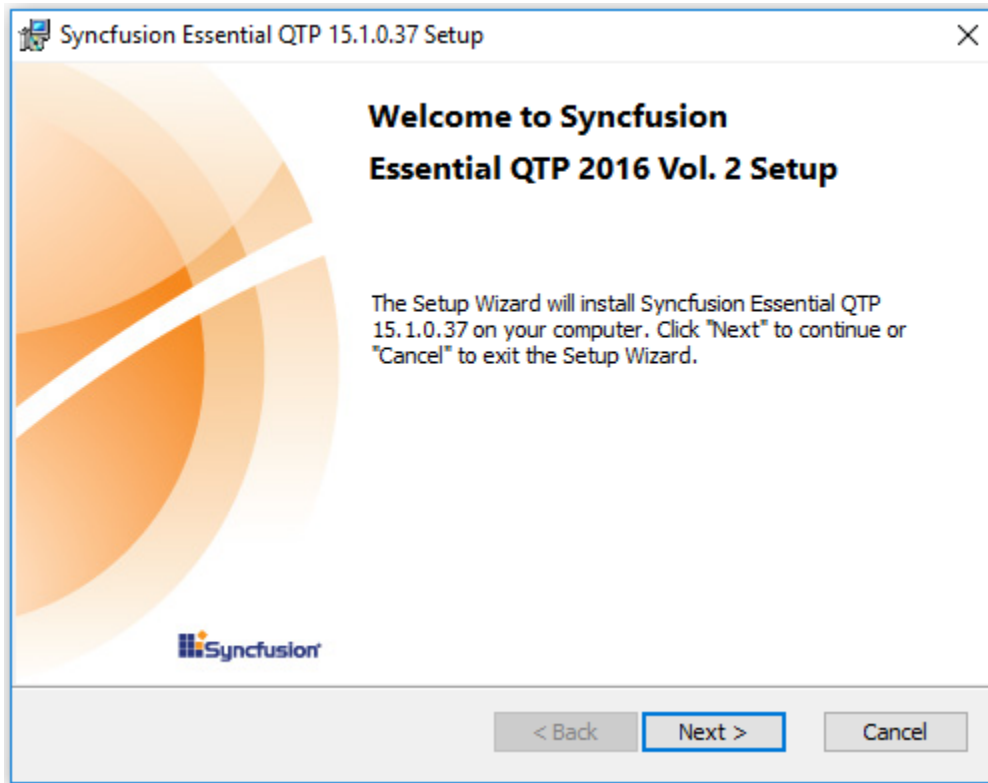
### Installation

You can download the QTP Essential Test Studio and install using the below steps,

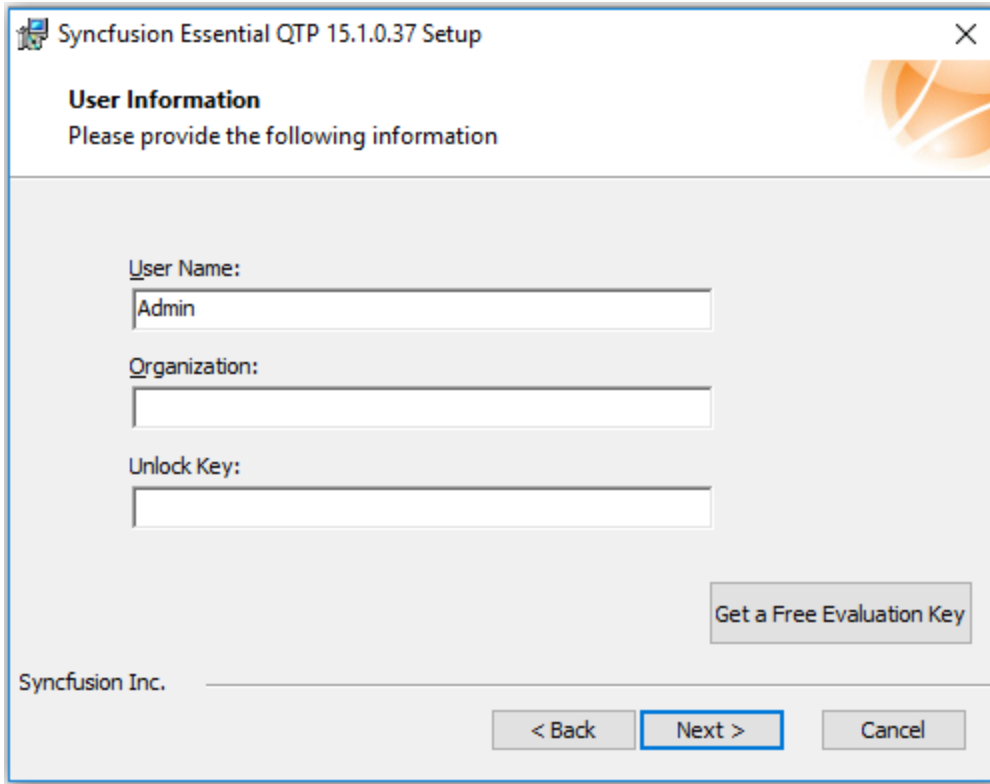
- 1.Double-click the Syncfusion Essential Test Studio Setup file.

**Note:**

Setup - Syncfusion Essential QuickTest Professional dialog box opens.



2.Click Next. The User Information dialog box opens.



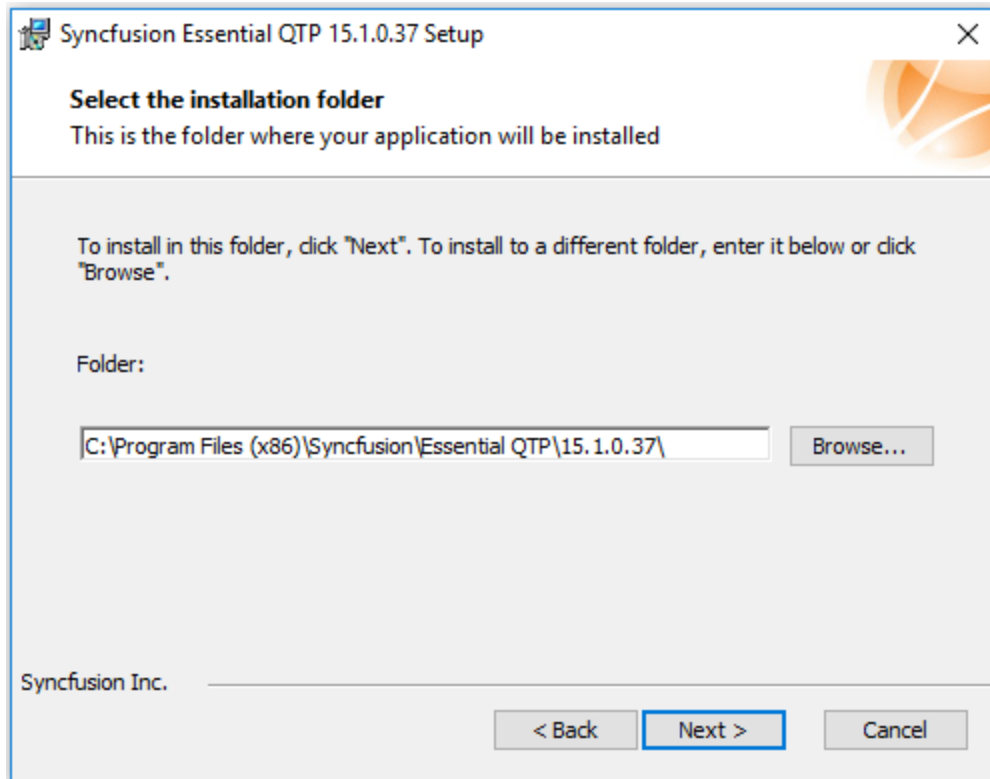
The screenshot shows a Windows-style dialog box titled "Syncfusion Essential QTP 15.1.0.37 Setup". The dialog has a standard title bar with a close button (X) in the top right corner. Below the title bar, the text "User Information" is displayed in bold, followed by the instruction "Please provide the following information". The main area of the dialog contains three text input fields: "User Name:" with the text "Admin" entered, "Organization:" which is empty, and "Unlock Key:" which is also empty. To the right of these fields is a button labeled "Get a Free Evaluation Key". At the bottom left, the text "Syncfusion Inc." is visible. At the bottom right, there are three buttons: "< Back", "Next >" (which is highlighted with a blue border), and "Cancel".

3. Enter the User Name, Organization and Unlock Key in the corresponding text boxes provided. 4. Click Next.

**Note:**

The unlock key is validated.

5. Select the installation folder dialog box opens.

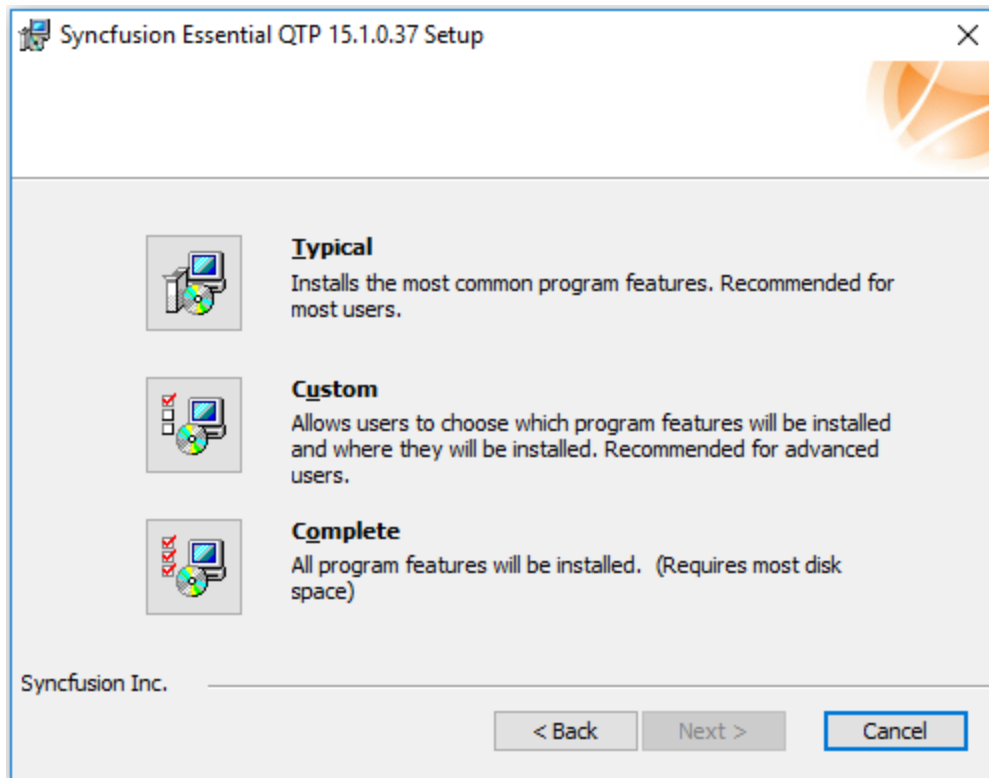


6.To install in the default location, click Next.

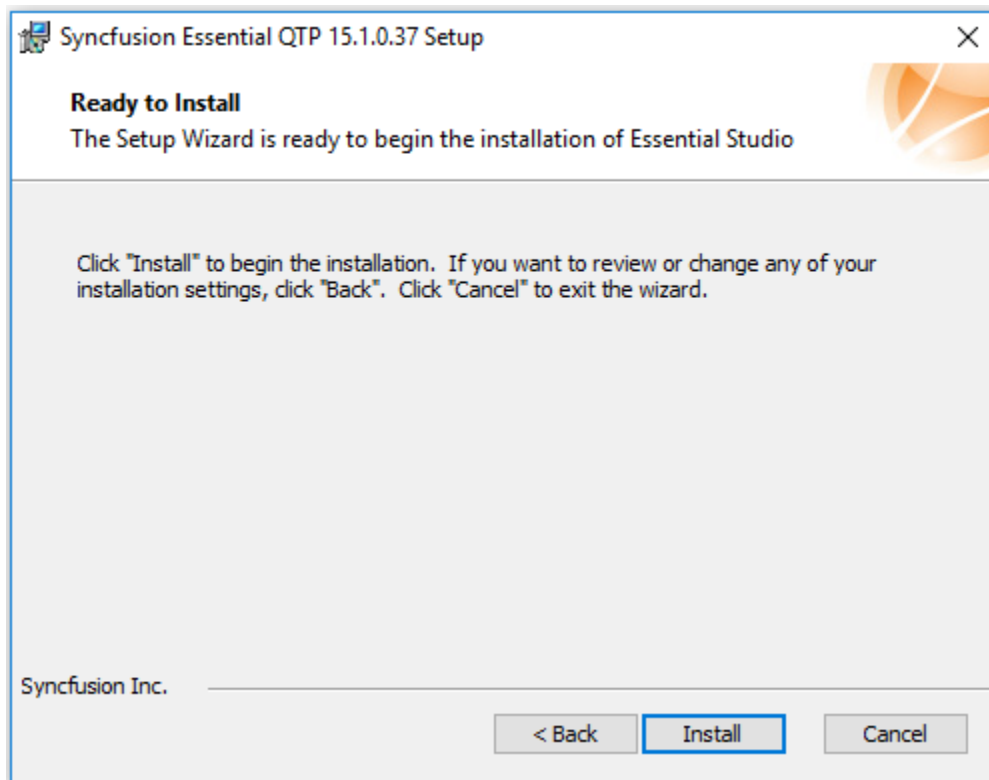
**Note:**

You can also browse to choose a location by clicking Browse.

7.Installation type dialog box opens.



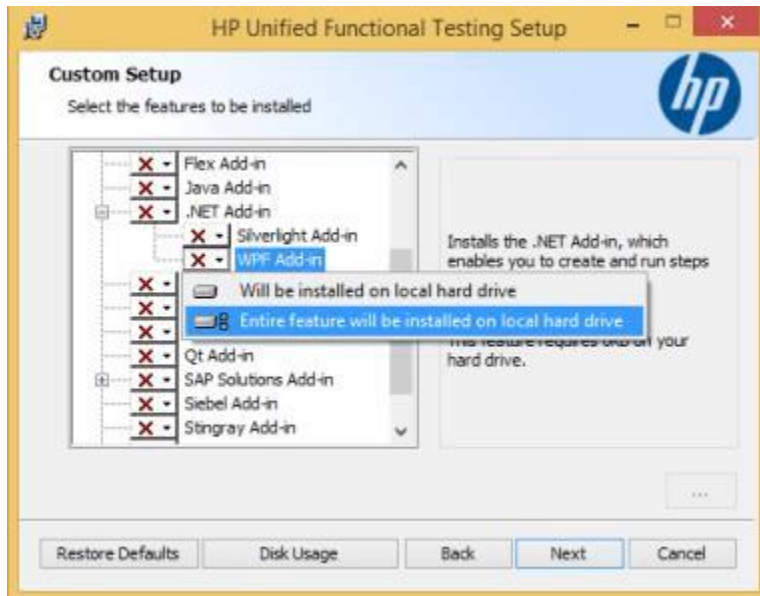
8. Choose from the options listed. For example, to install the complete setup, click Complete. 9. Click Next. The Ready to Install dialog box opens.



10. Click Install to continue with the installation.

### Enabling add-on support in UFT

You have to installed WPF Add-in while installing UFT by choosing below option.



### UFT - WPF Add-in installation

#### Configuring add-in assemblies for different .NET framework version

You need to configure Syncfusion add-in assemblies based on the framework version of the application that you are trying to automate.

For example, below are the steps to configure the custom add-in assemblies of syncfusion for SfDataGrid.

1. You need to ensure that **Syncfusion.CNG** and **SyncfusionTestObjects.XML** files are in the following location,

C:\Program Files\HP\QuickTestProfessional\dat\Extensibility\WPF\Syncfusion

C:\Program Files\HP\QuickTestProfessional\dat\Extensibility\WPF\

2. Also, ensure that **Syncfusion.SfGridQTP.WPF.dll** and **Syncfusion.GridExt.WPF.dll** are in the following location,

C:\Program Files\HP\QuickTest Professional\dat\Extensibility\WPF\Syncfusion

3. When your system does not contain **Syncfusion.CNG** , **SyncfusionTestObjects.XML** and

**Syncfusion.SfGridQTP.WPF.dll** files in the above mentioned location, you have to copy the files from the below location based on the framework version of the application you are trying to automate,

C:\Program Files\Syncfusion\Essential QTP\{{ site.releaseversion }}\WPF\bin\

### *Configuring add-in based on your application framework version*

By Default, Syncfusion QTP or UFT add-in configured based on the higher Framework version installed in your machine. if you have developed your application is lower framework version, you to configure the right version assemblies as below,

1. Copy the custom add-in assemblies of Syncfusion from the below installed location based on the framework version of application you are trying to automate,

C:\Program Files\Syncfusion\Essential QTP\{{ site.releaseversion }}\WPF\bin\ <Version>

2. Paste the copied assemblies to below location,

C:\ProgramFiles\HP\QuickTestProfessional\data\Extensibility\WPF\Syncfusion\

Create, Record and Run the tests

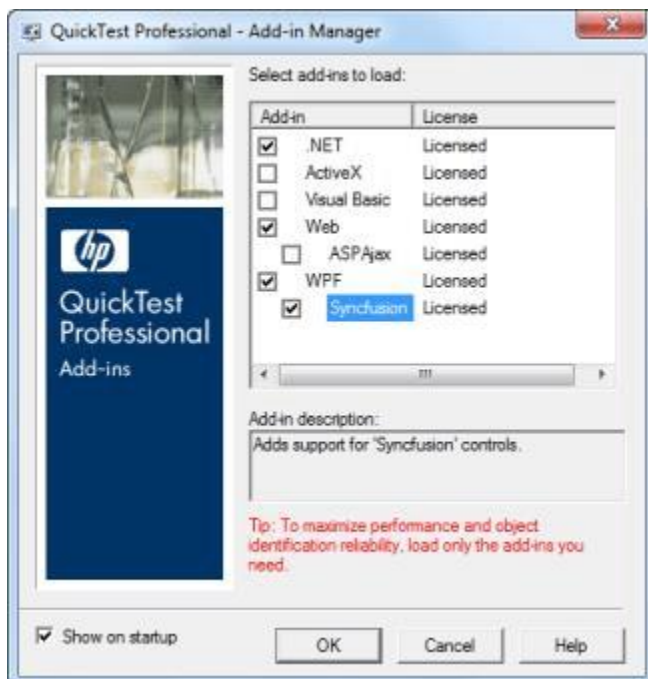
### *Creating a new Test*

1. Open QTP by double-clicking the QuickTest Professional icon.

#### **Note:**

The QuickTest Professional – Add-in Manager window is displayed.

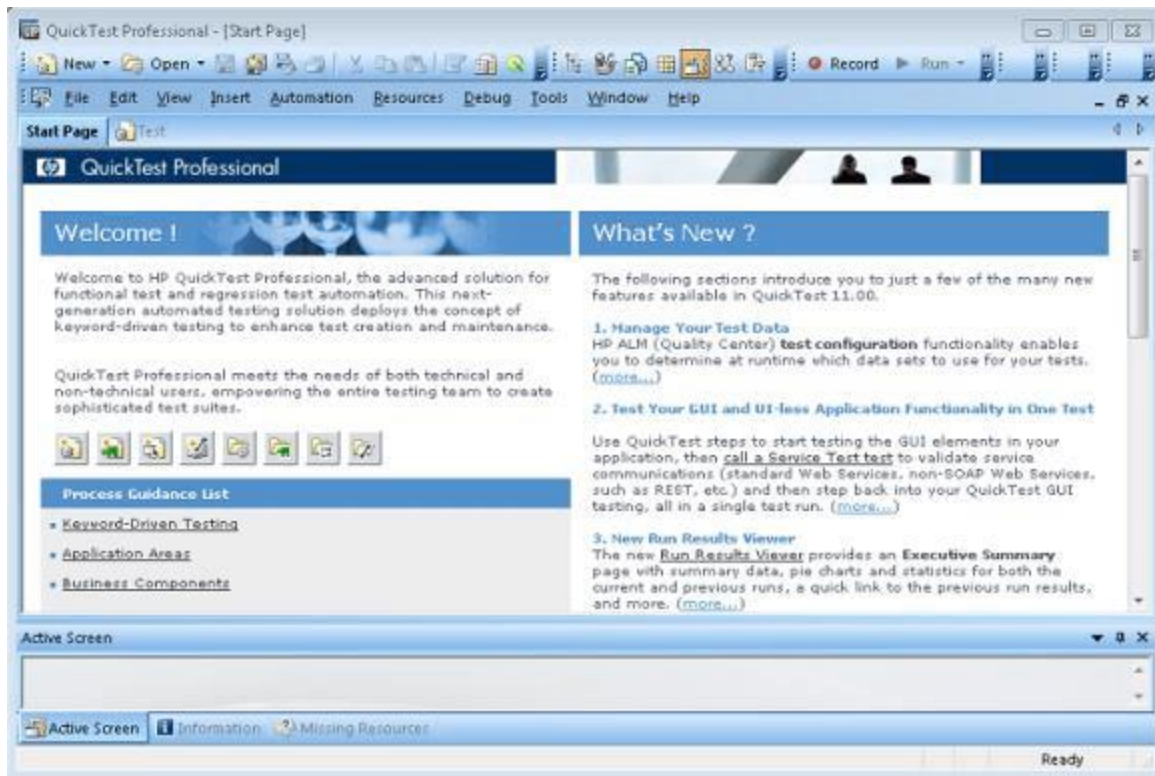
2. Select the WPF check box under the Add-in header. This ensures that WPF add-in is installed. Also, you need to check the Syncfusion add-in to detect the Syncfusion Controls. When it is not selected, you cannot access the Syncfusion controls.



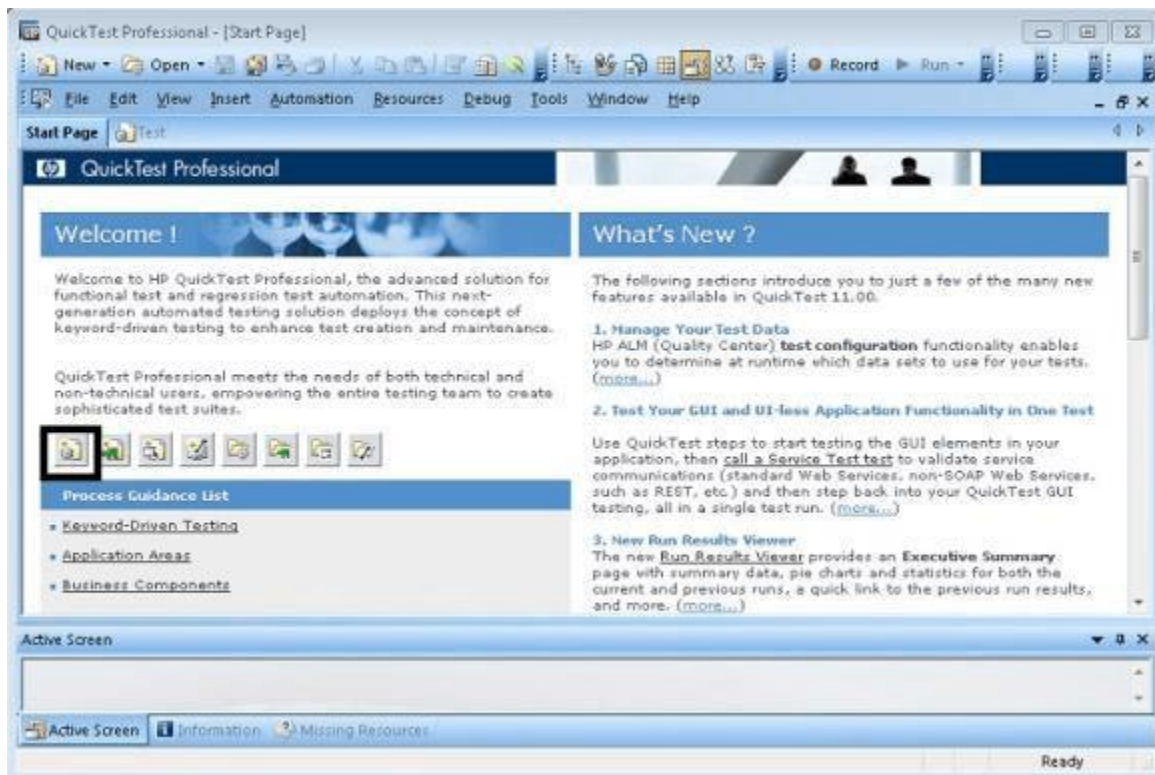
3. Click OK.

#### **Note:**

The QuickTest Professional – [Start Page] window opens. There are two tabs namely Start Page and Test in the main pane of the window. The content under the Start Page tab is displayed by default.

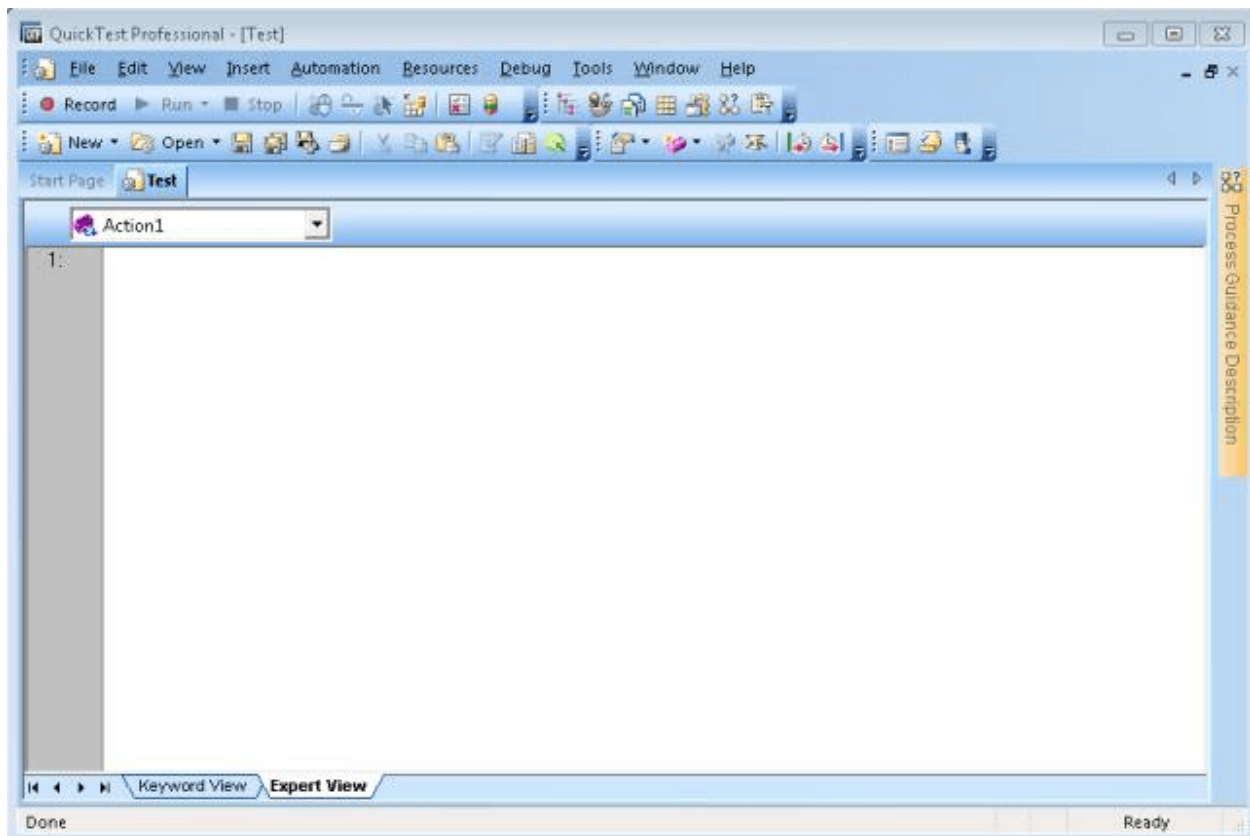


4. Click the New Test icon in the Start Page.

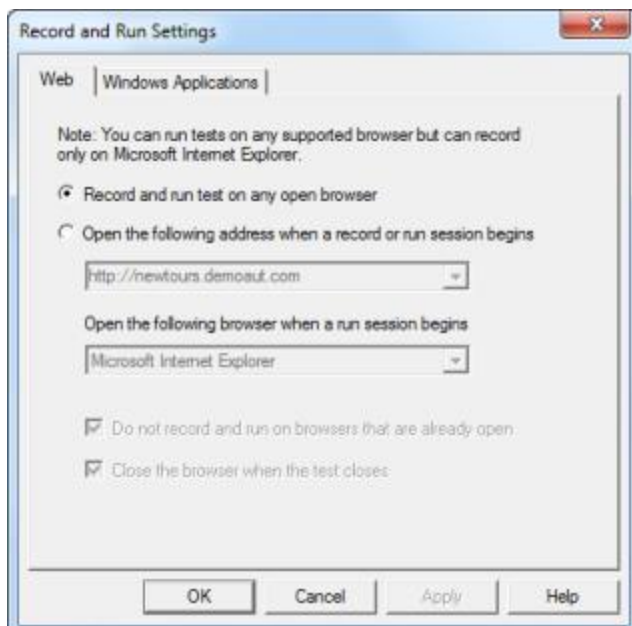




5. A new test is created. You can also create a new test by selecting the Test tab in the main pane of the window or Test sub-menu under the New menu in the menu bar. 6. Click Record in the toolbar to start the recording.

**Note:**

Record and Run Settings dialog box opens.



7. Select the Windows Application tab.

**Note:**

The content under the tab is displayed.



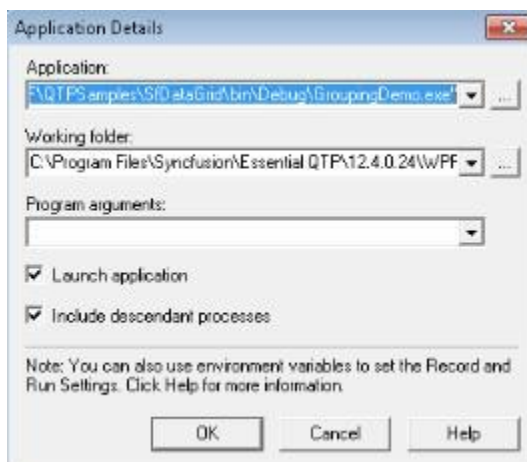
**Note:**


The Record and run only option is selected by default.

8. Select the following checkbox to ensure that only the applications opened by QuickTest and added applications are tested. 1. Applications opened by Quick Test 2. Applications opened via the Desktop (by the Window shell) 3. Applications specified below 9. To add an application for testing, click the + button in the Application details.


**Note:**

The Application Details dialog box opens.



10. For Application field, browse and select the path of the application that has to be tested by clicking 

button.

11. For working folder field, browse and select the path of the working folder by clicking 

button.

12. Select the Launch application check box, to launch the application immediately after clicking OK.

13. Select Include descendant processes check box, to include all the processes that are descendant to the current process.

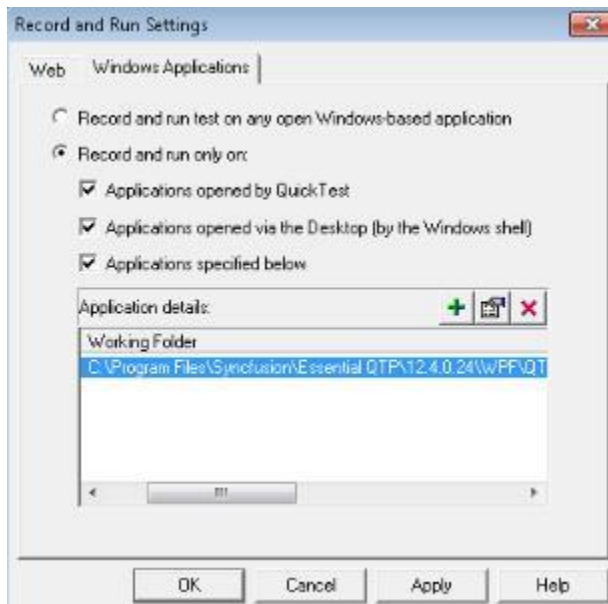
**Note:**

Both Launch application and Include descendant processes check boxes are selected by default

14. Click OK.

**Note:**

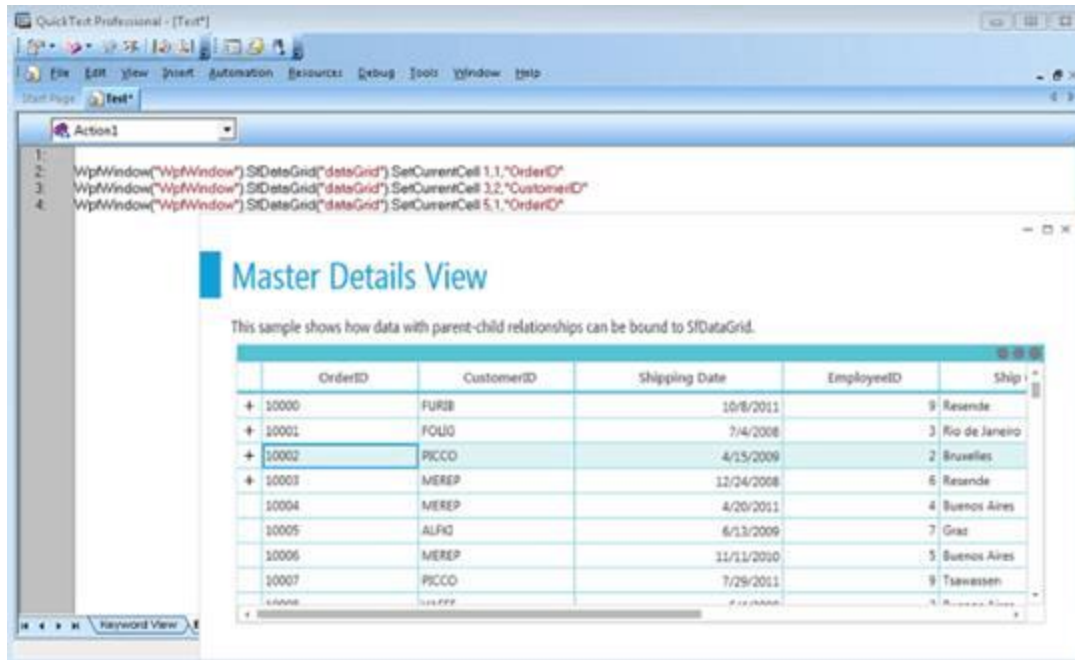
The path of the application and working folder are displayed in the Application details frame as shown in the following screenshot.



15. Click OK.

**Note:**

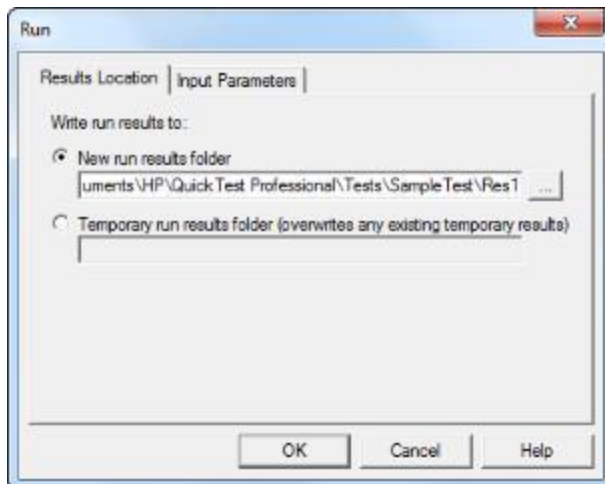
The recording starts. The application in the given path is opened as shown in the following screenshot.



### Running a Test

On recording, all the user actions performed in the control are just noted with the corresponding method names of the Syncfusion namespace. The errors can be checked while running a test. To run a test:

1. Click Run in the toolbar. The Run dialog box opens. The Results Location tab is selected by default.



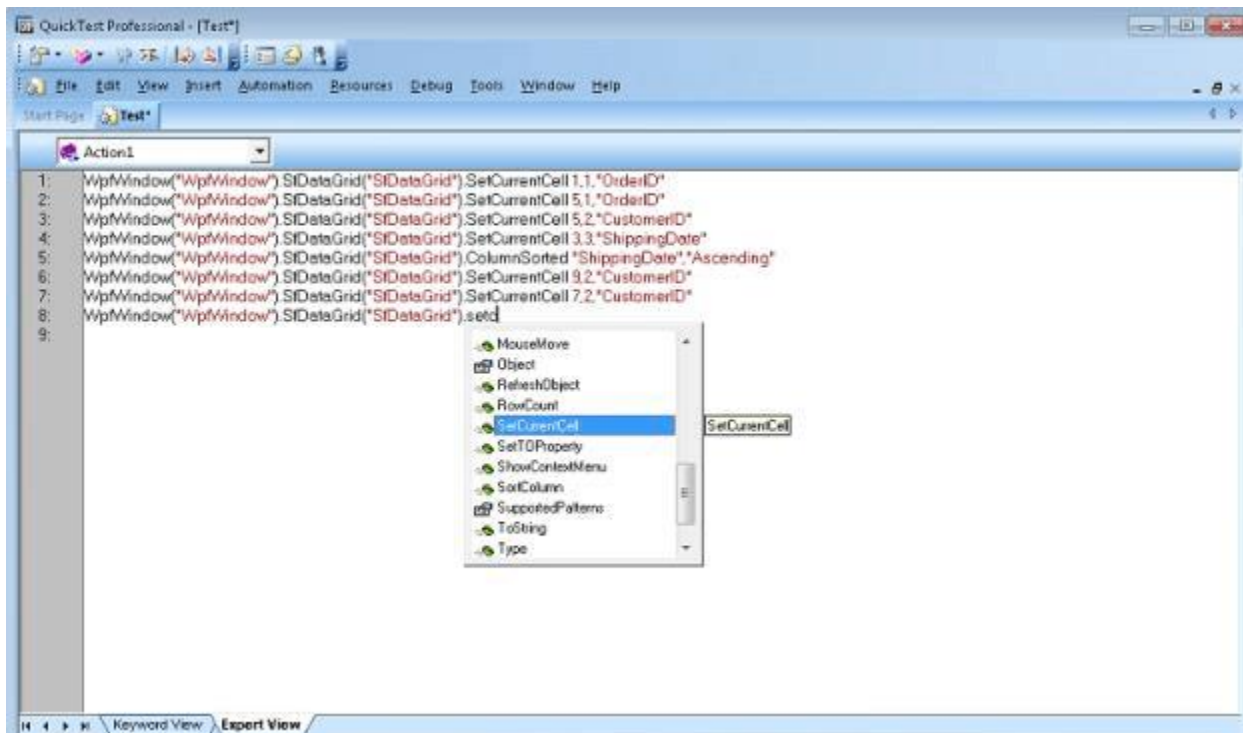
### Editing a Test

The editing of a test can be done either in the Keyword view or in the Expert view. You can switch between these views by selecting the required tab at the bottom left of the QTP screen.

### Editing in Expert View

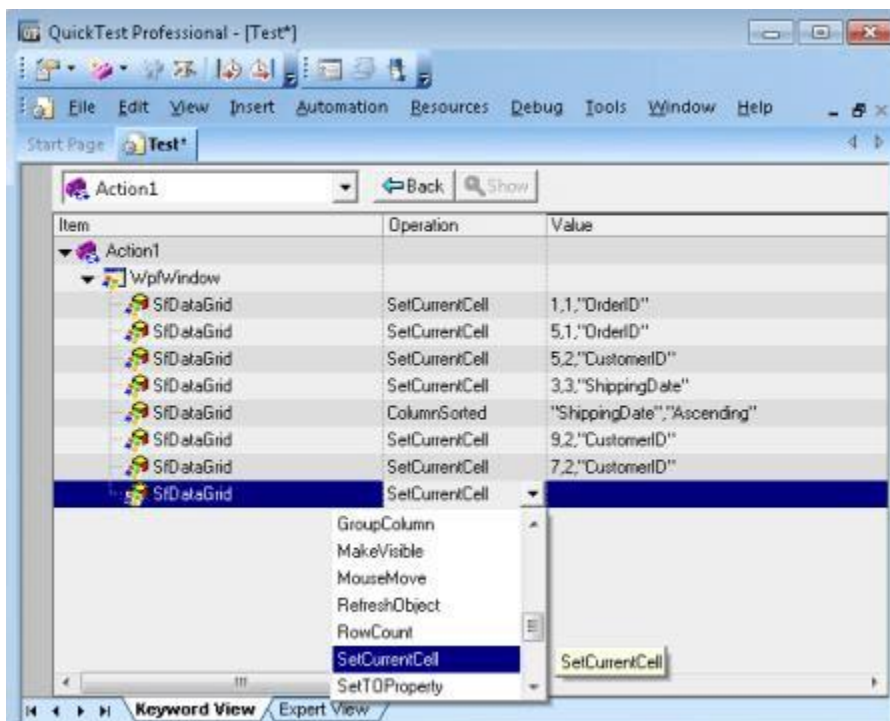
This view is especially provided for the experts in VB Script. In the Expert view, the VB scripts are generated while recording. You can also manually write scripts to the existing scripts in this view. So, this view can be used as a tool for managing the testing process in a more controlled manner. You can add scripts to trigger events manually.

The following image shows adding a script line to the Expert View pane.



#### Editing in Keyboard View

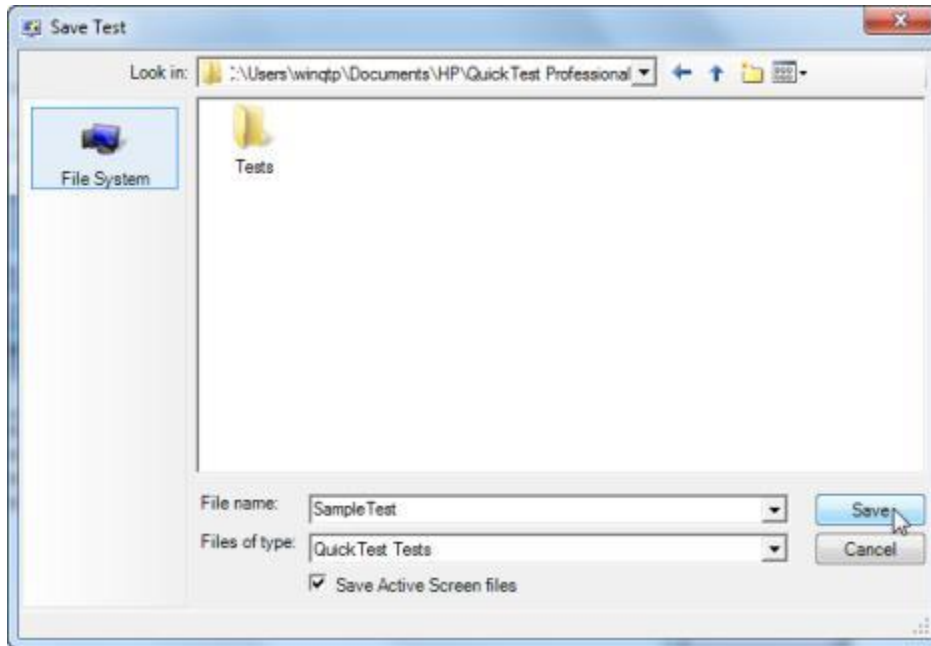
The keyword view is meant for people with no expertise in VB scripts. Keyword view contains the controls used, the user-actions or operations performed, values involved in the operation and the documentation summary in a table format. The controls used are listed under the Item header in a tree-view format as shown in the following screenshot:



### *Saving a Test*

Saving a test is as simple as saving any other document or picture. To save a test:

1. Click the Save button in the toolbar. The Save Test dialog box opens.



2. Select the location, to save the file from the Look in drop-down list.
3. Type the name of the file to be saved in the File name text box.
4. Click Save.
5. The test is saved.

### *Running a Saved Test*

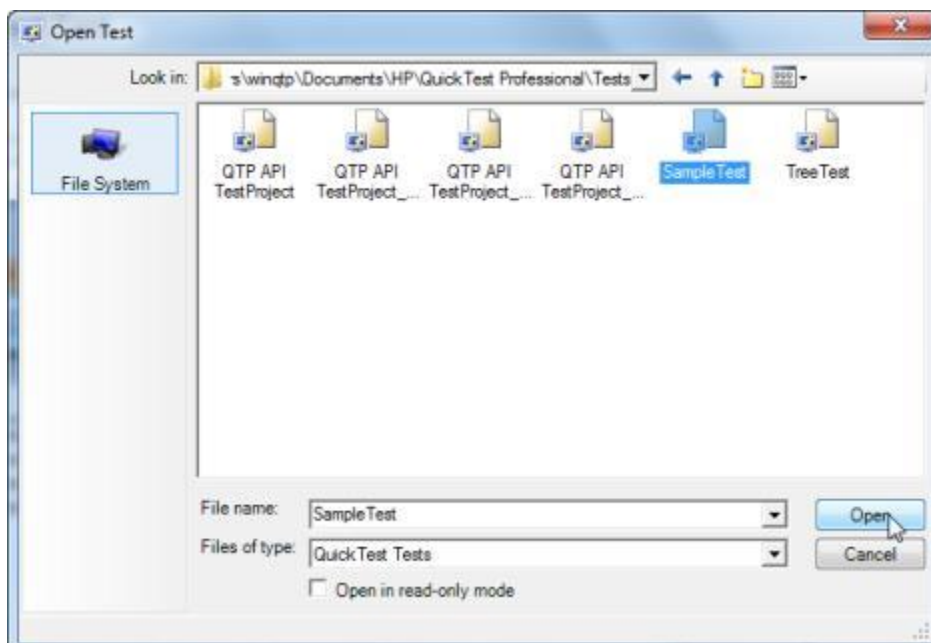
The tests that have been saved can be replayed later. For running such tests:

1. Click Open on the toolbar.

---

**Note:** The Open Test dialog box opens with a list of saved tests.

---



2.Select the required test. 3.Click Open.

**Note:**

The saved test is opened with its name and the complete path as the name of the window. By default, Expert View of the Test is opened.

**Supported Controls**

The following controls are supported for UFT/QTP testing

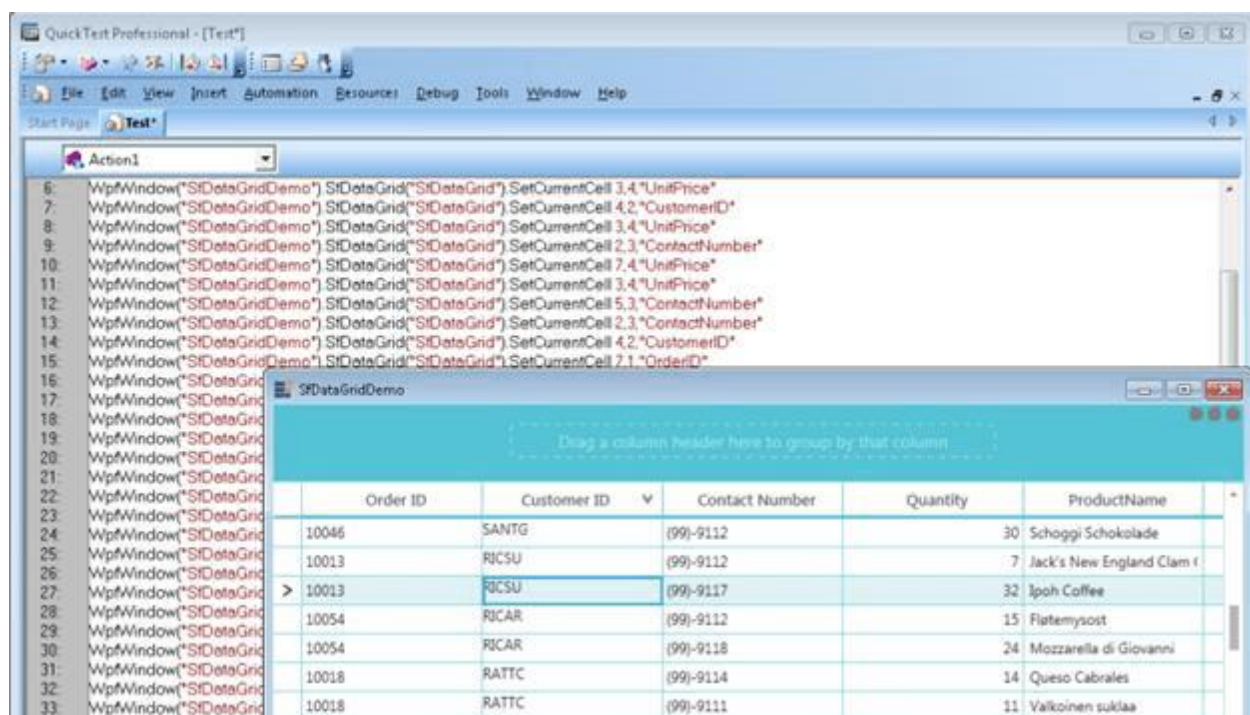
**SfDataGrid**

Method	Description	Parameters	Parameter Explanation	Return Type
void SetCurrentCell(int row,int col);	To Set the CurrentCell.	int row, int col	Passing current row and column index to the SetCurrentCell method	Void
void GroupColumn(string columnName);	To Group the Column	String columnName	Passing MappingName for a column	Void
void UnGroupColumn(string columnName);	To Un Group the column	String columnName	Passing MappingName for a column	Void
void SortColumn(string columnName,string state);	To Sort the column	string columnName, string state	Passing MappingName for a column, passing sorting state whether Ascending / Descending	Void
void BeginEdit(int row,int col);	To Enter the Edit mode for particular cell	int row,int col	Passing current row and column index to the BeginEdit method	Void
void EndEdit();	To EndEdit the current cell	NA	-	Void
void GetRowCount();	To get the row count of the SfDataGrid	NA	-	Int
void GetColumnCount();	To get the column count of the SfDataGrid	NA	-	Int



void GetCellValue(int rowIndex, int columnIndex);	To get the value of the cell	int rowIndex, int columnIndex	Passing row and column index for a cell	Object
---	---------------------------------	----------------------------------	---	--------

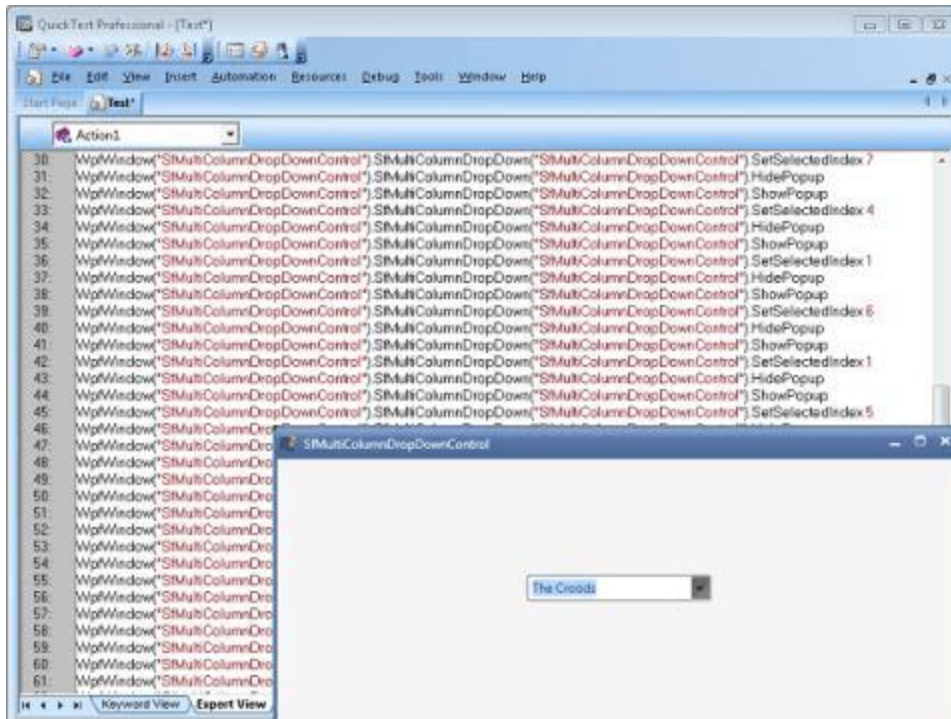
Properties	Description	Type	Data Type
Int RowCount	Gets the RowCount of the SfDataGrid	N/A	Int
Int ColumnCount	Gets the ColumnCount of the SfDataGrid	N/A	Int



### SfMulticolumnDropDownControl

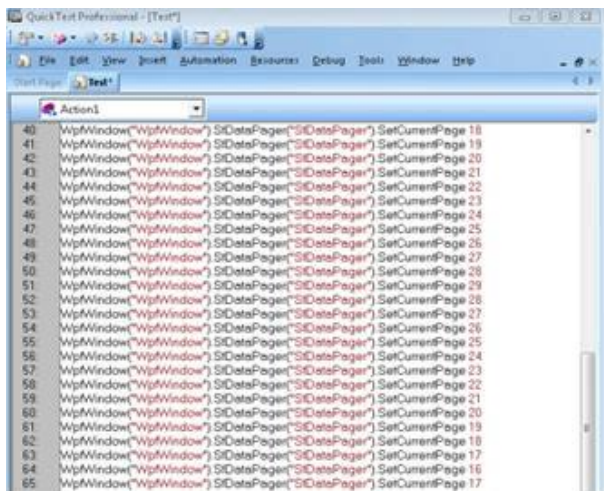
Method	Description	Parameters	Parameter Explanation	Return Type
void SetSelectedIndex(int index);	To set the SelectedIndex from the Popup	Int index	Passing columnindex to the SetSelectedIndex method	Void
void ShowPopup();	To open the Popup	NA	-	Void
void HidePopup();	To Close the Popup	NA	-	Void





## SfDataPager

Method	Description	Parameters	Parameter Explanation	Return Type
void SetCurrentPage(int pageIndex)	To set the current page in SfDataPager	Int pageIndex	Passing currentpage index to the SetCurrentPage method.	Void



OrderID	CustomerID	EmployeeID	Ship Name	Ship Address	Ship City	Ship Country
10256	WELLI	3	Wellington Import	Rua do Mercado, 12	Resende	Brazil
10257	HILAA	4	HILARIO-Alberto	Carrera 22 con Ave. Carlos	San Cristobal	Venezuela
10258	ERNSH	1	Ernst Handel	Kirchgasse 6	Graz	Austria
10259	CENTC	4	Centro comercial	Sierras de Granada, 916	Mexico D.F.	Mexico
10260	OTTIK	4	Ottlieb Kiskadee	Mehrheimerstr. 36	Köln	Germany
10261	QUEDE	4	Que Delicia	Rua de Panificacao	Rio de Janeiro	Brazil
10262	RATTC	8	Rattlesnake Canyon	2817 Milton Dr.	Albuquerque	USA
10263	ERNSH	9	Ernst Handel	Kirchgasse 6	Graz	Austria
10264	FOLKO	6	Folk och fa H&B	Järegatan 24	Bracke	Sweden
10265	BLONP	2	Blondel père et fils	24, place Kléber	Strasbourg	France
10266	WARTH	3	Wartian Herkku	Torikatu 38	Oulu	Finland
10267	FRANK	4	Frankenversand	Berliner Platz 43	München	Germany
10268	GROSR	8	GROSELLA-Restaurante	5P Ave. Los Palos	Caracas	Venezuela
10269	WHITC	5	White Clover Mart	1029 - 12th Ave. S	Seattle	USA
10270	WARTH	1	Wartian Herkku	Torikatu 38	Oulu	Finland
10271	SPLUR	6	Split Rail Beer & A.P.O.	Box 555	Lander	USA

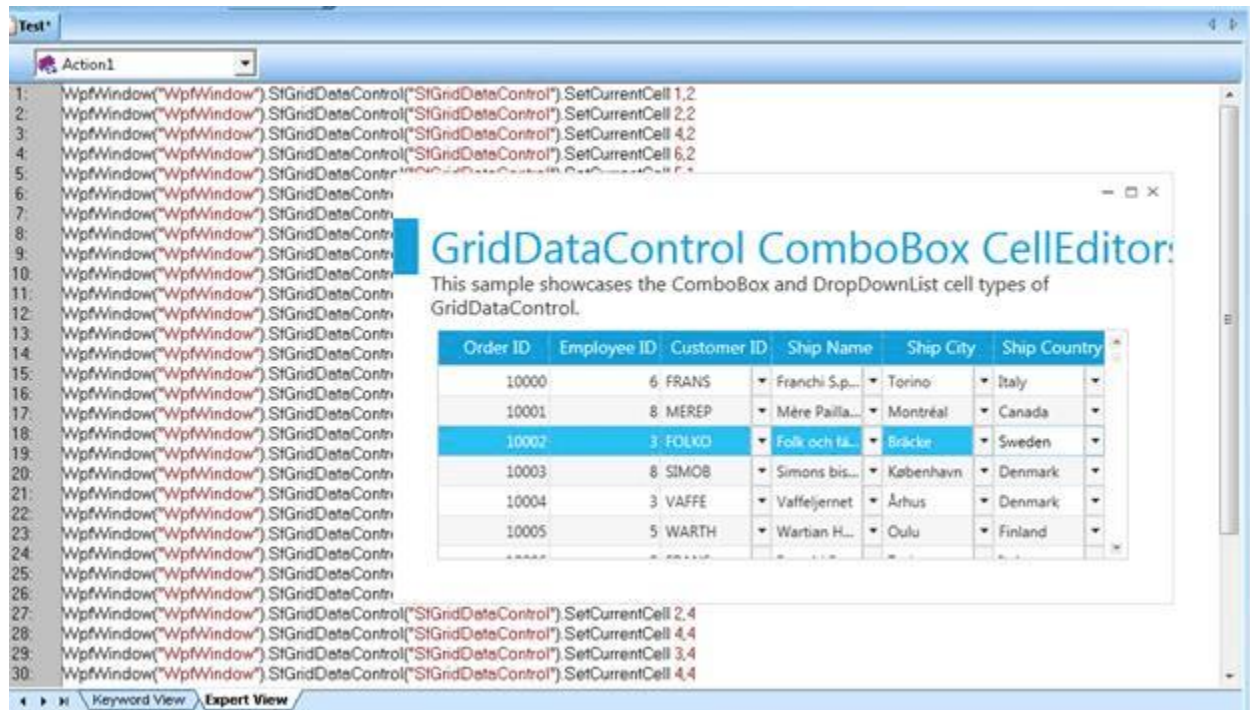
## GridDataControl

Method	Description	Parameters	Parameter Explanation	Return Type
bool IsColSorted(string colName);	Identify whether the column is sorted or not.	String colName	Passing the MappingName of the column	Bool
bool HasMappingName(string mappingName);	Identify whether the given MappingName is present for the Grid.	String MappingName	Passing MappingName for a column	Bool
bool IsGroupsExpanded();	Check whether the Groups are expanded or not	NA	-	Bool
bool IsRecord(int recordIndex);	Check the record based on the record index	int recordIndex	Passing record index	Bool
bool IsRecordExpanded(int recordIndex);	Check whether the record is Expanded	int recordIndex	Passing record index	Bool
bool IsGroupRow(int rowIndex);	Check whether the row is GroupRow	int rowIndex	Passing row index	Bool
Int GetChildCount();	Gets the Groups count	NA	-	Int
Void ScrollToRow(int row)	Move the cursor to specified row index	int row	Passing the row index	Void
ScrollToCol(int col)	Move the cursor to specified column index	int col	Passing the column index	Void
Void MoveTo(int row)	Move the current cell to particular row index	int row	Passing the row index	Void
Void SortColumn(string colName, string sortOrder)	Sorting the column	String colName, string sortOrder	Passing the column MappingName and sort direction	Void

Void GroupBy(string colName)	Grouping the column	String colName	Passing the column MappingName	Void
Void ExpandAllGroups()	To expanding the all groups	NA	-	Void
Void CollapseAllGroups()	To collapsing the all groups	NA	-	Void
Void ExpandGroup(int recordIndex)	To expand the group based on the record index.	int recordIndex	Passing the record index.	Void
Void CollapseGroup(int recordIndex)	To collapse the group based on the record index.	int recordIndex	Passing the record index.	Void
Void ExpandAllRecords()	To Expand all nested records.	NA	-	Void
Void CollapseAllRecords()	To Collapse all records	NA	-	Void
Void ExpandRecord(int recordIndex)	To Expand the particular record based on the record index.	int recordIndex	Passing the record index	Void
Void CollapseRecord(int recordIndex)	To collapse the particular record based on the record index.	int recordIndex	Passing the record index	Void
Void AddVisibleColumn(string MappingName, string HeaderText)	To add the new visiblecolumn	String MappingName, String HeaderText	Passing the MappingName and HeaderText for the column.	Void
Void InsertVisibleColumn(int insertAt, string MappingName, string headerText)	To insert the visible column for particular index.	int insertAt, string MappingName, string headerText	Passing the column index, MappingName and HeaderText for the column	Void
Void RemoveVisibleColumn(string colName)	To remove the visiblecolumn	String colName	Passing the MappingName for a column	Void
Void GetColSortOrder(string colName)	To get the column sorting order	String colName	Passing the MappingName for a column	Void

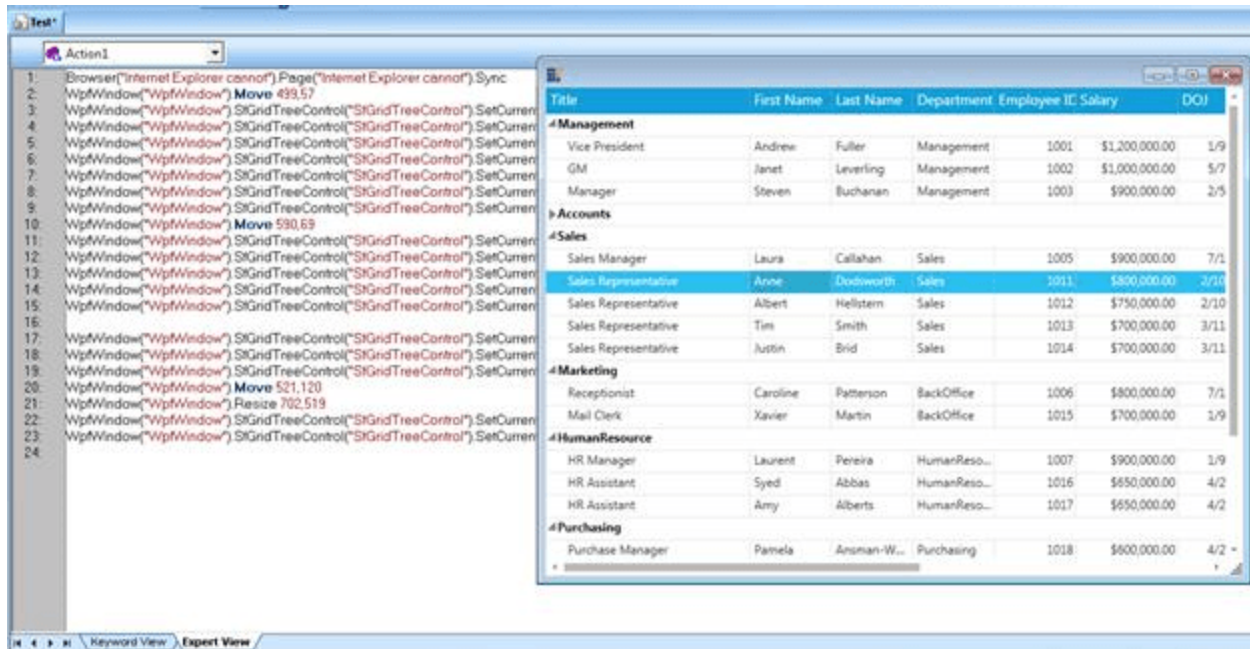
Void MoveCurrentCellTo(int rowIndex, int columnIndex)	To move the current cell to particular row and column index.	int rowIndex, int columnIndex	Passing the row and column index	Void
Void MoveCurrentCellTo(int rowIndex)	To move the current cell to particular row index.	int rowIndex	Passing the row index	Void
Void MoveTo(Object o)	To move the current cell to particular row and column index.	Object o	Passing the underlying object	Void
Bool IsRecordAvailable(Object o)	To check whether the record is available or not	Object o	Passing the underlying object	Bool
Bool IsReadOnly(int row, int col)	To check whether the cell is readonly or not	int row, int col	Passing the row and column index	Bool
Void CellClick(int row, int col)	To move the focus to the particular cell	int row, int col	Passing row and column index	Void
Void ResizeColumns(int fromColumn, int to, double width)	To resize the column	int fromColumn, int to, double width	Passing the from column index, to column index and width of the column	Void
Void ResizeRows(int fromRow, int to, int height)	To resize the row	int fromRow, int to, double height	Passing the from row index, to row index and height of the row	Void
Void GetCellValue(int row, int col)	To get the particular cell value	int row, int col	Passing the row and column index	Void
Void GetCellType(int row, int col)	To get the particular cell type	int row, int col	Passing the row and column index	Void
Double GetCellProperty(int row, int col, string value)	To get the x/y/height/width for particular cell	int row, int col	Passing the row and column, string value	Double

Void SetCurrentCell(int row, int col)	To set the currentcell to particular row and column index	int row, int col	Passing the row and column index	Void
---------------------------------------	---	------------------	----------------------------------	------



### GridTreeControl

Method	Description	Parameters	Parameter Explanation	Return Type
Void ExpandAllNodes()	To expand all nodes in GridTreeControl	-	-	Void
Void ExpandNode(int rowIndex)	To expand the particular code	int rowIndex	Passing the row index	Void
Void CollapseAllNodes()	To collapse the all nodes	-	-	Void
Void CollapseNode(int rowIndex)	To collapse the particular node	Int rowIndex	Passing the row index	Void



## Grid Control

Method	Description	Parameters	Return Type
bool SetCurrentCell(int row, int col)	Sets the CurrentCell based on Row and Column parameters.	int row, int col	bool
bool SetChkBoxCell(int row, int col, bool val)	Sets the CheckBoxCell value by using Row, Column and Value parameters.	int row, int col, bool val	bool
bool IsFormulaCell(int row, int col)	Finds whether the cell is a formula cell.	int row, int col	bool
int GetSelectedRowIndex()	Gets the selected row from Grid.	NA	int
int GetSelectedColIndex()	Gets the selected Column from Grid.	NA	int
void SetCellData(int row, int col, string val)	Sets the cell data by using Row, Column and Value parameters.	int row, int col, string val	void
void InsertColumns(int col, int count)	Inserts the column by using Count and Target parameters.	int col, int count	void



void InsertRows(int row, int count)	Inserts the row by using Count and Target parameters.	int row, int count	void
void RemoveColumns(int col, int count)	Removes the columns by using Row parameter.	int col, int count	void
void RemoveRows(int row, int count)	Removes the Rows by using Row parameter.	int row, int count	void
void MoveRows(int removeAtRowIndex, int count, int insertAtRowIndex)	Moves the row by using From and Target parameters.	int removeAtRowIndex, int count, int insertAtRowIndex	void
void MoveColumns(int removeAtColIndex, int count, int insertAtColIndex)	Moves the column by using From and Target parameters.	int removeAtColIndex, int count, int insertAtColIndex	void
void CellClick(int row, int col)	Performs a click action to the cell by using row and column parameters.	int row, int col	void
void ResizeColumns(int fromColumn, int to, int width)	Resizes the column by using column parameter.	int fromColumn, int to, int width	void
void ResizeRows(int fromRow, int to, int height)	Resizes the row by using row parameter.	int fromRow, int to, int height	void
void SetScrollPosition( int vScrollPosition, int hScrollPosition)	Sets the scroll position based on vScrollPosition and hScrollPosition parameters.	int vScrollPosition, int hScrollPosition	void
void SelectRange(int topRow, int leftCol, int bottomRow, int rightCol)	SelectRange by using Top,Left,Right and Bottom parameters.	int topRow, int leftCol, int bottomRow, int rightCol	void
void ScrollInToView(int row, int col)	Scrolls the cell into view, based on Row and Column parameters.	int row, int col	void
void HideRows(int fromRow, int to)	Hides rows based on Row parameter.	int fromRow, int to	void
void HideCols(int fromCol, int to)	Hides columns based on Column parameter.	int fromCol, int to	void

void ShowHiddenRows(int fromRow, int to)	Shows the hidden Rows in Grid.	int fromRow, int to	void
void ShowHiddenCols(int fromCol, int to)	Shows the hidden Columns in Grid.	int fromCol, int to	void
string GetCellType(int row, int col)	Gets the CellType as a string value based on Row and Column parameters.	int row, int col	string
string GetCellBackground(int row, int col)	Gets the background color as string value based on Row and Column parameter.	int row, int col	string
string GetCellForeground(int row, int col)	Gets the foreground color as string value, based on Row and Column parameters.	int row, int col	string
string GetFormattedText(int row, int col)	Gets the formatted text as string value based on Row and Column parameters.	int row, int col	string

Properties	Description	Type	Data Type
Int RowCount	Gets the RowCount of the SfDataGrid	N/A	Int
Int ColumnCount	Gets the ColumnCount of the SfDataGrid	N/A	Int

## Chart

Method	Description	Parameters	Type	Return Type	Refresh links
SetScrollPosition	To record the scrolling of Chart Area.	(int area, int axis, double zoomposition)	NA	Void	NA
ChartSegmentDragging	Denotes the Segment that is dragged.	(int series, int segment)	NA	Void	NA
ChartZoomedIn	To record the Zoom in of Chart Area	(int areaindex, double xzoomFactor,	NA	Void	NA



		double yzoomFactor)			
ChartZoomedOut	To record the zoom out Chart Area.	(int areaindex, double xzoomFactor, double yzoomFactor)	NA	Void	NA
ChartZoomReset	To record the Chart Area reset.	(int areaindex, double xzoomFactor, double yzoomFactor)	NA	Void	NA
ChartPanning	To record the panning of Chart Area.	(int area, double zoomXPosition, double zoomYPosition)	NA	Void	NA
LegendLocationChanged	To record the change in location of Legend.	(double dockX, double dockY)	NA	Void	NA
ContextMenuOpening	To record the ContextMenu support.	(double isOpen)	NA	Void	NA
InteractiveCursorLocationChanged	To record the Interactive Cursor.	(double offsetX, double offsetY, int areaindex)	NA	Void	NA

## SfChart

Method	Description	Parameters	Return Type
void SelectionChanged(int selectedIndex, int seriesIndex)	To record select the segment /series by using selected index and series index parameter	int selectedIndex, int seriesIndex	void
void ZoomChanged(double zoomFactor, double zoomPosition, int axisIndex);	To set the zoom factor and zoom position to the chart axis in order to zoom-in/ zoom-out chart.	double zoomFactor, double zoomPosition, int axisIndex	void

void PanChanged(double zoomPosition, int axisIndex)	To set the zoom position to the chart axis in order to pan the chart.	double zoomPosition, int axisIndex	void
void ResetZoom(int axisIndex);	To reset the zoom position and zoom factor value to the chart axis.	int axisIndex	void
void DraggingAnnotation(double x1, double x2, double y1, double y2, int index)	To set the selected annotation position by X1, X2, Y1 and Y2 parameters.	double x1, double x2, double y1, double y2, int index	Void

## PivotGrid

Method	Description	Parameters	Return Type
void ShowGroupingBar()	Sets the GroupingBar properties of PivotGrid when ShowGroupingBar value is true.	N/A	void
void ResizeColumns(int left, int right, double width);	Resizes the column of internal Grid for desired width.	int left, int right, double width	void
void ResizeRows(int top, int bottom, double height);	Resizes the row of internal Grid for desired height.	int top, int bottom, double height	void
bool LoadInBackgroundCompleted()	Returns the value as true once the LoadInBackground action has been completed.	N/A	bool
bool AllowSelection()	Returns the value when selecting the cells of internal Grid.	N/A	bool
bool AllowSelectionWithHeaders();	Return the value when selecting the value cells along with headers of PivotGrid.	N/A	bool

<code>void CellClick(int rowIndex, int colIndex);</code>	Performs the click action to the cell of internal Grid by using specific row, column.	<code>int rowIndex, int colIndex</code>	<code>void</code>
<code>void ExpandAll()</code>	Expands all the rows or columns of PivotGrid.	N/A	<code>void</code>
<code>void CollapseAll()</code>	Collapses all the rows or columns of PivotGrid.	N/A	<code>void</code>
<code>bool SortBegin()</code>	Indicates whether the sorting operation has been started when sorting the values of PivotGrid.	N/A	<code>bool</code>
<code>bool SortCompleted()</code>	Indicates whether the sorting operation has been completed when the values of PivotGrid have sorted.	N/A	<code>bool</code>
<code>bool CanDrop()</code>	Returns the value when the PivotItem has been dropped between GroupingBar fields.	N/A	<code>bool</code>
<code>bool ContextMenuOpening()</code>	Returns the value when opening the context menu of PivotItem.	N/A	<code>bool</code>
<code>bool FilterPopupOpened()</code>	Returns the value when clicking the filter popup of any PivotItem to filter the values.	N/A	<code>bool</code>
<code>bool FilterActionCompleted()</code>	Indicates whether the filtering action has been completed if the values are added in filters of PivotGrid.	N/A	<code>bool</code>

## Section 508 Compliance - Voluntary Product Accessibility Template

The following Voluntary Product Accessibility information refers to Syncfusion Essential Studio for WPF.

CRITERIA	SUPPORTING FEATURES
Section 1194.21 Software applications and operating systems.	Please refer to the Subpart B.
Section 1194.22 Web-based Intranet and Internet information and applications.	Not Applicable.
Section 1194.23 Telecommunications products.	Not Applicable.
Section 1194.24 Video and multimedia products.	Not Applicable.
Section 1194.25 Self-contained closed products.	Not Applicable.
Section 1194.26 Desktop and portable computers.	Not Applicable.
Section 1194.31 Functional performance criteria.	Please refer to the Subpart C.
Section 1194.41 Information, documentation, and support.	Please refer to the Subpart D.

[Subpart B – Technical Standards Section 1194.21](#)

Section 1194.21 Software Applications and Operating Systems - Detail Voluntary Product Accessibility Template

CRITERIA	SUPPORTING FEATURES	EXPLANATIONS
(a) When software is designed to run on a system that has a keyboard, product functions shall be executable from a keyboard, where the function itself or the result of performing a function can be discerned textually.	Supported with minor exceptions.	Essential Diagram WPF provides keyboard support for most of the functions.  All other products controls that accept focus provide keyboard support.
(b) Applications shall not disrupt or disable activated features of other products identified as accessibility features, where those features are developed and documented according to industry standards. Applications also shall not disrupt or disable activated features of any operating system that is identified as accessibility feature where the application programming interface for those accessibility features has been documented by the manufacturer of the operating system and is available to the product developer.	Fully Supported	All Essential Studio for WPF controls do not disable any accessibility features of other products.  Application developers using the controls should take care to preserve this.

<p>(c) A well-defined on-screen indication of the current focus shall be provided to move among interactive interface elements as the input focus changes. The focus shall be programmatically exposed so that Assistive Technology can track focus and focus changes.</p>	<p>Supported with minor exceptions</p>	<p>Essential Tools: Supported with minor exceptions. Controls provide a well-defined live indication of the current focus that moves among interactive interface elements as the input focus changes with minor exceptions such as Docking Manager, Ribbon.</p> <p>Essential Chart: Fully Supported.</p> <p>Essential Diagram: Fully Supported. Also provides hover styles and selection styles that highlight the position of the focus.</p> <p>Essential Schedule: Fully Supported.</p> <p>Essential Edit: Fully Supported.</p>
<p>(d) Sufficient information about user interface element including identity, operation and state of the element shall be available to Assistive Technology. When an image represents a program element, the information conveyed by the image must also be available in text.</p>	<p>Fully Supported</p>	<p>Essential Tools: Fully supported. Controls provide enough information regarding the user interface elements and program elements, which can be used in Assistive technologies.</p> <p>Essential Chart: Not Applicable.</p> <p>Essential Diagram: Fully supported. Control provides sufficient information available in text about user interface elements and program elements represented by images, in virtually all cases.</p> <p>Essential Gauge: Fully supported. Control provides enough information regarding the user interface elements and program elements, which can be used in Assistive technologies.</p> <p>Essential Edit: Not Applicable.</p>
<p>(e) When bitmap images are used to identify controls, status indicators, or other programmatic elements, the meaning assigned to those images shall be consistent throughout an application's performance.</p>	<p>Fully Supported</p>	<p>Essential Tools: Not applicable.</p> <p>Essential Chart: Not applicable.</p> <p>Essential Diagram: Fully supported.</p>

		Control provides functionality that complies with these criteria. The cursor changes according to the operation being performed and it is consistent throughout the program.  Essential Gauge: Not applicable.  Essential Edit: Not applicable.
(f) Textual information shall be provided through operating system functions for displaying text. The minimum information that shall be made available is text content, text input caret location and text attributes.	Fully Supported	Essential Diagram: Fully Supported. Control provides functionality that complies with these criteria. Text information is usually provided using the operating system only.  Essential Tools: Not applicable.  Essential Chart: Not applicable.  Essential Schedule: Not applicable.  Essential Edit: Not applicable.
(g) Applications shall not override user selected contrast, color selections and other individual display attributes.	Fully Supported	All Essential Studio for WPF controls provide functionality that conforms to these criteria.
(h) When animation is displayed, the information shall be displayable in at least one non-animated presentation mode at the option of the user.	Fully Supported	All Essential Studio for WPF controls provide functionality that conforms to these criteria.  Essential Diagram: The nodes can contain animations if desired and also any other content can be used.
(i) Color coding shall not be used as the only means of conveying information, indicating an action, prompting a response or distinguishing visual element.	Fully Supported	All Essential Studio for WPF controls provide functionality that conforms to these criteria.
(j) When a product permits a user to adjust color and contrast settings, a variety of color selections capable of producing a range of contrast levels shall be provided.	Supported with minor exceptions	Essential Diagram: Supported with minor exceptions. Control provides functionality that allows setting colors for almost all the elements by providing a large range of colors to choose from.

		Other products: Fully supported.
(k) Software shall not use flashing or blinking text, objects, or other elements having a flash or blink frequency greater than 2 Hz and lower than 55 Hz.	Fully Supported	All Essential Studio for WPF controls provide functionality that conforms to these criteria.
(l) When electronic forms are used, the form shall allow people using Assistive Technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.	Not Applicable	This criterion does not apply to Essential Studio for WPF controls.

**Subpart C - Section 1194.31 Functional Performance Criteria**

CRITERIA	SUPPORTING FEATURES	REMARKS AND EXPLANATIONS
(a) At least one mode of operation and information retrieval that does not require user vision shall be provided or support for Assistive Technology used by people who are blind or visually impaired shall be provided.	Partially Supported.	Essential Tools: Supported in the following WPF Controls - Supported for TabControlExt, Docking Manager related controls. Not supported in the other Essential Studio for WPF controls.
(b) At least one mode of operation and information retrieval that does not require visual acuity greater than 20/70 shall be provided in audio and enlarged print output working together or independently, or support for Assistive Technology used by people who are visually impaired shall be provided.	Not Applicable	This criterion does not apply to Essential Studio for WPF controls.
(c) At least one mode of operation and information retrieval that does not require user-hearing shall be provided or support for Assistive Technology used by people who are deaf or hard of hearing shall be provided.	Not Applicable.	This criterion does not apply to Essential Studio for WPF controls.
(d) Where audio information is important for the use of a product, at least one mode of operation and information retrieval shall be provided in an enhanced auditory fashion or support for assistive hearing devices shall be provided.	Not Applicable	This criterion does not apply to Essential Studio for WPF controls.

(e) At least one mode of operation and information retrieval that does not require user-speech shall be provided or support for Assistive Technology used by people with disabilities shall be provided.	Not Applicable.	This criterion does not apply to Essential Studio for WPF controls.
(f) At least one mode of operation and information retrieval that does not require fine motor control or simultaneous actions and that is operable with limited reach and strength shall be provided.	Fully Supported	Essential Studio for WPF controls provide functionality that conforms to these criteria.

### Subpart D - Section 1194.41 Information, documentation, and support

Criteria	Supporting Features	Remarks and explanations
Section 1194.41 (a) Product Support Documentation provided to end-users shall be made available in alternate formats upon request, at no additional charge.	Supported.	Online <a href="#">Documentation</a> is available.
Section 1194.41 (b) Accessibility and Compatibility Features. End-users shall have access to a description of the accessibility and compatibility features of products in alternate formats or alternate methods upon request, at no additional charge.	Supported.	Essential Studio WPF controls are accessible
1194.41 (c) Support Services for products shall accommodate the communication needs of end-users with disabilities.	Supported.	Online support <a href="#">Incidents</a> and <a href="#">Forums</a> are available.

Syncfusion does not promise that the information provided in this document will be error-free, or that any errors will be corrected, or that your use of the information will provide specific results. The document and its content are delivered on an “as-is” basis. All information provided is subject to change without notice. Syncfusion disclaims all warranties, express or implied, including any warranties of accuracy, non-infringement, merchantability and fitness for a particular purpose.

## SfAccordion

### WPF Accordion (SfAccordion) Overview

The [WPF Accordion](#) control organizes content into multiple collapsible sections that can be expanded on demand. Single or multiple content can be expanded at a time.



## Features

- ItemsSource - Any business object collection can be bound to the control
- SelectionMode - Number of items that can be expanded/selected based on **SelectionMode**

## Getting Started with WPF Accordion (SfAccordion)

### Assembly deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

You can find more details about installing the NuGet package in a WPF application in the following link:

### [How to install nuget packages](#)

### Create a simple application with SfAccordion

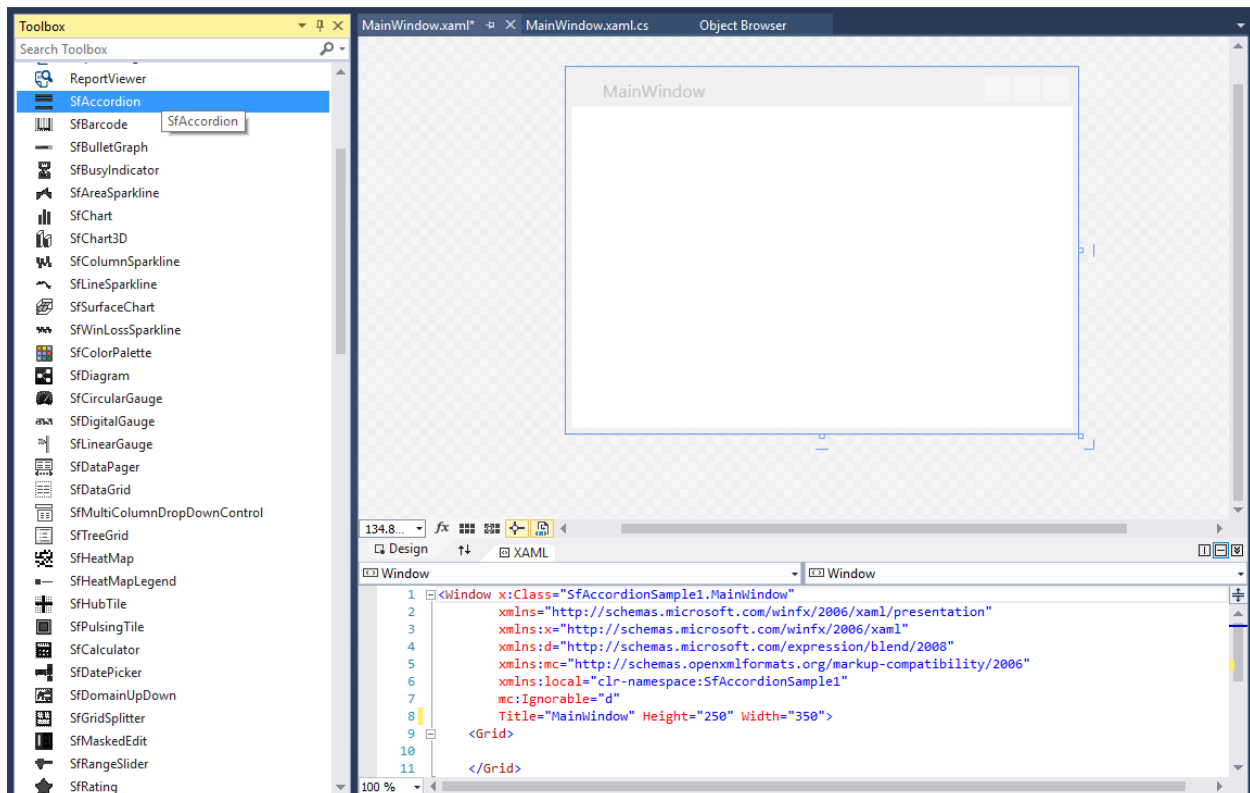
You can create a WPF application with SfAccordion control using the following steps:

#### Create a project

Create a new WPF project in Visual Studio to display the SfAccordion with functionalities.

#### Add control through designer

The SfAccordion control can be added to an application by dragging it from the toolbox to a designer view. The **Syncfusion.SfAccordion.WPF** assembly reference will be added automatically to the project.



#### Add control manually in XAML

To add the control manually in XAML, follow the given steps:

1. Add the **Syncfusion.SfAccordion.WPF** assembly reference to the project.
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page.
3. Declare the SfAccordion control in the XAML page.

### XML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="SfAccordionSample.MainWindow"
Title="SfAccordion Sample" Height="350" Width="525">
<Grid>
<!-- Adding SfAccordion control -->
<syncfusion:SfAccordion x:Name="SfAccordion" HorizontalAlignment="Center"
VerticalAlignment="Center" Width="100"/>
</Grid>
</Window>
```

### Add control manually in C\#

To add the control manually in C#, follow the given steps:

1. Add the **Syncfusion.SfAccordion.WPF** assembly reference to the project.
2. Import SfAccordion namespace using **Syncfusion.Windows.Controls.Layout**;
3. Create a SfAccordion instance, and add it to the window.

### C#

```
using Syncfusion.Windows.Controls.Layout;
namespace SfAccordionSample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            //Creating an instance of SfAccordion control
            SfAccordion accordion = new SfAccordion();
            //Adding SfAccordion as window content
            this.Content = accordion;
        }
    }
}
```

### Add items using SfAccordionItem

SfAccordion accepts [SfAccordionItem](#) as its child when adding it directly.

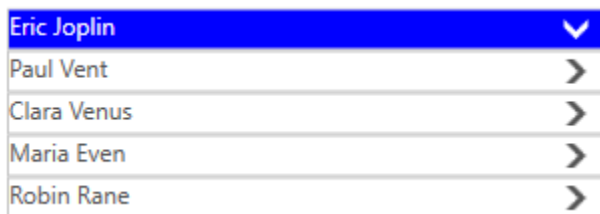
### XML

```
<syncfusion:SfAccordion HorizontalAlignment="Center" Width="300"
Height="200" VerticalAlignment="Center">
```

```
<syncfusion:SfAccordionItem Header="Eric Joplin"/>
<syncfusion:SfAccordionItem Header="Paul Vent"/>
<syncfusion:SfAccordionItem Header="Clara Venus"/>
<syncfusion:SfAccordionItem Header="Maria Even"/>
<syncfusion:SfAccordionItem Header="Robin Rane"/>
</syncfusion:SfAccordion>
```

**C#**

```
SfAccordion accordion = new SfAccordion();
//Instance of SfAccordionItem control
SfAccordionItem accordionItem1 = new SfAccordionItem();
SfAccordionItem accordionItem2 = new SfAccordionItem();
SfAccordionItem accordionItem3 = new SfAccordionItem();
SfAccordionItem accordionItem4 = new SfAccordionItem();
SfAccordionItem accordionItem5 = new SfAccordionItem();
//Setting header for SfAccordionItem
accordionItem1.Header = "Eric Joplin";
accordionItem2.Header = "Paul Vent";
accordionItem3.Header = "Clara Venus";
accordionItem4.Header = "Maria Even";
accordionItem5.Header = "Robin Rane";
accordion.Items.Add(accordionItem1);
accordion.Items.Add(accordionItem2);
accordion.Items.Add(accordionItem3);
accordion.Items.Add(accordionItem4);
accordion.Items.Add(accordionItem5);
this.Content = accordion;
```



## Bind data

SfAccordion accepts any business object collection to be bound using its [ItemsSource](#) property.

- **Model.cs**

**C#**

```
public class Person
{
    public ImageSource Image { get; set; }
    public string Name { get; set; }
    public string Position { get; set; }
    public string organization { get; set; }
    public string DateOfBirth { get; set; }
    public string Location { get; set; }
    public string Phone { get; set; }
    public string Email { get; set; }
}
```

```
public string TileColor { get; set; }
public Person(string name, ImageSource image, string position, string
organization, string birth, string location, string phone, string email)
{
    Name = name;
    Image = image;
    Position = position;
    organization = organization;
    DateOfBirth = birth;
    Location = location;
    Phone = phone;
    Email = email;
}
}
```

- **ViewModel.cs**

## C#

```
public class ViewModel
{
    private ObservableCollection<Person> _employee;
    public ObservableCollection<Person> Employees
    {
        get { return _employee; }
        set { _employee = value; }
    }
    public ViewModel()
    {
        Employees = new ObservableCollection<Person>();
        Employees.Add(new Person("Eric Joplin", new BitmapImage(new
Uri("pack://application:,,,/DataBinding;Component/Images/Emp_02.png")),
"Chairman", "Management", "27/09/1973", "Boston", "+800 9899 9929",
"Joplin@syncfusion.com"));
        Employees.Add(new Person("Paul Vent", new BitmapImage(new
Uri("pack://application:,,,/DataBinding;Component/Images/Emp_04.png")),
"Chief Executive Officer", "Management", "27/09/1975", "New York", "+800
9899 9930", "Paul@syncfusion.com"));
        Employees.Add(new Person("Clara Venus", new BitmapImage(new
Uri("pack://application:,,,/DataBinding;Component/Images/Emp_06.png")),
"Chief Executive Assistant", "Management", "27/09/1978", "California", "+800
9899 9931", "Clara@syncfusion.com"));
        Employees.Add(new Person("Maria Even", new BitmapImage(new
Uri("pack://application:,,,/DataBinding;Component/Images/Emp_11.png")),
"Executive Manager", "Operational Unit", "27/09/1970", "New York", "+800
9899 9932", "Maria@syncfusion.com"));
        Employees.Add(new Person("Mark Zune", new BitmapImage(new
Uri("pack://application:,,,/DataBinding;Component/Images/Emp_13.png")),
"Senior Executive", "Operational Unit", "27/09/1983", "Boston", "+800 9899
9933", "Mark@syncfusion.com"));
        Employees.Add(new Person("Robin Ranee", new BitmapImage(new
Uri("pack://application:,,,/DataBinding;Component/Images/Emp_16.png")),
"Manager", "Customer Service", "27/09/1985", "New Jersey", "+800 9899 9934",
"Robin@syncfusion.com"));
    }
}
```

```

Employees.Add(new Person("Chris Marker", new BitmapImage(new
Uri("pack://application:,,,/DataBinding;Component/Images/Emp_21.png")),
"Team Manager", "Customer Service", "27/09/1963", "California", "+800 9899
9935", "Chris@syncfusion.com"));
Employees.Add(new Person("Serra Sum", new BitmapImage(new
Uri("pack://application:,,,/DataBinding;Component/Images/Emp_23.png")),
"Coordinator", "Customer Service", "27/09/1961", "New York", "+800 9899
9936", "Serra@syncfusion.com"));
Employees.Add(new Person("Mathew Fleming", new BitmapImage(new
Uri("pack://application:,,,/DataBinding;Component/Images/Emp_25.png")),
"Recruitment Manager", "Human Resource", "27/09/1986", "Boston", "+800 9899
9937", "Mathew@syncfusion.com"));
}
}

```

- **MainWindow.Xaml**

```

<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>

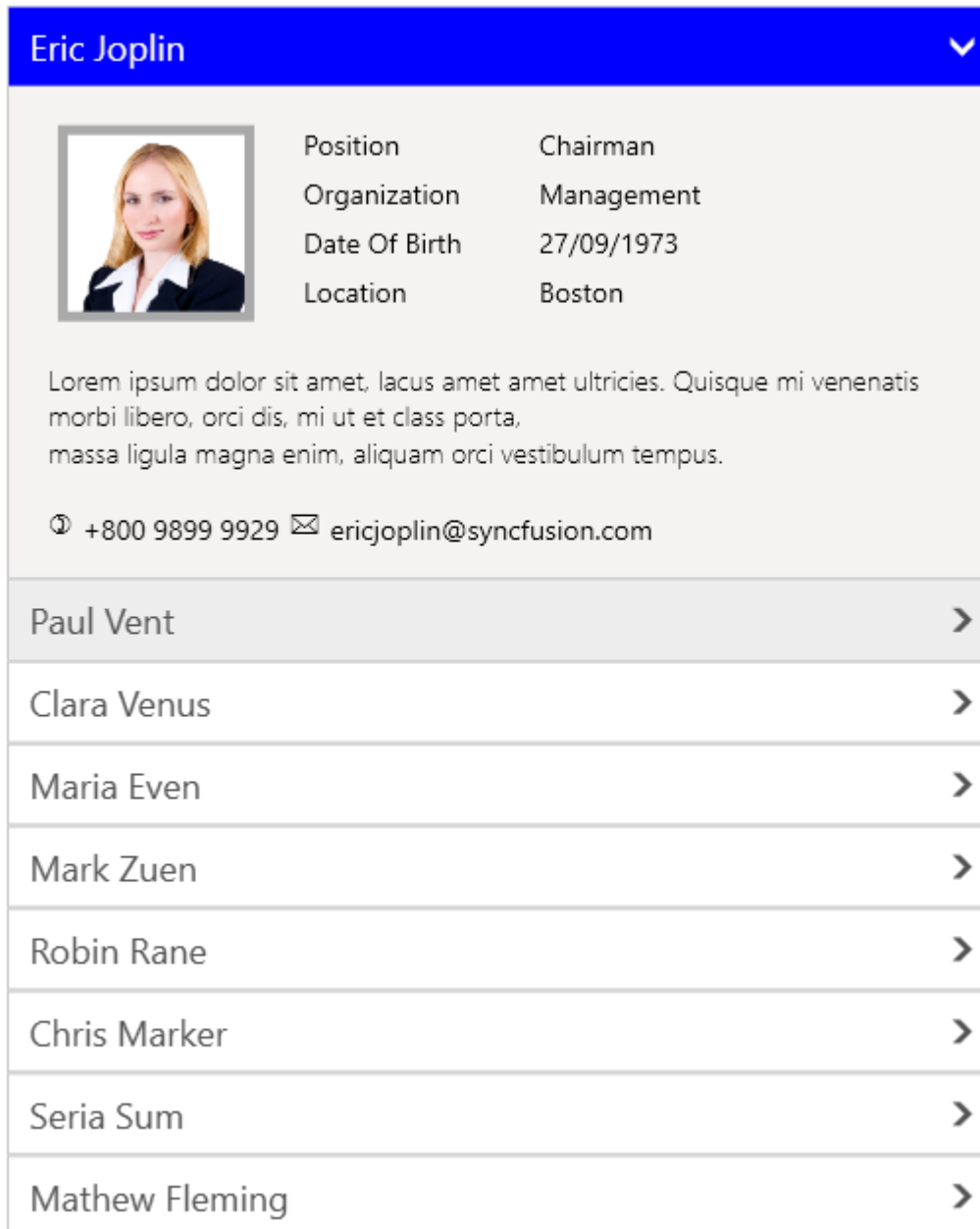
<syncfusion:SfAccordion x:Name="accordion" ItemsSource="{Binding Employees}"
HorizontalAlignment="Center" VerticalAlignment="Center" SelectionMode="ZeroOrOne" >

<syncfusion:SfAccordion.HeaderTemplate>
<DataTemplate>
<Grid>
<TextBlock Text="{Binding Name}" Width="450" FontSize="18" VerticalAlignment="Center" Margin="8"
/>
</Grid>
</DataTemplate>
</syncfusion:SfAccordion.HeaderTemplate>

<syncfusion:SfAccordion.ContentTemplate>
<DataTemplate>
<Grid>
<TextBlock Text="Position " />
<TextBlock Text="{Binding Position}" Grid.Column="1"/>
<TextBlock Text="Organization " Grid.Row="1"/>
<TextBlock Text="{Binding organization}" Grid.Row="1" Grid.Column="1"/>
<TextBlock Text="Date Of Birth " Grid.Row="2"/>
<TextBlock Text="{Binding DateOfBirth}" Grid.Row="2" Grid.Column="1"/>
<TextBlock Text="Location " Grid.Row="3"/>

```

```
<TextBlock Text="{Binding Location}" Grid.Row="3" Grid.Column="1"/>
<TextBlock Grid.ColumnSpan="2" Grid.Row="2" VerticalAlignment="Top" Margin="20" FontSize="14"
FontWeight="Light" >
<TextBlock TextWrapping="Wrap" Text= "Lorem ipsum dolor sit amet, lacus amet amet ultricies.
Quisque mi venenatis morbi libero, orci dis, mi ut et class porta, massa ligula magna enim, aliquam orci
vestibulum tempus."/>
</TextBlock>
<TextBlock Text=")" FontFamily="Wingdings"/>
<TextBlock Text="{Binding Phone}" Grid.Column="1" Margin="5 0" VerticalAlignment="Center"/>
<TextBlock Text="*" FontFamily="Wingdings" Grid.Column="2"/>
<TextBlock Text="{Binding Email}" Grid.Column="3" Margin="5 0" VerticalAlignment="Center"/>
</Grid>
</DataTemplate>
</syncfusion:SfAccordion.ContentTemplate>
</syncfusion:SfAccordion>
```



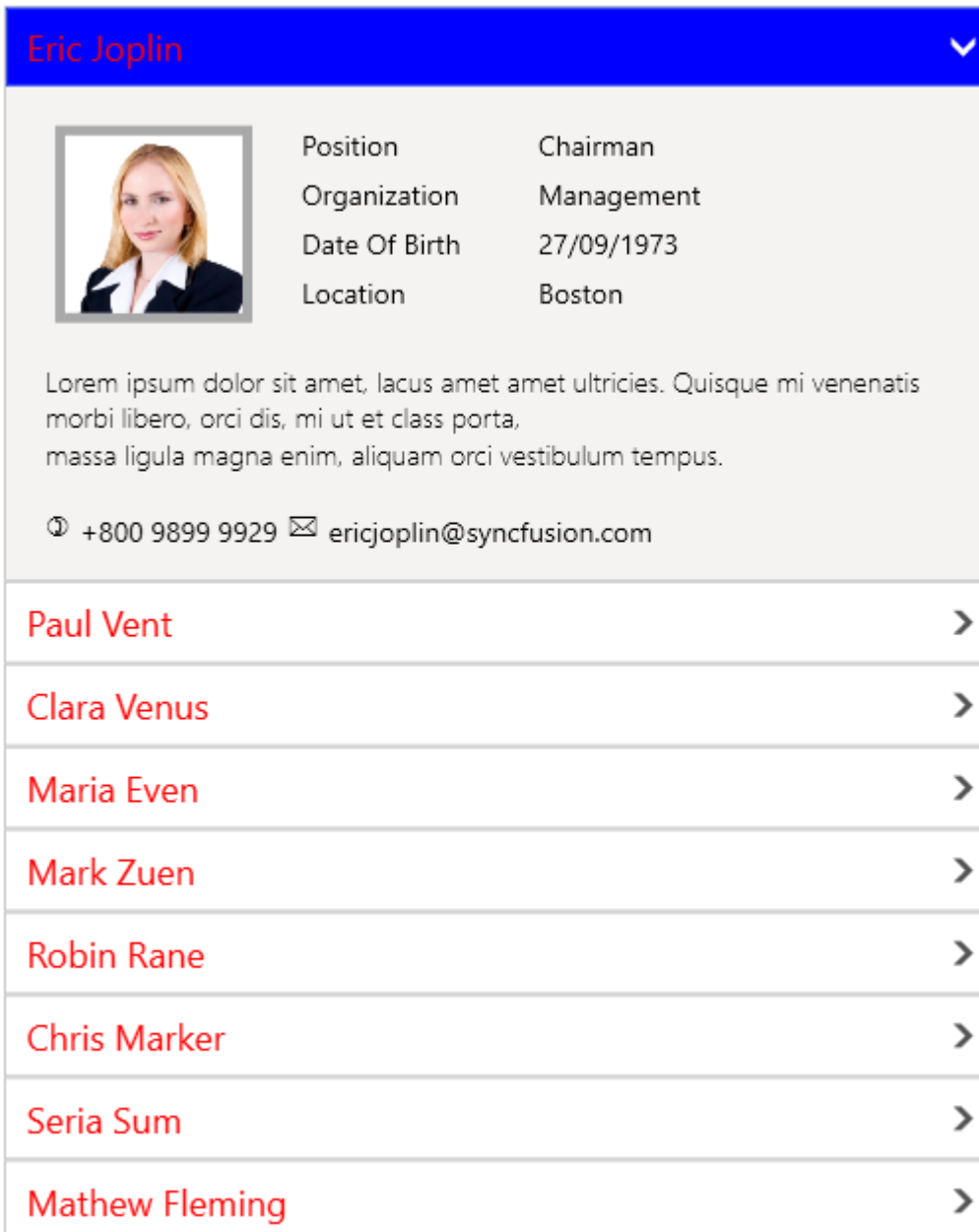
#### Apply template to item header

SfAccordion provides the [HeaderTemplate](#) property; it allows to apply a common data template to all accordion items header.

#### XML

```
<syncfusion:SfAccordion x:Name="accordion">
  <syncfusion:SfAccordion.HeaderTemplate>
    <DataTemplate>
      <Grid>
        <TextBlock Text="{Binding}" Foreground="Red"/>
      </Grid>
    </DataTemplate>
  </syncfusion:SfAccordion.HeaderTemplate>
</syncfusion:SfAccordion>
```

```
</syncfusion:SfAccordion>
```



Set content to children

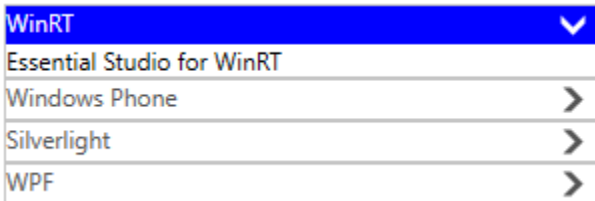
You can set content to SfAccordionItem using the [Content](#) property of SfAccordion.

#### XML

```
<syncfusion:SfAccordion Width="300" Height="300">
  <syncfusion:SfAccordionItem Header="WinRT" Content="Essential Studio for WinRT" />
  <syncfusion:SfAccordionItem Header="Windows Phone" Content="Essential Studio for Windows Phone 7" />
</syncfusion:SfAccordion>
```



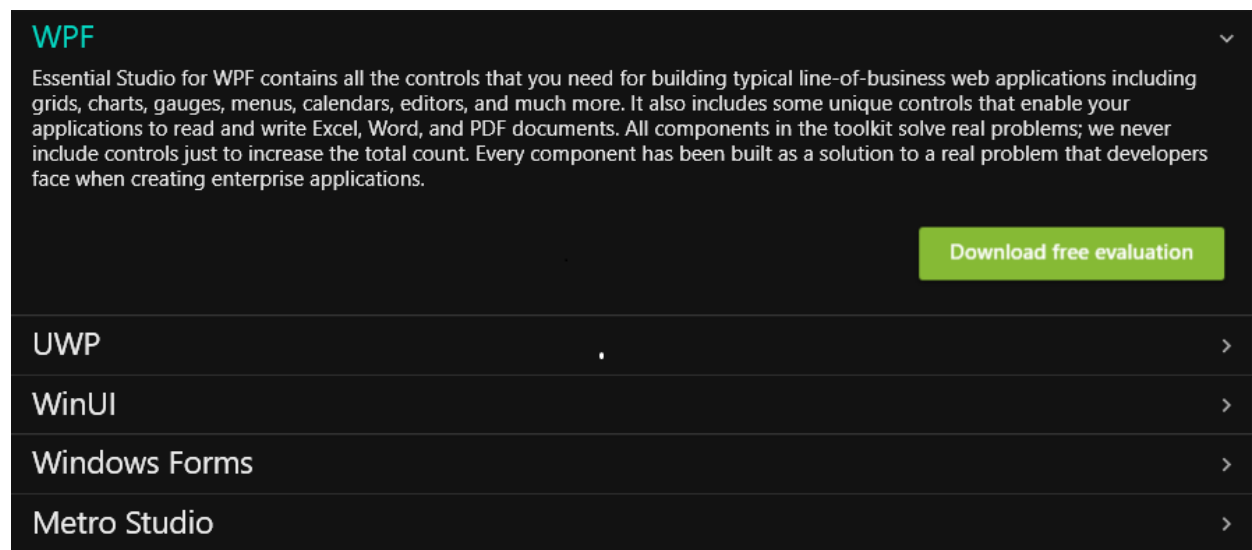
```
<syncfusion:SfAccordionItem Header="Silverlight" Content="Essential Studio
for Silverlight"/>
<syncfusion:SfAccordionItem Header="WPF" Content="Essential Studio for
WPF"/>
</syncfusion:SfAccordion>
```



### Theme

SfAccordion supports various built-in themes. Refer to the below links to apply themes for the SfAccordion,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Populating Items in WPF Accordion (SfAccordion)

SfAccordionItem are added as items of SfAccordion. Items can be added using Items or ItemSource property.

#### Using Items

SfAccordion accepts SfAccordionItem as its children when added directly.

#### Adding items to the control

Here five SfAccordionItems are added as the children of the SfAccordion.

#### XML

```
<layout:SfAccordion>
<layout:SfAccordionItem/>
<layout:SfAccordionItem/>
```

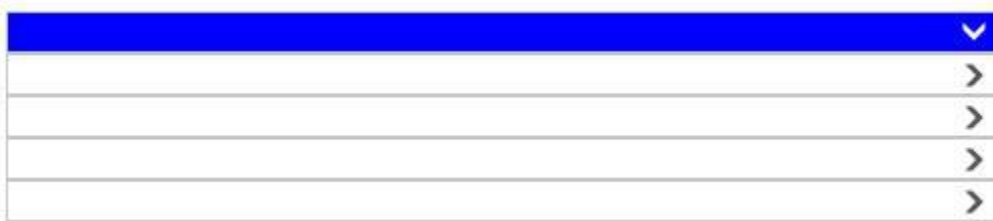
```
<layout:SfAccordionItem/>
<layout:SfAccordionItem/>
<layout:SfAccordionItem/>
</layout:SfAccordion>
```

**C#**

```
SfAccordion accordion = new SfAccordion();
accordion.Items.Add(new SfAccordionItem());
accordion.Items.Add(new SfAccordionItem());
accordion.Items.Add(new SfAccordionItem());
accordion.Items.Add(new SfAccordionItem());
accordion.Items.Add(new SfAccordionItem());
```

**VB.NET**

```
Dim accordion As New SfAccordion()
accordion.Items.Add(New SfAccordionItem())
accordion.Items.Add(New SfAccordionItem())
accordion.Items.Add(New SfAccordionItem())
accordion.Items.Add(New SfAccordionItem())
accordion.Items.Add(New SfAccordionItem())
```

*Setting Header for items*

**SfAccordionItem** provides a property [Header](#) that helps to set the header for the item. [Header](#) is visible in both expanded and collapsed state. Set the value as “WPF” for the first child and repeat the same procedure for the remaining children with values as “Silverlight”, “WinRT”, “Windows Phone” and “Universal”.

**XML**

```
<layout:SfAccordion>
<layout:SfAccordionItem Header="WPF"/>
<layout:SfAccordionItem Header="Silverlight"/>
<layout:SfAccordionItem Header="WinRT"/>
<layout:SfAccordionItem Header="Windows Phone"/>
<layout:SfAccordionItem Header="Universal"/>
</layout:SfAccordion>
```

**C#**

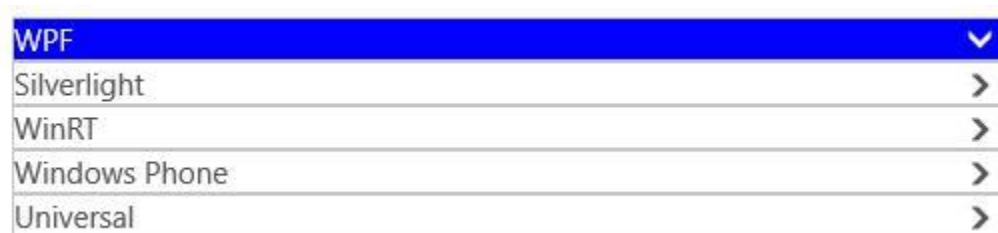
```
SfAccordion accordion = new SfAccordion();
accordion.Items.Add(new SfAccordionItem() { Header = "WPF" });
accordion.Items.Add(new SfAccordionItem() { Header = "Silverlight" });
accordion.Items.Add(new SfAccordionItem() { Header = "WinRT" });
accordion.Items.Add(new SfAccordionItem() { Header = "Windows Phone" });
```

```
accordion.Items.Add(new SfAccordionItem() { Header = "Universal" });
```

## VB.NET

```
Dim accordion As New SfAccordion()
accordion.Items.Add(New SfAccordionItem() With {.Header = "WPF"})
accordion.Items.Add(New SfAccordionItem() With {.Header = "Silverlight"})
accordion.Items.Add(New SfAccordionItem() With {.Header = "WinRT"})
accordion.Items.Add(New SfAccordionItem() With {.Header = "Windows Phone"})
accordion.Items.Add(New SfAccordionItem() With {.Header = "Universal"})
```

SfAccordion control is populated as follows:



### Setting Content for items

[Content](#) property helps to set the content for SfAccordionItem. SfAccordionItem is a ContentControl so that any object can be added as its content. Content is visible only in expanded state.

## XML

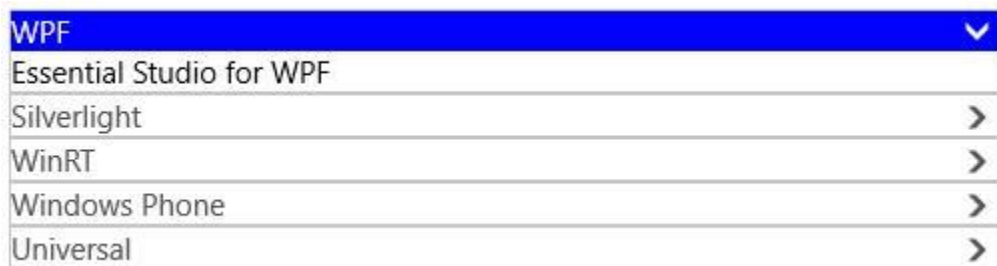
```
<layout:SfAccordion>
<layout:SfAccordionItem Header="WPF" Content="Essential Studio for WPF"/>
<layout:SfAccordionItem Header="Silverlight" Content="Essential Studio for Silverlight"/>
<layout:SfAccordionItem Header="WinRT" Content="Essential Studio for WinRT"/>
<layout:SfAccordionItem Header="Windows Phone" Content="Essential Studio for Windows Phone"/>
<layout:SfAccordionItem Header="Universal" Content="Essential Studio for Universal"/>
</layout:SfAccordion>
```

## C#

```
SfAccordion accordion = new SfAccordion();
accordion.Items.Add(new SfAccordionItem() { Header = "WPF", Content = "Essential Studio for WPF" });
accordion.Items.Add(new SfAccordionItem() { Header = "Silverlight", Content = "Essential Studio for Silverlight" });
accordion.Items.Add(new SfAccordionItem() { Header = "WinRT", Content = "Essential Studio for WinRT" });
accordion.Items.Add(new SfAccordionItem() { Header = "Windows Phone", Content = "Essential Studio for Windows Phone" });
accordion.Items.Add(new SfAccordionItem() { Header = "Universal", Content = "Essential Studio for Universal" });
```

**VB.NET**

```
Dim accordion As New SfAccordion()  
accordion.Items.Add(New SfAccordionItem() With {  
    .Header = "WPF",  
    .Content = "Essential Studio for WPF"  
})  
accordion.Items.Add(New SfAccordionItem() With {  
    .Header = "Silverlight",  
    .Content = "Essential Studio for Silverlight"  
})  
accordion.Items.Add(New SfAccordionItem() With {  
    .Header = "WinRT",  
    .Content = "Essential Studio for WinRT"  
})  
accordion.Items.Add(New SfAccordionItem() With {  
    .Header = "Windows Phone",  
    .Content = "Essential Studio for Windows Phone"  
})  
accordion.Items.Add(New SfAccordionItem() With {  
    .Header = "Universal",  
    .Content = "Essential Studio for Universal"  
})
```

**Using ItemsSource**

SfAccordion accepts any business object collection to be bound to its [ItemsSource](#) property.

*Adding items to the control*

Follow the below steps to add the Items through [ItemsSource](#) property.

**1.Create a model****C#**

```
public class Employee  
{  
    public string Name { get; set; }  
    public string Description { get; set; }  
}
```

**VB.NET**

```
Public Class Employee  
    Public Property Name() As String  
    Public Property Description() As String  
End Class
```

## 2.Create a collection of model

### C#

```
private List<Employee> employees;  
public List<Employee> Employees  
{  
    get { return employees; }  
    set { employees = value; }  
}
```

### VB.NET

```
Private employees_Renamed As List(Of Employee)  
Public Property Employees() As List(Of Employee)  
Get  
    Return employees_Renamed  
End Get  
Set(ByVal value As List(Of Employee))  
    employees_Renamed = value  
End Set  
End Property
```

## 3.Populate the collection

### C#

```
Employees = new List<Employee>();  
Employees.Add(new Employee() { Name = "James", Description = "Description  
about James" });  
Employees.Add(new Employee() { Name = "Linda", Description = "Description  
about Linda" });  
Employees.Add(new Employee() { Name = "Carl", Description = "Description  
about Carl" });  
Employees.Add(new Employee() { Name = "Niko", Description = "Description  
about Niko" });
```

### VB.NET

```
Employees = New List(Of Employee)()  
Employees.Add(New Employee() With {  
    .Name = "James",  
    .Description = "Description about James"  
})  
Employees.Add(New Employee() With {  
    .Name = "Linda",  
    .Description = "Description about Linda"  
})  
Employees.Add(New Employee() With {  
    .Name = "Carl",  
    .Description = "Description about Carl"  
})  
Employees.Add(New Employee() With {  
    .Name = "Niko",
```

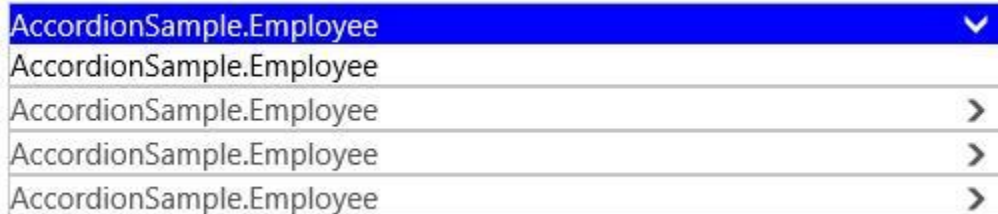
```
.Description = "Description about Niko"
})
```

4. Bind the Employees collection to [ItemsSource](#) property of **SfAccordion** Control

#### XML

```
<layout:SfAccordion ItemsSource="{Binding Employees}"/>
```

**SfAccordion** control is populated as follows:

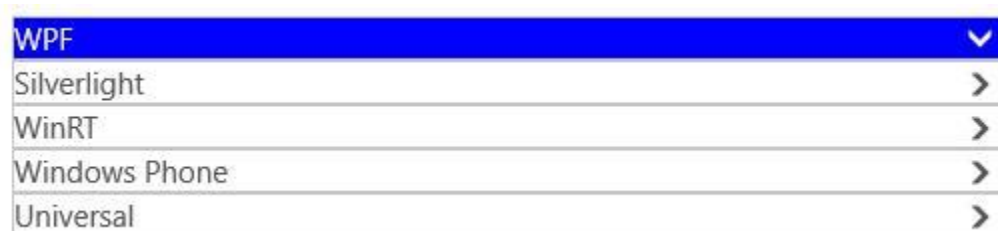


#### Setting Header for items

Header can be displayed using the property [DisplayMemberPath](#). This property is used to get the header from Model class. Header is visible in both expanded and collapsed state.

#### XML

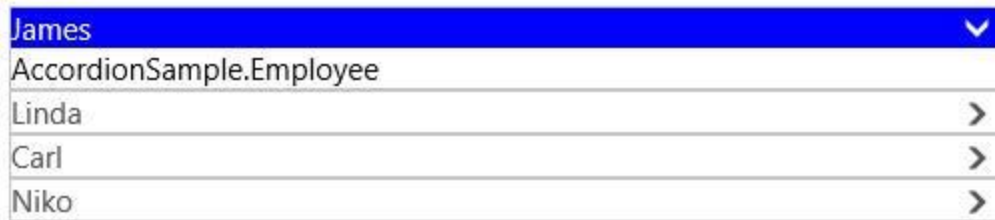
```
<layout:SfAccordion ItemsSource="{Binding Employees}"
    DisplayMemberPath="Name"/>
```



[HeaderTemplate](#) property can also be used to display the header. [HeaderTemplateSelector](#) property is also provided to apply header template based on the selection logic.

#### XML

```
<layout:SfAccordion ItemsSource="{Binding Employees}">
    <layout:SfAccordion.HeaderTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Name}"/>
        </DataTemplate>
    </layout:SfAccordion.HeaderTemplate>
</layout:SfAccordion>
```

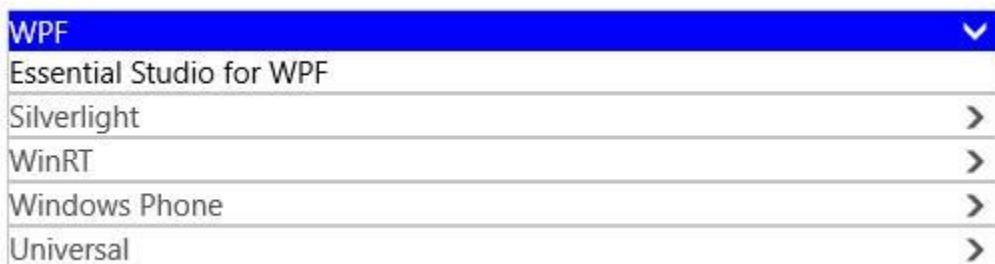


#### Setting Content for items

Content can be displayed using the [ContentTemplate](#) property. Content is visible only in the expanded state. [ContentTemplateSelector](#) property is also provided to apply content template based on the selection logic.

#### XML

```
<layout:SfAccordion ItemsSource="{Binding Employees}"
    DisplayMemberPath="Name">
    <layout:SfAccordion.ContentTemplate>
    <DataTemplate>
    <TextBlock Text="{Binding Description}"/>
    </DataTemplate>
    </layout:SfAccordion.ContentTemplate>
</layout:SfAccordion>
```



#### Selection Mode in WPF Accordion (SfAccordion)

**SfAccordion** provides a property [SelectionMode](#) that helps to decide the number of items that can be expanded or selected at a time. The values of [SelectionMode](#) are

1. One
2. OneOrMore
3. ZeroOrOne
4. ZeroOrMore

The default selection mode is One.

#### SelectionMode - One

- Only one item can be in expanded/selected state.
- One item must be in expanded/selected state.
- Does not allow to unselect all the item.

### SelectionMode - OneOrMore

- More than one item can be in expanded/selected state.
- One item must be in expanded/selected state.
- Does not allow to unselect all the item.

### SelectionMode - ZeroOrOne

- Only one item can be in expanded/selected state.
- Allows to unselect all the item.

### SelectionMode - ZeroOrMore

- More than one item can be in expanded/selected state.
- Allows to unselect all the item.

Here is an example showing OneOrMore [SelectionMode](#):

#### XML

```
<layout:SfAccordion SelectionMode="OneOrMore">
<layout:SfAccordionItem Header="WPF" Content="Essential Studio for WPF"/>
<layout:SfAccordionItem Header="Silverlight" Content="Essential Studio for
Silverlight"/>
<layout:SfAccordionItem Header="WinRT" Content="Essential Studio for
WinRT"/>
<layout:SfAccordionItem Header="Windows Phone" Content="Essential Studio for
Windows Phone"/>
<layout:SfAccordionItem Header="Universal" Content="Essential Studio for
Universal"/>
</layout:SfAccordion>
```

#### C#

```
accordion.SelectionMode = AccordionSelectionMode.OneOrMore;
```

#### VB.NET

```
accordion.SelectionMode = AccordionSelectionMode.OneOrMore
```





### Selecting Items in WPF Accordion (SfAccordion)

Items can be selected programmatically using the properties [SelectedIndex](#), [SelectedItem](#) and [SelectedItems](#).

#### Selecting item using SelectedIndex

[SelectedIndex](#) property is used to select an item using its index. It contains the index of most recently selected item in case of OneOrMore, ZeroOrMore SelectionModes.

#### Selecting item using SelectedItem

[SelectedItem](#) property is used to select an item using its instance. It contains the instance of most recently selected item in case of OneOrMore, ZeroOrMore SelectionModes.

#### Retrieving the selected items

[SelectedItems](#) property contains a collection of selected items instances for all the SelectionModes. It is a read only property and it cannot be set.

#### Retrieving the selected item indices

[SelectedIndices](#) property contains a collection of selected items indices for all the SelectionModes. It is a read only property and it cannot be set.

Here is an example showing the functioning of these properties in which items are selected in run time by touch:

#### XML

```
<Grid>
<StackPanel>
<TextBlock x:Name="selectedItem"/>
<TextBlock x:Name="selectedIndex"/>
<TextBlock x:Name="selectedItems"/>
<TextBlock x:Name="selectedIndices"/>
<layout:SfAccordion SelectionMode="ZeroOrMore" x:Name="accordion"
SelectedItemsChanged="accordion_SelectedItemsChanged">
<layout:SfAccordionItem Header="WPF" Content="Essential Studio for WPF"/>
<layout:SfAccordionItem Header="Silverlight" Content="Essential Studio for
Silverlight"/>
<layout:SfAccordionItem Header="WinRT" Content="Essential Studio for
WinRT"/>
<layout:SfAccordionItem Header="Windows Phone" Content="Essential Studio for
Windows Phone"/>
```

```
<layout:SfAccordionItem Header="Universal" Content="Essential Studio for
Universal"/>
</layout:SfAccordion>
</StackPanel>
</Grid>
```

**C#**

```
private void accordion_SelectedItemsChanged(object sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
{
    string items = string.Empty;
    string indices = string.Empty;
    foreach (var item in accordion.SelectedItems)
        items += (item as SfAccordionItem).Header + " , ";
    foreach (var item in accordion.SelectedIndices)
        indices += item + " , ";
    selectedItem.Text = "SelectedItem is : " + (accordion.SelectedItem as
SfAccordionItem).Header;
    selectedIndex.Text = "SelectedIndex is : " + accordion.SelectedIndex;
    selectedItem.Text = "SelectedItems are : " + items;
    selectedIndex.Text = "SelectedIndices are : " + indices;
}
```

**VB.NET**

```
Option Infer On
Private Sub accordion_SelectedItemsChanged(ByVal sender As Object, ByVal e
As System.Collections.Specialized.NotifyCollectionChangedEventArgs)
    Dim items As String = String.Empty
    Dim indices As String = String.Empty
    For Each item In accordion.SelectedItems
        items &= (TryCast(item, SfAccordionItem)).Header & " , "
    Next item
    For Each item In accordion.SelectedIndices
        indices &= item & " , "
    Next item
    selectedItem.Text = "SelectedItem is : " & (TryCast(accordion.SelectedItem,
SfAccordionItem)).Header
    selectedIndex.Text = "SelectedIndex is : " & accordion.SelectedIndex
    selectedItem.Text = "SelectedItems are : " & items
    selectedIndex.Text = "SelectedIndices are : " & indices
End Sub
```



### Selecting item using IsSelected

**SfAccordionItem** has a property [IsSelected](#) that determines whether the item is expanded or collapsed. More than one accordion item can have IsSelected as **True** based on the [SelectionMode](#).

- IsSelected=true – Item is expanded
- IsSelected=false – Item is collapsed

### XML

```
<layout:SfAccordion>
<layout:SfAccordionItem Header="Linda" IsSelected="True"
Content="Description about Linda">
</layout:SfAccordionItem>
</layout:SfAccordion>
```

### C#

```
SfAccordion accordion = new SfAccordion();
accordion.Items.Add(new SfAccordionItem() { Header = "Linda",
Content = "Description about Linda" , IsSelected = true });
```

### VB.NET

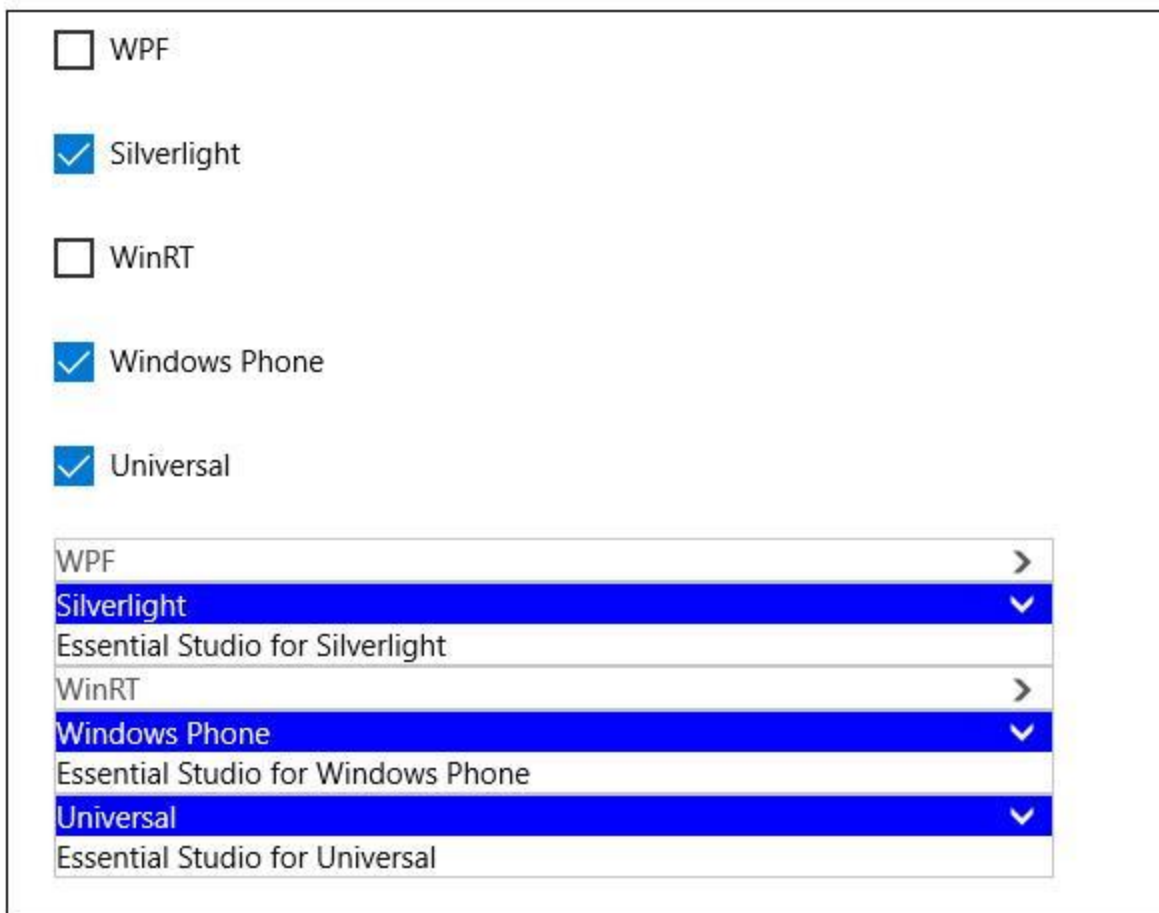
```
Dim accordion As New SfAccordion()
accordion.Items.Add(New SfAccordionItem() With {
.Header = "Linda",
.Content = "Description about Linda",
.IsSelected = True
```

```
}}
```

Here is an example showing the behavior of this property by binding [SfAccordionItem.IsSelected](#) property to `CheckBox.IsChecked` property in Two-way.

#### XML

```
<Grid>
<StackPanel>
<CheckBox Content="WPF" IsChecked="{Binding
ElementName=wpf, Path=IsSelected, Mode=TwoWay}" />
<CheckBox Content="Silverlight" IsChecked="{Binding
ElementName=silverlight, Path=IsSelected, Mode=TwoWay}" />
<CheckBox Content="WinRT" IsChecked="{Binding
ElementName=winrt, Path=IsSelected, Mode=TwoWay}" />
<CheckBox Content="Windows Phone" IsChecked="{Binding
ElementName=phone, Path=IsSelected, Mode=TwoWay}" />
<CheckBox Content="Universal" IsChecked="{Binding
ElementName=universal, Path=IsSelected, Mode=TwoWay}" />
<layout:SfAccordion SelectionMode="ZeroOrMore">
<layout:SfAccordionItem x:Name="wpf" Header="WPF" Content="Essential Studio
for WPF" />
<layout:SfAccordionItem x:Name="silverlight" Header="Silverlight"
Content="Essential Studio for Silverlight" />
<layout:SfAccordionItem x:Name="winrt" Header="WinRT" Content="Essential
Studio for WinRT" />
<layout:SfAccordionItem x:Name="phone" IsSelected="True" Header="Windows
Phone"
Content="Essential Studio for Windows Phone" />
<layout:SfAccordionItem x:Name="universal" Header="Universal"
Content="Essential Studio for Universal" />
</layout:SfAccordion>
</StackPanel>
</Grid>
```



#### Checking the lock state of an item

**SfAccordionItem** provides a read-only property [IsLocked](#) to check whether an item is locked or not. An accordion item is said to be locked when it cannot be unselected/collapsed.

For example: In One SelectionMode, the selected item cannot be collapsed directly by clicking on its header, it is locked. It can be unlocked by selecting another accordion item, now the newly selected item is locked.

#### Select All Items

**SfAccordion** provides a method [SelectAll](#) to select all the items. In One and ZeroOrOne SelectionModes, only the last item is selected.

#### C#

```
accordion.SelectAll();
```

#### VB.NET

```
accordion.SelectAll()
```

### Unselect All Items

**SfAccordion** provides a method [UnselectAll](#) to unselect all the items. In One [SelectionMode](#), there is no change in calling this method. In OneOrMore [SelectionMode](#), the element which has higher index remains selected whereas others are unselected.

### C#

```
accordion.UnselectAll();
```

### VB.NET

```
accordion.SelectAll()
```

### Notifying selected item change

[SelectedItemChanged](#) event is fired whenever an item is expanded or collapsed. The arguments of the event are

S.No	Argument	Item expanded	Item collapsed
1	OldStartingIndex	-1	Index of collapsed item
2	OldItems	null	Instance of collapsed
3	NewStartingIndex	SelectedIndex	-1
4	NewItems	SelectedItem	null

Here is an example to demonstrate the values of event arguments:

### XML

```
<Grid>
<StackPanel>
<TextBlock x:Name="oldStartingIndex"/>
<TextBlock x:Name="oldItems"/>
<TextBlock x:Name="newStartingIndex"/>
<TextBlock x:Name="newItems"/>
<TextBlock x:Name="selectedIndex"/>
<TextBlock x:Name="selectedItem"/>
<layout:SfAccordion SelectionMode="ZeroOrMore" x:Name="accordion"
SelectedItemChanged="accordion_SelectedItemsChanged">
<layout:SfAccordionItem Header="WPF" Content="Essential Studio for WPF"/>
<layout:SfAccordionItem Header="Silverlight"
Content="Essential Studio for Silverlight"/>
<layout:SfAccordionItem Header="WinRT" Content="Essential Studio for
WinRT"/>
<layout:SfAccordionItem Header="Windows Phone" Content="Essential Studio for
Windows Phone"/>
<layout:SfAccordionItem Header="Universal" Content="Essential Studio for
Universal"/>
</layout:SfAccordion>
</StackPanel>
</Grid>
```

**C#**

```

private void accordion_SelectedItemsChanged(object sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
{
    string olditems = string.Empty;
    string newitems = string.Empty;
    if (e.OldItems != null)
    {
        foreach (var item in e.OldItems)
            olditems += (item as SfAccordionItem).Header + " , ";
    }
    else
        olditems = "null";
    if (e.NewItems != null)
    {
        foreach (var item in e.NewItems)
            newitems += (item as SfAccordionItem).Header + " , ";
    }
    else
        newitems = "null";
    oldStartingIndex.Text = "OldStartingIndex is : " + e.OldStartingIndex;
    newStartingIndex.Text = "NewStartingIndex is : " + e.NewStartingIndex;
    selectedIndex.Text = "SelectedIndex is : " + accordion.SelectedIndex;
    selectedItem.Text = "SelectedItem is : " + ((accordion.SelectedItem ==
null)? "null":(accordion.SelectedItem as SfAccordionItem).Header);
    oldItems.Text = "OldItems are : " + olditems;
    newItems.Text = "NewItems are : " + newitems;
}

```

**VB.NET**

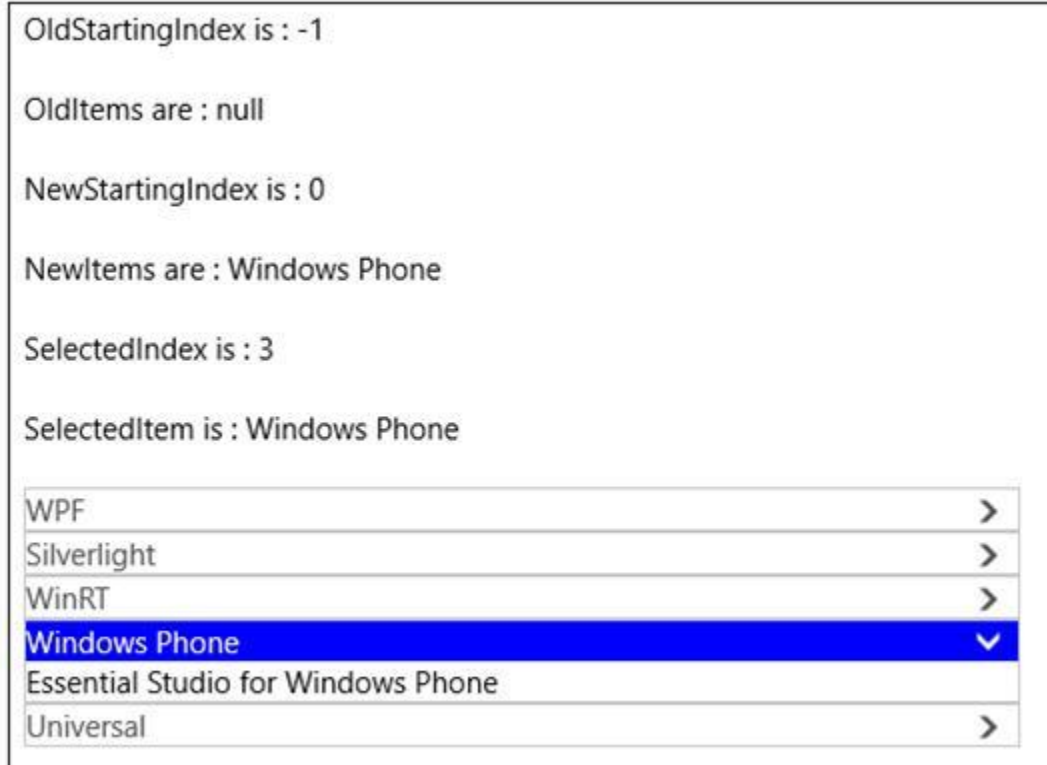
```

Option Infer On
Private Sub accordion_SelectedItemsChanged(ByVal sender As Object, ByVal e
As System.Collections.Specialized.NotifyCollectionChangedEventArgs)
    Dim olditems As String = String.Empty
    Dim newitems As String = String.Empty
    If e.OldItems IsNot Nothing Then
        For Each item In e.OldItems
            olditems &= (TryCast(item, SfAccordionItem)).Header & " , "
        Next item
    Else
        olditems = "null"
    End If
    If e.NewItems IsNot Nothing Then
        For Each item In e.NewItems
            newitems &= (TryCast(item, SfAccordionItem)).Header & " , "
        Next item
    Else
        newitems = "null"
    End If
    oldStartingIndex.Text = "OldStartingIndex is : " & e.OldStartingIndex
    newStartingIndex.Text = "NewStartingIndex is : " & e.NewStartingIndex
    selectedIndex.Text = "SelectedIndex is : " & accordion.SelectedIndex
    selectedItem.Text = "SelectedItem is : " & (If(accordion.SelectedItem Is
Nothing, "null", (TryCast(accordion.SelectedItem, SfAccordionItem)).Header))

```

```
oldItems.Text = "OldItems are : " & olditems  
newItems.Text = "NewItems are : " & newitems  
End Sub
```

Expand the item with header Windows Phone



Collapse the item with header Windows Phone



OldStartingIndex is : 0  
OldItems are : Windows Phone  
NewStartingIndex is : -1  
NewItems are : null  
SelectedIndex is : -1  
SelectedItem is : null

WPF	>
Silverlight	>
WinRT	>
Windows Phone	>
Universal	>

Notifying an item selection

[SfAccordionItem.Selected](#) event is fired whenever the item is selected/expanded.

#### XML

```
<layout:SfAccordionItem x:Name="wpf" Selected="Selected"
Header="WPF" Content="Essential Studio for WPF"/>
```

#### C#

```
private void Selected(object sender, RoutedEventArgs e)
{
}
```

#### VB.NET

```
Private Sub Selected(ByVal sender As Object, ByVal e As RoutedEventArgs)
End Sub
```

Notifying an item un-selection

[SfAccordionItem.Unselected](#) event is fired whenever the item is unselected/collapsed.

#### XML

```
<layout:SfAccordionItem x:Name="wpf" Unselected="Unselected"
Header="WPF" Content="Essential Studio for WPF"/>
```

#### C#

```
private void Unselected(object sender, RoutedEventArgs e)
```

```
{  
}
```

### VB.NET

```
Private Sub Unselected(ByVal sender As Object, ByVal e As RoutedEventArgs)  
End Sub
```

Notifying selection change

[SelectionChanged](#) event fires when an item is selected and unselected. It behaves same as that of [SelectedItemChanged](#) event.

The difference between these two events are the event argument parameters. The parameters of [SelectionChanged](#) event are AddedItems and RemovedItems. Added items have the list of recently selected items whereas RemovedItems have the list of recently unselected items.

### Appearance and Styling in WPF Accordion (SfAccordion)

Applying accent colors

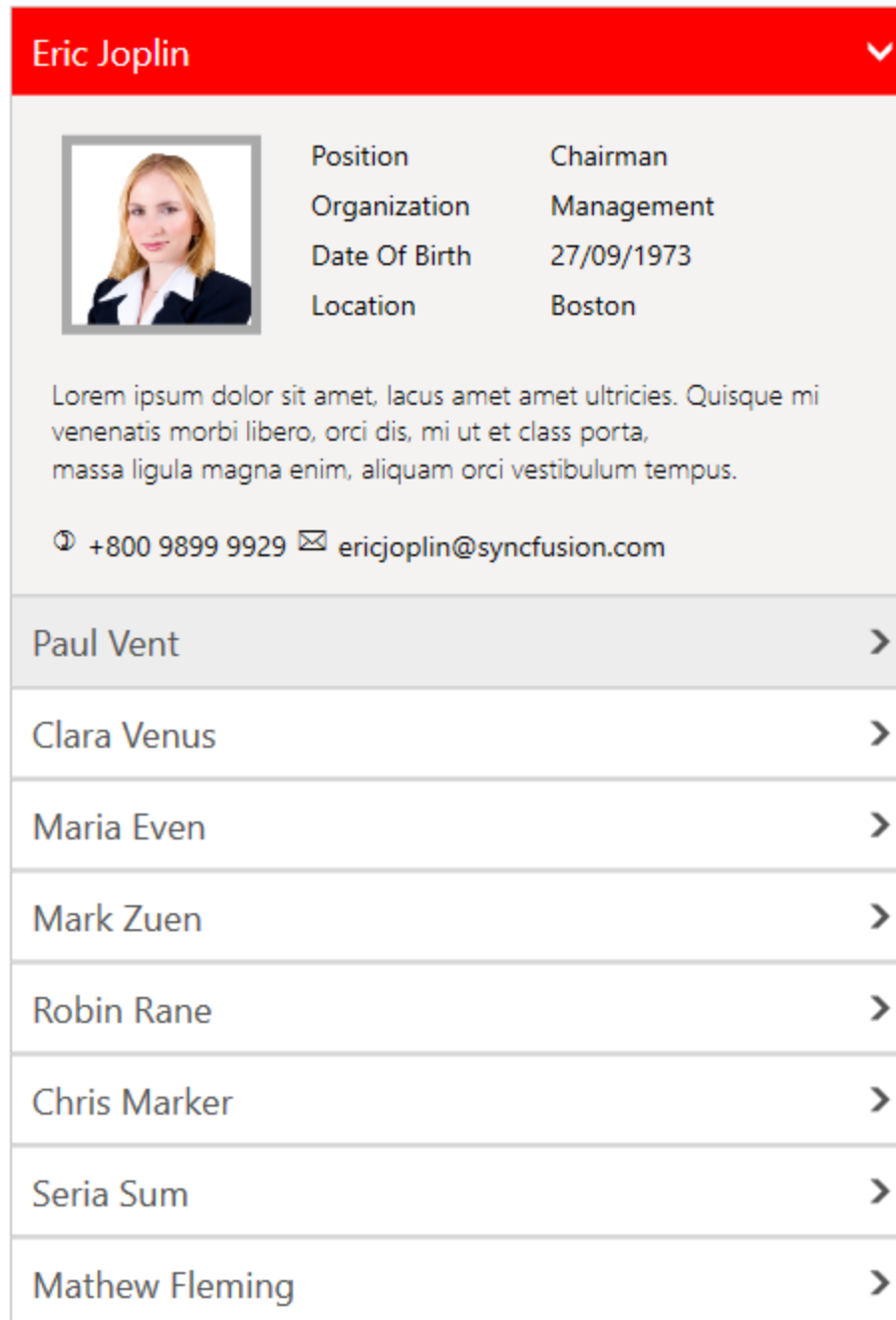
SfAccordion supports accent colors to highlight the hot spots of the control. You can customize the accent colors using the [AccentBrush](#) property.

### XML

```
<syncfusion:SfAccordion x:Name="accordion" AccentBrush="Red"  
HorizontalAlignment="Center" VerticalAlignment="Center"/>
```

### C#

```
accordion.AccentBrush = new SolidColorBrush() { Color =  
Windows.UI.Colors.Red };
```



#### Accordion header style

You can customize the appearance of SfAccordionItem header by setting the [HeaderTemplate](#) property of SfAccordion.

#### XML


```
<syncfusion:SfAccordion HorizontalAlignment="Center" Height="250"
VerticalAlignment="Center" Width="300">
<!-- For customizing header -->
<syncfusion:SfAccordion.HeaderTemplate>
<DataTemplate>
```

```

<TextBlock Text="{Binding}" Foreground="Red" Opacity="1"
FontFamily="Calibri" FontWeight="Bold" FontSize="20"/>
</DataTemplate>
</syncfusion:SfAccordion.HeaderTemplate>
<syncfusion:SfAccordionItem Header="WindowsForms"/>
<syncfusion:SfAccordionItem Header="WPF"/>
<syncfusion:SfAccordionItem Header="UWP"/>
<syncfusion:SfAccordionItem Header="WinRT"/>
</syncfusion:SfAccordion>

```

Eric Joplin



Position	Chairman
Organization	Management
Date Of Birth	27/09/1973
Location	Boston

Lorem ipsum dolor sit amet, lacus amet amet ultricies. Quisque mi  
 venenatis morbi libero, orci dis, mi ut et class porta,  
 massa ligula magna enim, aliquam orci vestibulum tempus.

☎ +800 9899 9929 ✉ ericjoplin@syncfusion.com

Paul Vent

>

Clara Venus

>

Maria Even

>

Mark Zuen

>

Robin Rane

>

Chris Marker

>

Seria Sum

>

Mathew Fleming

>

### Accordion expander style

You can customize the appearance of expander button by writing style of [AccordionButton](#), the edited style can be applied to accordion item by setting the [AccordionButtonStyle](#) property of SfAccordionItem.

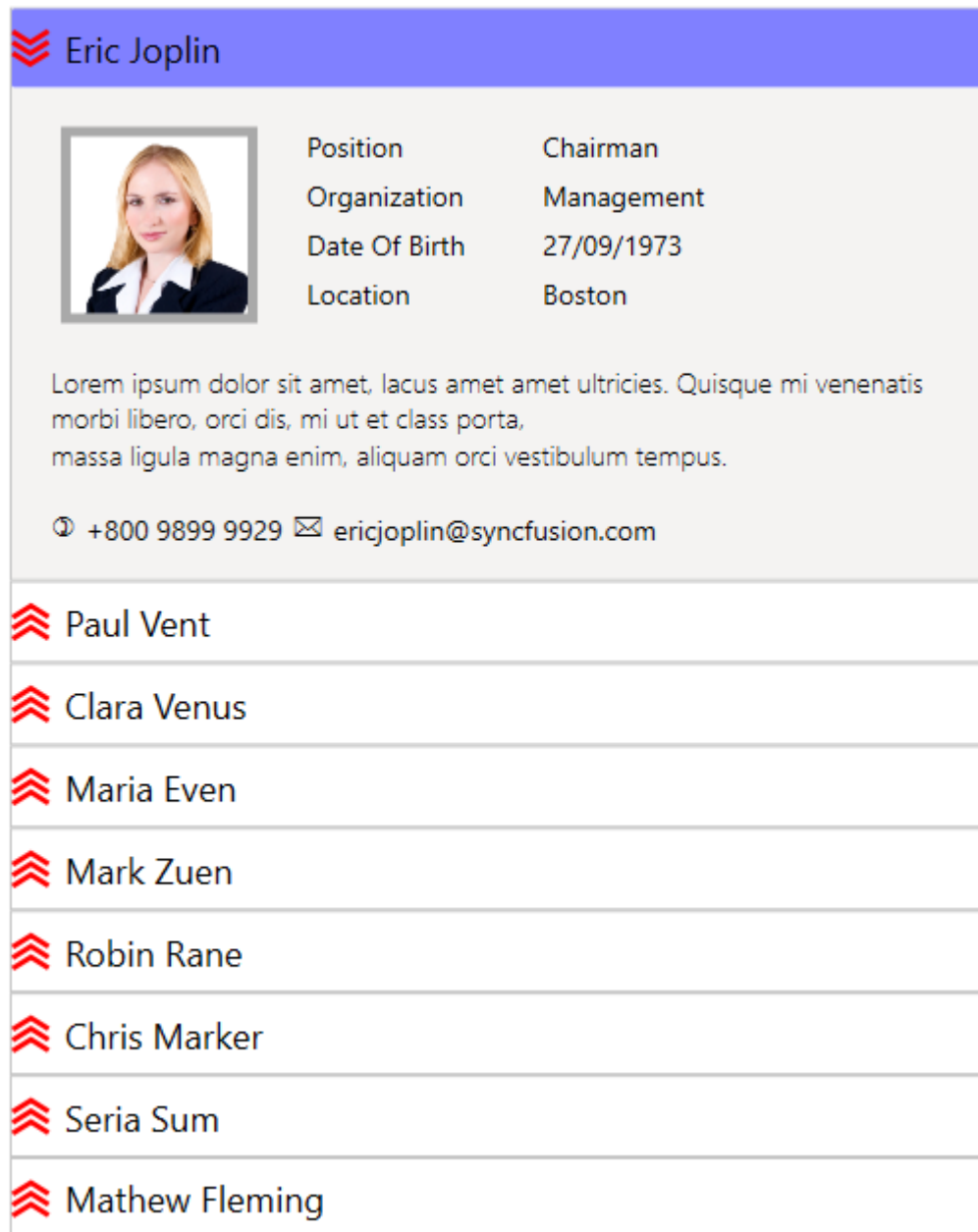
### XML

```
<!-- AccordionButton Style -->
<Style TargetType="syncfusion:AccordionButton" x:Key="expanderButtonStyle">
  <!-- Customization codes -->
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="ExpansionStates">
      <VisualState x:Name="Collapsed">
        <Storyboard>
          <DoubleAnimation BeginTime="00:00:00" Duration="00:00:00.3"
Storyboard.TargetName="icon"
Storyboard.TargetProperty="(UIElement.RenderTransform).(TransformGroup.Children)[2].(RotateTransform.Angle)" To="0" />
        </Storyboard>
      </VisualState>
      <VisualState x:Name="Expanded">
        <Storyboard>
          <DoubleAnimation BeginTime="00:00:00" Duration="00:00:00.3"
Storyboard.TargetName="icon"
Storyboard.TargetProperty="(UIElement.RenderTransform).(TransformGroup.Children)[2].(RotateTransform.Angle)" To="90" />
          <ColorAnimationUsingKeyFrames BeginTime="00:00:00"
Duration="00:00:00.0010000" Storyboard.TargetName="ExpandedBackground"
Storyboard.TargetProperty="(Border.Background).(SolidColorBrush.Color)">
            <SplineColorKeyFrame KeyTime="00:00:00" Value="DeepPink" />
          </ColorAnimationUsingKeyFrames>
          <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
Duration="00:00:00.0010000" Storyboard.TargetName="ExpandedBackground"
Storyboard.TargetProperty="(UIElement.Opacity)">
            <SplineDoubleKeyFrame KeyTime="00:00:00" Value="0.5" />
          </DoubleAnimationUsingKeyFrames>
        </Storyboard>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
  <Border x:Name="background" Background="{Binding Background}"
CornerRadius="1,1,1,1">
    <Grid>
      <Border Height="Auto" Margin="0,0,0,0" x:Name="ExpandedBackground"
VerticalAlignment="Stretch" Opacity="0" Background="Red"
BorderBrush="Yellow" BorderThickness="{TemplateBinding BorderThickness}"
CornerRadius="1,1,1,1" />
      <Grid Height="19" HorizontalAlignment="Center" x:Name="icon"
VerticalAlignment="Center" Width="19" RenderTransformOrigin="0.5,0.5"
Grid.Column="0" Grid.Row="0">
        <Grid.RenderTransform>
          <TransformGroup>
            <ScaleTransform />
            <SkewTransform />
            <RotateTransform Angle="-90" />
            <TranslateTransform />
          </TransformGroup>
        </Grid.RenderTransform>
      </Grid>
    </Border>
  </Border>
</Style>
```

```

<Path Height="Auto" HorizontalAlignment="Center" Margin="0,0,0,0"
x:Name="arrow" VerticalAlignment="Center" Width="Auto"
RenderTransformOrigin="0.5,0.5" Stroke="Red" StrokeThickness="1.5"
Stretch="Uniform" Data="M16.365994,20.000013L32.000027,29.802015
30.936978,31.497023 16.368008,22.360976 1.0660061,32.000013 2.7179741E-
05,30.308973z M16.366,10L32.000001,19.802 30.937001,21.497
16.368001,12.361001 1.0659999,22 0,20.309z M16.366,0L32.000001,9.802
30.937001,11.497001 16.368001,2.3610001 1.0659999,12 0,10.309z"/>
</Grid>
<ContentPresenter HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}" x:Name="header" Grid.Column="1" Grid.Row="0"
Grid.RowSpan="1" Content="{TemplateBinding Content}"
ContentTemplate="{TemplateBinding ContentTemplate}" />
</Grid>
</Border>
</Style>
<!--SfAccordion Control -->
<syncfusion:SfAccordion x:Name="accordion1" HorizontalAlignment="Right"
VerticalAlignment="Center" Width="180">
<!-- Setting AccordionButtonStyle -->
<syncfusion:SfAccordion.ItemContainerStyle>
<Style TargetType="syncfusion:SfAccordionItem">
<Setter Property="AccordionButtonStyle" Value="{StaticResource
expanderButtonStyle}"/>
</Style>
</syncfusion:SfAccordion.ItemContainerStyle>
</syncfusion:SfAccordion>

```

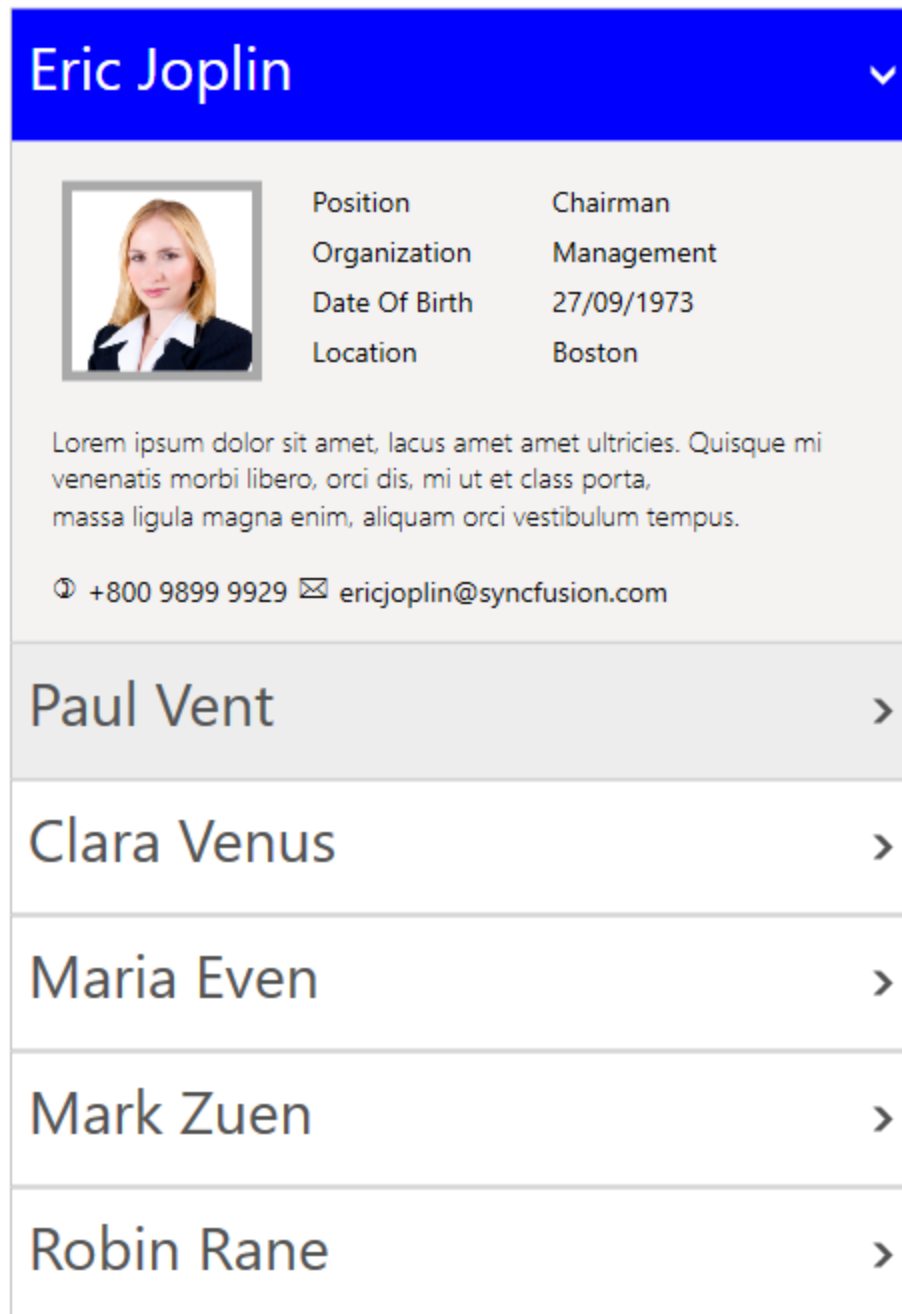


#### AccordionItem header height customization

You can customize the height of SfAccordionItem header by setting the [HeaderTemplate](#) property of SfAccordion.

#### XML

```
<!-- For customizinng header -->
<syncfusion:SfAccordion.HeaderTemplate>
  <DataTemplate>
    <TextBlock Text="{Binding Name}" Height="50" FontSize="30"
      VerticalAlignment="Center"/>
  </DataTemplate>
</syncfusion:SfAccordion.HeaderTemplate>
```



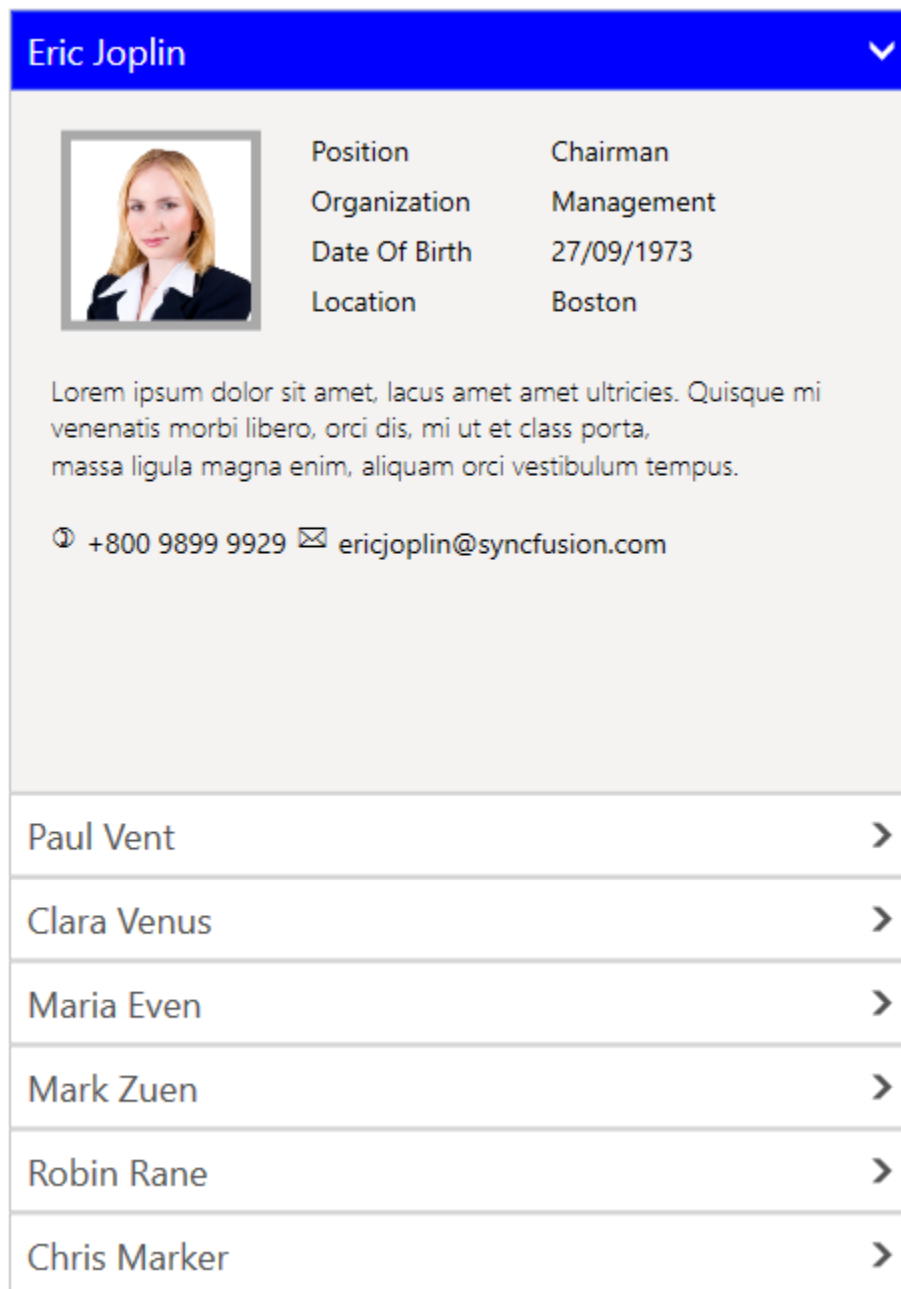
#### AccordionItem content height customization

You can customize the height of SfAccordionItem content by setting the [ContentTemplate](#) property of SfAccordion.

#### XML

```
<!-- For customizinng SfAccordionItem content -->
<syncfusion:SfAccordion.ContentTemplate>
  <DataTemplate>
    <TextBlock Text="{Binding Description}" Height="60"/>
  </DataTemplate>
</syncfusion:SfAccordion.ContentTemplate>
```





#### Enable or disable the animation behaviour

You can enable or disable the animation behaviour when its item is expanded or collapsed. It can be achieved by setting [TargetSize](#) and [Percentage](#) properties in [ExpandableContentControl](#) and writing the animation style in [SfAccordionItem](#), the edited style can be applied by using the [ItemContainerStyle](#) property of the SfAccordion control.

#### XML

```
<!-- SfAccordionItem Style -->
<Style x:Key="animationStyle" TargetType="syncfusion:SfAccordionItem">
<!-- Customization codes -->
```


```

<VisualStateManager.VisualStateGroups>
  <!-- Animation Style -->
  <VisualStateGroup x:Name="ExpansionStates">
    <VisualStateGroup.Transitions>
      <VisualTransition GeneratedDuration="0"/>
    </VisualStateGroup.Transitions>
    <!-- enable animation-->
    <VisualState x:Name="Collapsed">
      <Storyboard>
        <DoubleAnimationUsingKeyFrames BeginTime="00:00:01" Duration="00:00:10"
          Storyboard.TargetName="ExpandSite"
          Storyboard.TargetProperty="(layout:ExpandableContentControl.Percentage)">
          <SplineDoubleKeyFrame KeySpline="0.2,0,0,1" KeyTime="00:00:10.0" Value="0"
            />
        </DoubleAnimationUsingKeyFrames>
      </Storyboard>
    </VisualState>
    <VisualState x:Name="Expanded">
      <Storyboard>
        <DoubleAnimationUsingKeyFrames BeginTime="00:00:01" Duration="00:00:10"
          Storyboard.TargetName="ExpandSite"
          Storyboard.TargetProperty="(layout:ExpandableContentControl.Percentage)">
          <SplineDoubleKeyFrame KeySpline="0.2,0,0,1" KeyTime="00:00:10.0" Value="1"
            />
        </DoubleAnimationUsingKeyFrames>
      </Storyboard>
    </VisualState>
  </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border x:Name="Background" Padding="{TemplateBinding Padding}"
  BorderBrush="{TemplateBinding BorderBrush}"
  BorderThickness="{TemplateBinding BorderThickness}" CornerRadius="1,1,1,1">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>
    <syncfusion:AccordionButton Name="ExpanderButton" Style="{TemplateBinding
      AccordionButtonStyle}" Content="{TemplateBinding Header}"
      ContentTemplate="{TemplateBinding HeaderTemplate}"
      IsChecked="{TemplateBinding IsSelected}" IsTabStop="True" Grid.Row="0"
      Padding="0,0,0,0" Margin="0,0,0,0" FontFamily="{TemplateBinding FontFamily}"
      FontSize="{TemplateBinding FontSize}" FontStretch="{TemplateBinding
        FontStretch}" FontStyle="{TemplateBinding FontStyle}"
      FontWeight="{TemplateBinding FontWeight}" Foreground="{TemplateBinding
        Foreground}" HorizontalContentAlignment="{TemplateBinding
        HorizontalContentAlignment}" VerticalContentAlignment="{TemplateBinding
        VerticalContentAlignment}" HorizontalAlignment="{TemplateBinding
        HorizontalAlignment}" VerticalAlignment="{TemplateBinding
        VerticalAlignment}" Background="{TemplateBinding Background}" />
    <syncfusion:ExpandableContentControl Name="ExpandSite" Grid.Row="1"
      IsTabStop="False" Percentage="0" Content="{TemplateBinding Content}"
      ContentTemplate="{TemplateBinding ContentTemplate}" Margin="0,0,0,0"

```

```
Style="{StaticResource expandableContentStyle}" FontFamily="{TemplateBinding
FontFamily}" FontSize="{TemplateBinding FontSize}"
FontStretch="{TemplateBinding FontStretch}" FontStyle="{TemplateBinding
FontStyle}" FontWeight="{TemplateBinding FontWeight}"
Foreground="{TemplateBinding Foreground}"
HorizontalContentAlignment="{TemplateBinding HorizontalContentAlignment}"
VerticalContentAlignment="{TemplateBinding VerticalContentAlignment}"
HorizontalAlignment="{TemplateBinding HorizontalAlignment}"
VerticalAlignment="{TemplateBinding VerticalAlignment}" />
</Grid>
</Border>
</Style>
<!--SfAccordion Control -->
<syncfusion:SfAccordion x:Name="accordion1" HorizontalAlignment="Center"
VerticalAlignment="Center" ItemContainerStyle="{StaticResource
animationStyle}" />
```


Eric Joplin



Position	Chairman
Organization	Management
Date Of Birth	27/09/1973
Location	Boston

Lorem ipsum dolor sit amet, lacus amet amet ultricies. Quisque mi venenatis morbi libero, orci dis, mi ut et class porta, massa ligula magna enim, aliquam orci vestibulum tempus.


Paul Vent



Position	Chief Executive Officer
Organization	Management
Date Of Birth	27/09/1975
Location	New York

Lorem ipsum dolor sit amet, lacus amet amet ultricies. Quisque mi venenatis morbi libero, orci dis, mi ut et class porta,

Clara Venus



Position	Chief Executive Assistant
Organization	Management
Date Of Birth	27/09/1978
Location	California

Maria Even

>

Mark Zuen

>

Robin Rane

>

Chris Marker

>

Seria Sum

>

Mathew Fleming

>

## SfBarcode

### WPF Barcode (SfBarcode) Overview

The Barcode control helps rendering bar codes in desktop (WPF) application. The control can be merged with into any desktop application and easy to encode text using the supported symbol types. The basic structure of a bar code consists of a leading and trailing quiet zone, a start pattern, one or more data characters, optionally one or two check characters, and a stop pattern.



Barcode control rendering 1D bar code



Barcode control rendering 2D bar code

### Structure of the Control



### Structure of Barcode control

### Getting Started with WPF Barcode (SfBarcode)

#### Add Barcode control to an Application

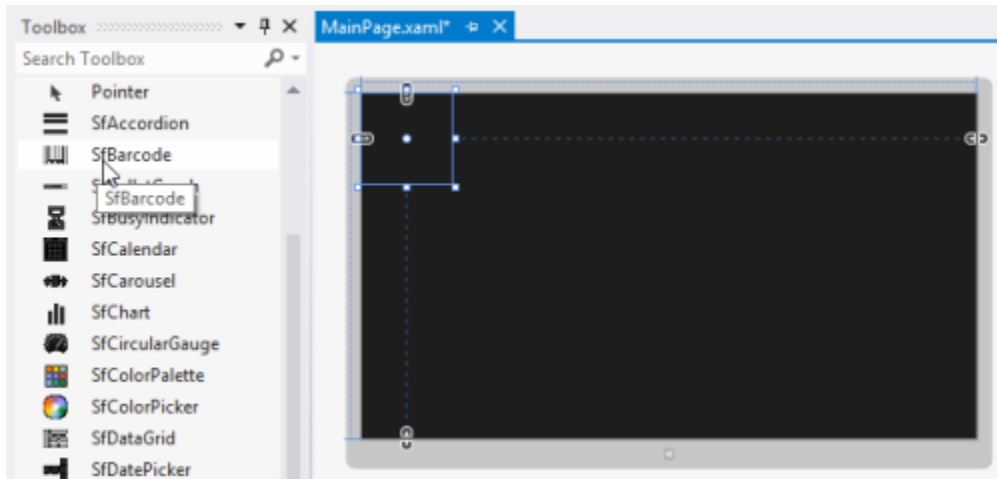
The following assembly reference is required for deploying Barcode.

#### C#

```
Namespace: Syncfusion.UI.Xaml.Controls.Barcode  
Assembly: Syncfusion.SfBarcode.WPF
```

To create the SfBarcode control in Visual Studio:

1. Create a new WPF project.
2. Drag the SfBarcode control from the Toolbox window to the Design View. An instance of the SfBarcode control is created in the Design view.



SfBarcode Control after Dragging to Design View

The following code example shows how to create the Barcode control from XAML:

#### XML

```
<Page xmlns:sync="using:Syncfusion.UI.Xaml.Controls.Barcode">  
<Grid>  
<sync:SfBarcode x:Name="barcode" Text="http://www.syncfusion.com"  
Symbology="QRBarcode">  
<sync:SfBarcode.SymbologySettings>  
<sync:QRBarcodeSetting XDimension="8"/>  
</sync:SfBarcode.SymbologySettings>  
</sync:SfBarcode>  
</Grid>  
</Page>
```

#### Text

The text to be encoded can be set using the Text property. By default, this original text will be displayed at the bottom of the bar code. The location of the text can be toggled between top and bottom using TextLocation property. The horizontal alignment of the text can be set using TextAlignment. The text brush and other various font customization can also be done using the built-in font properties. Optionally, the user can hide the barcode text by setting the DisplayText property to false.

#### XML

```
<sync:SfBarcode x:Name="barcode" Text="http://www.syncfusion.com"  
DisplayText="False" Symbology="QRBarcode"/>
```

### Rotation

Barcode control can be rotated to save space by using the Rotation property. The barcode can be rotated to 90, 180 and 270 degrees.

### Supported Barcode types in WPF Barcode (SfBarcode)

The following table contains the supported types and associated valid characters.

Symbol	Enum Value	Supported characters	Length
<a href="#">QR Code</a>	QRBarcode	[0-9]; [A-Z (upper-case only)]; [space \$ % * + - . / , :]; [Shift JIS characters]	variable
<a href="#">DataMatrix</a>	DataMatrix	All ASCII characters	
<a href="#">Code 39</a>	Code39	[0-9]; [A-Z]; [- . \$ / + % SPACE]	variable
<a href="#">Code 39 Extended</a>	Code39Extended	[0-9]; [A-Z]; [a-z]	variable
<a href="#">Code 11</a>	Code11	[0-9]; [-]	variable
<a href="#">Codabar</a>	Codabar	[0-9]; [- \$ : / . +]	variable
Code 32	Code32	[0-9]	8
<a href="#">Code 93</a>	Code93	[0-9]; [A-Z]; [- . \$ / + % SPACE]	variable
<a href="#">Code 93 Extended</a>	Code93Extended	All 128 ASCII characters	variable
<a href="#">Code 128A</a>	Code128A	[0-9]; [A-Z]; [NUL (0x00) SOH (0x01) STX (0x02) ETX (0x03) EOT(0x04) ENQ (0x05) ACK (0x06) BEL (0x07) BS (0x08) HT (0x09) LF (0x0A) VT(0x0B) FF (0x0C) CR (0x0D) SO (0x0E) SI (0x0F) DLE (0x10) DC1 (0x11) DC2(0x12) DC3 (0x13) DC4 (0x14) NAK (0x15) SYN (0x16) ETB (0x17) CAN(0x18) EM (0x19) SUB (0x1A) ESC (0x1B) FS (0x1C) GS (0x1D) RS (0x1E) US(0x1F) SPACE (0x20)]; [" ! \$ % & ' ( ) * + , - . / ; < = > ? @ [ / ] ^ _ ]	variable
<a href="#">Code 128B</a>	Code128B	[0-9]; [A-Z]; [a-z]; [SPACE (0x20) ! " # \$ % & ' ( ) * + , - . / ; < = > ? @ [ / ] ^ _ ` {   } ~ DEL (â€¢) ]	variable
<a href="#">Code 128C</a>	Code128C	ASCII 00-99(encodes each two digit with one code)	variable

### Barcode Customization in WPF Barcode (SfBarcode)

The color of the barcode can be customized by modifying the DarkBarBrush and LightBarBrush properties of the barcode control.

### HTML

```
<sync:SfBarcode x:Name="barcode" Text="82698640929" DarkBarBrush="Red"
LightBarBrush="Blue" Symbology="QRBarcode"/>
```

The DarkBarBrush represents the color of the dark bar (Black color usually) and the LightBarBrush represents the color of the gap between two adjacent black bars (White color usually).



Barcode color combinations- Red



Barcode color combinations- Blue

**Note:** The DarkBarBrush and LightBarBrush customizations are applicable only for one dimensional barcodes. In order for a barcode symbol to be recognized by a scanner, there must be an adequate contrast between the dark bars and the light spaces and not all the barcode scanners have support for colored barcodes.

### Symbology Settings in WPF Barcode (SfBarcode)

Each Barcode symbol can be associated with optional settings that may affect that specific bar code. The code sample below shows the settings of a code39 Barcode.

### HTML

```
<sync:SfBarcode.SymbologySettings>
<sync:Code39Setting BarHeight="45" EnableCheckDigit="False"
EncodeStartStopSymbols="True" NarrowBarWidth="1" ShowCheckDigit="False"/>
</sync:SfBarcode.SymbologySettings>
```

### 1D Barcode settings

The one dimensional barcodes have some of the settings in common, such as BarHeight which modifies the height of the bars and NarrowBarWidth which modifies the width ratio of the wide and narrow bars.

### HTML

```
<sync:SfBarcode.SymbologySettings>
<sync:Code39Setting BarHeight="45" NarrowBarWidth="1"/>
</sync:SfBarcode.SymbologySettings>
```

The one dimensional barcodes also has the error detection settings. The EnableCheckDigit property enables the redundancy check using a check digit, the decimal equivalent of a binary parity bit. It



consists of a single digit computed from the other digits in the message. The check digit can be shown in the barcode or kept hidden by using the ShowCheckDigit property.

The EncodeStartStopSymbols property adds Start and Stop symbols to signal a bar code reader that a bar code has been scanned.

#### HTML

```
<sync:SfBarcode.SymbologySettings>
<sync:Code39Setting EnableCheckDigit="False" EncodeStartStopSymbols="True"
ShowCheckDigit="False" />
</sync:SfBarcode.SymbologySettings>
```

#### 2D Barcode Settings

The two dimensional barcodes have a common XDimension property which modifies the block size of a two dimensional barcode.

##### DataMatrix Barcode settings

The DataMatrix barcode settings has the properties to modify the encoding and size of the DataMatrix barcode.

#### HTML

```
<sync:SfBarcode.SymbologySettings>
<sync:DataMatrixSetting XDimension="8" Encoding="ASCIINumeric"
Size="Size104x104" />
</sync:SfBarcode.SymbologySettings>
```

##### Encoding

The encoding of the DataMatrix barcode can be modified using the 'Encoding' property. The DataMatrixEncoding enumeration has the following four encoding schemes.

- ASCII
- ASCIINumeric
- Auto
- Base256

##### Size

The DataMatrix Barcode settings allow the user to specify the size of the barcode from a set of predefined sizes available in the DataMatrixSize enumeration.

Data Matrix size Table

Data Matrix Size	Description
Auto	Size is chosen based on the input data
Size10x10	Square matrix with 10 rows and 10 columns.
Size12x12	Square matrix with 12 rows and 12 columns.
Size14x14	Square matrix with 14 rows and 14 columns.

Size16x16	Square matrix with 16 rows and 16 columns.
Size18x18	Square matrix with 18 rows and 18 columns.
Size20x20	Square matrix with 20 rows and 20 columns.
Size22x22	Square matrix with 22 rows and 22 columns.
Size24x24	Square matrix with 24 rows and 24 columns.
Size26x26	Square matrix with 26 rows and 26 columns.
Size32x32	Square matrix with 32 rows and 32 columns.
Size36x36	Square matrix with 36 rows and 36 columns.
Size40x40	Square matrix with 40 rows and 40 columns.
Size44x44	Square matrix with 44 rows and 44 columns.
Size48x48	Square matrix with 48 rows and 48 columns.
Size52x52	Square matrix with 52 rows and 52 columns.
Size64x64	Square matrix with 64 rows and 64 columns.
Size72x72	Square matrix with 72 rows and 72 columns.
Size80x80	Square matrix with 80 rows and 80 columns.
Size88x88	Square matrix with 88 rows and 88 columns.
Size96x96	Square matrix with 96 rows and 96 columns.
Size104x104	Square matrix with 104 rows and 104 columns.
Size120x120	Square matrix with 120 rows and 120 columns.
Size132x132	Square matrix with 132 rows and 132 columns.
Size144x144	Square matrix with 144 rows and 144 columns.
Size8x18	Rectangular matrix with 8 rows and 18 columns.
Size8x32	Rectangular matrix with 8 rows and 32 columns.
Size12x26	Rectangular matrix with 12 rows and 26 columns.
Size12x36	Rectangular matrix with 12 rows and 36 columns.
Size16x36	Rectangular matrix with 16 rows and 36 columns.
Size16x48	Rectangular matrix with 16 rows and 48 columns.

#### [QRBarcode settings](#)

The QRBarcode settings has properties to modify the version, error correction level and Input mode of the QRBarcode.

## HTML

```
<sync:SfBarcode.SymbologySettings>
<sync:QRBarcodeSetting XDimension="8" ErrorCorrectionLevel="High"
InputMode="BinaryMode" Version="Auto" />
</sync:SfBarcode.SymbologySettings>
```

### Version

The QR Barcode uses version from 1 to 40. Version 1 measures 21 modules x 21 modules, Version 2 measures 25 modules x 25 modules and so on increasing in steps of 4 modules per side up to Version 40 which measures 177 modules x 177 modules. Each version has its own capacity. By default the QR Version is Auto, which will automatically set the version according to the input text length.

### Error correction level

The QR Barcode employs error correction to generate a series of error correction codewords which are added to the data code word sequence in order to enable the symbol to withstand damage without loss of data. There are four user-selectable levels of error correction, as shown in the table, offering the capability of recovery from the following amounts of damage. By default the Error correction level is Low.

Error Correction Level Table

Error Correction Level	Recovery Capacity % (approx.)
L	7
M	15
Q	25
H	30

### Input mode

There are three modes for the input as defined in the table. Each mode supports the specific set of Input characters. User may select the most suitable input mode. By default the Input mode is Binary Mode.

Input Mode Table

Input Mode	Possible characters
Numeric Mode	0,1,2,3,4,5,6,7,8,9
Alphanumeric Mode	0-9, A-Z (upper-case only), space, \$, %, *, +, -, ., /, :
Binary Mode	Shift JIS characters

## SfBulletGraph

### WPF Bullet Graph (SfBulletGraph) Overview

Bullet Graph is a variation of the bar graph, which is developed as a replacement for the dashboard gauges and meters. The bullet graph features a single, primary measure (for example, current year-to-

date revenue), and compares that measure to one or more other measures to enrich its meaning (for example, compared to a target). It displays the measures in the context of the qualitative range of performance, such as poor, satisfactory, and good.

#### Use cases

- It is used in the dashboard environment where the screen real-estate really counts.
- Bullet graph shows a ton of data in a very compact space.
- It is handy to visualize data.
- It is used in the revenue analysis and expense analysis.

#### Key Features in WPF Bullet Graph (SfBulletGraph)

[SfBulletGraph](#) is Composite UI element with following sub-parts:

- Ticks
- Label
- Caption
- Range
- Feature Measure

#### Ticks

Ticks are scale indicator which is used to specify the values on quantitative scale.

#### Label

A quantitative scale label specifies the numeric value according to the major ticks in the range of the scale.

#### Caption

The [Caption](#) for a bullet graph is used to specify a unique label describing the value represented in the bullet graph.

#### Range

A qualitative range is a visual element that ends at a specified [RangeEnd](#) at the beginning of the previous range's [RangeEnd](#). The qualitative ranges are arranged according to each [RangeEnd](#) value.

#### Performance measure

##### *Featured measure*

[FeaturedMeasure](#) is used to display the primary data, or the current value of the data that you are measuring. It should usually be encoded as a bar, like the bar on a bar graph, and be prominent.

##### *Comparative measure*

[ComparativeMeasure](#) should be less visually dominant than the featured measure. It should always be encoded as a short line that runs perpendicular to the orientation of the graph. A good example would be a target for YTD revenue. Whenever the featured measure intersects a comparative measure, the comparative measure should appear behind the featured measure.

#### Bullet graphs type

The Bullet Graph types can be differed by its orientation of the control, the control direction may be **horizontal** or **vertical**.

### Easy to use ###

SfBulletGraph is available in Visual Studio toolbox itself, you can easily drag and drop the control from toolbox.

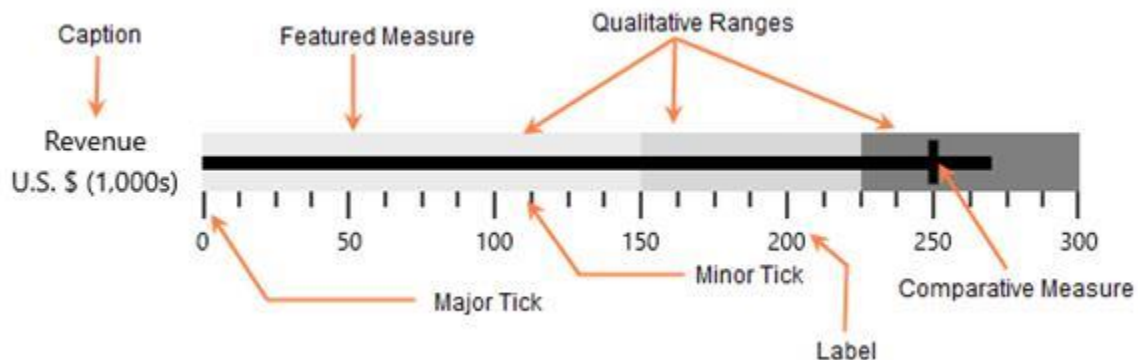
### Data Binding Support ##

The control can be bound to your application data from a variety of data sources in the form of binding declaration or by setting its properties directly.

### MVVM support ##

The Model-View-ViewModel (MVVM) pattern is to be followed to get better control customization.

### SfBulletGraph Elements ##



### Getting Started with WPF Bullet Graph (SfBulletGraph)

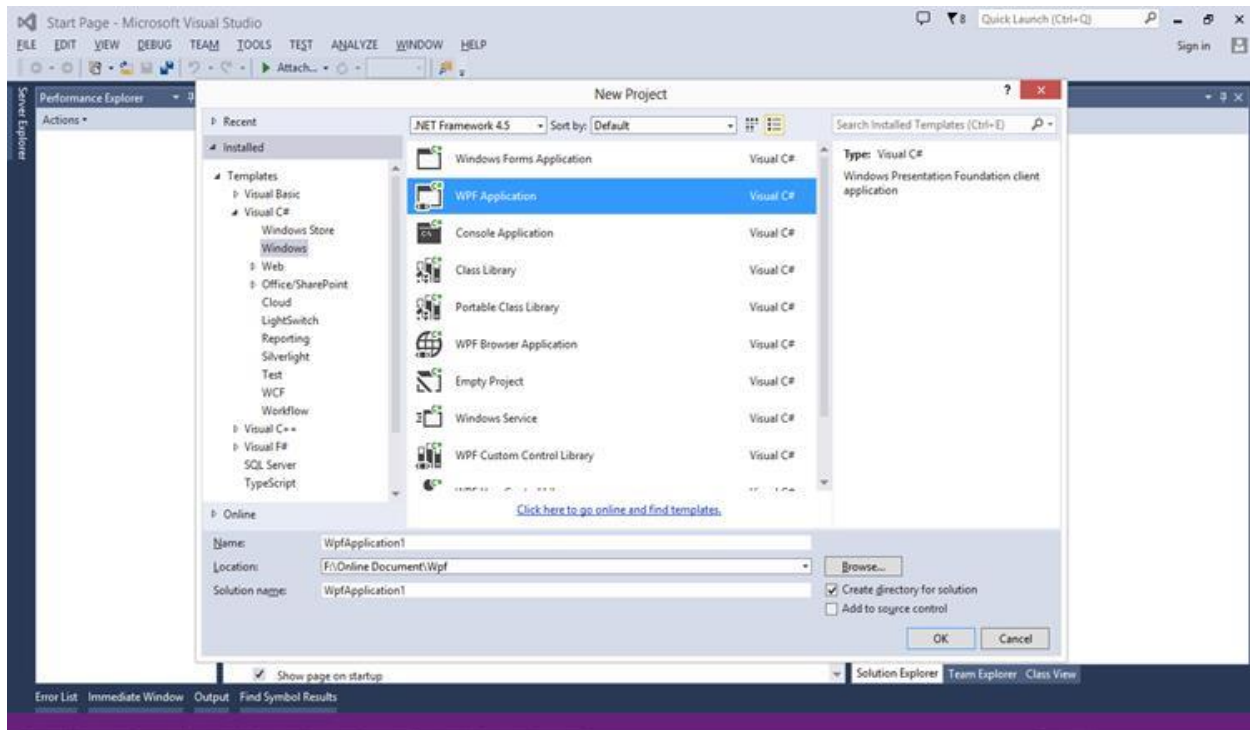
This section explains you the steps required to configure the [SfBulletGraph](#) and also explains the steps to add basic elements of [SfBulletGraph](#) using the various API's available within it.

#### Configuring SfBulletGraph using Syncfusion Reference Manager

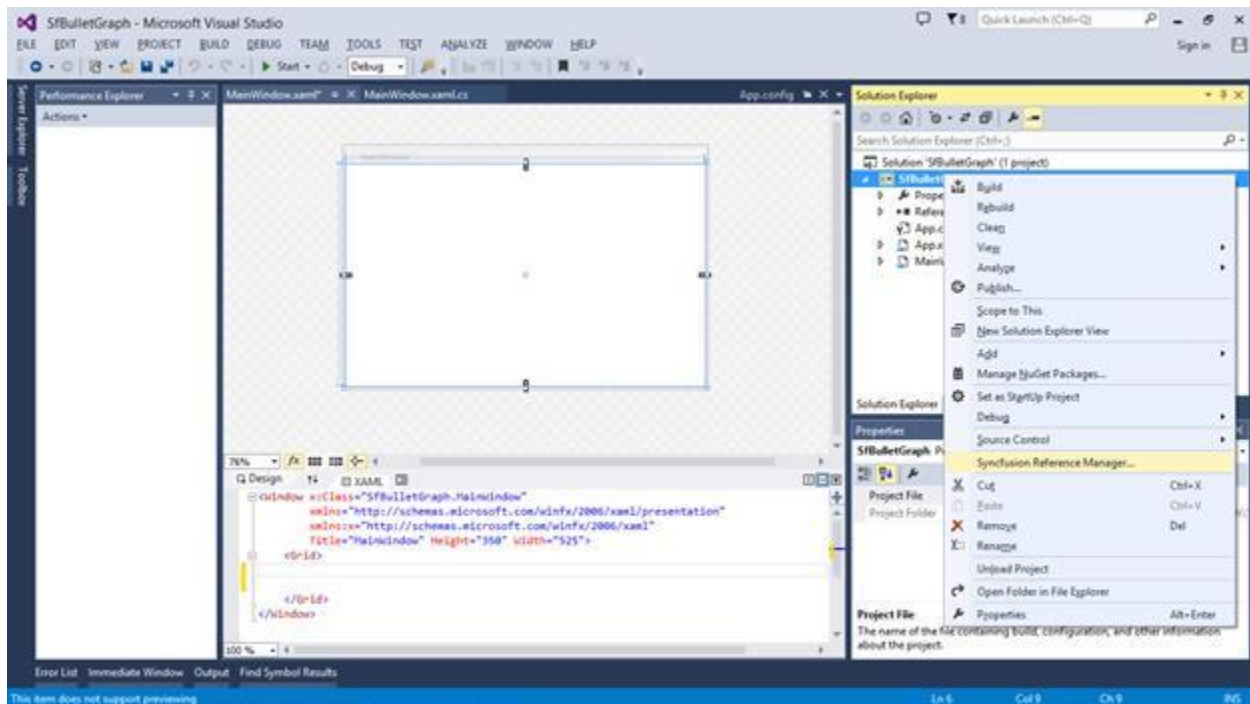
Syncfusion Reference Manager is used to add Syncfusion Tools.

Follow these steps to add [SfBulletGraph](#) Control using the Syncfusion Reference Manager.

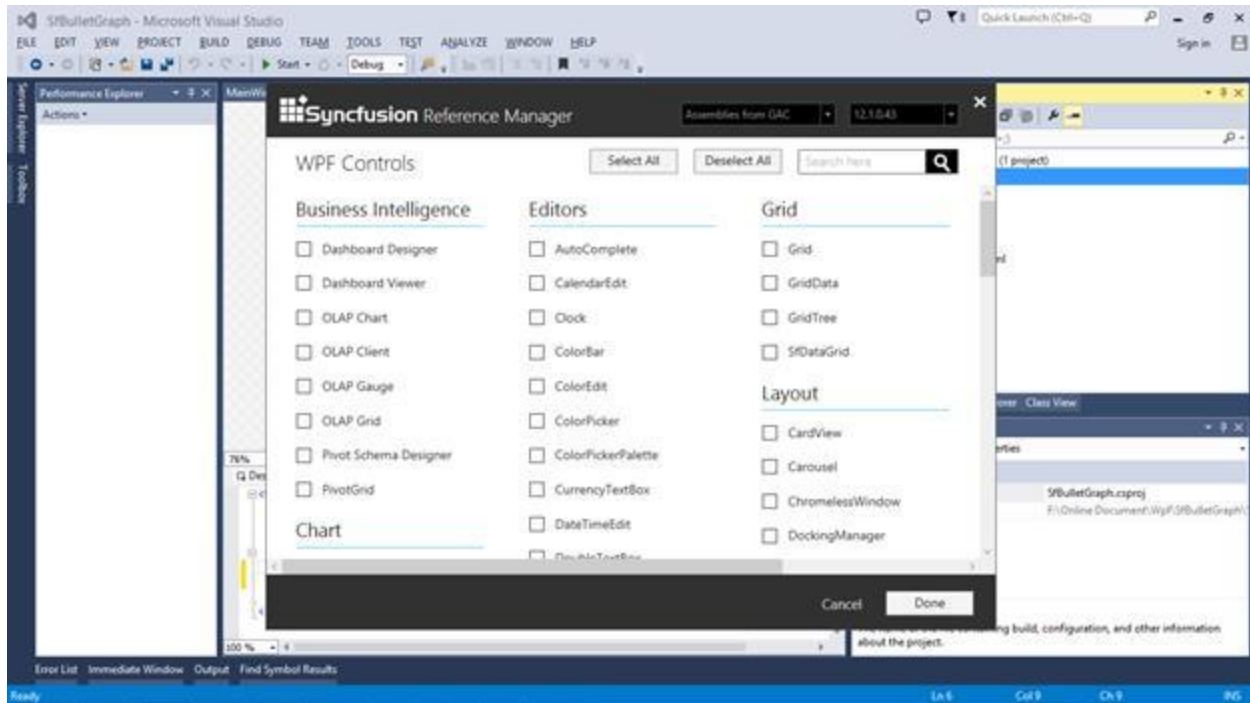
Create a simple WPF application using Visual Studio.



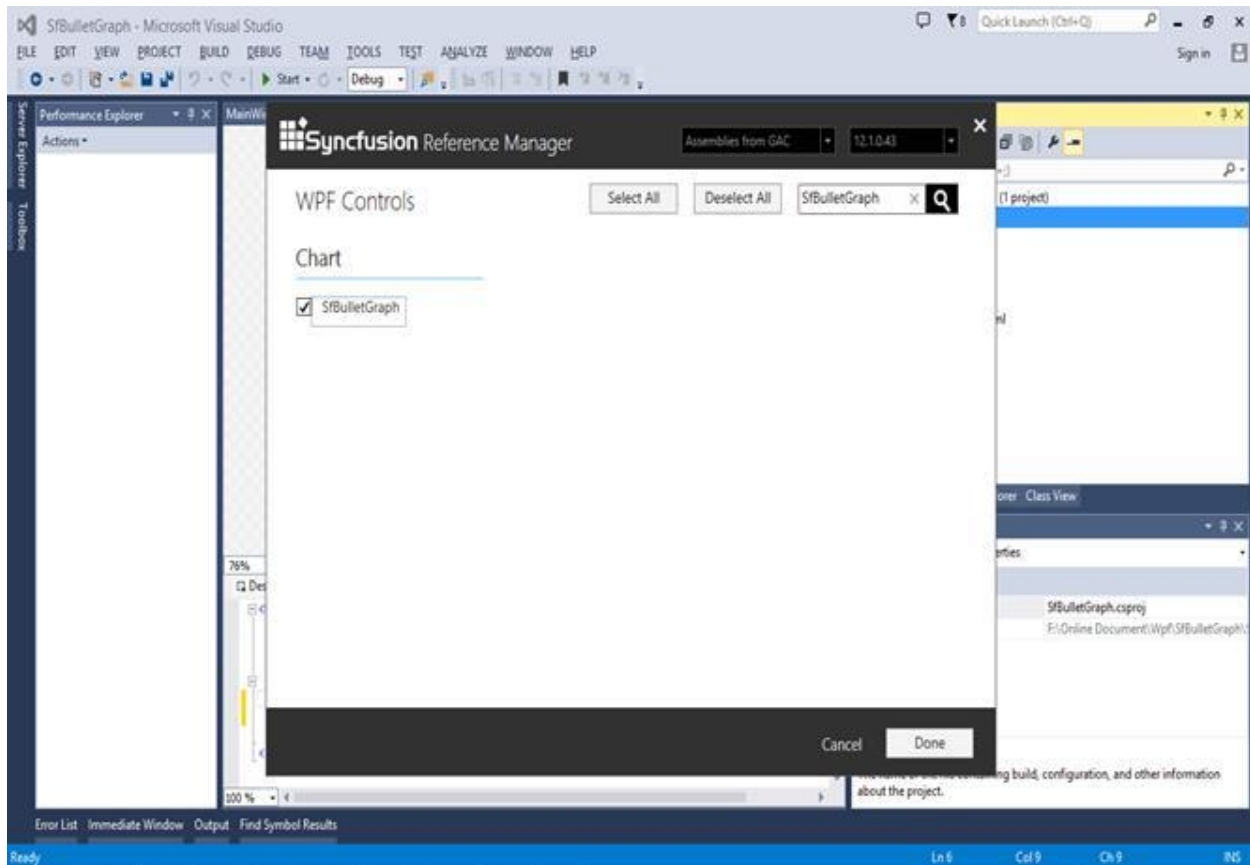
Right Click on the Project and select Syncfusion Reference Manager.



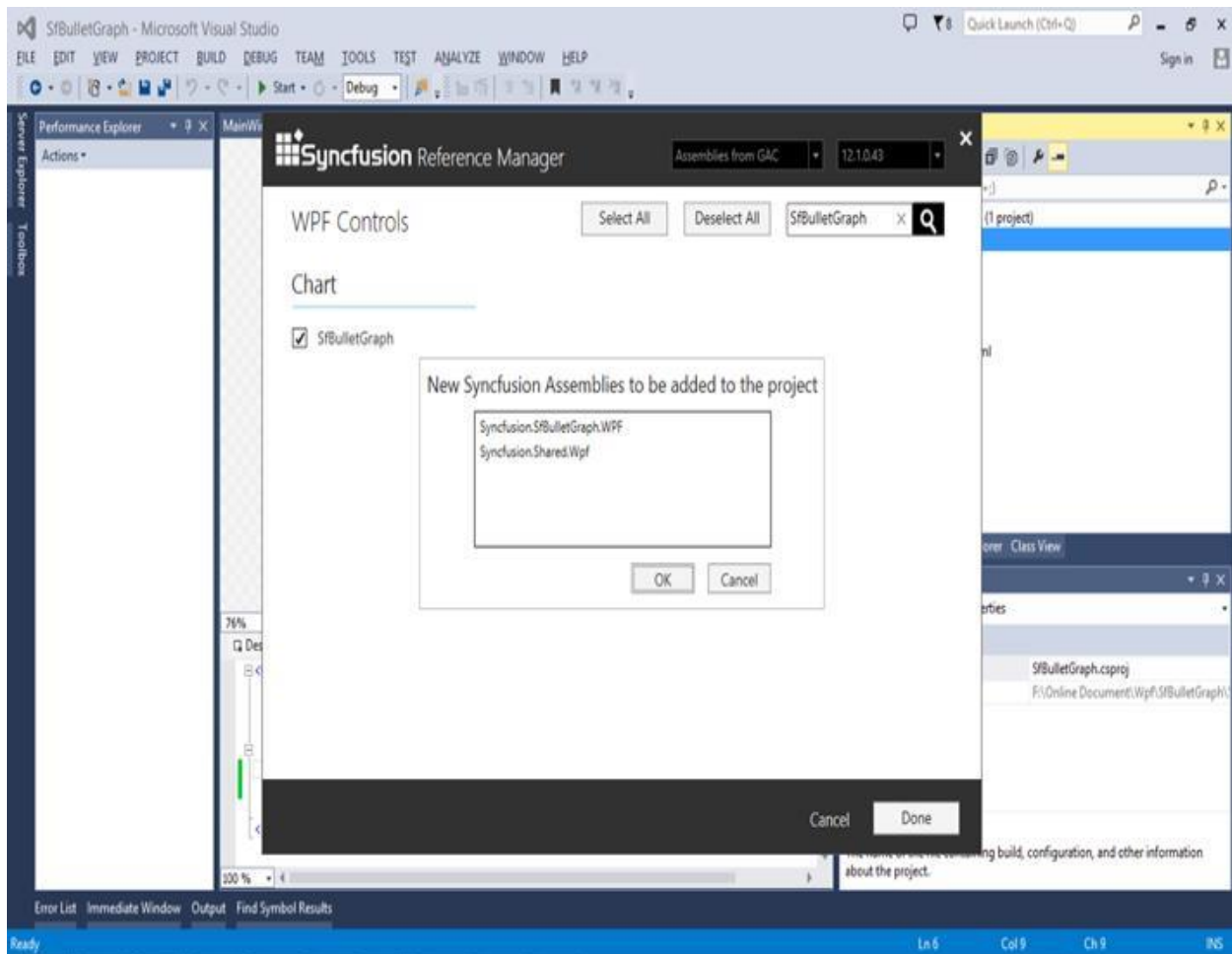
The Syncfusion Reference Manager Wizard will be opened as shown in the figure below.



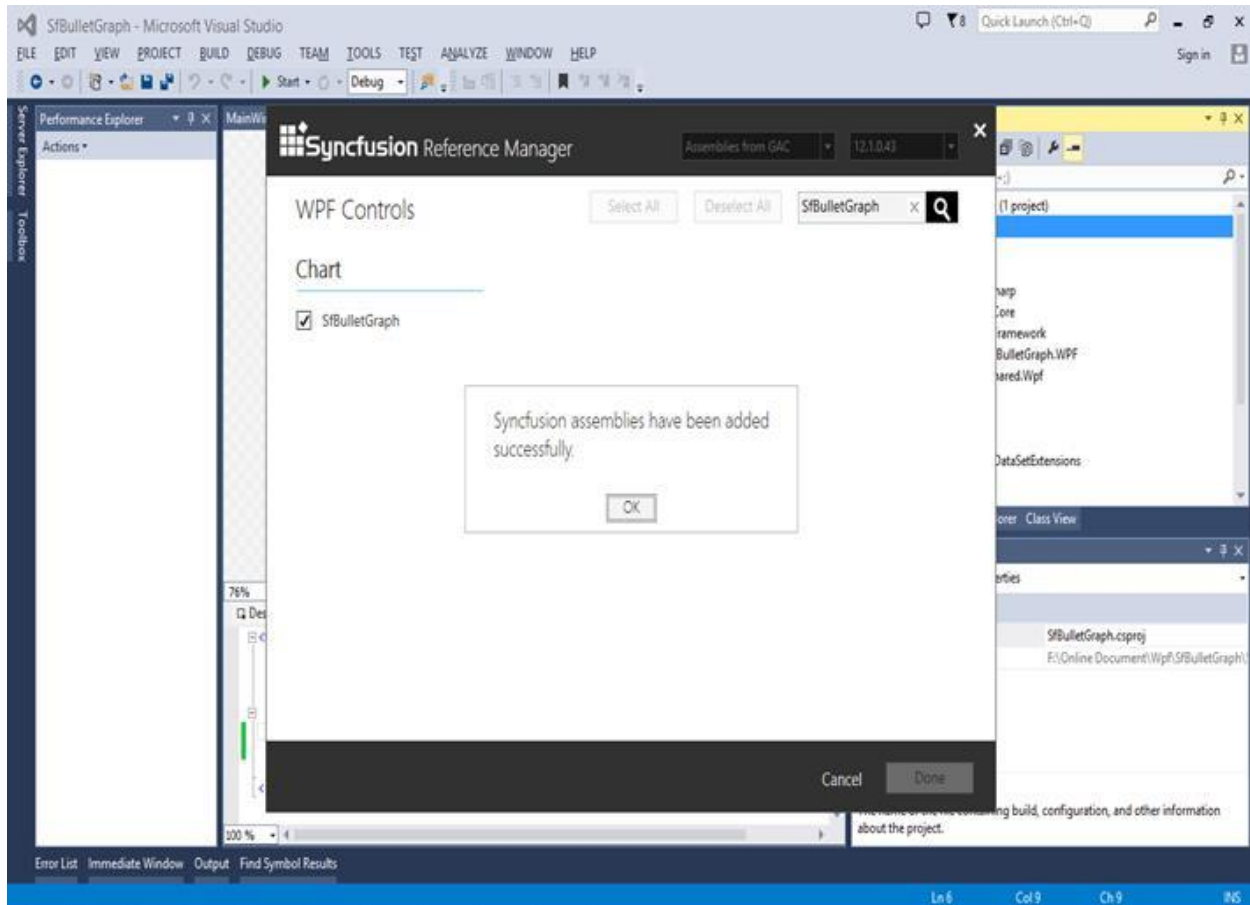
Search for **SfBulletGraph** using SearchBox and select the **SfBulletGraph** Control. Click **Done** to add selected **SfBulletGraph** control.



The **SfBulletGraph** assemblies will be automatically added to the project after clicking **OK**.







Create a namespace reference to the SfBulletGraph control using Syncfusion's global namespace reference **schemas.Syncfusion.com** or the SfBulletGraph control's namespace reference using **Syncfusion.UI.Xaml.BulletGraph** available in the **Syncfusion.SfBulletGraph.WPF** assembly.

### XML

```
xmlns:bulletgraph="http://schemas.syncfusion.com/wpf"
(or)
xmlns:bulletgraph="clr-namespace:Syncfusion.UI.Xaml.BulletGraph;assembly=Syncfusion.SfBulletGraph.WPF"
```

Add the following code to create a simple SfBulletGraph control.

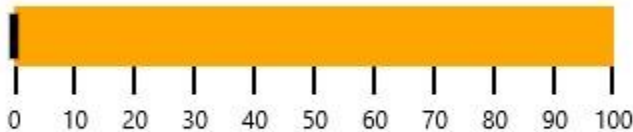
### XML

```
<Window x:Class="SfBulletGraph.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="350" Width="525">
<Grid x:Name="LayoutRoot">
<syncfusion:SfBulletGraph/>
</Grid>
</Window>
```

### C#

```
SfBulletGraph bulletgraph = new SfBulletGraph();  
this.Grid.Children.Add(bulletgraph);
```

The SfBulletGraph control will be created as shown in the figure below.



**Note:- The Syncfusion Reference Manager is available in versions 11.3.0.30 and later. It supports referencing assemblies from version 10.4.0.71 version to the current version and Syncfusion Reference Manager can be used only in Visual Studio 2010, 2012, 2013 and 2015.**

### Configuring SfBulletGraph

**SfBulletGraph** is available in the following assembly and namespace:

**Assembly:** Syncfusion.SfBulletGraph.Wpf

**Namespace:** Syncfusion.UI.Xaml.BulletGraph

Create a namespace reference to the SfBulletGraph control using Syncfusion's global namespace reference **schemas.Syncfusion.com** or the SfBulletGraph control's namespace reference using **Syncfusion.UI.Xaml.BulletGraph** available in the **Syncfusion.SfBulletGraph.WPF** assembly.

### XML

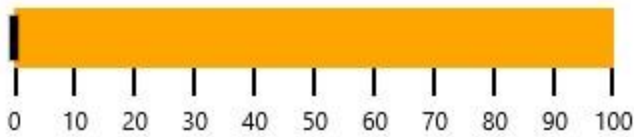
```
xmlns:bulletgraph="http://schemas.syncfusion.com/wpf"  
(or)  
xmlns:bulletgraph="clr-  
namespace:Syncfusion.UI.Xaml.BulletGraph;assembly=Syncfusion.SfBulletGraph.  
WPF"
```

### XML

```
<syncfusion:SfBulletGraph/>
```

### C#

```
SfBulletGraph bulletgraph = new SfBulletGraph();  
this.Grid.Children.Add(bulletgraph);
```



As you can see now in the above image, the SfBulletGraph displays its default elements. To customize its element, you have to add respective elements to SfBulletGraph, following section contains the steps to add the basic elements to SfBulletGraph.

### Adding Caption

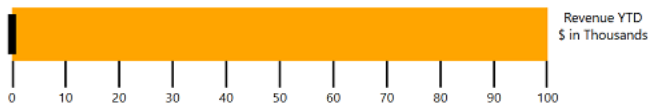
You can assign a caption to bullet graph by making use of [Caption](#) property and also you can position it either near or far using the [CaptionPosition](#) property.

### XML

```
<syncfusion:SfBulletGraph CaptionPosition="Far">
  <syncfusion:SfBulletGraph.Caption>
    <StackPanel Margin="0,0,10,0">
      <TextBlock Text="Revenue YTD" Foreground="Black"
        FontSize="13" HorizontalAlignment="Center"/>
      <TextBlock Text="$ in Thousands" Foreground="Black"
        FontSize="13" HorizontalAlignment="Center"/>
    </StackPanel>
  </syncfusion:SfBulletGraph.Caption>
</syncfusion:SfBulletGraph>
```

### C#

```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.CaptionPosition = BulletGraphCaptionPosition.Far;
TextBlock textBlock = new TextBlock() { Text = "Revenue YTD" };
TextBlock textBlock1 = new TextBlock() { Text = "$ in Thousands" };
StackPanel stackPanel = new StackPanel();
stackPanel.Children.Add(textBlock);
stackPanel.Children.Add(textBlock1);
bulletgraph.Caption = stackPanel;
grid.Children.Add(bulletgraph);
```



### Configuring Ticks and Labels

You can configure Ticks and Labels of Quantitative Scale by making use of following API's available in SfBulletGraph.

They are:

- Minimum

- Maximum
- Interval
- MinorTicksPerInterval
- MajorTickSize
- MinorTickSize
- MajorTickStroke
- LabelStroke
- MinorTickStroke

**XML**

```
<syncfusion:SfBulletGraph Orientation="Horizontal" Minimum="0" Maximum="10"
Interval="2"
ComparativeMeasure="7" FeaturedMeasure="5" FeaturedMeasureBarStroke="Black"
MajorTickStroke="Red"
MinorTickStroke="Green" MinorTicksPerInterval="3"
MajorTickSize="15" MinorTickSize="10">
  <syncfusion:SfBulletGraph.QualitativeRanges>
    <syncfusion:QualitativeRange RangeEnd="4.5"
RangeStroke="#EBEBEB"></syncfusion:QualitativeRange>
    <syncfusion:QualitativeRange RangeEnd="10"
RangeStroke="#7F7F7F"></syncfusion:QualitativeRange>
    <syncfusion:QualitativeRange RangeEnd="7.5"
RangeStroke="#D8D8D8"></syncfusion:QualitativeRange>
  </syncfusion:SfBulletGraph.QualitativeRanges>
</syncfusion:SfBulletGraph>
```

**C#**

```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.Interval = 2;
bulletgraph.FeaturedMeasure = 5;
bulletgraph.ComparativeMeasure = 7;
bulletgraph.MajorTickStroke = new SolidColorBrush(Colors.Red);
bulletgraph.MinorTickStroke = new SolidColorBrush(Colors.Green);
bulletgraph.FeaturedMeasureBarStroke = new SolidColorBrush(Colors.Black);
bulletgraph.MinorTicksPerInterval = 3;
bulletgraph.MinorTickSize = 10;
bulletgraph.MajorTickSize = 15;
bulletgraph.Orientation = Orientation.Horizontal;
QualitativeRange range1 = new QualitativeRange();
range1.RangeEnd = 4.5;
range1.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#EBEBEB");
QualitativeRange range2 = new QualitativeRange();
range2.RangeEnd = 10;
range2.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#7F7F7F");
QualitativeRange range3 = new QualitativeRange();
range3.RangeEnd = 7.5;
range3.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#D8D8D8");
bulletgraph.QualitativeRanges.Add(range1);
bulletgraph.QualitativeRanges.Add(range2);
bulletgraph.QualitativeRanges.Add(range3);
grid.Children.Add(bulletgraph);
```



### Adding Ranges

You can add ranges to bullet graph by creating ranges collection using [QualitativeRanges](#).

#### XML

```
<syncfusion:SfBulletGraph Orientation="Horizontal" Minimum="0" Maximum="10"
Interval="2"
QualitativeRangesSize="30" MinorTicksPerInterval="3" ComparativeMeasure="7"
FeaturedMeasure="5"
QuantitativeScaleLength="300" FeaturedMeasureBarStroke="Black"
MinorTickSize="10" MajorTickSize="14">
  <syncfusion:SfBulletGraph.QualitativeRanges>
    <syncfusion:QualitativeRange RangeEnd="4.5" RangeCaption="Bad"
RangeStroke="Red"
RangeOpacity="1">
  </syncfusion:QualitativeRange>
    <syncfusion:QualitativeRange RangeEnd="7.5"
RangeStroke="Yellow"
RangeOpacity="1">
  </syncfusion:QualitativeRange>
    <syncfusion:QualitativeRange RangeEnd="10"
RangeStroke="Green"
RangeOpacity="1">
  </syncfusion:QualitativeRange>
  </syncfusion:SfBulletGraph.QualitativeRanges>
</syncfusion:SfBulletGraph>
```

#### C#

```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.FeaturedMeasure = 5;
bulletgraph.ComparativeMeasure = 7;
bulletgraph.Interval = 2;
bulletgraph.MinorTickSize = 8;
bulletgraph.MinorTicksPerInterval = 3;
bulletgraph.QualitativeRangesSize = 30;
bulletgraph.QuantitativeScaleLength = 300;
bulletgraph.FeaturedMeasureBarStroke = new SolidColorBrush(Colors.Black);
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 4.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Red)
});
```

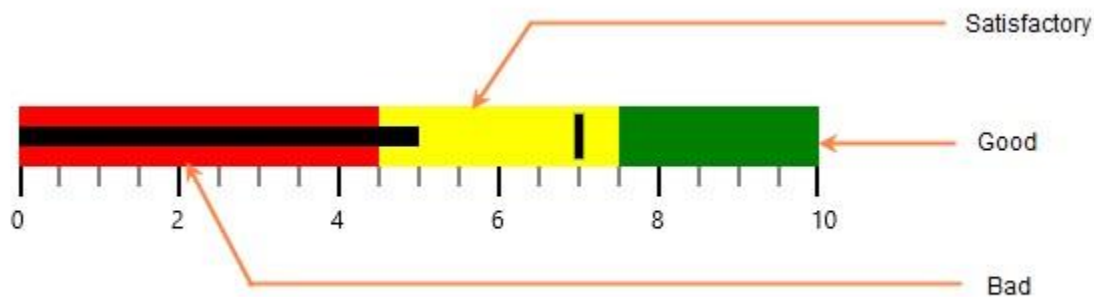
```

});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 7.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Yellow)
});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 10,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Green)
});
grid.Children.Add(bulletgraph);

```

SfBulletGraph ranges are displayed as follows.

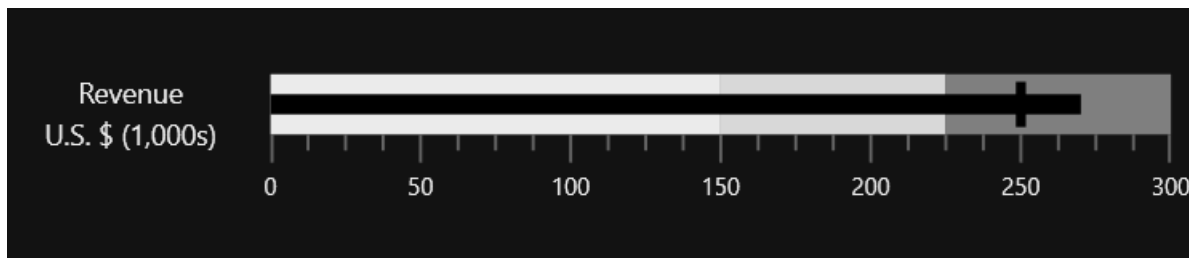
You can get the complete getting started sample [here](#).



### Theme

Bullet Graph supports various built-in themes. Refer to the below links to apply themes for the Bullet Graph,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Orientation in WPF Bullet Graph (SfBulletGraph)

By default, orientation of SfBulletGraph is horizontal. It can be customized by using [Orientation](#) property, respectively.

### XML

```

<syncfusion:SfBulletGraph Orientation="Vertical" Minimum="0" Maximum="10"
Interval="2">
<syncfusion:SfBulletGraph.QualitativeRanges>
<syncfusion:QualitativeRange RangeEnd="3"
RangeStroke="#EBEBEB"></syncfusion:QualitativeRange>
<syncfusion:QualitativeRange RangeEnd="7"
RangeStroke="#7F7F7F"></syncfusion:QualitativeRange>
<syncfusion:QualitativeRange RangeEnd="10"
RangeStroke="#D8D8D8"></syncfusion:QualitativeRange>
</syncfusion:SfBulletGraph.QualitativeRanges>
</syncfusion:SfBulletGraph>

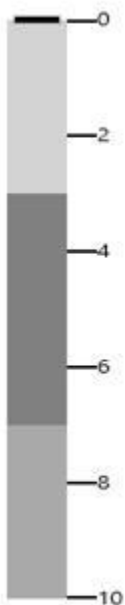
```

**C#**

```

SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.Interval = 2;
bulletgraph.Orientation = Orientation.Vertical;
QualitativeRange range1 = new QualitativeRange();
range1.RangeEnd = 3;
range1.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#EBEBEB");
QualitativeRange range2 = new QualitativeRange();
range2.RangeEnd = 7;
range2.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#7F7F7F");
QualitativeRange range3 = new QualitativeRange();
range3.RangeEnd = 10;
range3.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#D8D8D8");
bulletgraph.QualitativeRanges.Add(range1);
bulletgraph.QualitativeRanges.Add(range2);
bulletgraph.QualitativeRanges.Add(range3);
grid.Children.Add(bulletgraph);

```





### Flow Direction in WPF Bullet Graph (SfBulletGraph)

By default the flow direction of SfBulletGraph is Left to Right. It can be customized by using **FlowDirection** property respectively.

Note: When the orientation of [SfBulletGraph](#) is **Horizontal**, the default flow direction will be left to right and when the orientation of is **Vertical**, the default flow direction will be top to bottom.

#### XML

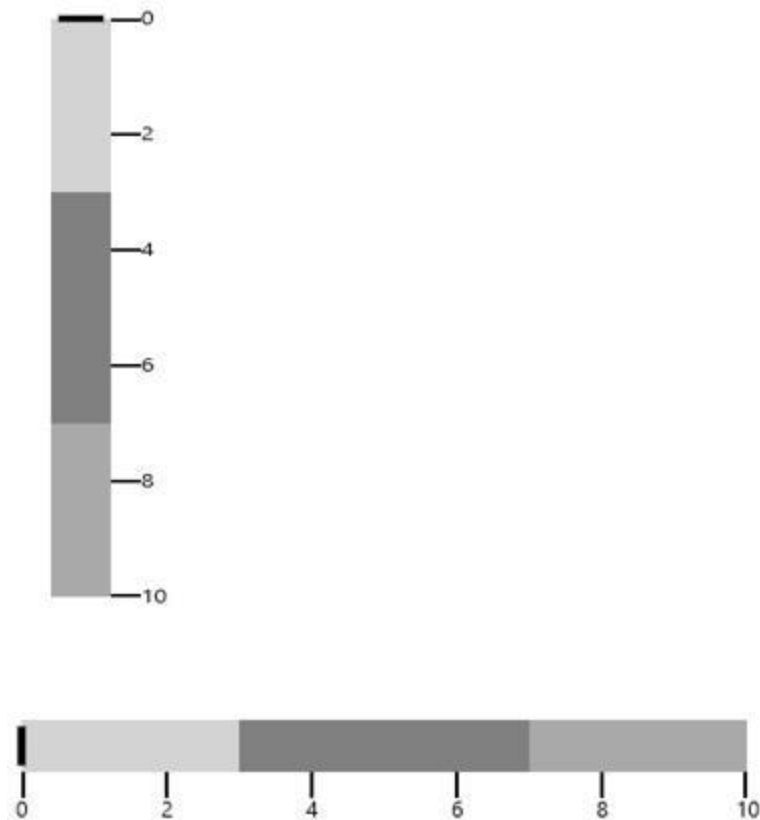
```
<syncfusion:SfBulletGraph Orientation="Horizontal" Minimum="0" Maximum="10"
Interval="2"
FlowDirection="LeftToRight">
<syncfusion:SfBulletGraph.QualitativeRanges>
<syncfusion:QualitativeRange RangeEnd="3"
RangeStroke="#EBEBEB"></syncfusion:QualitativeRange>
<syncfusion:QualitativeRange RangeEnd="7"
RangeStroke="#7F7F7F"></syncfusion:QualitativeRange>
<syncfusion:QualitativeRange RangeEnd="10"
RangeStroke="#D8D8D8"></syncfusion:QualitativeRange>
</syncfusion:SfBulletGraph.QualitativeRanges>
</syncfusion:SfBulletGraph>
```

#### C#

```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.Interval = 2;
bulletgraph.FlowDirection = FlowDirection.LeftToRight;
bulletgraph.Orientation = Orientation.Vertical;
QualitativeRange range1 = new QualitativeRange();
range1.RangeEnd = 3;
range1.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#EBEBEB");
QualitativeRange range2 = new QualitativeRange();
range2.RangeEnd = 7;
range2.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#7F7F7F");
QualitativeRange range3 = new QualitativeRange();
range3.RangeEnd = 10;
range3.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#D8D8D8");
bulletgraph.QualitativeRanges.Add(range1);
bulletgraph.QualitativeRanges.Add(range2);
bulletgraph.QualitativeRanges.Add(range3);
grid.Children.Add(bulletgraph);
```

Refer below screenshot for flow direction **LeftToRight**.



**XML**

```
<syncfusion:SfBulletGraph Orientation="Horizontal" Minimum="0" Maximum="10"
Interval="2"
FlowDirection="RightToLeft">
<syncfusion:SfBulletGraph.QualitativeRanges>
<syncfusion:QualitativeRange RangeEnd="3"
RangeStroke="#EBEBEB"></syncfusion:QualitativeRange>
<syncfusion:QualitativeRange RangeEnd="7"
RangeStroke="#7F7F7F"></syncfusion:QualitativeRange>
<syncfusion:QualitativeRange RangeEnd="10"
RangeStroke="#D8D8D8"></syncfusion:QualitativeRange>
</syncfusion:SfBulletGraph.QualitativeRanges>
</syncfusion:SfBulletGraph>
```

**C#**

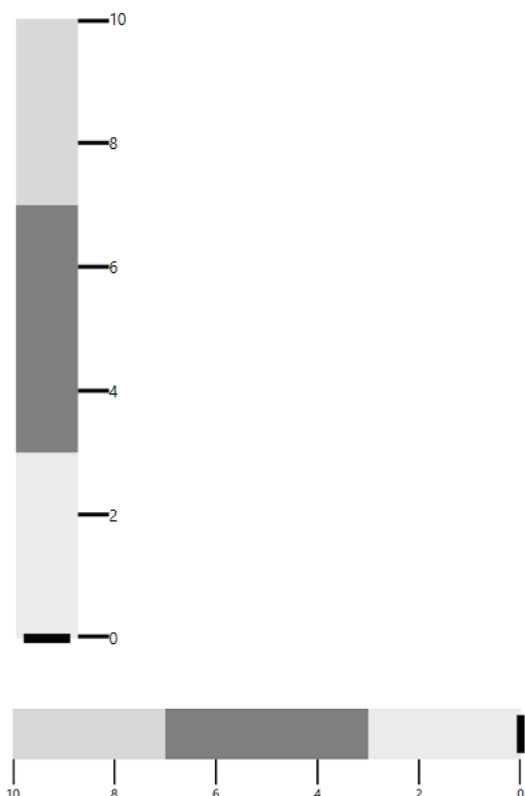
```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.Interval = 2;
bulletgraph.FlowDirection = FlowDirection.RightToLeft;
bulletgraph.Orientation = Orientation.Vertical;
QualitativeRange range1 = new QualitativeRange();
range1.RangeEnd = 3;
range1.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#EBEBEB");
QualitativeRange range2 = new QualitativeRange();
```

```

range2.RangeEnd = 7;
range2.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#7F7F7F");
QualitativeRange range3 = new QualitativeRange();
range3.RangeEnd = 10;
range3.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#D8D8D8");
bulletgraph.QualitativeRanges.Add(range1);
bulletgraph.QualitativeRanges.Add(range2);
bulletgraph.QualitativeRanges.Add(range3);
grid.Children.Add(bulletgraph);

```

Refer the following screenshot for flow direction **RightToLeft**.



### Quantitative Scale in WPF Bullet Graph (SfBulletGraph)

Quantitative scale contains two major components such as ticks and labels which gives the looks of bar graph. It defines the frequency of labels and tick marks with overall [Minimum](#) and [Maximum](#) values for declared [Interval](#). The length of the quantitative scale can be customized by using the [QuantitativeScaleLength](#) property.

#### XML

```

<syncfusion:SfBulletGraph QuantitativeScaleLength="300" Minimum="0"
Maximum="10" Interval="2"/>

```

#### C#

```

SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.QuantitativeScaleLength = 300;
bulletgraph.Minimum = 0;

```

```
bulletgraph.Maximum = 10;
bulletgraph.Interval = 2;
grid.Children.Add(bulletgraph);
```



### Ticks in WPF Bullet Graph (SfBulletGraph)

Quantitative scale is displayed with two types of ticks:

- Major ticks, the primary scale indicators.
- Minor ticks, the secondary scale indicators that fall in between the major ticks.

### Customizing Ticks

The [Interval](#) property is used to calculate the Major tick count for a SfBulletGraph. Like ticks, small ticks are calculated using the [MinorTicksPerInterval](#) property.

The stroke of the major and minor ticks is customized by setting the [MajorTickStroke](#) and [MinorTickStroke](#) properties. The size can be modified by using the [MajorTickSize](#) and [MinorTickSize](#) properties. By setting [MajorTickStrokeThickness](#) and [MinorTickStrokeThickness](#), the stroke's thickness can be customized.

### XML

```
<syncfusion:SfBulletGraph Interval="2" Minimum="0" Maximum="10"
MinorTicksPerInterval="3"
MajorTickSize="15"
MinorTickSize="10"
MajorTickStroke="Red"
MinorTickStroke="Green">
</syncfusion:SfBulletGraph>
```

### C#

```
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.MinorTicksPerInterval = 3;
bulletgraph.MajorTickSize = 15;
bulletgraph.MinorTickSize = 10;
bulletgraph.MajorTickStroke = new SolidColorBrush(Colors.Red);
bulletgraph.MinorTickStroke = new SolidColorBrush(Colors.Green);
grid.Children.Add(bulletgraph);
```



### TickPosition

The ticks in the scale can be placed above or below the ranges of the quantitative scale by choosing the options available in the [TickPosition](#) property.

They are:

1. Below (Default)
2. Above
3. Cross

### XML

```
<syncfusion:SfBulletGraph TickPosition="Cross">
</syncfusion:SfBulletGraph>
```

### C#

```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.TickPosition = BulletGraphTicksPosition.Cross;
grid.Children.Add(bulletgraph);
```



### Labels in WPF Bullet Graph (SfBulletGraph)

A quantitative scale label specifies the numeric value according to the major ticks in the range of the scale.

#### Customizing Labels

The labels can be positioned far away from the quantitative scale by using the [LabelOffset](#) property and the default value of this is 0. The foreground of the label is customized by setting [LabelStroke](#). By setting [LabelSize](#), the font size of the labels is modified. The label content can be formatted by using [LabelFormat](#) property.

### XML

```
<syncfusion:SfBulletGraph LabelSize="20" Minimum="0" Maximum="10"
LabelOffset="5" Interval="2"
LabelStroke="Red"
LabelFormat="C" >
</syncfusion:SfBulletGraph>
```

### C#

```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.LabelOffset = 5;
bulletgraph.LabelSize = 20;
bulletgraph.LabelFormat = "C";
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.Interval = 2;
bulletgraph.LabelStroke = new SolidColorBrush(Colors.Red);
```

```
grid.Children.Add(bulletgraph);
```



### Label Position

The labels in the scale can be placed above or below the qualitative ranges by choosing the following options available in the [LabelPosition](#) property.

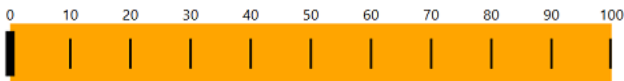
1. Below (Default)
2. Above

### XML

```
<syncfusion:SfBulletGraph LabelPosition="Above" TickPosition="Cross">
</syncfusion:SfBulletGraph>
```

### C#

```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.LabelPosition = BulletGraphLabelsPosition.Above;
this.Grid.Children.Add(bulletgraph);
```



### Caption in WPF Bullet Graph (SfBulletGraph)

The [Caption](#) for a bullet graph is used to specify a unique label describing the value represented in the bullet graph.

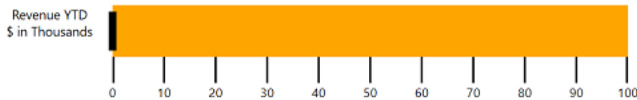
### XML

```
<syncfusion:SfBulletGraph >
<syncfusion:SfBulletGraph.Caption>
<StackPanel Margin="0,0,10,0">
<TextBlock Text="Revenue YTD" Foreground="Black"
FontSize="13" HorizontalAlignment="Center"/>
<TextBlock Text="$ in Thousands" Foreground="Black"
FontSize="13" HorizontalAlignment="Center"/>
</StackPanel>
</syncfusion:SfBulletGraph.Caption>
</syncfusion:SfBulletGraph>
```

### C#

```
SfBulletGraph bulletgraph = new SfBulletGraph();
TextBlock textBlock = new TextBlock() { Text = "Revenue YTD" };
TextBlock textBlock1 = new TextBlock() { Text = "$ in Thousands" };
```

```
StackPanel stackPanel = new StackPanel();
stackPanel.Children.Add(textBlock);
stackPanel.Children.Add(textBlock1);
bulletgraph.Caption = stackPanel;
grid.Children.Add(bulletgraph);
```



### Caption Position

The caption in the bullet graph can be placed in the start or end of the quantitative scale by choosing from one of the two options available in the [CaptionPosition](#) property. They are:

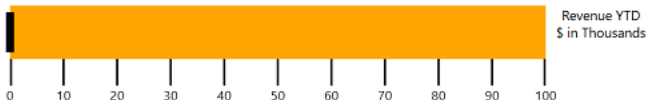
1. Near (Default)
2. Far

### XML

```
<syncfusion:SfBulletGraph CaptionPosition="Far">
  <syncfusion:SfBulletGraph.Caption>
    <StackPanel Margin="0,0,10,0">
      <TextBlock Text="Revenue YTD" Foreground="Black"
        FontSize="13" HorizontalAlignment="Center"/>
      <TextBlock Text="$ in Thousands" Foreground="Black"
        FontSize="13" HorizontalAlignment="Center"/>
    </StackPanel>
  </syncfusion:SfBulletGraph.Caption>
</syncfusion:SfBulletGraph>
```

### C#

```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.CaptionPosition = BulletGraphCaptionPosition.Far;
TextBlock textBlock = new TextBlock() { Text = "Revenue YTD" };
TextBlock textBlock1 = new TextBlock() { Text = "$ in Thousands" };
StackPanel stackPanel = new StackPanel();
stackPanel.Children.Add(textBlock);
stackPanel.Children.Add(textBlock1);
bulletgraph.Caption = stackPanel;
grid.Children.Add(bulletgraph);
```



## Measures in WPF Bullet Graph (SfBulletGraph)

### Featured Measure

Featured measure is used to display the primary data, or the current value of the data that you are measuring. It should usually be encoded as a bar.

*Customizing featured measure*

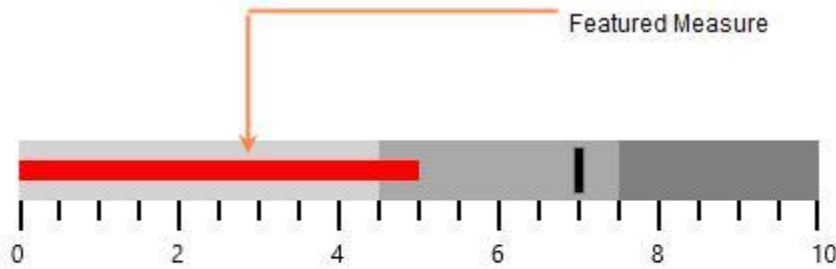
The value of the featured measure of the bullet graph is set by the [FeaturedMeasure](#) property. By setting the [FeaturedMeasureBarStroke](#) property, the stroke of the feature measure bar can be customized. The thickness of the featured measure bar is modified by using [FeaturedMeasureBarStrokeThickness](#).

**XML**

```
<syncfusion:SfBulletGraph FeaturedMeasure="5" FeaturedMeasureBarStroke="Red"
ComparativeMeasure="7"
FeaturedMeasureBarStrokeThickness="10" Minimum="0" Maximum="10" Interval="2"
MinorTicksPerInterval="3"
MinorTickSize="10" MajorTickSize="14">
  <syncfusion:SfBulletGraph.QualitativeRanges>
    <syncfusion:QualitativeRange RangeEnd="4.5"
RangeStroke="#EBEBEB"></syncfusion:QualitativeRange>
    <syncfusion:QualitativeRange RangeEnd="10"
RangeStroke="#7F7F7F"></syncfusion:QualitativeRange>
    <syncfusion:QualitativeRange RangeEnd="7.5"
RangeStroke="#D8D8D8"></syncfusion:QualitativeRange>
  </syncfusion:SfBulletGraph.QualitativeRanges>
</syncfusion:SfBulletGraph>
```

**C#**

```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.Interval = 2;
bulletgraph.FeaturedMeasure = 5;
bulletgraph.ComparativeMeasure = 7;
bulletgraph.FeaturedMeasureBarStrokeThickness = 10;
bulletgraph.FeaturedMeasureBarStroke = new SolidColorBrush(Colors.Red);
bulletgraph.MinorTicksPerInterval = 3;
bulletgraph.MinorTickSize = 10;
bulletgraph.MajorTickSize = 14;
QualitativeRange range1 = new QualitativeRange();
range1.RangeEnd = 4.5;
range1.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#EBEBEB");
QualitativeRange range2 = new QualitativeRange();
range2.RangeEnd = 10;
range2.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#7F7F7F");
QualitativeRange range3 = new QualitativeRange();
range3.RangeEnd = 7.5;
range3.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#D8D8D8");
bulletgraph.QualitativeRanges.Add(range1);
bulletgraph.QualitativeRanges.Add(range2);
bulletgraph.QualitativeRanges.Add(range3);
grid.Children.Add(bulletgraph);
```



### Comparative Measure

Comparative measure should be less visually dominant than the featured measure. It should always be encoded as a short line that runs perpendicular to the orientation of the graph. A good example would be a target for YTD revenue. Whenever the featured measure intersects a comparative measure, the comparative measure should appear behind the featured measure.

### Customizing comparative measure

The value of the comparative measure is set by using the [ComparativeMeasure](#) property. By setting the [ComparativeMeasureSymbolStroke](#) property, the stroke of the comparative measure symbol is customized. The thickness of the comparative measure symbol is modified by using [ComparativeMeasureSymbolStrokeThickness](#).

### XML

```
<syncfusion:SfBulletGraph FeaturedMeasure="5"
FeaturedMeasureBarStroke="Black" ComparativeMeasure="7"
ComparativeMeasureSymbolStrokeThickness="6"
ComparativeMeasureSymbolStroke="Red" Minimum="0" Maximum="10" Interval="2"
MinorTicksPerInterval="3"
MinorTickSize="10" MajorTickSize="14">
  <syncfusion:SfBulletGraph.QualitativeRanges>
    <syncfusion:QualitativeRange RangeEnd="4.5"
RangeStroke="#EBEBEB"></syncfusion:QualitativeRange>
    <syncfusion:QualitativeRange RangeEnd="10"
RangeStroke="#7F7F7F"></syncfusion:QualitativeRange>
    <syncfusion:QualitativeRange RangeEnd="7.5"
RangeStroke="#D8D8D8"></syncfusion:QualitativeRange>
  </syncfusion:SfBulletGraph.QualitativeRanges>
</syncfusion:SfBulletGraph>
```

### C#

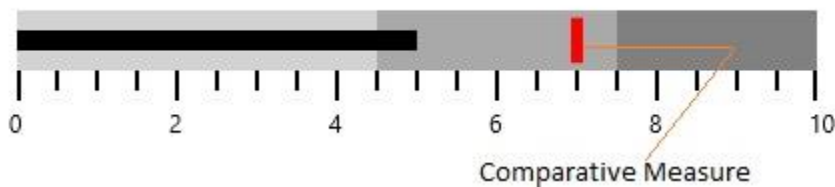
```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.Interval = 2;
bulletgraph.FeaturedMeasure = 5;
bulletgraph.ComparativeMeasure = 7;
bulletgraph.ComparativeMeasureSymbolStroke = new
SolidColorBrush(Colors.Red);
bulletgraph.ComparativeMeasureSymbolStrokeThickness = 6;
bulletgraph.FeaturedMeasureBarStroke = new SolidColorBrush(Colors.Black);
```



```

bulletgraph.MinorTicksPerInterval = 3;
bulletgraph.MinorTickSize = 10;
bulletgraph.MajorTickSize = 14;
QualitativeRange range1 = new QualitativeRange();
range1.RangeEnd = 4.5;
range1.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#EBEBEB");
QualitativeRange range2 = new QualitativeRange();
range2.RangeEnd = 10;
range2.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#7F7F7F");
QualitativeRange range3 = new QualitativeRange();
range3.RangeEnd = 7.5;
range3.RangeStroke = (Brush)new BrushConverter().ConvertFrom("#D8D8D8");
bulletgraph.QualitativeRanges.Add(range1);
bulletgraph.QualitativeRanges.Add(range2);
bulletgraph.QualitativeRanges.Add(range3);
grid.Children.Add(bulletgraph);

```



### Ranges in WPF Bullet Graph (SfBulletGraph)

Ranges for a bullet graph are a collection of qualitative ranges. A qualitative range is a visual element that begins at [RangeStart](#) and ends at a specified [RangeEnd](#) at the beginning of the previous range's [RangeEnd](#). The qualitative ranges are arranged according to each [RangeEnd](#) value.

#### Customizing Range

The width of the ranges can be customized by setting the [QualitativeRangesSize](#) property. By changing [RangeStroke](#) of the qualitative range, the stroke of the range can be personalized. By setting the [RangeOpacity](#) of the qualitative range, the opacity of the range is modified.

#### XML

```

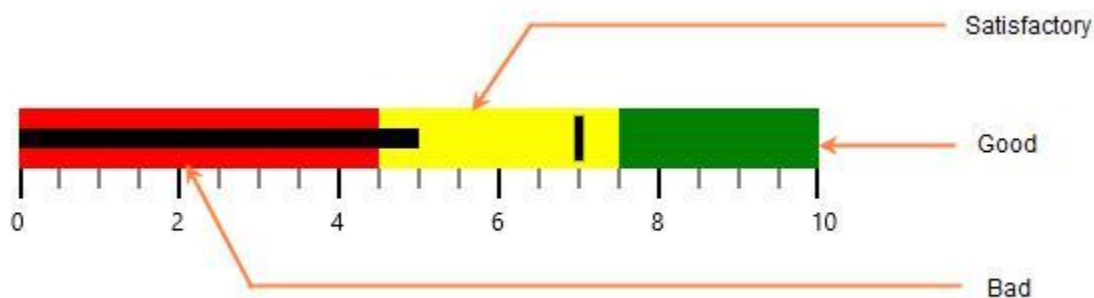
<syncfusion:SfBulletGraph QualitativeRangesSize="30" Minimum="0"
Maximum="10" Interval="2"
MinorTicksPerInterval="3" MinorTickSize="8" FeaturedMeasure="5"
ComparativeMeasure="7">
  <syncfusion:SfBulletGraph.QualitativeRanges>
    <syncfusion:QualitativeRange RangeEnd="4.5"
RangeStroke="Red"
RangeOpacity="1">
    </syncfusion:QualitativeRange>
    <syncfusion:QualitativeRange RangeEnd="7.5"
RangeStroke="Yellow"
RangeOpacity="1">
    </syncfusion:QualitativeRange>
    <syncfusion:QualitativeRange RangeEnd="10"
RangeStroke="Green"
RangeOpacity="1">
    </syncfusion:QualitativeRange>
  </syncfusion:SfBulletGraph.QualitativeRanges>

```

```
</syncfusion:SfBulletGraph>
```

**C#**

```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.FeaturedMeasure = 5;
bulletgraph.ComparativeMeasure = 7;
bulletgraph.Interval = 2;
bulletgraph.MinorTickSize = 8;
bulletgraph.MinorTicksPerInterval = 3;
bulletgraph.QualitativeRangesSize = 30;
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 4.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Red)
});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 7.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Yellow)
});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 10,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Green)
});
Grid.SetColumn(bulletgraph, 0);
grid.Children.Add(bulletgraph);
```

**Binding RangeStroke to Ticks and Labels**

By setting [BindWithRangeStrokeToLabels](#), the stroke of the labels is set related to the stroke of the specified ranges. Similarly, by setting [BindWithRangeStrokeToTicks](#), the stroke of the ticks is set related to the stroke of the specified ranges.

**XML**

```

<syncfusion:SfBulletGraph QualitativeRangesSize="30" Minimum="0"
Maximum="10" Interval="2"
BindRangeStrokeToLabels="True" BindRangeStrokeToTicks="True"
MinorTicksPerInterval="3" MinorTickSize="8" FeaturedMeasure="5"
ComparativeMeasure="7">
<syncfusion:SfBulletGraph.QualitativeRanges>
<syncfusion:QualitativeRange RangeEnd="4.5"
RangeStroke="Red"
RangeOpacity="1">
</syncfusion:QualitativeRange>
<syncfusion:QualitativeRange RangeEnd="7.5"
RangeStroke="Yellow"
RangeOpacity="1">
</syncfusion:QualitativeRange>
<syncfusion:QualitativeRange RangeEnd="10"
RangeStroke="Green"
RangeOpacity="1">
</syncfusion:QualitativeRange>
</syncfusion:SfBulletGraph.QualitativeRanges>
</syncfusion:SfBulletGraph>

```

**C#**

```

SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.FeaturedMeasure = 5;
bulletgraph.ComparativeMeasure = 7;
bulletgraph.Interval = 2;
bulletgraph.MinorTickSize = 8;
bulletgraph.MinorTicksPerInterval = 3;
bulletgraph.QualitativeRangesSize = 30;
bulletgraph.BindRangeStrokeToLabels = true;
bulletgraph.BindRangeStrokeToTicks = true;
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 4.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Red)
});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 7.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Yellow)
});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 10,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Green)
});
Grid.SetColumn(bulletgraph, 0);
grid.Children.Add(bulletgraph);

```



### Tooltip in WPF Bullet Graph (SfBulletGraph)

Tooltip in **SfBulletGraph** is used to view the values of **FeaturedMeasure**, **ComparativeMeasure** and **QualitativeRange** in required design.

This tooltip is displayed when the mouse is over the **FeaturedMeasure**, **ComparativeMeasure** or **QualitativeRange**. Whereas, in touch device it is displayed on holding over the **FeaturedMeasure**, **ComparativeMeasure** and **QualitativeRange** of **SfBulletGraph**.

The **SfBulletGraph** tooltip is displayed only when the [ShowToolTip](#) property is set to true.

### XML

```
<syncfusion:SfBulletGraph QualitativeRangesSize="30" Minimum="0"
Maximum="10" Interval="2"
MinorTicksPerInterval="3" MinorTickSize="8" FeaturedMeasure="5"
ShowToolTip="True" FeaturedMeasureBarStroke="Black"
ComparativeMeasure="4" >
  <syncfusion:SfBulletGraph.QualitativeRanges>
    <syncfusion:QualitativeRange RangeEnd="4.5"
RangeStroke="Red"
RangeOpacity="1">
    </syncfusion:QualitativeRange>
    <syncfusion:QualitativeRange RangeEnd="7.5"
RangeStroke="Yellow"
RangeOpacity="1">
    </syncfusion:QualitativeRange>
    <syncfusion:QualitativeRange RangeEnd="10"
RangeStroke="Green"
RangeOpacity="1">
    </syncfusion:QualitativeRange>
  </syncfusion:SfBulletGraph.QualitativeRanges>
</syncfusion:SfBulletGraph>
```

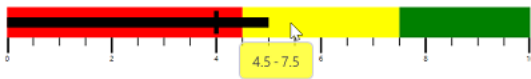
### C#

```
SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.FeaturedMeasure = 5;
bulletgraph.ComparativeMeasure = 4;
bulletgraph.Interval = 2;
bulletgraph.MinorTickSize = 8;
bulletgraph.MinorTicksPerInterval = 3;
bulletgraph.QualitativeRangesSize = 30;
bulletgraph.FeaturedMeasureBarStroke = new SolidColorBrush(Colors.Black);
bulletgraph.ShowToolTip = true;
bulletgraph.QualitativeRanges.Add(new QualitativeRange())
```

```

{
    RangeEnd = 4.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Red)
});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 7.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Yellow)
});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 10,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Green)
});
grid.Children.Add(bulletgraph);

```



### Customizing FeaturedMeasureToolTip

You can display the value of [FeaturedMeasure](#) of SfBulletGraph in ToolTip that is used to view the FeaturedMeasure value. The [FeaturedMeasureToolTipTemplate](#) is DataTemplate type by using this property, the default appearance of the FeaturedMeasure tooltip can be customized.

### XML

```

<Grid x:Name="grid">
<Grid.Resources>
<DataTemplate x:Key="tooltipTemplate">
<Border BorderBrush="#D3D3D3" BorderThickness="1.5" Background="#232323"
CornerRadius="5">
<TextBlock Text="{Binding}" FontSize="14" Foreground="#D3D3D3" Margin="12
8"/>
</Border>
</DataTemplate>
</Grid.Resources>
<syncfusion:SfBulletGraph QualitativeRangesSize="30" Minimum="0"
Maximum="10" Interval="2"
MinorTicksPerInterval="3" MinorTickSize="8" FeaturedMeasure="5"
ShowToolTip="True" FeaturedMeasureBarStroke="Black"
ComparativeMeasure="4" FeaturedMeasureToolTipTemplate="{StaticResource
tooltipTemplate}">
<syncfusion:SfBulletGraph.QualitativeRanges>
<syncfusion:QualitativeRange RangeEnd="4.5"
RangeStroke="Red"
RangeOpacity="1">
</syncfusion:QualitativeRange>
<syncfusion:QualitativeRange RangeEnd="7.5"
RangeStroke="Yellow"

```

```

RangeOpacity="1">
</syncfusion:QualitativeRange>
<syncfusion:QualitativeRange RangeEnd="10"
RangeStroke="Green"
RangeOpacity="1">
</syncfusion:QualitativeRange>
</syncfusion:SfBulletGraph.QualitativeRanges>
</syncfusion:SfBulletGraph>
</Grid>

```

## C#

```

SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.FeaturedMeasure = 5;
bulletgraph.ComparativeMeasure = 4;
bulletgraph.Interval = 2;
bulletgraph.MinorTickSize = 8;
bulletgraph.MinorTicksPerInterval = 3;
bulletgraph.QualitativeRangesSize = 30;
bulletgraph.FeaturedMeasureBarStroke = new SolidColorBrush(Colors.Black);
bulletgraph.ShowToolTip = true;
bulletgraph.FeaturedMeasureToolTipTemplate =
grid.Resources["tooltipTemplate"] as DataTemplate;
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 4.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Red)
});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 7.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Yellow)
});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 10,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Green)
});
grid.Children.Add(bulletgraph);

```



### Customizing ComparativeMeasureToolTip

You can display the value of [ComparativeMeasure](#) of SfBulletGraph in ToolTip that is used to view the ComparativeMeasure value. The [ComparativeMeasureToolTipTemplate](#) is DataTemplate type by using this property, the default appearance of the ComparativeMeasure tooltip can be customized.

**XML**

```

<Grid x:Name="grid">
<Grid.Resources>
<DataTemplate x:Key="tooltipTemplate">
<Border BorderBrush="#D3D3D3" BorderThickness="1.5" Background="#232323"
CornerRadius="5">
<TextBlock Text="{Binding}" FontSize="14" Foreground="#D3D3D3" Margin="12
8"/>
</Border>
</DataTemplate>
</Grid.Resources>
<syncfusion:SfBulletGraph QualitativeRangesSize="30" Minimum="0"
Maximum="10" Interval="2"
MinorTicksPerInterval="3" MinorTickSize="8" FeaturedMeasure="5"
ShowToolTip="True" FeaturedMeasureBarStroke="Black"
ComparativeMeasure="4" ComparativeMeasureToolTipTemplate="{StaticResource
tooltipTemplate}">
<syncfusion:SfBulletGraph.QualitativeRanges>
<syncfusion:QualitativeRange RangeEnd="4.5"
RangeStroke="Red"
RangeOpacity="1">
</syncfusion:QualitativeRange>
<syncfusion:QualitativeRange RangeEnd="7.5"
RangeStroke="Yellow"
RangeOpacity="1">
</syncfusion:QualitativeRange>
<syncfusion:QualitativeRange RangeEnd="10"
RangeStroke="Green"
RangeOpacity="1">
</syncfusion:QualitativeRange>
</syncfusion:SfBulletGraph.QualitativeRanges>
</syncfusion:SfBulletGraph>
</Grid>

```

**C#**

```

SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.FeaturedMeasure = 5;
bulletgraph.ComparativeMeasure = 4;
bulletgraph.Interval = 2;
bulletgraph.MinorTickSize = 8;
bulletgraph.MinorTicksPerInterval = 3;
bulletgraph.QualitativeRangesSize = 30;
bulletgraph.FeaturedMeasureBarStroke = new SolidColorBrush(Colors.Black);
bulletgraph.ShowToolTip = true;
bulletgraph.ComparativeMeasureToolTipTemplate =
grid.Resources["tooltipTemplate"] as DataTemplate;
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
RangeEnd = 4.5,
RangeOpacity = 1,
RangeStroke = new SolidColorBrush(Colors.Red)
});

```

```

bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 7.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Yellow)
});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 10,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Green)
});
grid.Children.Add(bulletgraph);

```



### Customizing QualitativeRangeToolTip

You can display the value of QualitativeRange of SfBulletGraph in ToolTip that is used to view the Start and End value of QualitativeRange. The [QualitativeRangeToolTipTemplate](#) is DataTemplate type by using this property, the default appearance of the QualitativeRange tooltip can be customized.

### XML

```

<Grid x:Name="grid">
  <Grid.Resources>
    <DataTemplate x:Key="tooltipTemplate">
      <Border BorderBrush="Black" BorderThickness="1.5" CornerRadius="5">
        <Border Background="{Binding RangeStroke}" Opacity="0.7" CornerRadius="5">
          <StackPanel Orientation="Horizontal" Margin="12 8" >
            <TextBlock Text="{Binding RangeEnd}" FontSize="14" Foreground="Black"/>
          </StackPanel>
        </Border>
      </Border>
    </DataTemplate>
  </Grid.Resources>
  <syncfusion:SfBulletGraph QualitativeRangesSize="30" Minimum="0"
    Maximum="10" Interval="2"
    MinorTicksPerInterval="3" MinorTickSize="8" FeaturedMeasure="5"
    ShowToolTip="True" FeaturedMeasureBarStroke="Black"
    ComparativeMeasure="4"
    QualitativeRangeToolTipTemplate="{StaticResource tooltipTemplate}">
    <syncfusion:SfBulletGraph.QualitativeRanges>
      <syncfusion:QualitativeRange RangeEnd="4.5"
        RangeStroke="Red"
        RangeOpacity="1">
      </syncfusion:QualitativeRange>
      <syncfusion:QualitativeRange RangeEnd="7.5"
        RangeStroke="Yellow"
        RangeOpacity="1">
      </syncfusion:QualitativeRange>
      <syncfusion:QualitativeRange RangeEnd="10"

```



```

RangeStroke="Green"
RangeOpacity="1">
</syncfusion:QualitativeRange>
</syncfusion:SfBulletGraph.QualitativeRanges>
</syncfusion:SfBulletGraph>
</Grid>

```

**C#**

```

SfBulletGraph bulletgraph = new SfBulletGraph();
bulletgraph.Minimum = 0;
bulletgraph.Maximum = 10;
bulletgraph.FeaturedMeasure = 5;
bulletgraph.ShowToolTip = "True";
bulletgraph.ComparativeMeasure = 4;
bulletgraph.Interval = 2;
bulletgraph.MinorTickSize = 8;
bulletgraph.MinorTicksPerInterval = 3;
bulletgraph.QualitativeRangesSize = 30;
bulletgraph.FeaturedMeasureBarStroke = new SolidColorBrush(Colors.Black);
bulletgraph.QualitativeRangeToolTipTemplate =
grid.Resources["tooltipTemplate"] as DataTemplate;
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 4.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Red)
});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 7.5,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Yellow)
});
bulletgraph.QualitativeRanges.Add(new QualitativeRange()
{
    RangeEnd = 10,
    RangeOpacity = 1,
    RangeStroke = new SolidColorBrush(Colors.Green)
});
grid.Children.Add(bulletgraph);

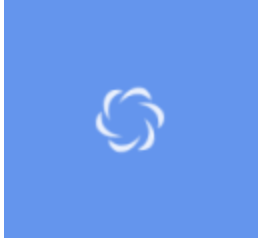
```



## SfBusyIndicator

### WPF Busy Indicator (SfBusyIndicator) Overview

The Busy Indicator control includes over 37 built-in animations that can be displayed within your applications.



## Busy Indicator

### Key Features

- **IsBusy** – The control will show the animation when this property is set.
- **Header** – A user customized string that is displayed while showing animation.
- **Sizing** – The control view box height and width can be set.
- **AnimationTypes** – There are over 37 built-in animations that can be set with the control.

### Getting Started with WPF Busy Indicator (SfBusyIndicator)

Namespace: Syncfusion.Windows.Controls.Notification.

Assembly: Syncfusion.SfBusyIndicator.WPF (in Syncfusion.SfBusyIndicator.WPF.dll)

The following code example shows how to create the SfBusyIndicator from XAML and code behind respectively.

#### XML

```
<Page xmlns:Notification="clr-namespace:Syncfusion.Windows.Controls.Notification;assembly=Syncfusion.SfBusyIndicator.WPF">
  <Grid Background="CornflowerBlue">
    <Notification:SfBusyIndicator/>
  </Grid>
</Page>
```

#### C#

```
SfBusyIndicator busyIndicator = new SfBusyIndicator();
```

---

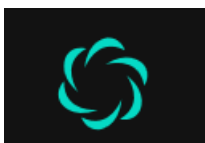
**Note:** View [sample](#) in GitHub

---

#### Theme

SfBusyIndicator supports various built-in themes. Refer to the below links to apply themes for the SfBusyIndicator,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



## IsBusy in WPF Busy Indicator (SfBusyIndicator)

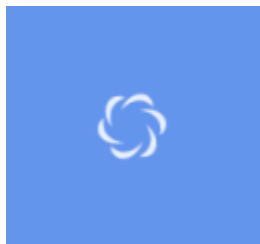
The IsBusy property in the BusyIndicator control is used to determine whether an animation needs to be executed or not.

### XML

```
<Grid Background="CornflowerBlue">
<Navigation:SfBusyIndicator IsBusy="True"/>
</Grid>
```

### C#

```
SfBusyIndicator busyIndicator = new SfBusyIndicator();
busyIndicator.IsBusy = true;
```



Busy Indicator

**Note:** View [sample](#) in GitHub

## Header in WPF Busy Indicator (SfBusyIndicator)

### Header

The Header is displayed below the animation. The Header property can be used to get or set the content which indicates the information related to loading.

### XML

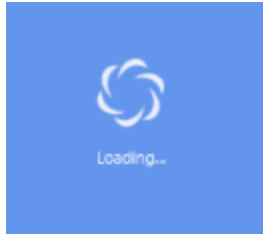
```
<!--To set the header for SfBusyIndicator-->
<Grid Background="CornflowerBlue">
<Notification:SfBusyIndicator Header="Loading..." Foreground="White" />
</Grid>
```

### C#

```
// To set the Header for SfBusyIndicator
SfBusyIndicator SfBusyIndicator = new SfBusyIndicator();
SfBusyIndicator.Header = "Loading..";
grid1.Children.Add(SfBusyIndicator);
```

### VB.NET

```
'To set the Header for SfBusyIndicator
Dim SfBusyIndicator As New SfBusyIndicator()
SfBusyIndicator.Header = "Loading.."
grid1.Children.Add(SfBusyIndicator)
```



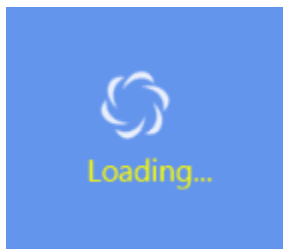
Busy Indicator with header

## Header Template

Header template can be used to get or set the template that defines how the header section of the SfBusyIndicator control is displayed.

**XML**

```
<!--To set the HeaderTemplate for SfBusyIndicator-->
<Notification:SfBusyIndicator Grid.Row="0" Foreground="White"
Background="CornflowerBlue" >
<Notification:SfBusyIndicator.HeaderTemplate>
<DataTemplate >
<TextBlock Text="Loading..." TextAlignment="Center" FontSize="15"
Width="100" Foreground="Yellow" />
</DataTemplate>
</Notification:SfBusyIndicator.HeaderTemplate>
</Notification:SfBusyIndicator>
```



Busy Indicator with header template

**Note:** View [sample](#) in GitHub

## Sizing in WPF Busy Indicator (SfBusyIndicator)

## ViewBoxHeight

The ViewBoxHeight property allows the user to set the height of the ViewBox.

**XML**

```
<!--To set the ViewboxHeight for SfBusyIndicator-->
<Notification:SfBusyIndicator Foreground="White" Header="Loading..."
Background="CornflowerBlue" ViewboxHeight="200" >
</Notification:SfBusyIndicator>
```

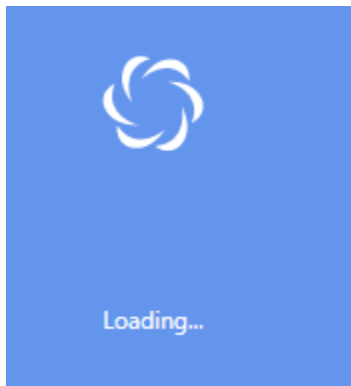
**C#**

```
// To set the ViewboxHeight for SfBusyIndicator
SfBusyIndicator SfBusyIndicator = new SfBusyIndicator();
SfBusyIndicator.Header = "Loading..";
```

```
SfBusyIndicator.Foreground = Brushes.White;
SfBusyIndicator.Background = Brushes.CornflowerBlue;
SfBusyIndicator.ViewboxHeight = 200;
Grid1.Children.Add(SfBusyIndicator);
```

**VB.NET**

```
' To set the ViewboxHeight for SfBusyIndicator
Dim SfBusyIndicator As New SfBusyIndicator()
SfBusyIndicator.Header = "Loading.."
SfBusyIndicator.Foreground = Brushes.White
SfBusyIndicatorBackground = Brushes.CornflowerBlue
SfBusyIndicator.ViewboxHeight = 200
Grid1.Children.Add(SfBusyIndicator)
```

**ViewBoxWidth**

ViewBoxWidth property allows the user to set the width of the ViewBox.

**XML**

```
<!--To set the ViewBoxWidth for SfBusyIndicator-->
<Notification:SfBusyIndicator Foreground="White" Header="Loading..."
Background="CornflowerBlue" ViewboxWidth="50" >
</Notification:SfBusyIndicator>
```

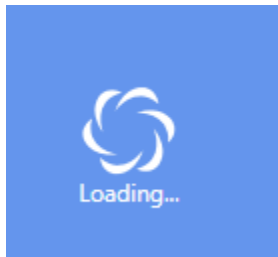
**C#**

```
// To set the ViewBoxWidth for SfBusyIndicator
SfBusyIndicator SfBusyIndicator = new SfBusyIndicator();
SfBusyIndicator.Header = "Loading..";
SfBusyIndicator.Foreground = Brushes.White;
SfBusyIndicator.Background = Brushes.CornflowerBlue;
SfBusyIndicator.ViewboxWidth = 50;
Grid1.Children.Add(SfBusyIndicator);
```

**VB.NET**

```
' To set the ViewBoxWidth for SfBusyIndicator
Dim SfBusyIndicator As New SfBusyIndicator()
SfBusyIndicator.Header = "Loading.."
SfBusyIndicator.Foreground = Brushes.White
```

```
SfBusyIndicator.Background = Brushes.CornflowerBlue  
SfBusyIndicator.ViewboxWidth = 50  
Grid1.Children.Add(SfBusyIndicator)
```



Busy Indicator with height and width

**Note:** View [sample](#) in GitHub

### AnimationTypes in WPF Busy Indicator (SfBusyIndicator)

The AnimationTypes property for the SfBusyIndicator allows the user to set one of the animations from the built-in animations as the busy indicator.

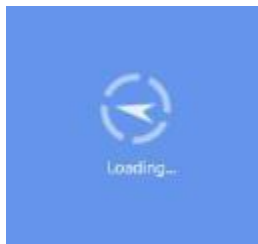
**Note:** AnimationSpeed property is not applicable for Fluent animation type.

#### XML

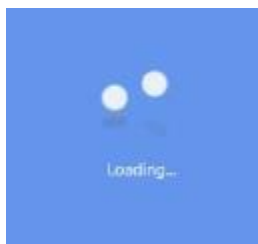
```
<Grid Background="CornflowerBlue">  
<Notification:SfBusyIndicator AnimationType="Flight"/>  
</Grid>
```

#### C#

```
SfBusyIndicator busyIndicator = new SfBusyIndicator();  
busyIndicator.AnimationType = AnimationTypes.Flight;
```



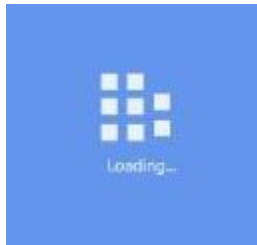
Busy Indicator with ArrowTrack type animation



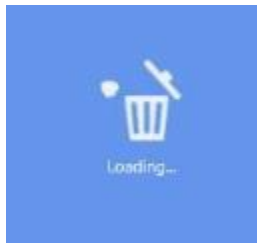
Busy Indicator with Ball type animation



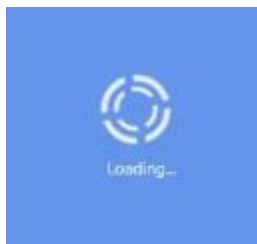
Busy Indicator with Battery type animation



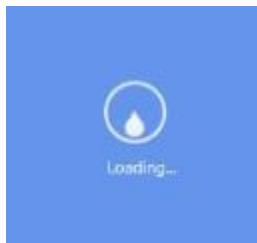
Busy Indicator with Box type animation



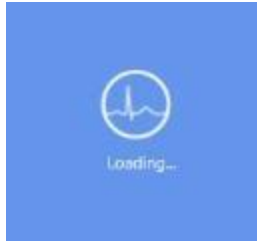
Busy Indicator with Delete type animation



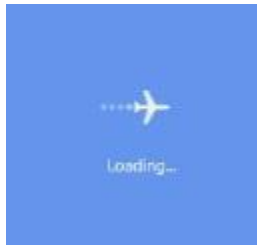
Busy Indicator with DoubleCircle type animation



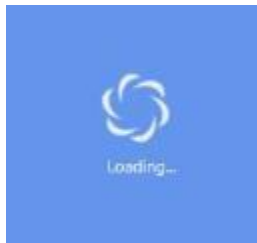
Busy Indicator with Drop type animation



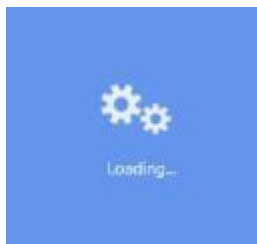
Busy Indicator with ECG type animation



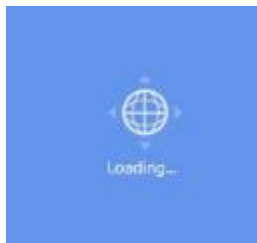
Busy Indicator with Flight type animation



Busy Indicator with Flower type animation

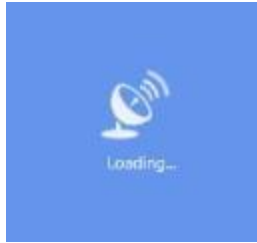


Busy Indicator with Gear type animation

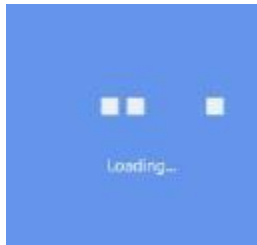


Busy Indicator with Globe type animation

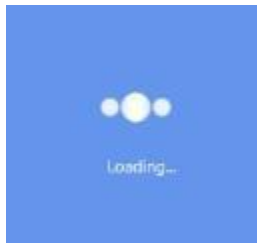




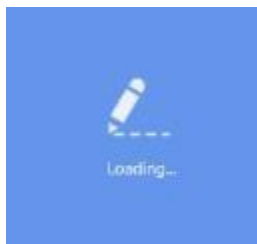
Busy Indicator with GPS type animation



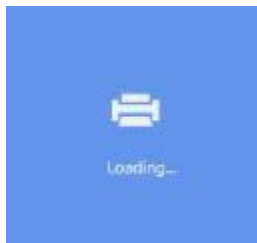
Busy Indicator with HorizontalPulsingBox type animation



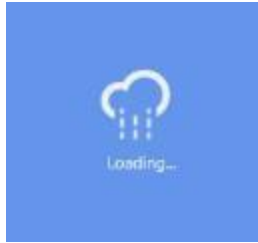
Busy Indicator with Liquid type animation



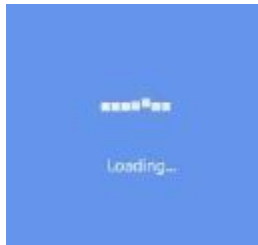
Busy Indicator with Pen type animation



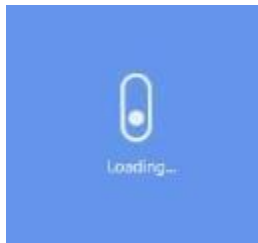
Busy Indicator with Print type animation



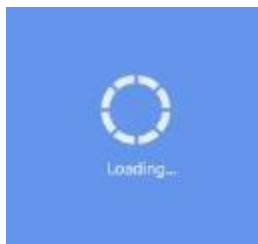
Busy Indicator with Rain type animation



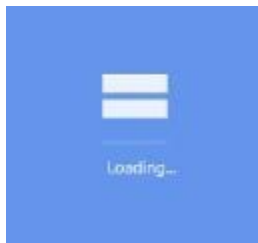
Busy Indicator with Rectangle type animation



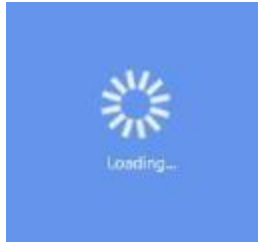
Busy Indicator with Rotation type animation



Busy Indicator with SingleCircle type animation



Busy Indicator with SliceBox type animation



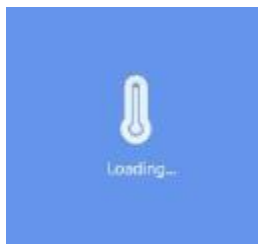
Busy Indicator with SlicedCircle type animation



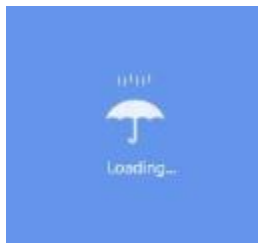
Busy Indicator with Snow type animation



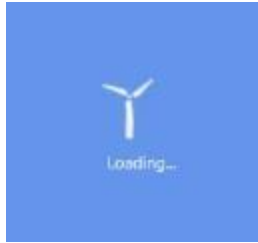
Busy Indicator with Sunny type animation



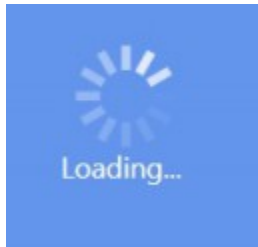
Busy Indicator with Temperature type animation



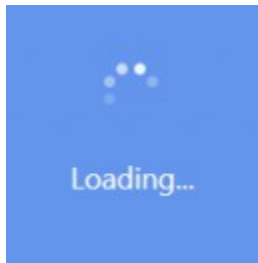
Busy Indicator with Umbrella type animation



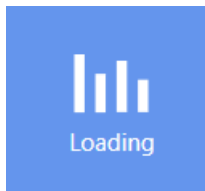
Busy Indicator with Windmill type animation



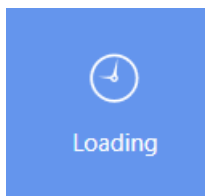
Busy Indicator with Cupertino type animation



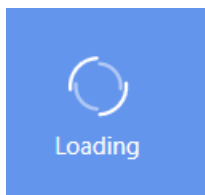
Busy Indicator with DotCircle type animation



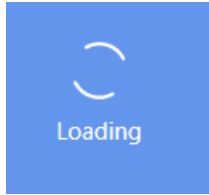
Busy Indicator with BarChart type animation



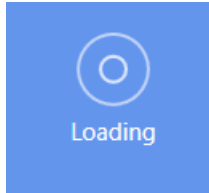
Busy Indicator with Clock type animation



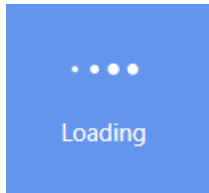
Busy Indicator with DoubleRing type animation



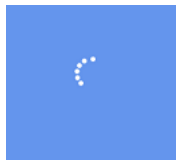
Busy Indicator with DualRing type animation



Busy Indicator with Ripple type animation



Busy Indicator with Message type animation



Busy Indicator with Fluent type animation

---

**Note:** View [sample](#) in GitHub

---

## SfBadge

### WPF Badge (SfBadge) Overview

[Badge](#) control used to notify users of new or unread messages, notifications, or the status of something.

Badge control



### Key features

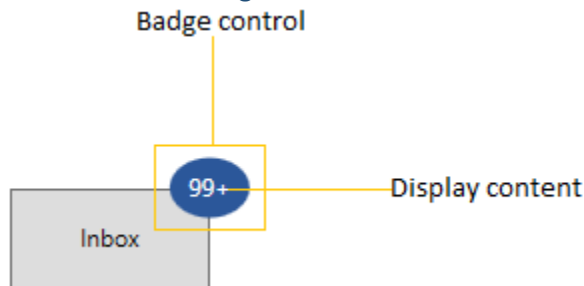
- Position - Position the **Badge** text around the **Badge** content.
- Predefined colors - Customize the **Badge** background with predefined colors for the success, warning, error and information states.
- Animation - Use animations for **Badge** text.
- Predefined and custom shapes for **Badge** control.
- Rotation - Rotate the **Badge** to any angle.

- Alignment - Align the **Badge** at any position
- Custom UI - Customized UI for control and its content.

### Getting Started with WPF Badge (SfBadge)

This section explains the steps required to add the [Badge](#) control and its elements such as shapes, alignment and predefined colors. This section covers only basic features needed to get started with Syncfusion **Badge** control.

#### Structure of SfBadge control



#### Assembly deployment

Refer to the [Control Dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

Refer to this [documentation](#) to find more details about installing nuget packages in a WPF application.

#### Adding WPF SfBadge via XAML

To add the **SfBadge** manually in XAML, follow these steps:

1. Create a new WPF project in Visual Studio.
2. Add the following required assembly references to the project:
  - Syncfusion.Shared.WPF
  - Syncfusion.Tools.WPF
3. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf>, and declare the **SfBadge** in XAML page.

#### XML

```
<Window
  x:Class="GettingStarted.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:notification="http://schemas.syncfusion.com/wpf"
  mc:Ignorable="d">
  <Grid>
    <notification:SfBadge Name="badge" />
  </Grid>
</Window>
```

#### Adding WPF SfBadge via C#

To add the **SfBadge** control manually in C#, follow these steps:

1. Create a new WPF application via Visual Studio.
2. Add the following assembly references to the project,
  - Syncfusion.Shared.WPF
  - Syncfusion.Tools.WPF
3. Include the required namespace and create an instance of **SfBadge** and add it to the window.
4. Declare the **SfBadge** control using C#.

### C#

```
using Syncfusion.Windows.Controls.Notification;
namespace GettingStarted
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            this.InitializeComponent();
            // Creating an instance of the Badge control
            SfBadge badge = new SfBadge();
            grid.Children.Add(badge);
        }
    }
}
```

### Adding badge for a Button

If you want to assign **Badge** for any objects, create the **Badge** and assign the badge to the **SfBadge.Badge** property. Before that you need to create a **SfBadge.Badge** object and add that object to the parent control.

Here, **Badge** control added for the **Button** control.

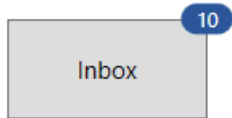
### XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Content="10"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

### C#

```
//Creating Badge control
SfBadge sfBadge = new SfBadge();
sfBadge.Name = "badge";
sfBadge.Content = 10;
//Create button control as container for badge
Button button = new Button();
button.Width = 100;
```

```
button.Height = 50;  
button.Content = "Primary";  
//Assigning Badge control to button  
SfBadge.SetBadge(button, sfBadge);
```



**Note:** Download demo application from [GitHub](#)

### Setting Badge display content

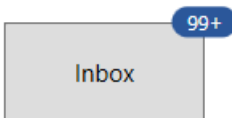
If you want to set or change the display content of the **Badge**, use the **Content** property. The default value of **Content** property is **null**.

#### XML

```
<Button Width="100"  
Height="50"  
Content="Inbox">  
<notification:SfBadge.Badge>  
<notification:SfBadge Content="99+"  
x:Name="badge"/>  
</notification:SfBadge.Badge>  
</Button>
```

#### C#

```
badge.Content = "99+";
```



**Note:** Download demo application from [GitHub](#)

### Alignment of Badge

you can align the **Badge** either horizontally or vertically by using the **HorizontalAlignment** or **VerticalAlignment** properties. The default value of **HorizontalAlignment** property is **Right** and **VerticalAlignment** property is **Top**.

#### XML

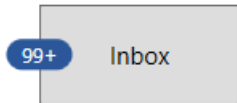
```
<Button Width="100"  
Height="50"  
Content="Inbox">  
<notification:SfBadge.Badge>  
<notification:SfBadge HorizontalAlignment="Left"  
VerticalAlignment="Center"  
Content="99+"  
x:Name="badge"/>  
</notification:SfBadge.Badge>
```



```
</Button>
```

**C#**

```
badge.HorizontalAlignment = HorizontalAlignment.Left;
badge.VerticalAlignment = VerticalAlignment.Center;
badge.Content = "99+";
```



**Note:** Download demo application from [GitHub](#)

**Positioning of Badge**

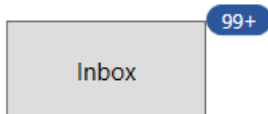
You can change the horizontal or vertical position of the **Badge** either inside, outside or in the middle by using the [HorizontalAnchor](#) and [VerticalAnchor](#) properties. The default value of [HorizontalAnchor](#) and [VerticalAnchor](#) properties is **Center**.

**XML**

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge HorizontalAnchor="Outside"
VerticalAnchor="Center"
Content="10"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

**C#**

```
badge.HorizontalAnchor = BadgeAnchor.Outside;
badge.VerticalAnchor = BadgeAnchor.Center;
badge.Content = "10";
```



**Note:** Download demo application from [GitHub](#)

**Place the Badge any where on the container**

If you want to place the **Badge** anywhere on any shaped container, use the [HorizontalPosition](#) or [VerticalPosition](#) properties. The value range for [HorizontalPosition](#) and [VerticalPosition](#) properties is 0 to 1. The default value of [HorizontalPosition](#) property is 1 and [VerticalPosition](#) property is 0.

For example, if you use any circular containers, you can easily place the **Badge** anywhere by using the [HorizontalPosition](#) and [VerticalPosition](#) properties.

## XML

```
<Image Source="/Images/avatar.png"
Width="100"
Height="100" >
<notification:SfBadge.Badge>
<notification:SfBadge Shape="None"
HorizontalPosition="0.9"
VerticalPosition="0.8"
x:Name="badge">
<notification:SfBadge.Content>
<Ellipse Width="20"
Height="20"
Fill="LimeGreen"/>
</notification:SfBadge.Content>
</notification:SfBadge>
</notification:SfBadge.Badge>
</Image>
```

## C#

```
badge.HorizontalPosition = 0.9;
badge.VerticalPosition = 0.8;
```



**Note:** Download demo application from [GitHub](#)

## Adding badge without BadgeContainer

You can directly add the **Badge** to any objects without using the **SfBadge.Badge** container.

## C#

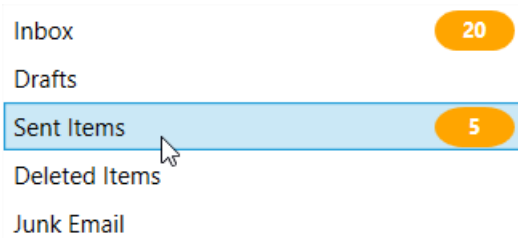
```
public class ViewModel {
public string ItemName { get; set; }
public int? UnreadMessageount { get; set; }
}

public class ViewModel {
public List<Model> MailItems { get; set; }
public ViewModel() {
MailItems = new List<Model>();
MailItems.Add(new Model()
{
ItemName = "Inbox",
UnreadMessageount = 20
});
MailItems.Add(new Model()
{
ItemName = "Drafts",
UnreadMessageount = null
});
};
```

```
MailItems.Add(new Model()
{
    ItemName = "Sent Items",
    UnreadMessagecount = 5
});
MailItems.Add(new Model()
{
    ItemName = "Deleted Items",
    UnreadMessagecount = null
});
MailItems.Add(new Model()
{
    ItemName = "Junk Email",
    UnreadMessagecount = null
});
}
```

## XML

```
<Window.DataContext>
<local:ViewModel></local:ViewModel>
</Window.DataContext>
<Grid>
<ListView BorderThickness="1"
BorderBrush="LightGray"
ItemsSource="{Binding MailItems}"
SelectedIndex="0"
VerticalAlignment="Center"
HorizontalAlignment="Center">
<ListView.ItemTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="150"/>
<ColumnDefinition Width="100"/>
</Grid.ColumnDefinitions>
<ContentPresenter Grid.Column="0"
Content="{Binding ItemName}"
VerticalAlignment="Center"/>
<notification:SfBadge x:Name="badge4"
Grid.Column="1"
Height="20"
Width="40"
Background="Orange"
Content="{Binding UnreadMessagecount}"
Shape="Oval"/>
</Grid>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</Grid>
```



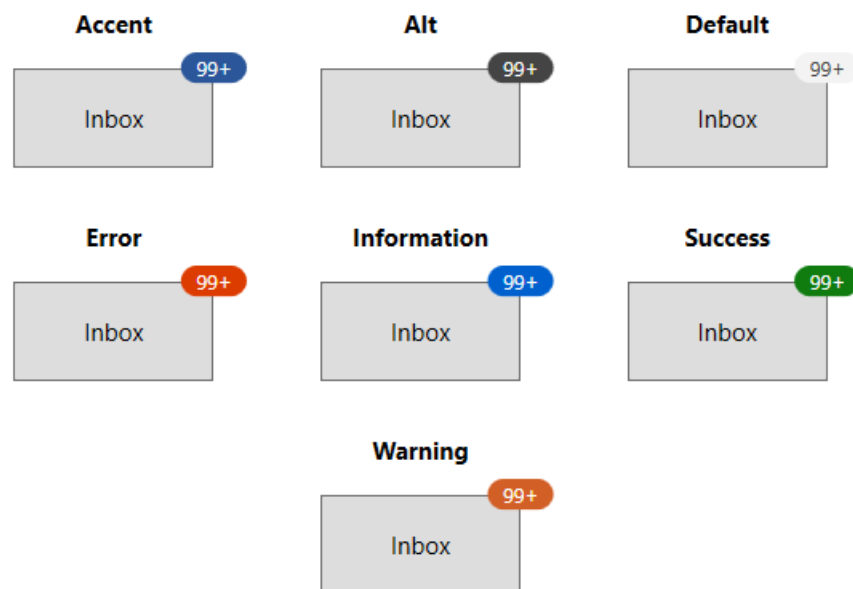
**Note:** Download demo application from [GitHub](#)

### Predefined colors for displaying the badges

You can change background color of the **Badge** by using the **Fill** property. Based on the value of **Fill** property, respective background color will be applied to the **Badge**. The default value of **Fill** property is **Accent**.

The **Badge** supports the following different essential states :

- Accent - DarkSlateBlue background will be applied
- Alt - DarkSlateGray background will be applied
- Default - WhiteSmoke background will be applied
- Error - OrangeRed background will be applied
- Information - RoyalBlue background will be applied
- Success - Green background will be applied
- Warning - Chocolate background will be applied



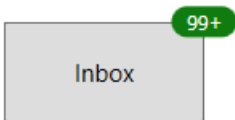
### XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Fill="Success"
Content="99+"
x:Name="badge"/>
```

```
</notification:SfBadge.Badge>
</Button>
```

### C#

```
badge.Fill = BadgeFill.Success;
badge.Content = "99+";
```

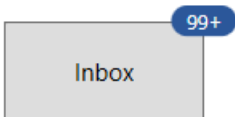


**Note:** Download demo application from [GitHub](#)

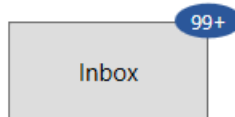
### Predefined shapes for displaying the Badge

You can change the default shape to either **Rectangle**, **Oval** or **Ellipse** by using **Shape** property. If you want to display the **Badge** content without any default shapes, use the **Shape** property value as **None**. The default value of **Shape** property is **Oval**.

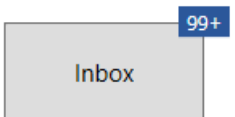
#### Default (Oval)



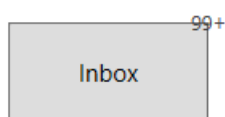
#### Ellipse



#### Rectangle



#### None

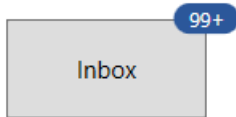


### XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Shape="Oval"
Content="99+"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

### C#

```
badge.Shape = BadgeShape.Oval;
badge.Content = "99+";
```



**Note:** Download demo application from [GitHub](#)

### Animate when content changes

You can enable the **Scale** or **Opacity** based animation for displaying the **Badge** text by using **AnimationType** property. You can only see the animation when you change the text of the **Badge**. The default value of **AnimationType** property is **None**.

### XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge AnimationType="Scale"
Shape="Ellipse"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
<StackPanel Orientation="Vertical">
<TextBlock Text="Badge content"
TextAlignment="Center" />
<notification:UpDown MinValue="1"
MaxValue="10"
Step="1"
NumberDecimalDigits="0"
Value="1"
x:Name="badgeContent"
ValueChanged="BadgeContent_ValueChanged"/>
```

### C#

```
badge.AnimationType = BadgeAnimationType.Scale;
badge.Shape = BadgeShape.Ellipse;
badgeContent.ValueChanged += BadgeContent_ValueChanged;
```

### C#

```
private void BadgeContent_ValueChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    this.badge.Content = Convert.ToInt32(e.NewValue).ToString();
}
```

### Scaling based animation



*Opacity based animation*

**Note:** Download demo application from [GitHub](#)

**Custom Content Formats**

You can format the numbers which are displayed in the **Badge** content by using the converters. For example, you can display the number as **99+** which is greater than or equal to **100**.

**XML**

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge x:Name="badge"
Shape="Ellipse"
AnimationType="None"/>
</notification:SfBadge.Badge>
</Button>
<StackPanel Orientation="Vertical">
<TextBlock Text="Badge content"
TextAlignment="Center" />
<syncfusion1:UpDown MinValue="1"
MaxValue="10000000"
Step="1"
NumberDecimalDigits="0"
Value="1"
x:Name="badgeContent"
ValueChanged="BadgeContent_ValueChanged"/>
</StackPanel>
```

**C#**

```
badge.AnimationType = BadgeAnimationType.None;
badge.Shape = BadgeShape.Ellipse;
badgeContent.ValueChanged += BadgeContent_ValueChanged;
```

**C#**

```
private void BadgeContent_ValueChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    int content_Value = Convert.ToInt32(e.NewValue);
    if (content_Value <= 99)
    {
        this.badge.Content = content_Value.ToString();
    }
    else if (content_Value <= 999)
    {
        this.badge.Content = "99+";
    }
}
```

```

else if (content_Value < 99999)
{
    this.badge.Content = (content_Value / 1000).ToString("0.#") + "K";
}
else if (content_Value < 999999)
{
    this.badge.Content = (content_Value / 1000).ToString("#,0K");
}
else if (content_Value < 9999999)
{
    this.badge.Content = (content_Value / 1000000).ToString("0.#") + "M";
}
else
{
    this.badge.Content = (content_Value / 1000000).ToString("#,0M");
}
}

```

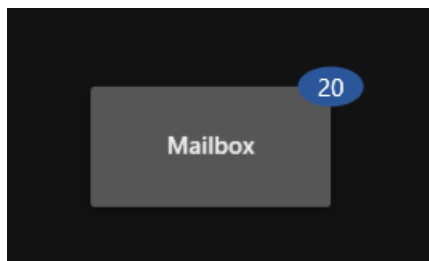


**Note:** Download demo application from [GitHub](#)

### Theme

Badge supports various built-in themes. Refer to the below links to apply themes for the Badge,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Customization in WPF Badge (SfBadge)

This section explains the customization features available in the WPF [Badge](#) control.

#### Adding badge for a Button

If you want to assign [Badge](#) for any objects, create the [Badge](#) and assign the badge to the [SfBadge.Badge](#) property. Before that you need to create a [SfBadge.Badge](#) object and add that object to the parent control.

Here, [Badge](#) control added for the [Button](#) control.

#### XML

```

<Button Width="100"
Height="50"

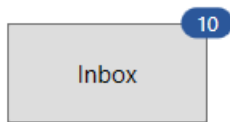
```



```
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Content="10"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

**C#**

```
/Creating Badge control
SfBadge sfBadge = new SfBadge();
sfBadge.Name = "badge";
sfBadge.Content = 10;
//Create button control as container for badge
Button button = new Button();
button.Width = 100;
button.Height = 50;
button.Content = "Primary";
//Assigning Badge control to button
SfBadge.SetBadge(button, sfBadge);
```



**Note:** Download demo application from [GitHub](#)

## Adding badge without BadgeContainer

You can directly add the `Badge` to any objects without using the `SfBadge.Badge` container.

**C#**

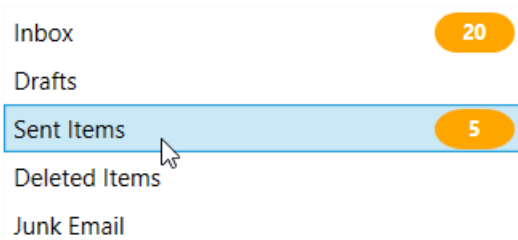
```
public class MailItem
{
    public string ItemName { get; set; }
    public int? UnreadMessageount { get; set; }
}

public class ViewModel {
    public List<MailItem> MailItems { get; set; }
    public ViewModel() {
        MailItems = new List<MailItem>();
        MailItems.Add(new MailItem()
        {
            ItemName = "Inbox",
            UnreadMessageount = 20
        });
        MailItems.Add(new MailItem()
        {
            ItemName = "Drafts",
            UnreadMessageount = null
        });
        MailItems.Add(new MailItem()
        {
            ItemName = "Sent Items",
```

```
UnreadMessageount = 5
});
MailItems.Add(new MailItem()
{
    ItemName = "Deleted Items",
    UnreadMessageount = null
});
MailItems.Add(new MailItem()
{
    ItemName = "Junk Email",
    UnreadMessageount = null
});
}
```

### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<ListView BorderThickness="1"
BorderBrush="LightGray"
ItemsSource="{Binding MailItems}"
SelectedIndex="0"
VerticalAlignment="Center"
HorizontalAlignment="Center">
<ListView.ItemTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="150"/>
<ColumnDefinition Width="100"/>
</Grid.ColumnDefinitions>
<ContentPresenter Grid.Column="0"
Content="{Binding ItemName}"
VerticalAlignment="Center"/>
<notification:SfBadge x:Name="badge4"
Grid.Column="1"
Height="20"
Width="40"
Background="Orange"
Content="{Binding UnreadMessageount}"
Shape="Oval"/>
</Grid>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</Grid>
```



**Note:** Download demo application from [GitHub](#)

### Setting Badge display content

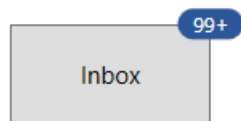
If you want to set or change the display content of the **Badge**, use the **Content** property. The default value of **Content** property is **null**.

#### XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Content="99+"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

#### C#

```
badge.Content = "99+";
```



**Note:** Download demo application from [GitHub](#)

### Custom UI for Badge content

You can change the appearance of **Badge** content by using **ContentTemplate** property. The **DataContext** of **ContentTemplate** property is **Content**.

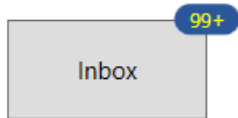
#### XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Content="99+"
x:Name="badge">
<notification:SfBadge.ContentTemplate>
<DataTemplate>
<Grid>
<TextBlock Text="{Binding}"
Foreground="Yellow"/>
</Grid>
</DataTemplate>
```

```

</notification:SfBadge.ContentTemplate>
</notification:SfBadge>
</notification:SfBadge.Badge>
</Button>

```



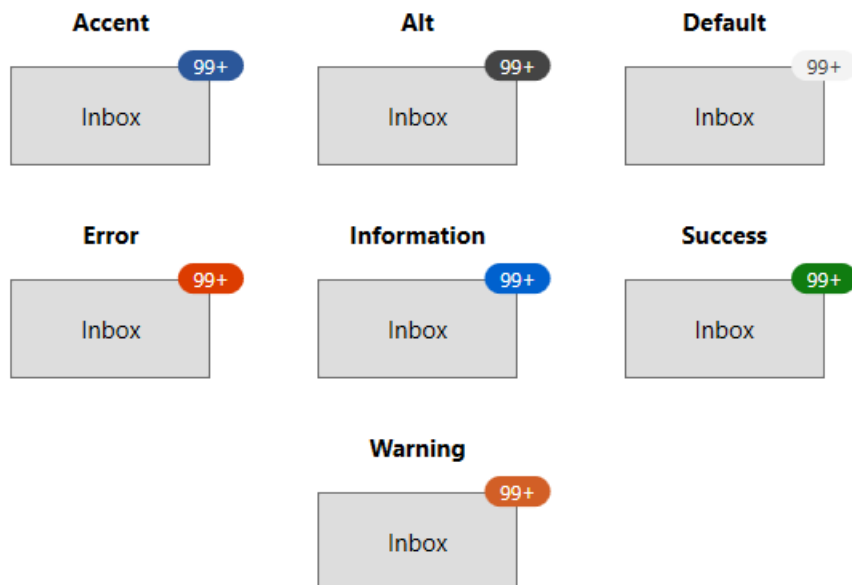
**Note:** Download demo application from [GitHub](#)

#### Predefined colors for displaying the badges

You can change background color of the **Badge** by using the **Fill** property. Based on the value of **Fill** property, respective background color will be applied to the **Badge**. The default value of **Fill** property is **Accent**.

The **Badge** supports the following different essential states :

- Accent - DarkSlateBlue background will be applied
- Alt - DarkSlateGray background will be applied
- Default - WhiteSmoke background will be applied
- Error - OrangeRed background will be applied
- Information - RoyalBlue background will be applied
- Success - Green background will be applied
- Warning - Chocolate background will be applied



#### XML

```

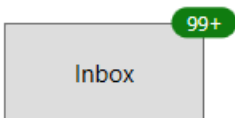
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>

```

```
<notification:SfBadge Fill="Success"
Content="99+"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

### C#

```
badge.Fill = BadgeFill.Success;
badge.Content = "99+";
```



**Note:** Download demo application from [GitHub](#)

### Custom colors for displaying the badges

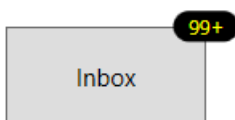
If you want to change the background color of the **Badge** other than the default **Fill** colors, use the **Background** property. You can also change foreground of the **Badge** by using the **Foreground** property. The default value of **Background** and **Foreground** properties is **null**.

### XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Background="Black"
Foreground="Yellow"
Content="99+"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

### C#

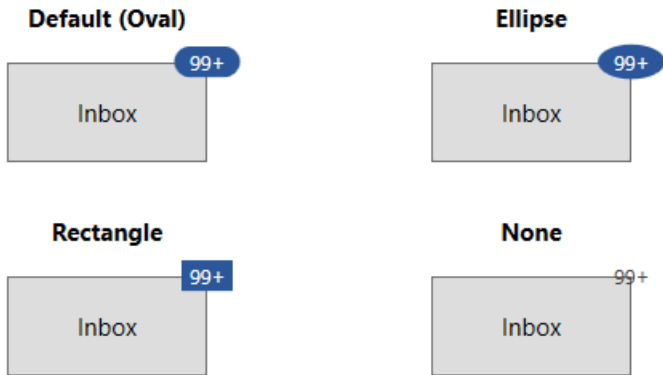
```
badge.Background = Brushes.Black;
badge.Foreground = Brushes.Yellow;
badge.Content = "99+";
```



**Note:** Download demo application from [GitHub](#)

### Predefined shapes for displaying the Badge

You can change the default shape to either **Rectangle**, **Oval** or **Ellipse** by using **Shape** property. If you want to display the **Badge** content without any default shapes, use the **Shape** property value as **None**. The default value of **Shape** property is **Ellipse**.

**XML**

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Shape="Oval"
Content="99+"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

**C#**

```
badge.Shape = BadgeShape.Oval;
badge.Content = "99+";
```



**Note:** Download demo application from [GitHub](#)

**Custom shape for displaying the Badge**

If you want to change the shape of the **Badge** other than the default shapes, use the [CustomShape](#) property. You can enable the custom shapes by setting the **Shape** property value as **Custom**.

**XML**

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Shape="Custom"
CustomShape="M16,0C17.300003,0.49999999 18.399994,1.6000063
19.199997,3.3000189 19.300003,3.3000189 19.5,3.1999823 19.600006,3.1999823
19.800003,3.3999945 20,3.4999703 20.199997,3.8000194 21.5,2.9999701
22.800003,2.6000067 24,2.8000189 24.600006,3.6000069 25,4.6000072
25,5.8000194 25.5,5.8000194 25.899994,5.899995 26.300003,6.1000076
26.399994,6.3999954 26.5,6.6000076 26.600006,6.8999954 28.199997,6.8000198
29.600006,7.1999832 30.5,7.8999959 30.699997,9.1999837 30.199997,10.699984
```

```

29.100006,12.199985L29.399994,12.499972C29.399994,12.800021
29.300003,12.999972 29.199997,13.300021 30.600006,13.999972
31.600006,14.999973 32,15.999973 31.600006,16.899998 30.899994,17.699986
29.800003,18.399998 30,18.800022 30.100006,19.199986 30.199997,19.60001
30,19.800022 29.800003,19.999973 29.600006,20.199986 30.399994,21.499975
30.800003,22.800024 30.600006,23.999975 29.5,24.800024 27.899994,25.199988
26,24.999975 26,25.100012 25.899994,25.300024 25.899994,25.4
25.600006,25.499975 25.399994,25.600012 25.100006,25.600012
25.199997,26.999975 24.800003,28.300024 24.100006,29.199989
23.100006,29.400002 22,29.100014 20.800003,28.499975 20.5,28.900002
20.199997,29.199989 19.899994,29.400002 19.600006,29.400002
19.300003,29.300026 19,29.199989 18.300003,30.499977 17.199997,31.499977
16.100006,31.900002 14.800003,31.400002 13.699997,30.199989
12.899994,28.600012 12.800003,28.600012 12.600006,28.699988 12.5,28.699988
12.300003,28.499975 12.100006,28.4 11.899994,28.100012 10.5,28.999977
9.1999969,29.400002 8,29.199989 7.3999939,28.4 7,27.4 7,26.199988
6.5,26.199988 6.1000061,26.100012 5.6999969,25.9 5.6000061,25.600012
5.5,25.4 5.3999939,25.100012 3.8000031,25.199988 2.3999939,24.800024
1.5,24.100012 1.3000031,22.800024 1.8000031,21.300024
2.8999939,19.800022L2.6000061,19.499973C2.6000061,19.199986
2.6999969,18.999973 2.8000031,18.699986 1.3999939,17.999973
0.3999939,16.999973 0,15.999973 0.3999939,15.10001 1.1000061,14.300021
2.1999969,13.600009 2,13.199985 1.8999939,12.800021 1.8000031,12.399997
2,12.199985 2.1999969,11.999972 2.5,11.800021 1.6999969,10.499971
1.3000031,9.1999837 1.5,7.9999715 2.5,7.1999832 4.1000061,6.8000198
6,6.999971 6,6.8999954 6.1000061,6.6999832 6.1000061,6.6000076
6.3999939,6.499971 6.6999969,6.3999954 7,6.3999954 6.8999939,4.899995
7.3000031,3.6999825 8,2.8000189 9,2.6000067 10.100006,2.8999945
11.300003,3.4999705 11.600006,3.1000067 11.899994,2.8000189
12.199997,2.6000067 12.5,2.6000067 12.800003,2.6999823 13.100006,2.8000189
13.800003,1.3999941 14.800003,0.39999388 16,0z"
Content="10"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>

```



**Note:** Download demo application from [GitHub](#)

### Animate when content changes

You can enable the **Scale** or **Opacity** based animation for displaying the **Badge** text by using **AnimationType** property. You can only see the animation when you change the text of the **Badge**. The default value of **AnimationType** property is **None**.

### XML

```

<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge AnimationType="Scale"
x:Name="badge"/>

```

```

</notification:SfBadge.Badge>
</Button>
<StackPanel Orientation="Vertical">
<TextBlock Text="Badge content"
TextAlignment="Center" />
<notification:UpDown MinValue="1"
MaxValue="10"
Step="1"
NumberDecimalDigits="0"
Value="1"
x:Name="badgeContent"
ValueChanged="BadgeContent_ValueChanged"/>

```

**C#**

```

badge.AnimationType = BadgeAnimationType.Scale;
badgeContent.ValueChanged += BadgeContent_ValueChanged;

```

**C#**

```

private void BadgeContent_ValueChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    this.badge.Content = Convert.ToInt32(e.NewValue).ToString();
}

```

*Scaling based animation**Opacity based animation*

**Note:** Download demo application from [GitHub](#)

*Stroke customization*

You can change stroke color and its thickness by using the [Stroke](#) and [StrokeThickness](#) properties. The default value of [Stroke](#) property is `null` and [StrokeThickness](#) property is `0`.

**XML**

```

<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Stroke="Red"
StrokeThickness="3"
Content="99+"

```



```
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

**C#**

```
badge.Stroke = Brushes.Red;
badge.StrokeThickness = 3;
badge.Content = "99+";
```



**Note:** Download demo application from [GitHub](#)

**Display number formatting**

You can format the numbers which are displayed in the **Badge** content by using the converters. For example, you can display the number as **99+** which is greater than or equal to **100**.

**XML**

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge x:Name="badge"
Shape="Ellipse"
AnimationType="None"/>
</notification:SfBadge.Badge>
</Button>
<StackPanel Orientation="Vertical">
<TextBlock Text="Badge content"
TextAlignment="Center" />
<notification:UpDown MinValue="1"
MaxValue="10000000"
Step="1"
NumberDecimalDigits="0"
Value="1"
x:Name="badgeContent"
ValueChanged="BadgeContent_ValueChanged"/>
```

**C#**

```
badge.AnimationType = BadgeAnimationType.None;
badge.Shape = BadgeShape.Ellipse;
badgeContent.ValueChanged += BadgeContent_ValueChanged;
```

**C#**

```
private void BadgeContent_ValueChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
```

```

int content_Value = Convert.ToInt32(e.NewValue);
if (content_Value <= 99)
{
    this.badge.Content = content_Value.ToString();
}
else if (content_Value <= 999)
{
    this.badge.Content = "99+";
}
else if (content_Value < 99999)
{
    this.badge.Content = (content_Value / 1000).ToString("0.#") + "K";
}
else if (content_Value < 999999)
{
    this.badge.Content = (content_Value / 1000).ToString("#,0K");
}
else if (content_Value < 9999999)
{
    this.badge.Content = (content_Value / 1000000).ToString("0.#") + "M";
}
else
{
    this.badge.Content = (content_Value / 1000000).ToString("#,0M");
}
}

```



**Note:** Download demo application from [GitHub](#)

### Change Badge size

You can change the size of **Badge** by using the **Width** and **Height** properties. The default value of **Width** property is **40** and **Height** property is **30**.

### XML

```

<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Width="50"
Height="30"
Content="99+"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>

```

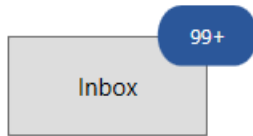
### C#

```

badge.Width = 50;
badge.Height = 30;

```

```
badge.Content = "99+";
```



**Note:** Download demo application from [GitHub](#)

### Text formatting

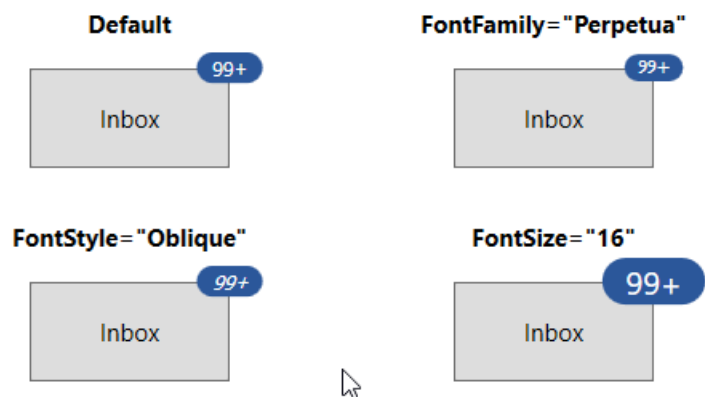
You can change the style, size and font family of the `Badge` display content by using the `FontFamily`, `FontStyle` and `FontSize` properties. The default value of `FontFamily` property is `Segoe UI`, `FontStyle` property is `Normal` and `FontSize` property is `14`.

### XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge FontFamily="Perpetua"
FontStyle="Oblique"
Shape="Ellipse"
FontSize="20"
Content="99+"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

### C#

```
badge.FontFamily = new FontFamily("Perpetua");
badge.FontSize = 20;
badge.Shape = BadgeShape.Ellipse;
badge.Content = "99+";
```



**Note:** Download demo application from [GitHub](#)

### Change opacity of Badge

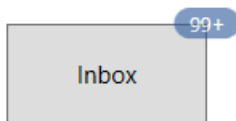
You can change opacity of the **Badge** by using the **Opacity** property. Value range of **Opacity** property is **0** to **1**. The default value of **Opacity** property is **1**.

#### XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Opacity="0.6"
Content="99+"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

#### C#

```
badge.Opacity = 0.6;
badge.Content = "99+";
```



**Note:** Download demo application from [GitHub](#)

### Hide the Badge

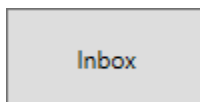
You can hide the **Badge** by setting the **Visibility** property value as **Collapsed**. Badge will be hidden when its content is **null**. The default value of **Visibility** property is **Visible**.

#### XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge Visibility="Collapsed"
Content="99+"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

#### C#

```
badge.Visibility = Visibility.Collapsed;
badge.Content = "99+";
```



Alignment and positioning in WPF Badge (SfBadge)

This section explains the alignment and positioning functionalities available in the WPF [Badge](#) control.

Alignment of Badge

you can align the **Badge** either horizontally or vertically by using the **HorizontalAlignment** or **VerticalAlignment** properties. The default value of **HorizontalAlignment** property is **Right** and **VerticalAlignment** property is **Top**.

```
<style>
table, td, th {
text-align: center;
}
</style>
```

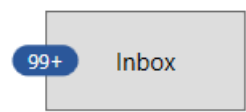
	HorizontalAlignment			
VerticalAlignment	Left	Center	Right	Stretch
Top				
Center				
Bottom				
Stretch				

XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge HorizontalAlignment="Left"
VerticalAlignment="Center"
Content="99+"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

C#

```
badge.HorizontalAlignment = HorizontalAlignment.Left;
badge.VerticalAlignment = VerticalAlignment.Center;
badge.Content = "99+";
```



**Note:** Download demo application from [GitHub](#)

Positioning of Badge

You can change the horizontal or vertical position of the **Badge** either inside, outside or in the middle by using the [HorizontalAnchor](#) and [VerticalAnchor](#) properties. It will be placed based on the value of **HorizontalAlignment** and **VerticalAlignment** properties. The default value of **HorizontalAnchor** and **VerticalAnchor** properties is **Center**.

For example, you will change the **HorizontalAnchor** and **VerticalAnchor** property values on when the value of **HorizontalAlignment** properties is **Right** and **VerticalAlignment** property is **Top**. **Badge** will be positioned as follows,

```
<style>
table, td, th {
text-align: center;
}
</style>
```

	HorizontalAnchor			
VerticalAnchor	Inside	Center	OutSide	
Inside				
Center				
Outside				

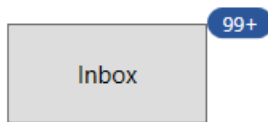
**XML**

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge HorizontalAnchor="Outside"
VerticalAnchor="Center"
Content="10"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

**C#**

```
badge.HorizontalAnchor = BadgeAnchor.Outside;
```

```
badge.VerticalAnchor = BadgeAnchor.Center;  
badge.Content = "10";
```



**Note:** Download demo application from [GitHub](#)

Place the Badge any where on the container

If you want to place the Badge anywhere on any shaped container, use the [HorizontalPosition](#) or [VerticalPosition](#) properties. The value range for [HorizontalPosition](#) and [VerticalPosition](#) properties is 0 to 1. The default value of [HorizontalPosition](#) property is 1 and [VerticalPosition](#) property is 0.

For example, if you use any circular containers, you can easily place the Badge anywhere by using the [HorizontalPosition](#) and [VerticalPosition](#) properties.

### XML

```
<Image Source="/Images/avatar.png"  
Width="100"  
Height="100" >  
  <notification:SfBadge.Badge>  
    <notification:SfBadge Shape="None"  
      HorizontalPosition="0.9"  
      VerticalPosition="0.8"  
      x:Name="badge">  
        <notification:SfBadge.Content>  
          <Ellipse Width="20"  
            Height="20"  
            Fill="LimeGreen"/>  
        </notification:SfBadge.Content>  
      </notification:SfBadge>  
    </notification:SfBadge.Badge>  
  </Image>
```

### C#

```
badge.HorizontalPosition = 0.9;  
badge.VerticalPosition = 0.8;
```



**Note:** Download demo application from [GitHub](#)

Custom alignment and positioning of Badge

You can customize the horizontal or vertical position of the Badge either inside, outside or in the middle with any point by using the `HorizontalPosition` & `VerticalPosition` properties and `HorizontalAnchorPosition` & `VerticalAnchorPosition` properties. This will effective only on by setting the `HorizontalAnchor` and `VerticalAnchor` properties value as `Custom`. The value range for `HorizontalAnchorPosition` and `VerticalAnchorPosition` properties is 0 to 1. The default value of `HorizontalAnchorPosition` and `VerticalAnchorPosition` properties is 0.

```
<style>
table, td, th {
text-align: center;
}
</style>
```

	HorizontalPosition & HorizontalAnchorPosition		
VerticalPosition & VerticalAnchorPosition	0	0.5	1
0			
0.5			
1			

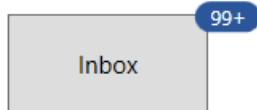
XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge HorizontalAnchor="Custom"
VerticalAnchor="Custom"
HorizontalAnchorPosition="0.2"
VerticalAnchorPosition="0.4"
Content="99+"
x:Name="badge2"/>
</notification:SfBadge.Badge>
</Button>
```

C#

```
badge.HorizontalAnchor = BadgeAnchor.Custom;
badge.VerticalAnchor = BadgeAnchor.Custom;
badge.HorizontalAnchorPosition = 0.2;
badge.VerticalAnchorPosition = 0.4;
badge.Content = "99+";
```





---

**Note:** Download demo application from [GitHub](#)

---

### Badge content alignment

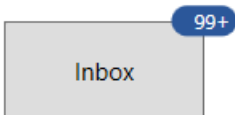
you can place the **Badge** content either horizontally or vertically by using the **HorizontalAlignment** or **VerticalContentAlignment** properties. The default value of **HorizontalAlignment** and **VerticalContentAlignment** properties is **Center**.

### XML

```
<Button Width="100"
Height="50"
Content="Inbox">
<notification:SfBadge.Badge>
<notification:SfBadge HorizontalContentAlignment="Right"
VerticalContentAlignment="Center"
Content="10"
x:Name="badge"/>
</notification:SfBadge.Badge>
</Button>
```

### C#

```
badge.HorizontalAlignment = HorizontalAlignment.Right;
badge.VerticalContentAlignment = VerticalAlignment.Center;
badge.Content = "99+";
```



---

**Note:** Download demo application from [GitHub](#)

---

## ButtonAdv

### WPF Button (ButtonAdv) Overview

The Button (or ButtonAdv) is a basic button control with image options and multi-line support which is used to design complex forms and applications. The control is shipped with a commanding support that provides compatibility in MVVM design pattern by attaching commands to the control, which will get executed when it gets clicked.

### Key features

- **Size Mode** - Predefined sizes, such as small, normal and large, can be set to the button.
- **Image** - Provides options for loading image in button.
- **Command Binding** - Provides support to execute any action on clicking the instance.
- **Multiline** - Provides support for displaying multiple lines of text in large button.

- **Checkable** - Provides support for clicking the button that looks like the toggle button.
- **Localization** - Provides support to customize the text in the user interface based on the local culture.
- **Right-to-left (RTL)** - The text direction and layout of the control can be displayed in the right-to-left direction.

### Getting Started with WPF Button (ButtonAdv)

This section provides an overview of how to work with [WPF Button](#) control. It describes the control structure, the control initialization and the image setting to the control.

#### Control structure



#### Assembly deployment

Refer [ButtonAdv](#) control dependencies section to get the list of assemblies or [NuGet package](#) needs to be added as reference to use the ButtonAdv control in any application.

#### Creating simple application with Button

In this walk through, will create WPF application that contains Button control. By the following ways, one can add the controls:

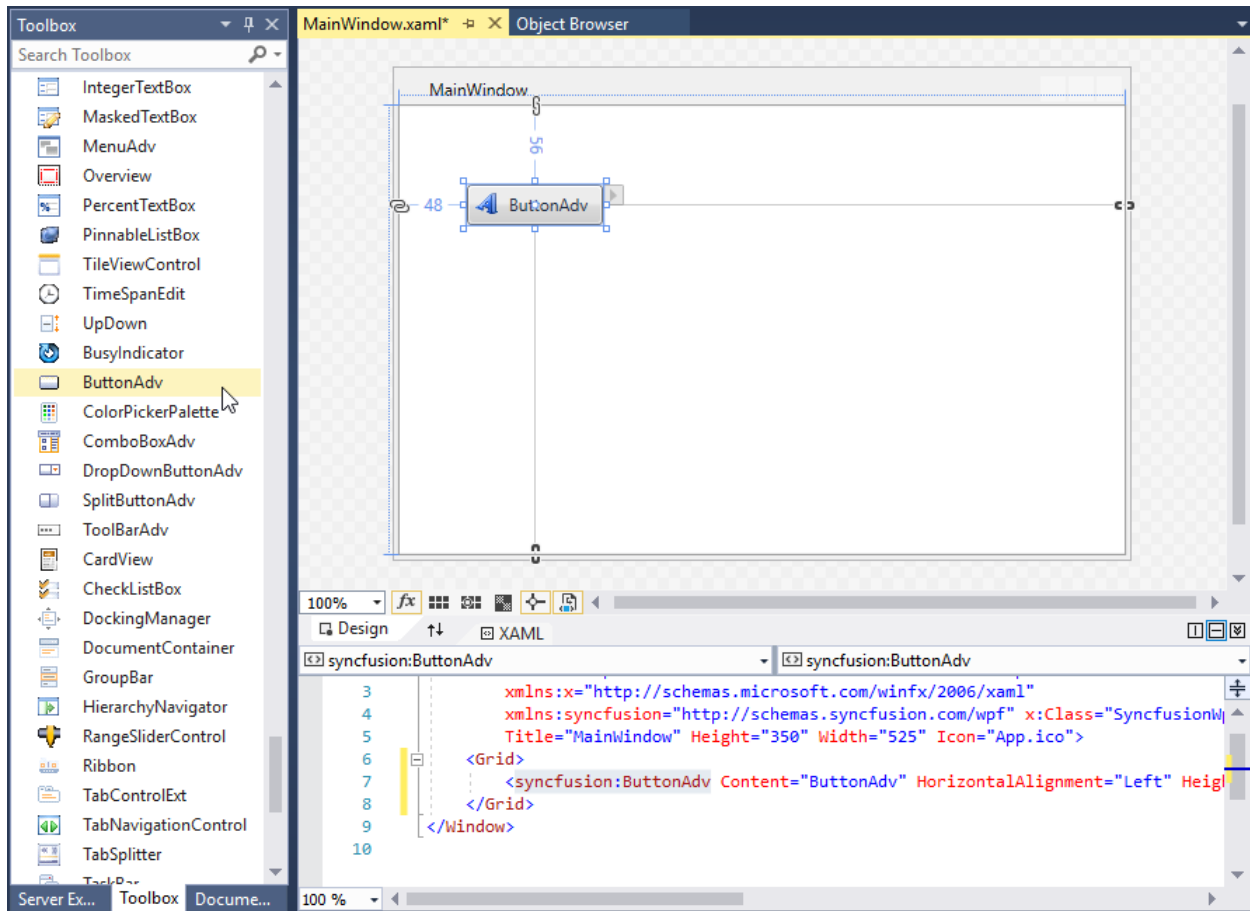
1. [Adding control via designer](#)
2. [Adding control manually in XAML](#)
3. [Adding control manually in C#](#)

#### Adding control via designer

WPF Button control can be added to the application by dragging **ButtonAdv** from toolbox and dropping it in designer view. After dropping the control in designer view, the assembly **Syncfusion.Shared.WPF** gets added into the project automatically. The following code snippets will also be added into the XAML.

#### XML

```
<syncfusion:ButtonAdv x:Name="buttonAdv" Label="ButtonAdv"/>
```



**Note:** syncfusion in XAML is an auto generated namespace.

#### Adding control manually in XAML

In order to add the control manually in XAML, follow the below steps.

1. Add the below required assembly reference to the project.
  - Syncfusion.Shared.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> or the control namespace `Syncfusion.Windows.Tools.Controls` in XAML page.
3. Declare ButtonAdv control in XAML page.

#### XML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:Buttonadv_GetStart_Sample"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:Syncfusion="http://schemas.microsoft.com/netfx/2009/xaml/presentation"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<syncfusion:ButtonAdv Height="44" VerticalAlignment="Center"
HorizontalAlignment="Center" Width="162"/>
</Grid>
```

```
</Window>
```

#### Adding control manually in C#

In order to add the control manually in C#, do the below steps.

1. Add the below required assembly reference to the project.
  - Syncfusion.Shared.WPF
2. Import the **Syncfusion.Windows.Tools.Controls** namespace.
3. Create ButtonAdv control instance and add it to the window.

#### XML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:Button_Getting_Started"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:Syncfusion="http://schemas.microsoft.com/netfx/2009/xaml/presentation"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid x:Name="Root">
</Grid>
</Window>
```

#### C#

```
using Syncfusion.Windows.Tools.Controls;
namespace Button_Getting_Started
{
public partial class MainWindow : Window
{
public MainWindow()
{
InitializeComponent();
ButtonAdv button = new ButtonAdv();
button.Height=44;
button.Width=31;
Root.Children.Add(button);
}
}
}
```

#### Setting label

The label on the button is a text that explains its action to the end-user. Apply the text by using the [Label](#) property.

#### XML

```
<syncfusion:ButtonAdv SmallIcon="image\usersmall.png" Label="Log-in"/>
```

#### C#

```
ButtonAdv button = new ButtonAdv();
button.Label = "Log-in";
```

```
button.SmallIcon = new BitmapImage(new Uri("image/usersmall.png" ,  
UriKind.RelativeOrAbsolute));
```



### Setting size mode

Size mode is used to render button control in different pre-defined sizes based on application demand. Apply the size mode by setting the [SizeMode](#) property.

The **SizeMode** is an enumeration which contains the following values:

- Small
- Normal
- Large

### Small mode

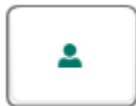
When the mode is set to small, the control is displayed without the label. Only icon will be present in it.

### XML

```
<syncfusion:ButtonAdv SizeMode="Small" Label="Log-in" SmallIcon  
="image\usersmall.png"/>
```

### C#

```
ButtonAdv button = new ButtonAdv();  
button.Label = "Log-in";  
button.SizeMode = SizeMode.Small;  
button.SmallIcon = new BitmapImage(new Uri("image/usersmall.png",  
UriKind.RelativeOrAbsolute));
```



### Normal mode

In a normal size button, a small image with the text on the side will be displayed.

### XML

```
<syncfusion:ButtonAdv SizeMode="Normal" Label="Log-in" SmallIcon  
="image\usersmall.png"/>
```

### C#

```
ButtonAdv button = new ButtonAdv();  
button.Label = "Log-in";  
button.SizeMode = SizeMode.Normal;  
button.SmallIcon = new BitmapImage(new Uri("image/usersmall.png",  
UriKind.RelativeOrAbsolute));
```



### Large mode

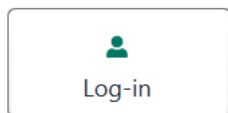
In a large size button, a large image along with the text at the bottom will be displayed.

### XML

```
<syncfusion:ButtonAdv SizeMode="Large" Label="Log-in" LargeIcon
="image\userlarge.png"/>
```

### C#

```
ButtonAdv button = new ButtonAdv();
button.Label = "Log-in";
button.SizeMode = SizeMode.Large;
button.LargeIcon = new BitmapImage(new Uri("image/userlarge.png",
UriKind.RelativeOrAbsolute));
```



### Setting icon template

The [IconTemplate](#) property provides support for setting up any type of image such as path data, font icons, etc. to the ButtonAdv. The icon will automatically resize the template content according to its size provided in the data template.

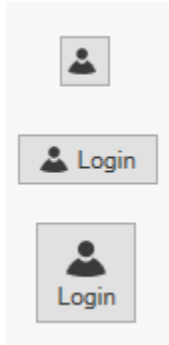
### XML

```
<Window x:Class="ButtonAdv_IconTemplate.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:ButtonAdv_IconTemplate"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.Resources>
<DataTemplate x:Key="smallIconTemplate">
<Grid Width="12" Height="16">
<Path
Data="M21.576999,13.473151C26.414003,15.496185 30.259996,20.071221
31.999999,25.86432 15.448002,32.143386 0,25.86432 0,25.86432
1.7140042,20.158227 5.4690005,15.632174 10.202001,13.564156
11.338002,15.514191 13.444005,16.827195 15.862003,16.827195
18.317996,16.827195 20.455997,15.474182 21.576999,13.473151z
M16.000003,0C19.617999,1.5323894E-07 22.550998,2.9330488 22.550998,6.5510722
22.550998,10.170134 19.617999,13.102144 16.000003,13.102144
12.381993,13.102144 9.4489957,10.170134 9.4489957,6.5510722
9.4489957,2.9330488 12.381993,1.5323894E-07 16.000003,0z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
```

```

</DataTemplate>
<DataTemplate x:Key="normalIconTemplate">
<Grid Width="16" Height="16">
<Path
Data="M21.576999,13.473151C26.414003,15.496185 30.259996,20.071221
31.999999,25.86432 15.448002,32.143386 0,25.86432 0,25.86432
1.7140042,20.158227 5.4690005,15.632174 10.202001,13.564156
11.338002,15.514191 13.444005,16.827195 15.862003,16.827195
18.317996,16.827195 20.455997,15.474182 21.576999,13.473151z
M16.000003,0C19.617999,1.5323894E-07 22.550998,2.9330488 22.550998,6.5510722
22.550998,10.170134 19.617999,13.102144 16.000003,13.102144
12.381993,13.102144 9.4489957,10.170134 9.4489957,6.5510722
9.4489957,2.9330488 12.381993,1.5323894E-07 16.000003,0z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="largeIconTemplate">
<Grid Width="16" Height="16">
<Path
Width="16"
Height="16"
Data="M21.576999,13.473151C26.414003,15.496185 30.259996,20.071221
31.999999,25.86432 15.448002,32.143386 0,25.86432 0,25.86432
1.7140042,20.158227 5.4690005,15.632174 10.202001,13.564156
11.338002,15.514191 13.444005,16.827195 15.862003,16.827195
18.317996,16.827195 20.455997,15.474182 21.576999,13.473151z
M16.000003,0C19.617999,1.5323894E-07 22.550998,2.9330488 22.550998,6.5510722
22.550998,10.170134 19.617999,13.102144 16.000003,13.102144
12.381993,13.102144 9.4489957,10.170134 9.4489957,6.5510722
9.4489957,2.9330488 12.381993,1.5323894E-07 16.000003,0z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
</DataTemplate>
</Window.Resources>
<StackPanel>
<syncfusion:ButtonAdv x:Name="smallIcon" SizeMode="Small" Label="Login"
HorizontalAlignment="Center" VerticalAlignment="Center" Margin="10"
IconTemplate="{StaticResource smallIconTemplate}">
</syncfusion:ButtonAdv>
<syncfusion:ButtonAdv x:Name="normalIcon" SizeMode="Normal" Label="Login"
HorizontalAlignment="Center" VerticalAlignment="Center" Margin="10"
IconTemplate="{StaticResource normalIconTemplate}">
</syncfusion:ButtonAdv>
<syncfusion:ButtonAdv x:Name="largeIcon" SizeMode="Large" Label="Login"
HorizontalAlignment="Center" VerticalAlignment="Center" Margin="10"
IconTemplate="{StaticResource largeIconTemplate}">
</syncfusion:ButtonAdv>
</StackPanel>
</Window>

```



**Note:** The [ButtonAdv](#) loads the icon in the following priority order.

- [IconTemplate](#)
- [LargeIcon](#)
- [SmallIcon](#)

Setting icon template selector

The [IconTemplateSelector](#) property that allows you to specify a different data template based on the value given in the data templates.

#### XML

```
<Window x:Class="TemplateSelector_ButtonAdv.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:TemplateSelector_ButtonAdv"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.Resources>
<DataTemplate x:Key="newIcon">
<Grid Width="12" Height="16">
<Path
Margin="0.5"
Data="M0,0 L5.9999999,0 11,5 11,15 0,15 z"
Fill="White"
Stretch="Fill" />
<Path
Data="M7,1.7070007 L7,5 10.292999,5 z M1,1 L1,15 11,15 11,6 6,6 6,1 z M0,0
L6.7070007,0 12,5.2929993 12,16 0,16 z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="OpenIcon">
<Grid Width="16" Height="16">
<Path
Margin="0.5,0.5,0.738,0.502"
Data="M0,0 L5,0 6,1 12,1 12,3.4999998 11.499065,3.9999996
14.716998,3.9999996 11.92699,10.999 4.1853847,10.984859 0,10.982999 z"
Fill="White"
```



```

Stretch="Fill" />
<Path
Data="M5.162991,5.0009986 L1.7839907,10.979999 4.3081884,10.984653
5.0009999,10.984999 5.0009999,10.98593 12.088991,10.999 14.480014,5.0009986
z M0,0 L5.7069998,0 6.7069998,1 13,1 13,3.9999998 12,3.9999998 12,1.9999998
6.2930002,1.9999998 5.2930002,1 0.99999994,1 0.99999994,10.335325
4.5790062,4.0009986 15.954991,4.0009986 12.765994,12.000998
4.552258,11.98482 0,11.982999 z"
Fill="#FF3A3A38"
Stretch="Fill" />
</Grid>
</DataTemplate>
<local:TemplateSelector x:Key="IconTemp" NewIcon="{StaticResource newIcon}"
OpenIcon="{StaticResource OpenIcon}"/>
</Window.Resources>
<Grid>
<StackPanel VerticalAlignment="Center">
<CheckBox Name="Check" IsChecked="True" Checked="Check_Checked"
Unchecked="Check_Unchecked" HorizontalAlignment="Center" Command="{Binding
CheckCommand}" Content="ChangeIcon"/>
<syncfusion:ButtonAdv HorizontalAlignment="Center" Margin="10"
Content="{Binding IsChecked}" Label="IconTemplateSelector"
IconTemplateSelector="{StaticResource IconTemp}"/>
</StackPanel>
</Grid>
</Window>

```

## C#

```

public class TemplateSelector : DataTemplateSelector
{
    public DataTemplate NewIcon { get; set; }
    public DataTemplate OpenIcon { get; set; }
    public override DataTemplate SelectTemplate(object item, DependencyObject
container)
    {
        if (item == null)
        {
            return OpenIcon;
        }
        if ((item as Model).IsChecked)
        {
            return NewIcon;
        }
        return base.SelectTemplate(item, container);
    }
}

```

**Note:** The [ButtonAdv](#) loads the icon in the following priority order.

- [IconTemplateSelector](#)
- [IconTemplate](#)
- [Largelcon](#)
- [SmallIcon](#)

### Setting image

The image option helps to provide pictorial representation of the button. Image can be added either using the [SmallIcon](#) property or [LargeIcon](#) property.

**SmallIcon** — This property will be used to set the image when size mode is **Normal** or **Small**.

**LargeIcon** — This property will be used to set the image when size mode is **Large**.

The **SmallIcon** property can be set as follows:

#### XML

```
<syncfusion:ButtonAdv SizeMode="Small" SmallIcon="image\syncfusion.png"/>
```

#### C#

```
ButtonAdv button = new ButtonAdv();  
button.SizeMode = SizeMode.Small;  
button.SmallIcon = new BitmapImage(new Uri("image/syncfusion.png",  
UriKind.RelativeOrAbsolute));
```



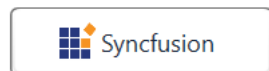
The **SmallIcon** property can be set even when the SizeMode is **Normal**.

#### XML

```
<syncfusion:ButtonAdv SizeMode="Normal" SmallIcon="syncfusion.png"  
Label="Syncfusion" SmallIcon="image\syncfusion.png"/>
```

#### C#

```
ButtonAdv button = new ButtonAdv();  
button.Label = "Syncfusion";  
button.SizeMode = SizeMode.Normal;  
button.SmallIcon = new BitmapImage(new Uri("image/syncfusion.png",  
UriKind.RelativeOrAbsolute));
```



The **LargeIcon** property can be set as follows:

#### XML

```
<syncfusion:ButtonAdv SizeMode="Large" LargeIcon="syncfusion.png"  
Label="Syncfusion" SmallIcon="image\syncfusion.png"/>
```

#### C#

```
ButtonAdv button = new ButtonAdv();  
button.Label = "Syncfusion";  
button.SizeMode = SizeMode.Large;
```

```
button.SmallIcon = new BitmapImage(new Uri("image/syncfusion.png",  
UriKind.RelativeOrAbsolute));
```



### Setting icon width and height

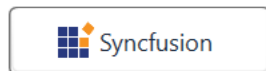
Icon width and icon height can be set using [IconWidth](#) and [IconHeight](#) properties respectively.

### XML

```
<syncfusion:ButtonAdv x:Name="button1" SizeMode="Normal" IconHeight="20"  
IconWidth="20" Label="Syncfusion" SmallIcon ="image\syncfusion.png"/>
```

### C#

```
ButtonAdv button1 = new ButtonAdv();  
button1.Label = "Syncfusion";  
button1.IconWidth=20;  
button1.IconHeight=20;  
button1.SmallIcon = new BitmapImage(new Uri("image/syncfusion.png",  
UriKind.RelativeOrAbsolute));
```



### XML

```
<syncfusion:ButtonAdv x:Name="button2" SizeMode="Normal" IconHeight="30"  
IconWidth="30" Label="Syncfusion" SmallIcon ="image\syncfusion.png" />
```

### C#

```
ButtonAdv button2 = new ButtonAdv();  
button2.Label = "Syncfusion";  
button2.IconWidth=30;  
button2.IconHeight=30;  
button2.SmallIcon = new BitmapImage(new Uri("image/syncfusion.png",  
UriKind.RelativeOrAbsolute));
```



**Note:** View [sample](#) in GitHub. This sample showcases how to add button control and its basic features like image sizing options and size modes.

### IsDefault mode

The [IsDefault](#) property indicates whether the ButtonAdv is a Default button and is used to activate the ButtonAdv by pressing using Enter key. When setting the IsDefault property to true, the user can invoke the button by pressing the **Enter** key.

### XML

```
<syncfusion:ButtonAdv x:Name="defaultButton" Label="Default" Grid.Column="1"
Grid.Row="1" VerticalAlignment="Top" HorizontalAlignment="Center"
Click="ButtonAdv_Click" IsDefault="True" />
```

### IsCancel mode

The [IsCancel](#) property indicates whether the ButtonAdv is a Cancel button and is used to activate the button by using Escape key. When setting the IsCancel property to true, the user can invoke the button by pressing the [Escape](#) key.

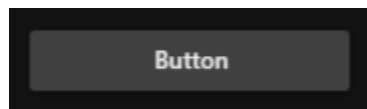
### XML

```
<syncfusion:ButtonAdv x:Name="cancelButton" Label="Cancel" Grid.Column="1"
Grid.Row="1" VerticalAlignment="Top" HorizontalAlignment="Center"
Click="ButtonAdv_Click" IsCancel="True" />
```

### Theme

WPF Button supports various built-in themes. Refer to the below links to apply themes for the Button,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### MVVM in WPF Button (ButtonAdv)

The command and command parameter properties allow to execute any action on clicking the button control.

- **Command** - The [Command](#) property accept all commands derived from interface [ICommand](#).
- **CommandParameter** - The [CommandParameter](#) property allows the user to provide additional data required in the command handler in-order to perform any operation.

### XML

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:Buttonadv_Mvvm"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="Buttonadv_Mvvm.MainWindow"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:ButtonViewModel/>
</Window.DataContext>
<Grid VerticalAlignment="Center">
<Grid.ColumnDefinitions>
```

```

<ColumnDefinition Width="200"/>
<ColumnDefinition Width="*/>
</Grid.ColumnDefinitions>
<CheckBox IsChecked="{Binding CanPerformAction}" Grid.Column="0"
Content="Can perform action in button"/>
<syncfusion:ButtonAdv SizeMode="Large" LargeIcon="image\userlarge.png"
Label="Log in"
Command="{Binding ClickCommand}"
Grid.Column="1"
CommandParameter="Action completed"
Height="68" Width="78"/>
</Grid>
</Window>

```

## C#

```

public class DelegateCommand<T> : ICommand
{
    private Predicate<T> _canExecute;
    private Action<T> _method;
    bool _canExecuteCache = true;
    /// <summary>
    /// Initializes a new instance of the <see cref="DelegateCommand"/> class.
    /// </summary>
    /// <param name="method">The method.</param>
    public DelegateCommand(Action<T> method)
    : this(method, null)
    {
    }
    /// <summary>
    /// Initializes a new instance of the <see cref="DelegateCommand"/> class.
    /// </summary>
    /// <param name="method">The method.</param>
    /// <param name="canExecute">The can execute.</param>
    public DelegateCommand(Action<T> method, Predicate<T> canExecute)
    {
        _method = method;
        _canExecute = canExecute;
    }
    /// <summary>
    /// Defines the method that determines whether the command can execute in
    its current state.
    /// </summary>
    /// <param name="parameter">Data used by the command. If the command does
    not require data to be passed, this object can be set to null.</param>
    /// <returns>
    /// true if this command can be executed; otherwise, false.
    /// </returns>
    public bool CanExecute(object parameter)
    {
        if (_canExecute != null)
        {
            bool tempCanExecute = _canExecute((T)parameter);
            if (_canExecuteCache != tempCanExecute)
            {
                _canExecuteCache = tempCanExecute;
            }
        }
    }
}

```

```
this.RaiseCanExecuteChanged();
}
}
return _canExecuteCache;
}
/// <summary>
/// Raises CanExecuteChanged event to notify changes in command status.
/// </summary>
public void RaiseCanExecuteChanged()
{
    if (CanExecuteChanged != null)
    {
        CanExecuteChanged(this, new EventArgs());
    }
}
/// <summary>
/// Defines the method to be called when the command is invoked.
/// </summary>
/// <param name="parameter">Data used by the command. If the command does
not require data to be passed, this object can be set to null.</param>
public void Execute(object parameter)
{
    if (_method != null)
        _method.Invoke((T)parameter);
}
#region ICommand Members
/// <summary>
/// 
/// </summary>
public event EventHandler CanExecuteChanged;
#endregion
}
public class ButtonViewModel : NotificationObject
{
    private bool _canperformaction = true;
    public ButtonViewModel()
    {
        ClickCommand = new DelegateCommand<object>(ClickAction,
        CanPerformClickAction);
    }
    public bool CanPerformAction
    {
        get
        {
            return _canperformaction;
        }
        set
        {
            _canperformaction = value;
            this.ClickCommand.RaiseCanExecuteChanged();
            this.RaisePropertyChanged("CanPerformAction");
        }
    }
    private bool CanPerformClickAction(object parameter)
    {
        return CanPerformAction;
    }
}
```

```
public DelegateCommand<object> ClickCommand { get; set; }
private void ClickAction(object parameter)
{
    MessageBox.Show(parameter.ToString());
}
```

**Note:** View [sample](#) in GitHub. This sample showcases how to bind commands to the `ButtonAdv` control.

### Multiline Text in WPF Button (ButtonAdv)

Multiline support is used to render text content of the Button control in multiple lines for precise view. One can apply the multiline text by using the [IsMultiLine](#) property.

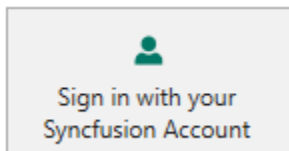
**Note:** This property is only applicable for large size mode button.

#### XML

```
<syncfusion:ButtonAdv x:Name="ButtonAdv" IsMultiLine="True"
    LargeIcon="image1/employee.png" Label="Sign in with your Syncfusion Account"
    SizeMode="Large" />
```

#### C#

```
ButtonAdv button = new ButtonAdv();
button.SizeMode = SizeMode.Large;
button.LargeIcon = new BitmapImage(new Uri("employee.png"));
button.IsMultiLine="true";
button.Label = "Sign in with your Syncfusion Account";
```



### Toggle State in WPF Button (ButtonAdv)

The button control can also be used as a toggle button, similar to the on/off view when the [IsCheckable](#) property is set to **true**. The default value of this property is **false**.

**Note:** In addition to [IsCheckable](#) property, the [IsChecked](#) property helps to check the button by default. In other words, during initial rendering, the button will appear in **on** state using the later property.

#### XML

```
<syncfusion:ButtonAdv Label="Log-in" SmallIcon="image/employee.png"
    SizeMode="Normal" IsCheckable="True" IsChecked="True" />
```

#### C#

```
ButtonAdv button = new ButtonAdv();
button.Label = "Log-in";
button.SizeMode = SizeMode.Normal;
button.SmallIcon = new BitmapImage(new Uri("employee.png"));
```

```
button.IsCheckedable = true;  
button.IsChecked = true;
```



Checkable Button control

### Events in WPF Button (ButtonAdv)

The ButtonAdv comprises of the pre-defined events that are illustrated below.

#### Click

The events occurs when the button control is clicked and any action can be handled in the respective event handler.

#### XML

```
<syncfusion:ButtonAdv Click="button_Click"/>
```

#### C#

```
ButtonAdv button = new ButtonAdv();  
button.Click += new RoutedEventHandler(button_Click);  
private void button_Click(object sender, RoutedEventArgs e)  
{  
}
```

#### Checked

The event occurs when the button control is utilized as toggle button, that is, when [IsCheckable](#) property value is set to **true**. Any action can be handled in the respective event handler.

#### XML

```
<syncfusion:ButtonAdv IsCheckable="true" Checked="button_Checked"/>
```

#### C#

```
ButtonAdv button = new ButtonAdv();  
button.IsCheckedable = true;  
button.Checked += new RoutedEventHandler(button_Checked);  
private void button_Checked(object sender, RoutedEventArgs e)  
{  
}
```

### Styles and Templates in WPF Button (ButtonAdv)

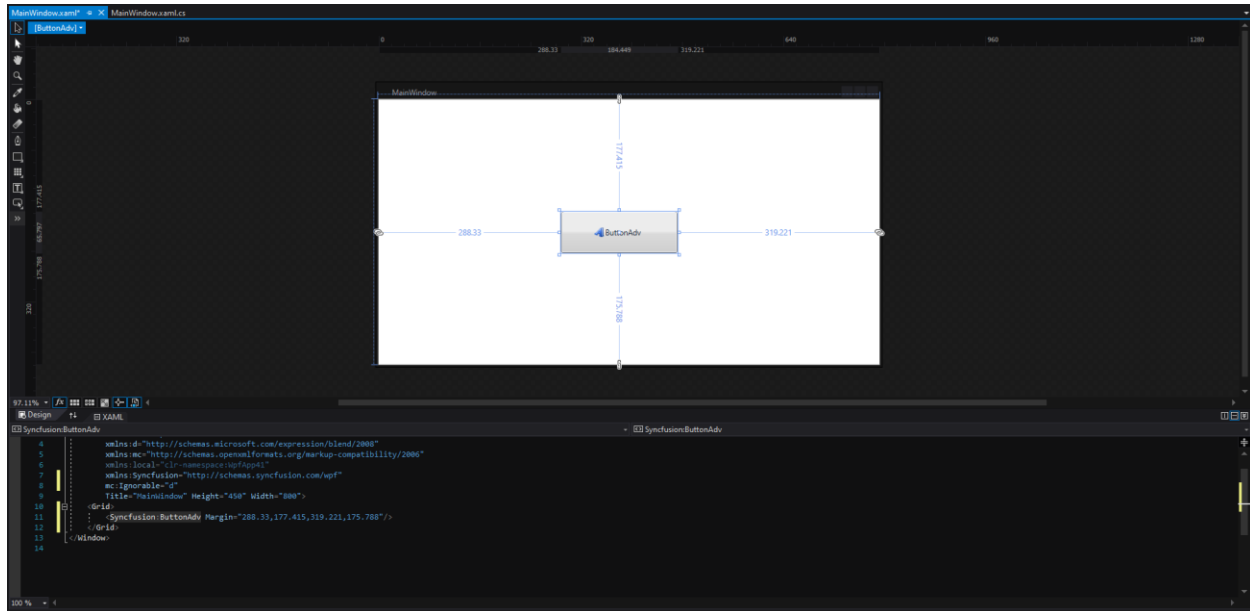
WPF styles and templates is a suite of features that allow developers and designers to create visual compelling effects and consistent appearance of the products.

This document provides information to change the visual appearance of the Button control. In addition, one can edit the structure of the Button control by using Blend and Visual Studio that helps to customize their appearances.

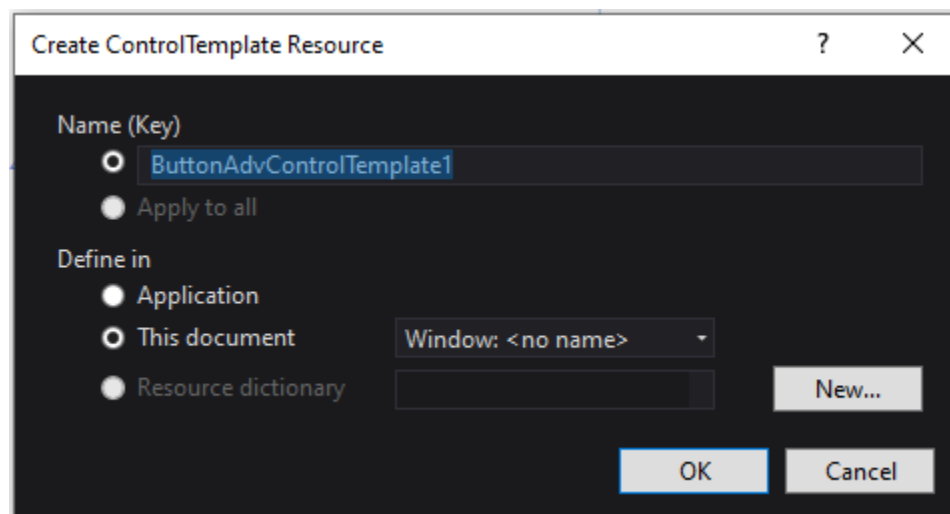
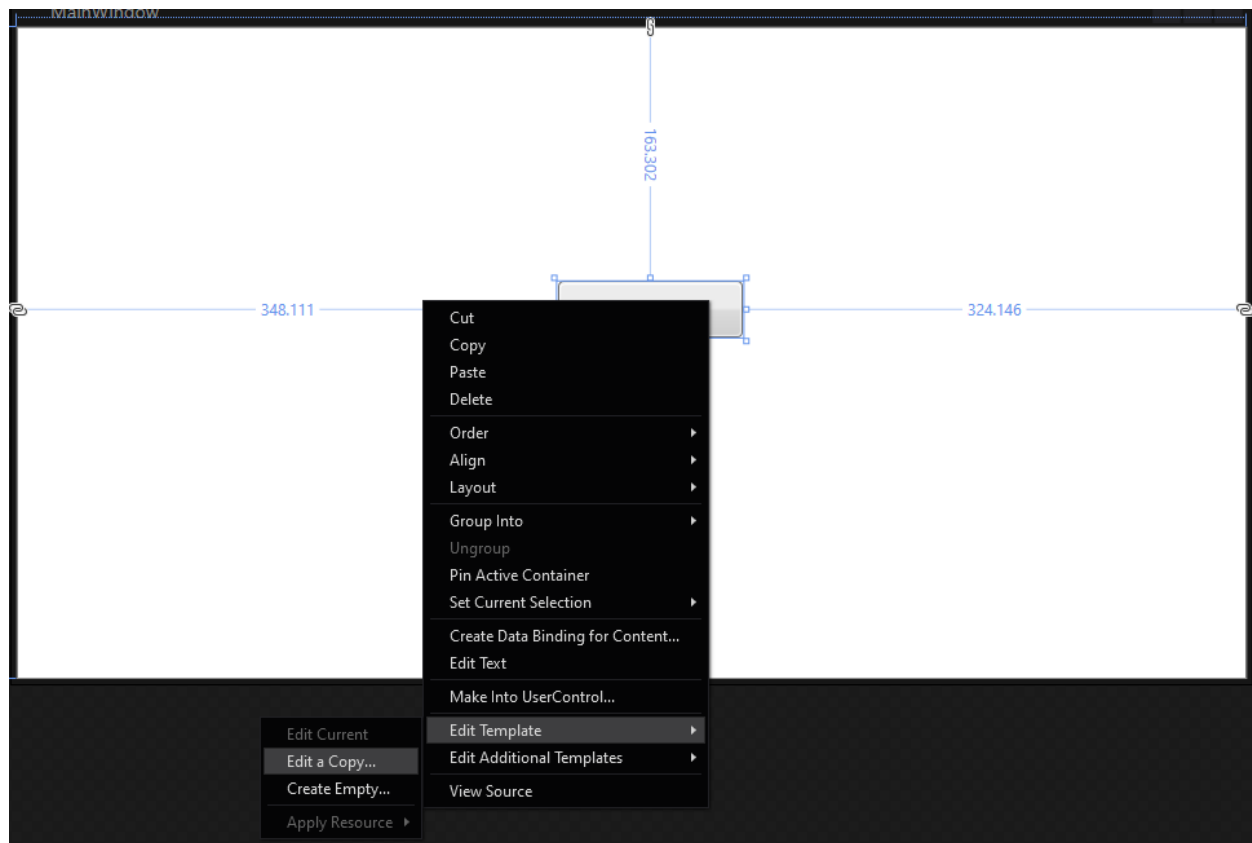


## Edit appearance in Expression Blend

- Open the application in Expression Blend.
- Select the Button control from the window.



- Right click on the button control and choose the menu option **Edit Template**. It will comprise of following two options.
- **Edit a Copy...** – Edit a copy of the default style. When selecting this option, a new dialog opens as follows.



The **Create Style Resource** dialog allows to enter the name or change the style name, as well as choose the location for the style. When **OK** is pressed, the Button control style is generated by the Expression Blend in the **Resource** section. The generated XAML can be edited in XAML view or in Visual Studio.

- **Create Empty...** - Creates an empty button style. Selecting this option will open the **Create ControlTemplate Resource** dialog which allows the user to enter the name or change the control template name, as well as choose the location for the template.

All resources will be displayed on the XAML file of the application after performing above steps. These resources can be edited to create a new Style.

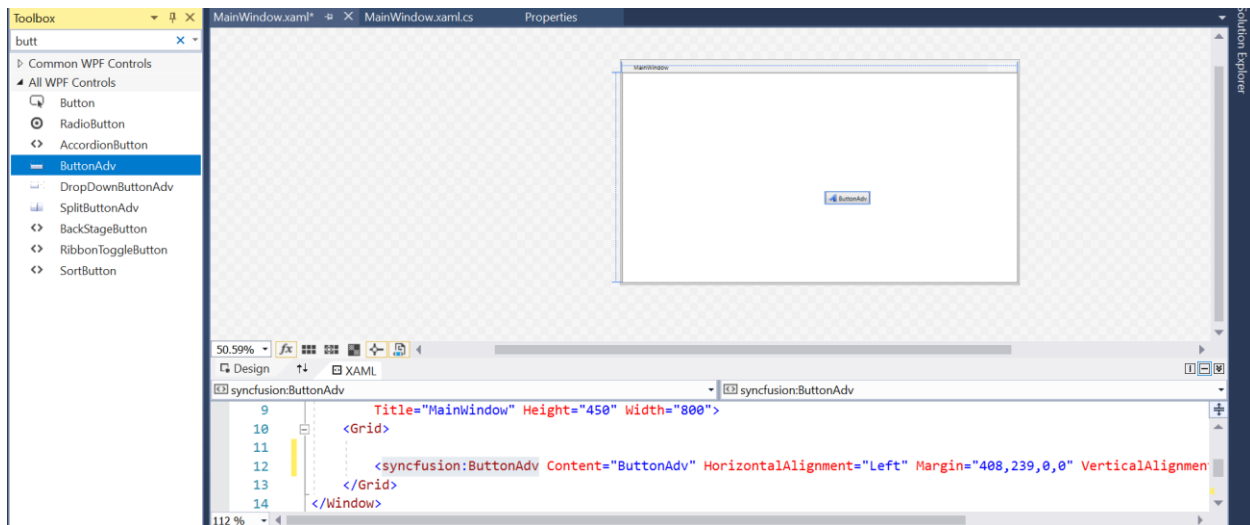
```
<Window.Resources>
<Style x:Key="ButtonAdvStyle1" TargetType="{x:Type syncfusion:ButtonAdv}"...>
</Window.Resources>
```



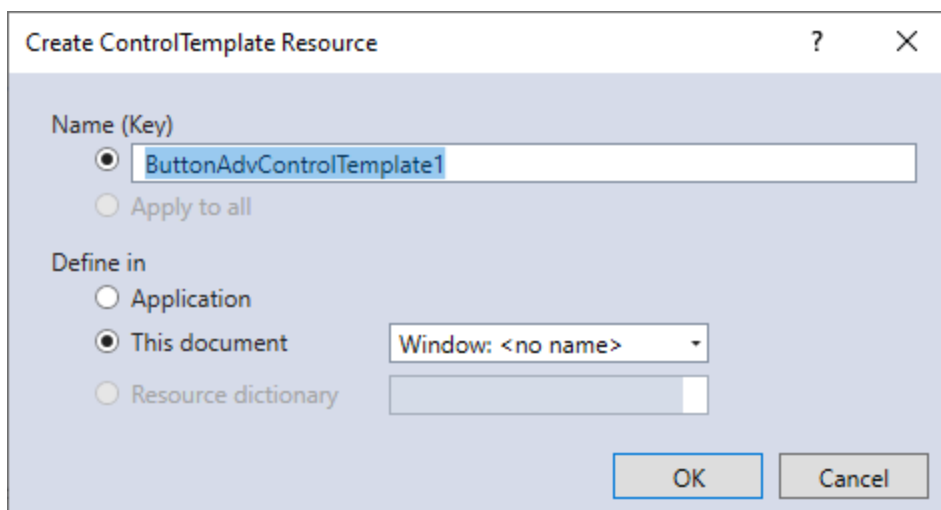
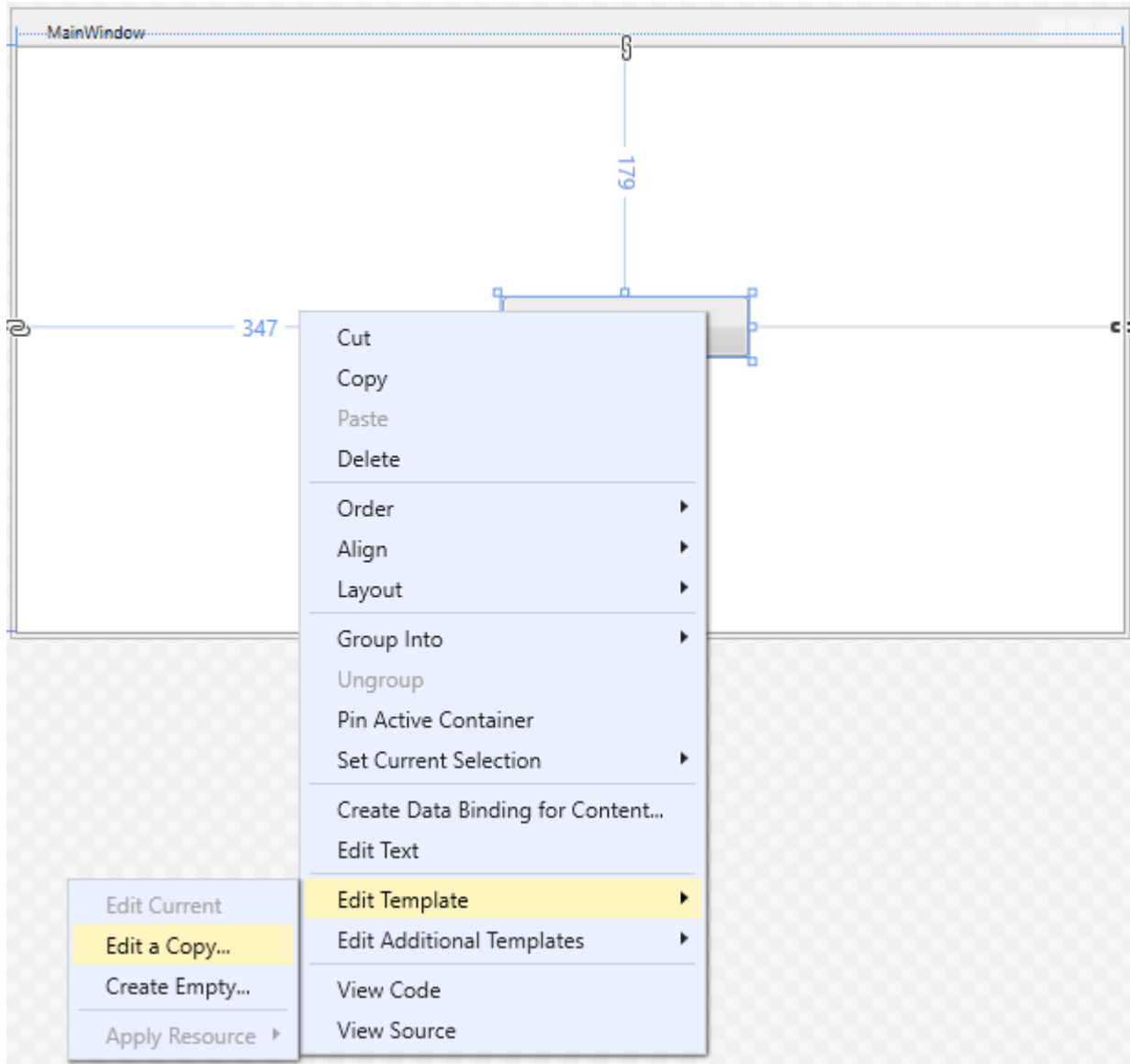
Button control edited in Expression Blend

[Edit appearance in Visual Studio](#)

- Open the application in Visual Studio.
- Open design view and select the Button control. Now right Click on the Button control and you can see some menu options showing up.



- On choosing menu option **Edit Template**, it further comprise of following two options.
- **Edit a Copy...** – Edit a copy of the default style. When selecting this option, a new dialog opens as follows.



The **Create ControlTemplate Resource** dialog allows to enter the name or change the control template name, as well as choose the location for the template. When **OK** is pressed, the Button control template is generated in the **Resource** section. The generated XAML can be edited in XAML view.

- **Create Empty...** - Creates an empty Button style. Selecting this option will open the **Create ControlTemplate Resource** dialog which allows the user to enter the name or change the control template name, as well as choose the location for the template.

All resources will be displayed on the XAML file of the application after performing above steps. These resources can be edited to create a new Style.

```
<Window.Resources>
  <ControlTemplate x:Key="ButtonAdvControlTemplate1" TargetType="{x:Type syncfusion:ButtonAdv}"...>
</Window.Resources>
```

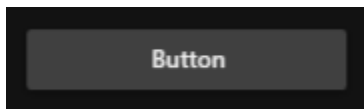


Button control edited in Visual Studio

### Themes in WPF Button (ButtonAdv)

Button supports various built-in themes. Refer to the below links to apply themes for the Button,

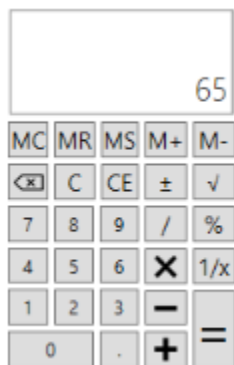
- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



## SfCalculator

### WPF Calculator (SfCalculator) Overview

The [SfCalculator](#) control allows us to perform mathematical operations as like in calculator.



### Key features

- **Value** – The Value property retrieves the current value calculated from the calculator. This is a **Read-Only** property.
- **Memory** – The Memory property retrieves the value stored in memory.

- DefaultValue - The Default value property shows default value in the value TextBox.
- DisplayText - The DisplayText property shows watermark in the value TextBox.

### Getting Started with WPF Calculator (SfCalculator)

This section provides a quick overview for working with the [WPF Calculator](#) (SfCalculator).

#### Assembly deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the [SfCalculator](#) control in any application.

You can find more details about installing the NuGet package in a WPF application in the following link:

[How to install nuget packages](#)

#### Creating Application with SfCalculator control

In this walk through, user will create a WPF application that contains [SfCalculator](#) control.

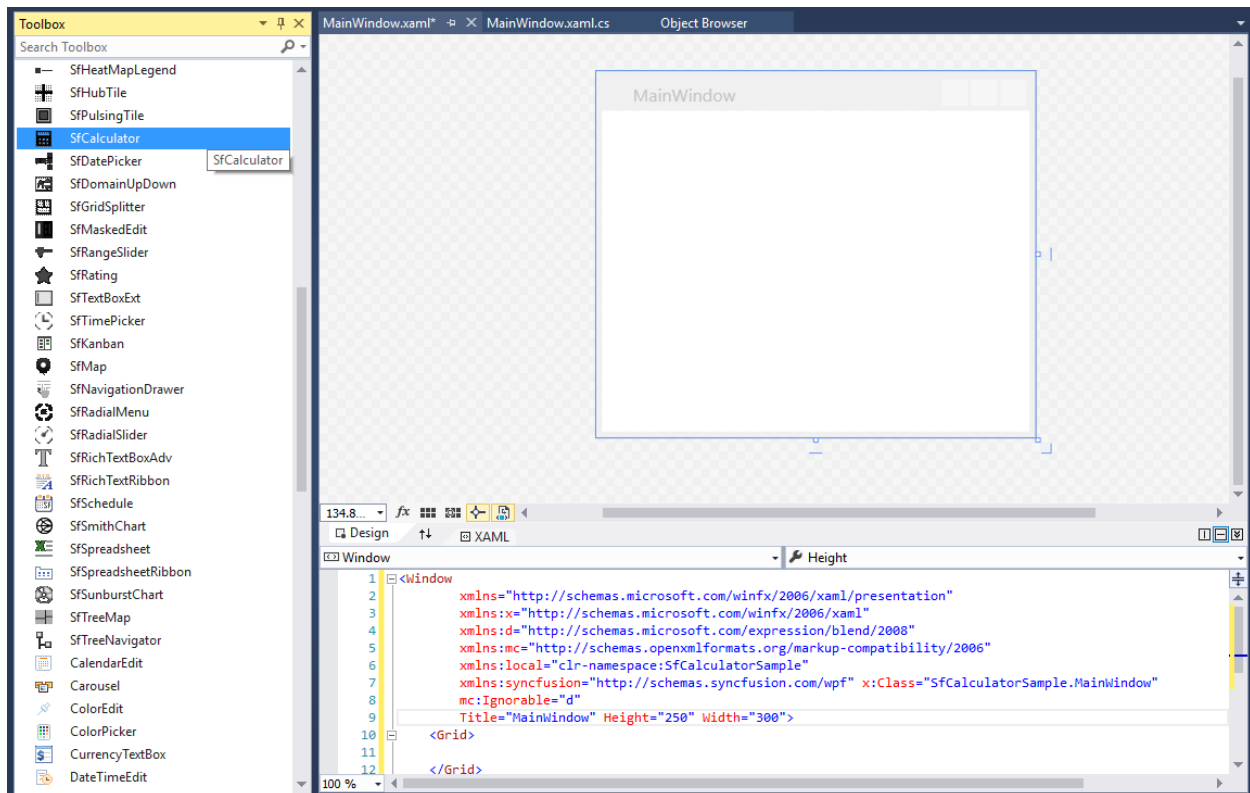
1. [Creating project](#)
2. [Adding control via designer](#)
3. [Adding control manually in XAML](#)
4. [Adding control manually in C#](#)

#### Creating project

Below section provides detailed information to create new project in Visual Studio to display [SfCalculator](#) control. The required [assemblies](#) will be added automatically.

#### Add control via designer

The [SfCalculator](#) control can be added to an application by dragging it from the toolbox to a designer view. The following assembly references are added automatically:



### Add control manually in XAML

In order to add [SfCalculator](#) control manually in XAML, do the below steps,

1. Add the below required assembly references to the project,
  - Syncfusion.SfInput.WPF
  - Syncfusion.Shared.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in XAML page.
3. Declare [SfCalculator](#) in XAML page.

### XML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="SfCalculatorSample.MainWindow"
Title="SfCalculator Sample" Height="350" Width="525">
<Grid>
<syncfusion:SfCalculator x:Name="sfCalculator" HorizontalAlignment="Center"
VerticalAlignment="Center" Width="100"/>
</Grid>
</Window>
```

### Add control manually in C#

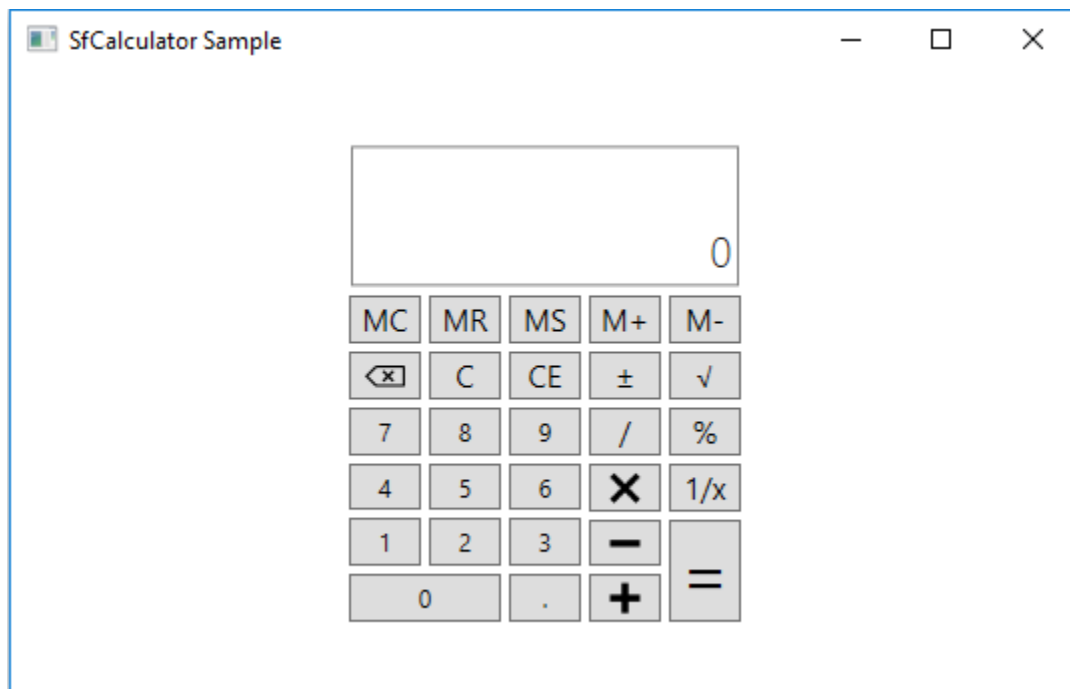
In order to add [SfCalculator](#) control manually in C#, do the below steps,

1. Add the below required assembly references to the project,

- Syncfusion.SfInput.WPF
  - Syncfusion.Shared.WPF
2. Import SfCalculator namespace **Syncfusion.Windows.Controls.Input**.
  3. Create SfCalculator control instance and add it to the window.

### C#

```
using Syncfusion.Windows.Controls.Input;
namespace SfCalculatorSample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            //Creating an instance of SfCalculator control
            SfCalculator sfCalculator = new SfCalculator();
            //Adding SfCalculator as window content
            this.Content = sfCalculator;
        }
    }
}
```



### Setting watermark

You can set watermark for [SfCalculator](#) control using [DefaultText](#) property.

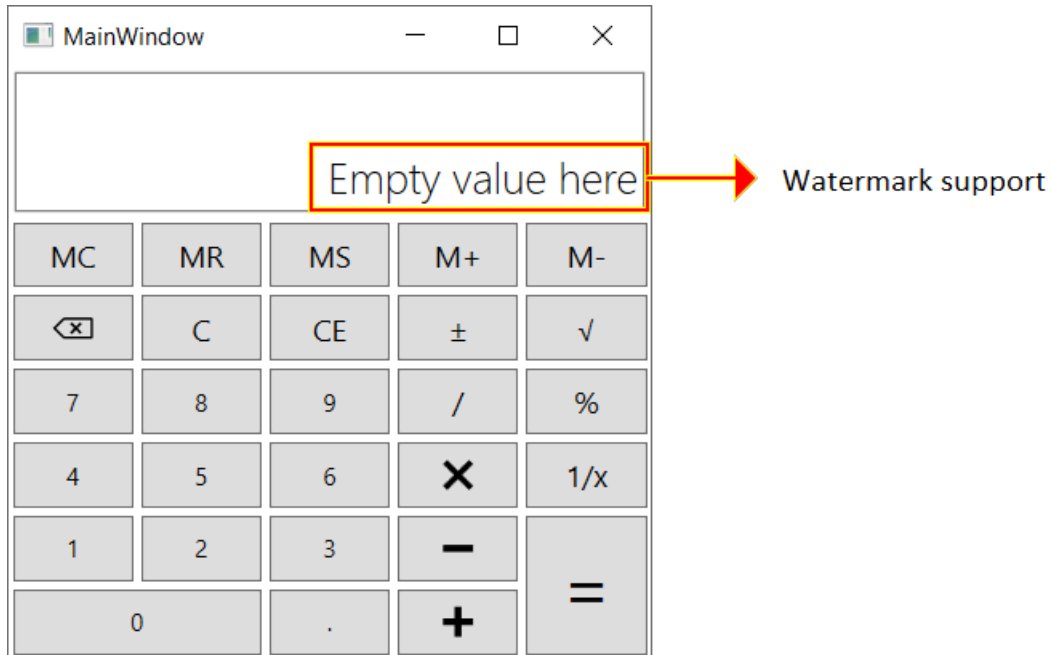
### XML



```
<syncfusion:SfCalculator HorizontalAlignment="Stretch" DisplayText="Empty value here" />
```

### C#

```
SfCalculator sfCalculator = new SfCalculator()  
{  
    HorizontalAlignment = HorizontalAlignment.Stretch,  
    DisplayText = "Empty value here"  
};
```



### Setting value

You can set the value to be displayed on the [SfCalculator](#) control using [DefaultValue](#) property.

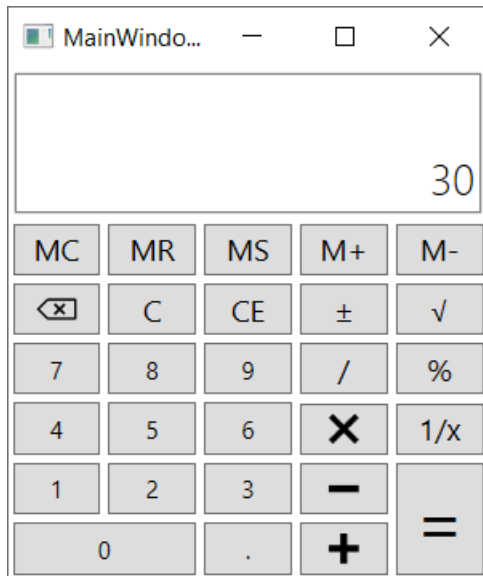
**Note:** The [Value](#) property of [SfCalculator](#) is **Read-only** which will allow you to get the value calculated from last expression and will be in decimal format.

### XML

```
<syncfusion:SfCalculator HorizontalAlignment="Stretch" DefaultValue="30" />
```

### C#

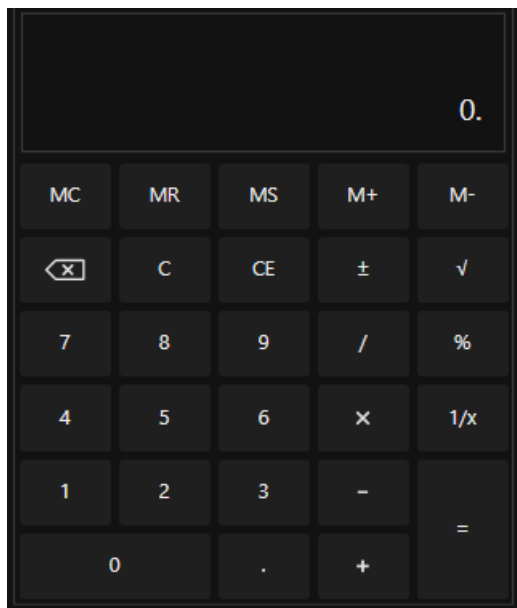
```
sfCalculator.DefaultValue = 30;
```



### Theme

The WPF Calculator (SfCalculator) supports various built-in themes. Refer to the below links to apply themes for the SfCalculator,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Memory in WPF Calculator (SfCalculator)

[Memory](#) property in the [SfCalculator](#) control is used to determine the value stored in the memory. The following buttons are available in the SfCalculator control to perform memory related operations.

---

**Note:** Memory property is a **Read-Only** property that is of decimal type.

---

### MS

MS performs memory storage. Clicking MS button will store the current value to the memory which can be retrieve and used later whenever necessary.

### MR

MR performs memory restore. Clicking MR button will restore the value stored in the memory. It retrieves the value stored in the memory for the further usage in the calculation.

### M+

Clicking M+ button will increment the value already stored in the memory by the value which we want to add. The newly calculated value gets stored in the memory now.

### M-

Clicking M- button will decrement the value already stored in the memory by the value which we want to subtract. The newly calculated value gets stored in the memory now.

### MC

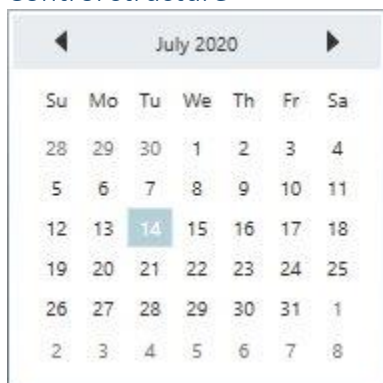
Clicking MC button, clears the value stored in the memory by resetting memory to 0.

## CalendarEdit

### WPF Calendar (CalendarEdit) Overview

The [CalendarEdit](#) control provides implementation of DateTime values in a calendar form. It therefore supports full range of dates allowed by that object. Effectively, any date ranging from year 0 to year 9999 A.D. can be displayed. It displays a calendar through where you can navigate to any day in a selected year. Its template can be easily modified just like any other WPF control. It is added with in-built animation to navigate between months.

### Control structure



### Features

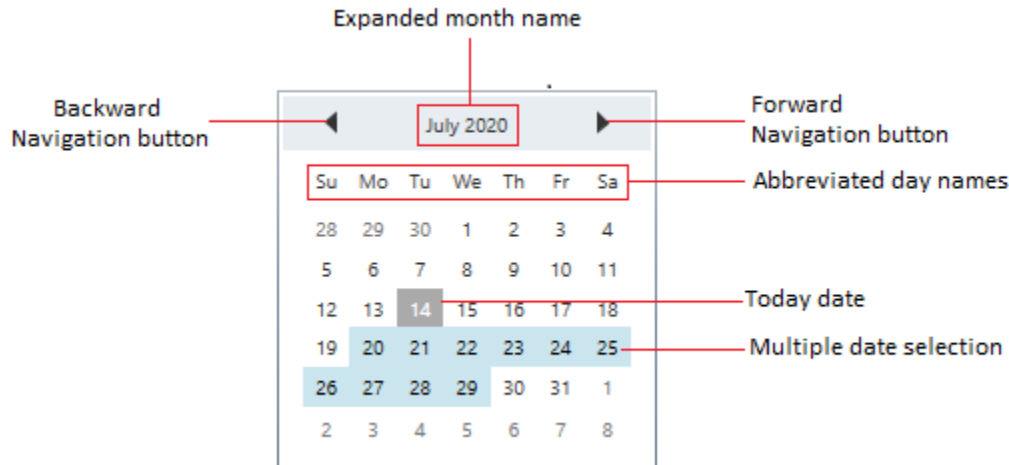
- **CalendarEdit** supports different Culture Types.
- Different built-in skins are available to give a good look and feel to the control.
- It has options to change the direction of month while navigation.
- The appearance of the header and selection border can be customized using different brushes.
- You can select more than one "Date" value at a time.
- Display area of Calendar control is limited using abbreviated days and months.
- A ToolTip can be set to the required date, or by row and column.

## Getting Started with WPF Calendar (CalendarEdit)

This section explains how to create a WPF [CalendarEdit](#) and explains about its structure.

### Structure of CalendarEdit

The various elements of [CalendarEdit](#) are illustrated in the following screenshot.



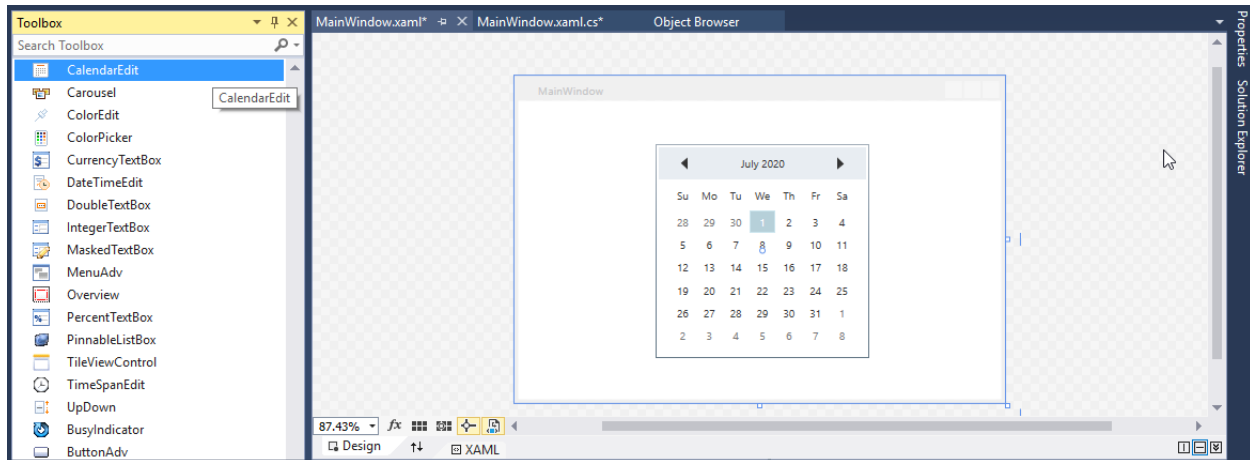
### Assembly deployment

Refer to the [Control Dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

Refer to this [documentation](#) to find more details about installing nuget packages in a WPF application.

### Adding WPF CalendarEdit via designer

1) The [CalendarEdit](#) can be added to an application by dragging it from the toolbox to a designer view. The following dependent assemblies will be added automatically: \* Syncfusion.Shared.WPF



2) Set the properties for [CalendarEdit](#) in design mode using the SmartTag feature.

### Adding WPF CalendarEdit via XAML

To add the [CalendarEdit](#) manually in XAML, follow these steps:

1) Create a new WPF project in Visual Studio. 2) Add the following required assembly references to the project: \* Syncfusion.Shared.WPF 3) Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf>, and declare the [CalendarEdit](#) in XAML page.

## XML

```
<Window x:Class="CalendarEdit_sample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:CalendarEdit_sample"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid Name="grid">
<syncfusion:CalendarEdit Name="calendarEdit"
Height="200"
Width="200"/>
</Grid>
</Window>
```

### Adding WPF CalendarEdit via C#

To add the `CalendarEdit` manually in C#, follow these steps:

1) Create a new WPF application via Visual Studio. 2) Add the following required assembly references to the project: \* Syncfusion.Shared.WPF 3) Include the required namespace.

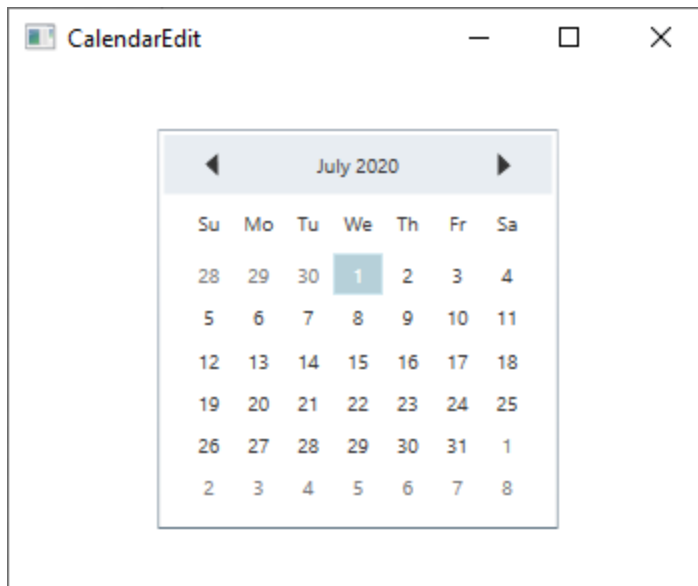
#### C#

```
using Syncfusion.Windows.Shared;
```

4) Create an instance of `CalendarEdit`, and add it to the window.

#### C#

```
// Creating an instance of the CalendarEdit
CalendarEdit calendarEdit = new CalendarEdit();
// Setting height and width to CalendarEdit
calendarEdit.Height = 200;
calendarEdit.Width = 200;
```



**Note:** [View Sample in GitHub](#)

Select a date

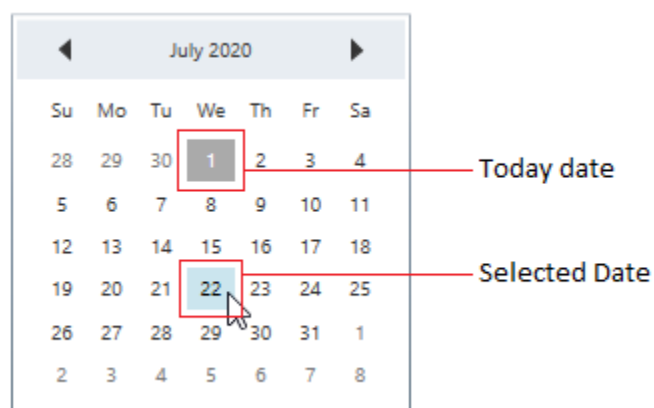
You can select a date in the `CalendarEdit` control by mouse click on the specific date. You can get the selected date by using the `Date` property.

#### XML

```
<!-- Selecting date -->
<syncfusion:CalendarEdit Name="calendarEdit" />
```

#### C#

```
CalendarEdit calendarEdit = new CalendarEdit();
```



**Note:** [View Sample in GitHub](#)

*Select a date programmatically*

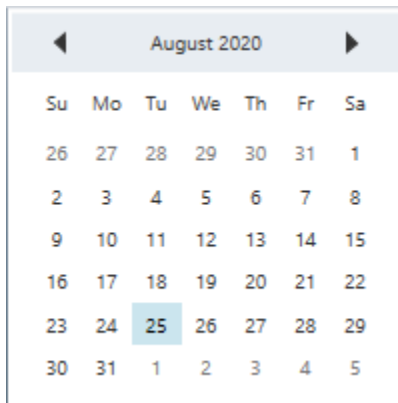
You can set selected date programmatically by setting the date value to the [Date](#) property.

#### XML

```
<!-- Selecting date programmatically -->
<syncfusion:CalendarEdit Date="25/08/2020"
Name="calendarEdit"/>
```

### C#

```
//Selecting date programmatically
calendarEdit.Date = new DateTime(2020, 08, 25);
```



**Note:** [View Sample in GitHub](#)

### Select multiple dates

You can select a multiple dates by drag and move the mouse from required start date to end date.

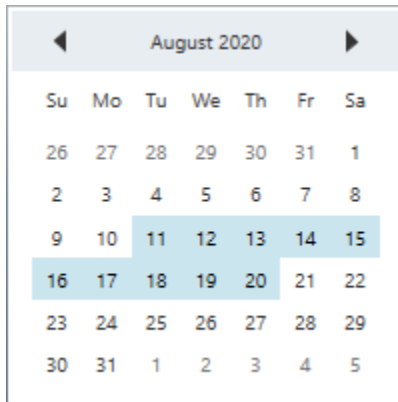
You can get the selected dates from the [SelectedDates](#) property. You can restrict the multiple date selection by setting [AllowMultiplySelection](#) property as `false`. The default value of [AllowMultiplySelection](#) property is `true`.

### XML

```
<!-- Selecting multiple dates-->
<syncfusion:CalendarEdit Name="calendarEdit"
AllowMultiplySelection="True"/>
```

### C#

```
//Selecting multiple dates
calendarEdit.AllowMultiplySelection = true;
```



**Note:** [View Sample in GitHub](#)

*Select a multiple dates programmatically*

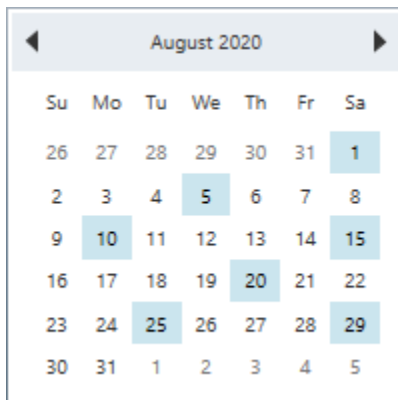
You can select a multiple dates programmatically by setting the dates to the [SelectedDatesList](#) property.

### XML

```
<!-- Selecting date -->
<syncfusion:CalendarEdit Date="08/25/2020"
AllowMultiplySelection="True"
Name="calendarEdit"/>
```

### C#

```
//Selecting multiple date programmatically
calendarEdit.AllowMultiplySelection = true;
calendarEdit.SelectedDatesList = new List<Date>();
calendarEdit.SelectedDatesList.Add(new Date(2020, 08, 01));
calendarEdit.SelectedDatesList.Add(new Date(2020, 08, 05));
calendarEdit.SelectedDatesList.Add(new Date(2020, 08, 10));
calendarEdit.SelectedDatesList.Add(new Date(2020, 08, 15));
calendarEdit.SelectedDatesList.Add(new Date(2020, 08, 20));
calendarEdit.SelectedDatesList.Add(new Date(2020, 08, 29));
```



*Restrict date selection*

You can restrict the users from selecting a date within the particular range by specifying [MinDate](#) and [MaxDate](#) in [CalendarEdit](#) control.

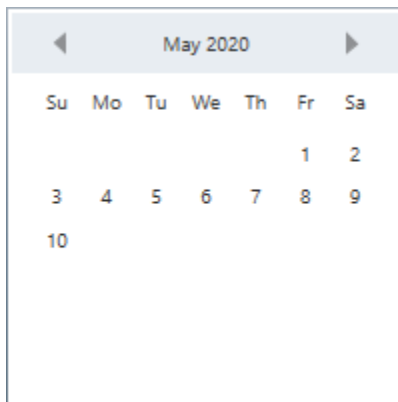


**XML**

```
<!--Setting date range -->
<syncfusion:CalendarEdit MinDate="05/1/2020"
MaxDate="05/10/2020"
Name="calendarEdit"/>
```

**C#**

```
CalendarEdit calendarEdit = new CalendarEdit();
calendarEdit.MinDate = new DateTime(2020, 05, 01);
calendarEdit.MaxDate = new DateTime(2020, 05, 10);
```



**Note:** [View Sample in GitHub](#)

**Block dates**

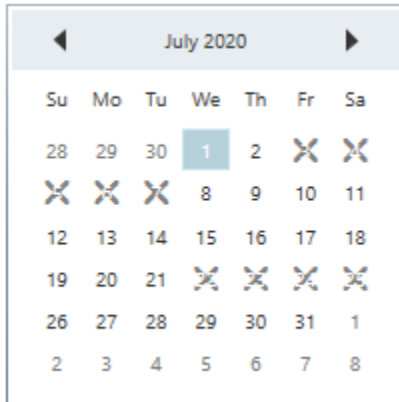
If you want to block particular dates from the date selection, add that date ranges to the [BlackoutDates](#) collection. You can add more block out date ranges to the [BlackoutDates](#) collection.

**XML**

```
<syncfusion:CalendarEdit Name="calendarEdit">
<syncfusion:CalendarEdit.BlackoutDates>
<syncfusion:BlackoutDatesRange StartDate="07/03/2020"
EndDate="07/07/2020" />
<syncfusion:BlackoutDatesRange StartDate="07/22/2020"
EndDate="07/25/2020" />
</syncfusion:CalendarEdit.BlackoutDates>
</syncfusion:CalendarEdit>
```

**C#**

```
calendarEdit.BlackoutDates.Add(new BlackoutDatesRange() {
StartDate = new DateTime(2020, 08, 03),
EndDate = new DateTime(2020, 08, 07)});
calendarEdit.BlackoutDates.Add(new BlackoutDatesRange() {
StartDate = new DateTime(2020, 08, 22),
EndDate = new DateTime(2020, 08, 25)});
```



**Note:** [View Sample in GitHub](#)

### Special days

You can differentiate the special day from other days by setting that date value to the [SpecialDate.Date](#) property and adding [SpecialDate.Date](#) into the [SpecialDates](#) collection. You can use the [SpecialDate.CellTemplate](#) property to customize the [SpecialDate](#) day cell appearance.

### C#

```
//ViewModel.cs
public class ViewModel {
    private SpecialDatesCollection specialDates;
    public SpecialDatesCollection SpecialDates {
        get { return specialDates; }
        set { specialDates = value; }
    }
    public ViewModel() {
        SpecialDates = new SpecialDatesCollection();
    }
}
```

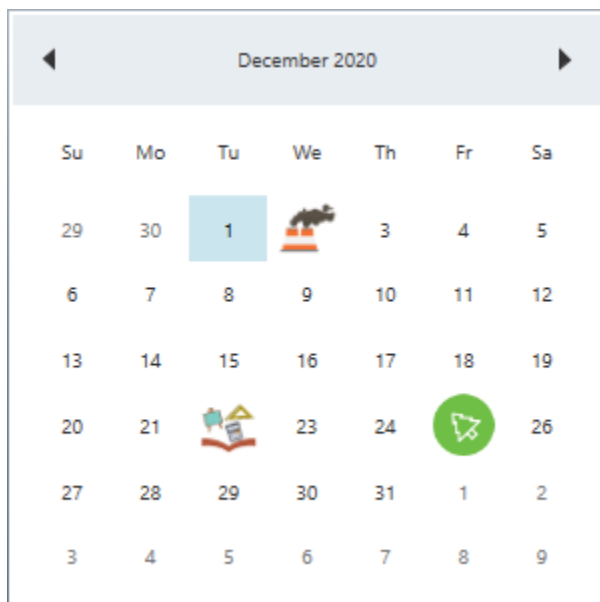
### XML

```
<Window.Resources>
<DataTemplate x:Key="WorldEnvironmentDay" >
<Image Source="Resources\Icon_Environmental day.png" />
</DataTemplate>
<DataTemplate x:Key="EngineersDay" >
<Image Source="Resources\Icon_Engineer day.png" />
</DataTemplate>
<DataTemplate x:Key="PollutionPreventionDay" >
<Image Source="Resources\Icon_Pollution day.png" />
</DataTemplate>
<DataTemplate x:Key="NationalMathematicsDay" >
<Image Source="Resources\Icon_Mathematics day.png" />
</DataTemplate>
<DataTemplate x:Key="Christmas" >
<Image Source="Resources\Christmas.png" />
</DataTemplate>
<local:ViewModel x:Key="viewModel">
<local:ViewModel.SpecialDates>
```

```

<syncfusion:SpecialDate Date="06/05/2020" CellTemplate="{StaticResource
WorldEnvironmentDay }"/>
<syncfusion:SpecialDate Date="09/15/2020" CellTemplate="{StaticResource
EngineersDay }"/>
<syncfusion:SpecialDate Date="12/02/2020" CellTemplate="{StaticResource
PollutionPreventionDay }"/>
<syncfusion:SpecialDate Date="12/22/2020" CellTemplate="{StaticResource
NationalMathematicsDay }"/>
<syncfusion:SpecialDate Date="12/25/2020" CellTemplate="{StaticResource
Christmas }"/>
</local:ViewModel.SpecialDates>
</local:ViewModel>
</Window.Resources>
<Grid>
<syncfusion:CalendarEdit DataContext="{StaticResource viewModel}"
SpecialDates="{Binding SpecialDates}"
Name="calendarEdit" />
</Grid>

```



**Note:** [View Sample in GitHub](#)

### Display week numbers

If you want to know the week number of the currently displayed dates, use the [ShowWeekNumbers](#) property as `true`. The default value of `ShowWeekNumbers` property is `false`.

### XML

```

<!--Showing Week number-->
<syncfusion:CalendarEdit Name="calendarEdit"
ShowWeekNumbers="True"/>

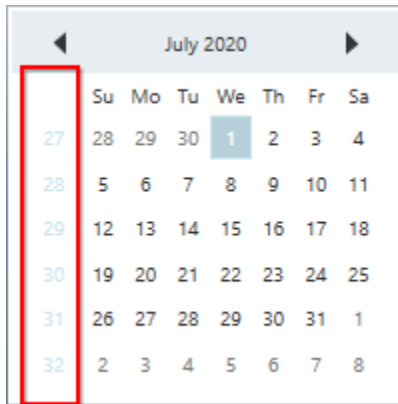
```

### C#

```

//Shows week numbers
calendarEdit.ShowWeekNumbers = true;

```

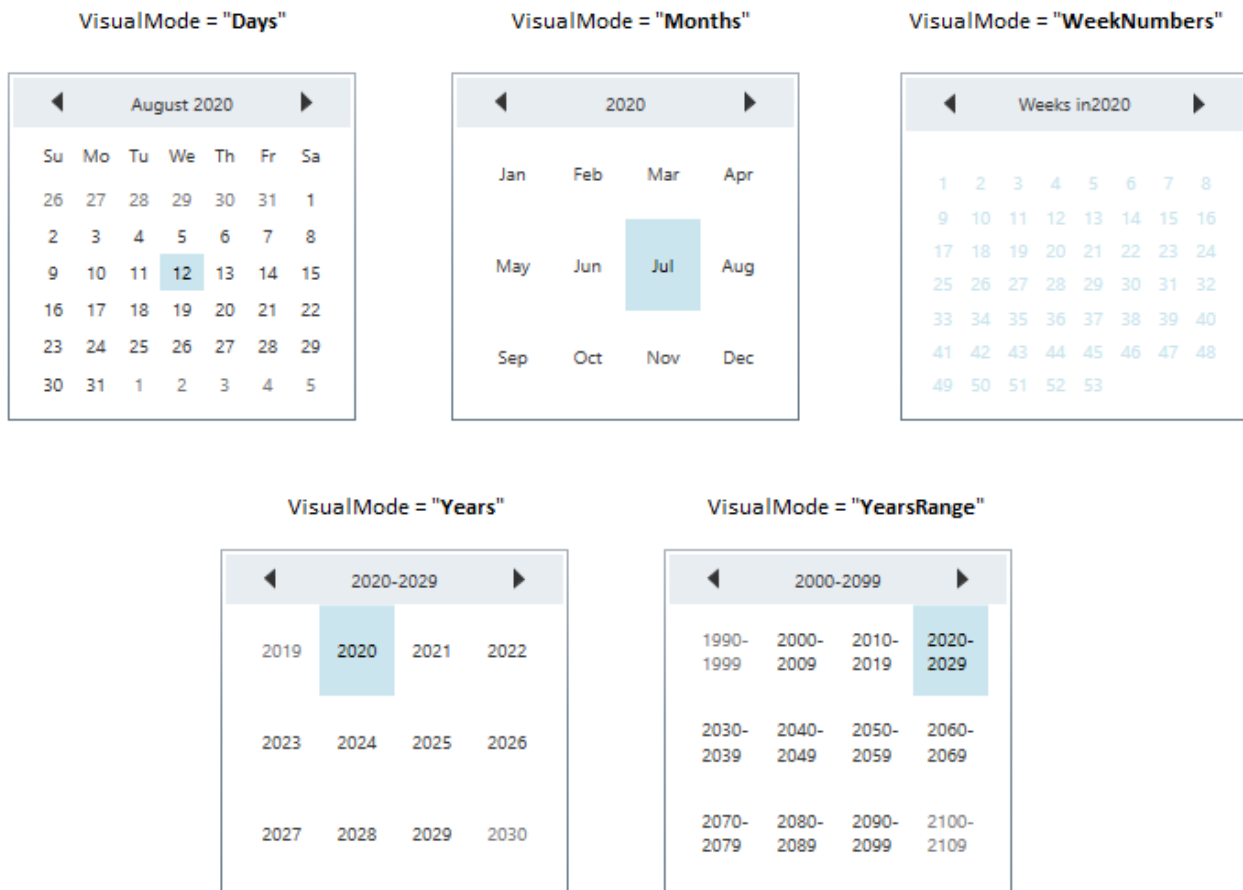


**Note:** [View Sample in GitHub](#)

Change default view (Month, Year, Decade)

By default, the days are displayed in the `CalendarEdit`.

You can change the default calendar view as week numbers, month, years or years range mode by setting the respective value to the `VisualMode` property. The default value of `VisualMode` property is `Days`.

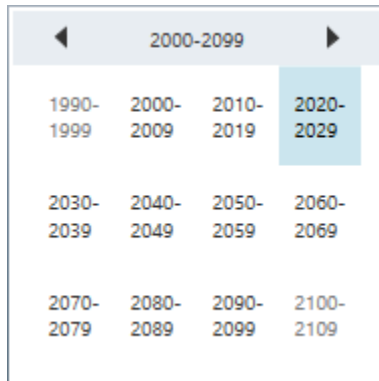


## XML

```
<syncfusion:CalendarEdit Name="calendarEdit"
VisualMode="YearsRange"/>
```

**C#**

```
calendarEdit.VisualMode = CalendarVisualMode.YearsRange;
```



**Note:** [View Sample in GitHub](#)

**Setting Culture**

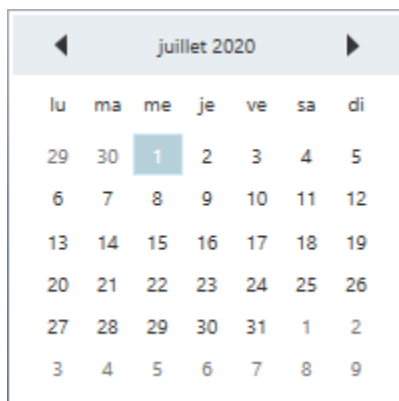
You can change the culture for **CalendarEdit** control by setting the required culture to the **Culture** property.

**XML**

```
<!--Setting french culture-->
<syncfusion:CalendarEdit Name="calendarEdit"
Culture="fr-FR"/>
```

**C#**

```
//Setting french culture
calendarEdit.Culture = new CultureInfo("fr-FR");
```



### Show full month and week name

You can display full month names and week day names by setting the [ShowAbbreviatedDayNames](#) and [ShowAbbreviatedMonthNames](#) properties as `false`. The default value of `ShowAbbreviatedDayNames` and `ShowAbbreviatedDayNames` property is `true`.

### XML

```
<syncfusion:CalendarEdit ShowAbbreviatedDayNames="False"
ShowAbbreviatedMonthNames="False"
Name="calendarEdit"/>
```

### C#

```
CalendarEdit calendarEdit = new CalendarEdit();
calendarEdit.ShowAbbreviatedDayNames = false;
calendarEdit.ShowAbbreviatedMonthNames = false;
```

Expanded month and week day names

July 2020						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Abbreviated month and week day names

Jul 2020						
Su	Mo	Tu	We	Th	Fr	Sa
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

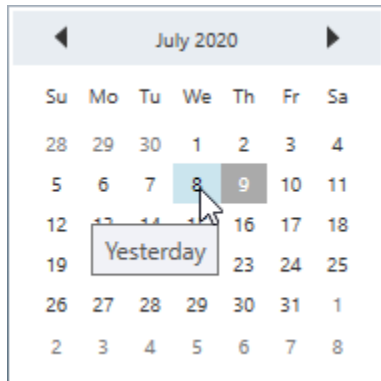
**Note:** [View Sample in GitHub](#)

### Tooltip for particular days

You can set tooltip for particular days in the `CalendarEdit` control by using the [SetToolTip\(Date,ToolTip\)](#) method. You can pass the specific date and tooltip to the `SetToolTip(Date,ToolTip)` method.

### C#

```
Date yesterday = new Date(DateTime.Now.Year, DateTime.Now.Month,
DateTime.Now.Day-1);
Date today = new Date(DateTime.Now.Year, DateTime.Now.Month,
DateTime.Now.Day);
CalendarEdit calendarEdit = new CalendarEdit();
//Setting tooltip for yesterday and today dates
calendarEdit.SetToolTip(yesterday, new ToolTip() { Content = "Yesterday" });
calendarEdit.SetToolTip(today, new ToolTip() { Content = "Today" });
```



### Selected date changed notification

The selected date changed in `CalendarEdit` can be examined using [DateChanged](#) event. The `DateChanged` event contains the old and newly selected date time values in the `OldValue` and `NewValue` properties.

### XML

```
<syncfusion:CalendarEdit DateChanged="CalendarEdit_DateChanged"
Name="calendarEdit"/>
```

### C#

```
CalendarEdit calendarEdit = new CalendarEdit();
calendarEdit.DateChanged += CalendarEdit_DateChanged;
```

You can handle the event as follows,

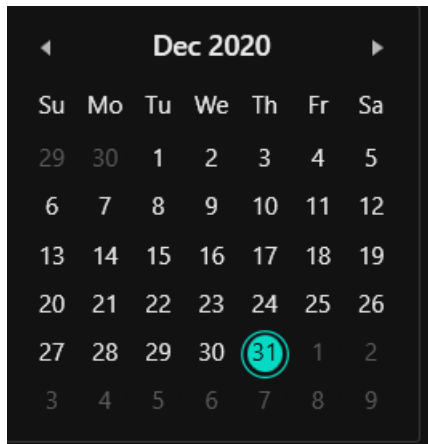
### C#

```
private void CalendarEdit_DateChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
    //Get old and new selected date values
    var oldValue = e.OldValue;
    var newValue = e.NewValue;
}
```

### Theme

`CalendarEdit` supports various built-in themes. Refer to the below links to apply themes for the `CalendarEdit`,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Date-Selection in WPF Calendar (CalendarEdit)

This section explains how to select a date and custom UI of the WPF [CalendarEdit](#) control.

#### Select a date

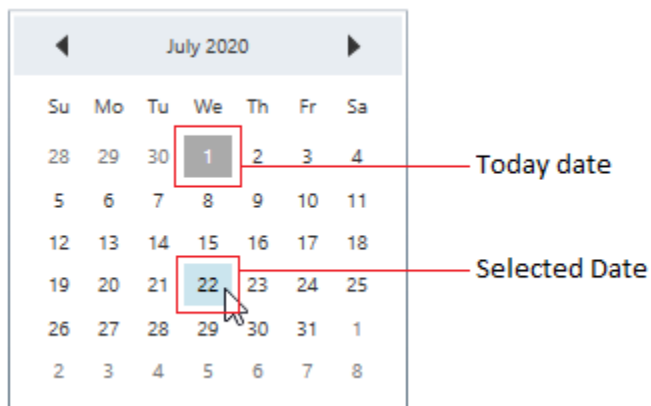
You can select a date in the `CalendarEdit` control by mouse click on the specific date. You can get the selected date by using the `Date` property.

#### XML

```
<!-- Selecting date -->
<syncfusion:CalendarEdit Name="calendarEdit" />
```

#### C#

```
CalendarEdit calendarEdit = new CalendarEdit();
```



**Note:** [View Sample in GitHub](#)

#### Select a date programmatically

You can set selected date programmatically by setting the date value to the `Date` property.

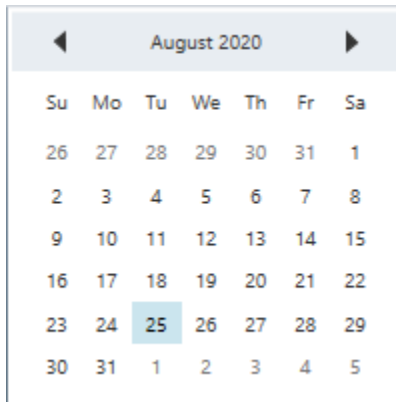
#### XML

```
<!-- Selecting date programmatically -->
<syncfusion:CalendarEdit Date="25/08/2020"
Name="calendarEdit"/>
```



**C#**

```
//Selecting date programmatically
calendarEdit.Date = new DateTime(2020, 08, 25);
```



**Note:** [View Sample in GitHub](#)

## Select multiple dates

You can select a multiple dates by drag and move the mouse from required start date to end date.

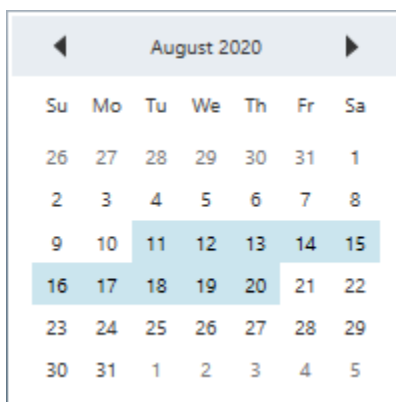
You can get the selected dates from the [SelectedDates](#) property. You can restrict the multiple date selection by setting [AllowMultiplySelection](#) property as `false`. The default value of [AllowMultiplySelection](#) property is `true`.

**XML**

```
<!-- Selecting multiple dates-->
<syncfusion:CalendarEdit Name="calendarEdit"
AllowMultiplySelection="True"/>
```

**C#**

```
//Selecting multiple dates
calendarEdit.AllowMultiplySelection = true;
```



**Note:** [View Sample in GitHub](#)

### Select specific multiple dates

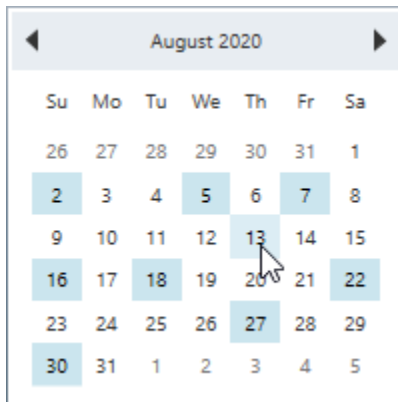
You can select a specific multiple dates by pressing the **Ctrl** key and select required dates using mouse click.

### XML

```
<!-- Selecting multiple dates-->
<syncfusion:CalendarEdit Name="calendarEdit"
AllowMultiplySelection="True"/>
```

### C#

```
//Selecting multiple dates
calendarEdit.AllowMultiplySelection = true;
```



**Note:** [View Sample in GitHub](#)

### Select a multiple dates programmatically

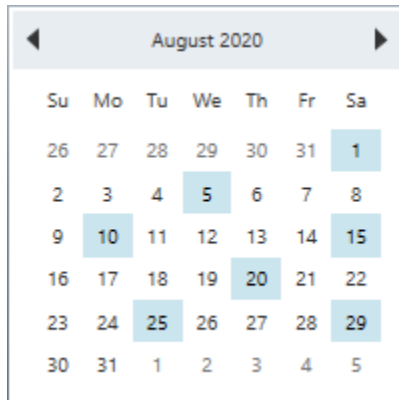
You can select a multiple dates programmatically by setting the dates to the [SelectedDatesList](#) property.

### XML

```
<!-- Selecting date -->
<syncfusion:CalendarEdit Date="08/25/2020"
AllowMultiplySelection="True"
Name="calendarEdit"/>
```

### C#

```
//Selecting multiple date programmatically
calendarEdit.AllowMultiplySelection = true;
calendarEdit.SelectedDatesList = new List<Date>();
calendarEdit.SelectedDatesList.Add(new Date(2020, 08, 01));
calendarEdit.SelectedDatesList.Add(new Date(2020, 08, 05));
calendarEdit.SelectedDatesList.Add(new Date(2020, 08, 10));
calendarEdit.SelectedDatesList.Add(new Date(2020, 08, 15));
calendarEdit.SelectedDatesList.Add(new Date(2020, 08, 20));
calendarEdit.SelectedDatesList.Add(new Date(2020, 08, 29));
```



### Select multiple dates using key navigation

You can select a multiple dates by pressing the **Shift** with **Arrow** keys. If you want to select multiple dates in forward direction, press the **Shift + Down** or **Shift + Right** keys. If you want to select a date in backward direction, press the **Shift + UP** or **Shift + Left** keys. You can get the selected dates from the **SelectedDates** property.

For example, if you start selecting the date from **18 Nov 2020** using key navigation, it will be select the dates as follows,

### XML

```
<!-- Selecting multiple dates-->
<syncfusion:CalendarEdit Name="calendarEdit"
AllowMultiplySelection="True"/>
```

### C#

```
//Selecting multiple dates
calendarEdit.AllowMultiplySelection = true;
```

**Backward direction**

Shift + Left arrow

November 2020						
Su	Mo	Tu	We	Th	Fr	Sa
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

Shift + Top arrow

November 2020						
Su	Mo	Tu	We	Th	Fr	Sa
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

**Forward direction**

Shift + Right arrow

November 2020						
Su	Mo	Tu	We	Th	Fr	Sa
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

Shift + Down arrow

November 2020						
Su	Mo	Tu	We	Th	Fr	Sa
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

*Select specific multiple dates using key navigation*

You can select or unselect the specific multiple dates by using the **Ctrl**, **Arrow** keys and **Space** bar.

- **Ctrl + Arrow** keys → To move the focus to the top, left, bottom and right cells with maintaining the old selection.
- **Space** bar → To select the currently focused item.
- **Ctrl + Space** bar → To remove an item from multiple selection that is already selected.

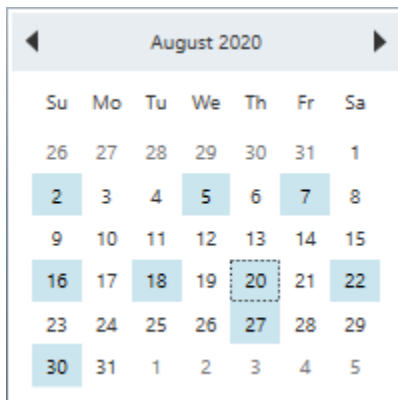
**XML**

```
<!-- Selecting multiple dates-->
<syncfusion:CalendarEdit Name="calendarEdit"
AllowMultiplySelection="True"/>
```

**C#**

```
//Selecting multiple dates
```

```
calendarEdit.AllowMultiplySelection = true;
```

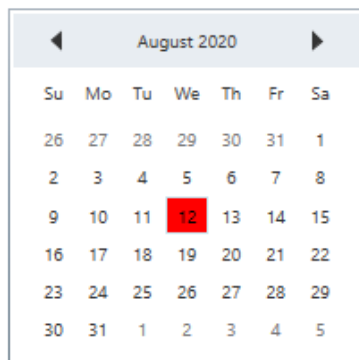


**Note:** [View Sample in GitHub](#)

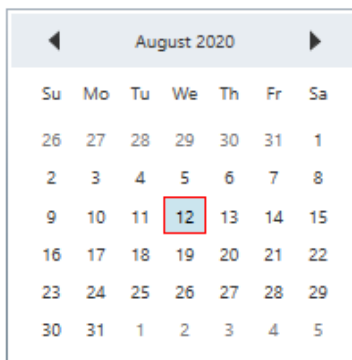
Highlight selected date

If you want to highlight the selected date, change it's foreground, background or border brush by using the [SelectionForeground](#), [SelectedDayCellBackground](#) and [SelectedDayCellBorderBrush](#) and properties. You can also change the mouse hover background and border brush for the selected day cell by using the [SelectedDayCellHoverBackground](#) and [SelectionBorderBrush](#) properties.

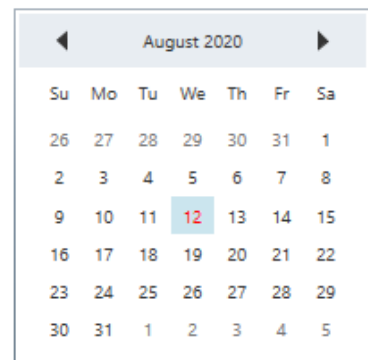
SelectedDayCell Background



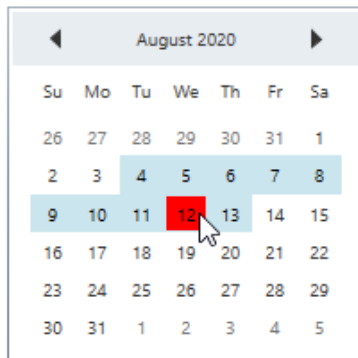
SelectedDayCell BorderBrush



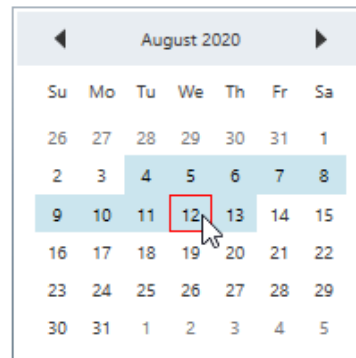
SelectedDayCell Foreground



SelectedDayCell HoverBackground



SelectedDayCell HoverBorderBrush



## XML

```
<syncfusion:CalendarEdit SelectedDayCellBackground="Yellow"
```

```

SelectedDayCellBorderBrush="Blue"
SelectionForeground="Red"
SelectedDayCellHoverBackground="Green"
SelectionBorderBrush="Red"
Name="calendarEdit" />

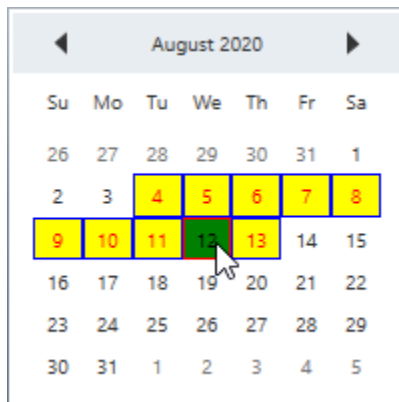
```

**C#**

```

calendarEdit.SelectedDayCellBackground = Brushes.Yellow;
calendarEdit.SelectedDayCellBorderBrush = Brushes.Blue;
calendarEdit.SelectionForeground = Brushes.Red;
calendarEdit.SelectedDayCellHoverBackground = Brushes.Green;
calendarEdit.SelectionBorderBrush = Brushes.Red;

```



**Note:** [View Sample in GitHub](#)

**Get today date**

If you want to know the today date, use the [TodayDate](#) property. It contains the today date of the `CalendarEdit` control.

**C#**

```

CalendarEdit calendarEdit = new calendarEdit();
//get the today date
var today_Date= calendarEdit.TodayDate;

```

**Display today date**

If you want to display the today date in the `CalendarEdit` control, use the [TodayRowIsVisible](#) property value as `true`. It will display the today date in the bottom-left corner of the `CalendarEdit` control. The default value of `TodayRowIsVisible` property is `false`.

**XML**

```

<syncfusion:CalendarEdit TodayRowIsVisible="True"
Name="calendarEdit" />

```

**C#**

```

//Enable the today row
calendarEdit.TodayRowIsVisible = true;

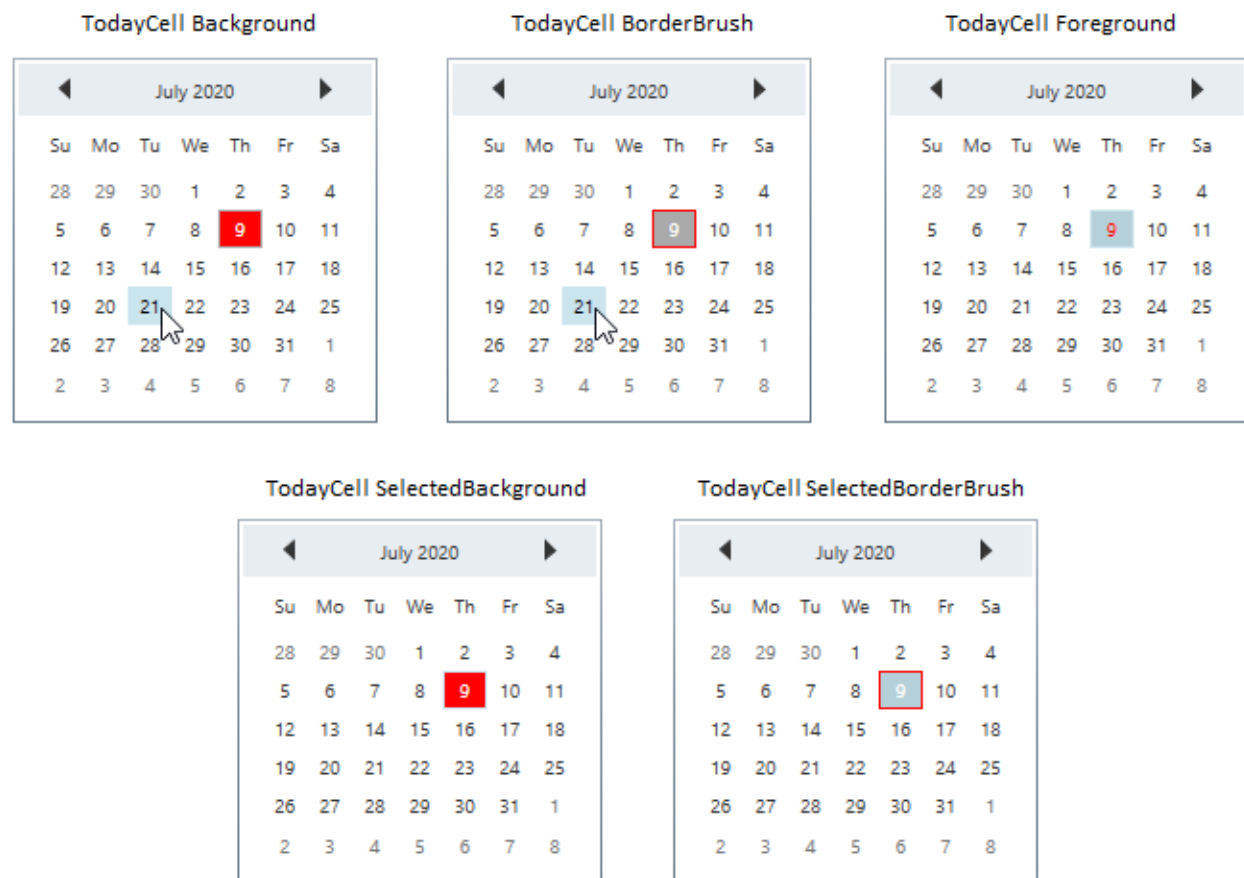
```



**Note:** [View Sample in GitHub](#)

#### Highlight today date

If you want to highlight the today date, change its foreground, background or border brush by using [TodayCellForeground](#), [TodayCellBackground](#) and [TodayCellBorderBrush](#) properties. You can also change the selected border brush and background of the today date by using the [TodayCellSelectedBorderBrush](#) and [TodayCellSelectedBackground](#) properties.



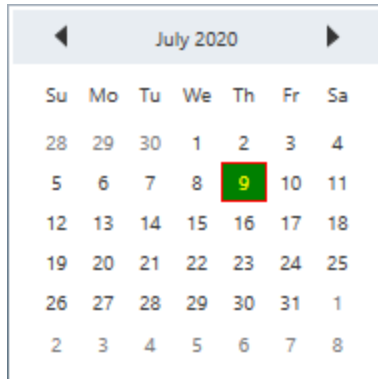
#### XML

```
<syncfusion:CalendarEdit TodayCellSelectedBorderBrush="Red">
```

```
TodayCellSelectedBackground="Green"
TodayCellForeground="Yellow"
Name="calendarEdit" />
```

**C#**

```
calendarEdit.TodayCellSelectedBorderBrush = Brushes.Red;
calendarEdit.TodayCellSelectedBackground = Brushes.Green;
calendarEdit.TodayCellForeground = Brushes.Yellow;
```



**Note:** [View Sample in GitHub](#)

Differentiate current month days from other days

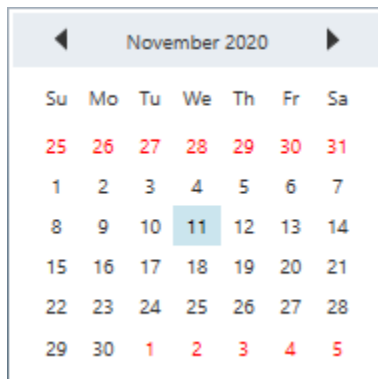
If you want to differentiate currently selected month days from previous or next month days, change the previous and next month days foreground by using the [NotCurrentMonthForeground](#) property. The default value of `NotCurrentMonthForeground` property is `Gray`.

**XML**

```
<syncfusion:CalendarEdit NotCurrentMonthForeground="Red"
Name="calendarEdit" />
```

**C#**

```
calendarEdit.NotCurrentMonthForeground = Brushes.Red;
```



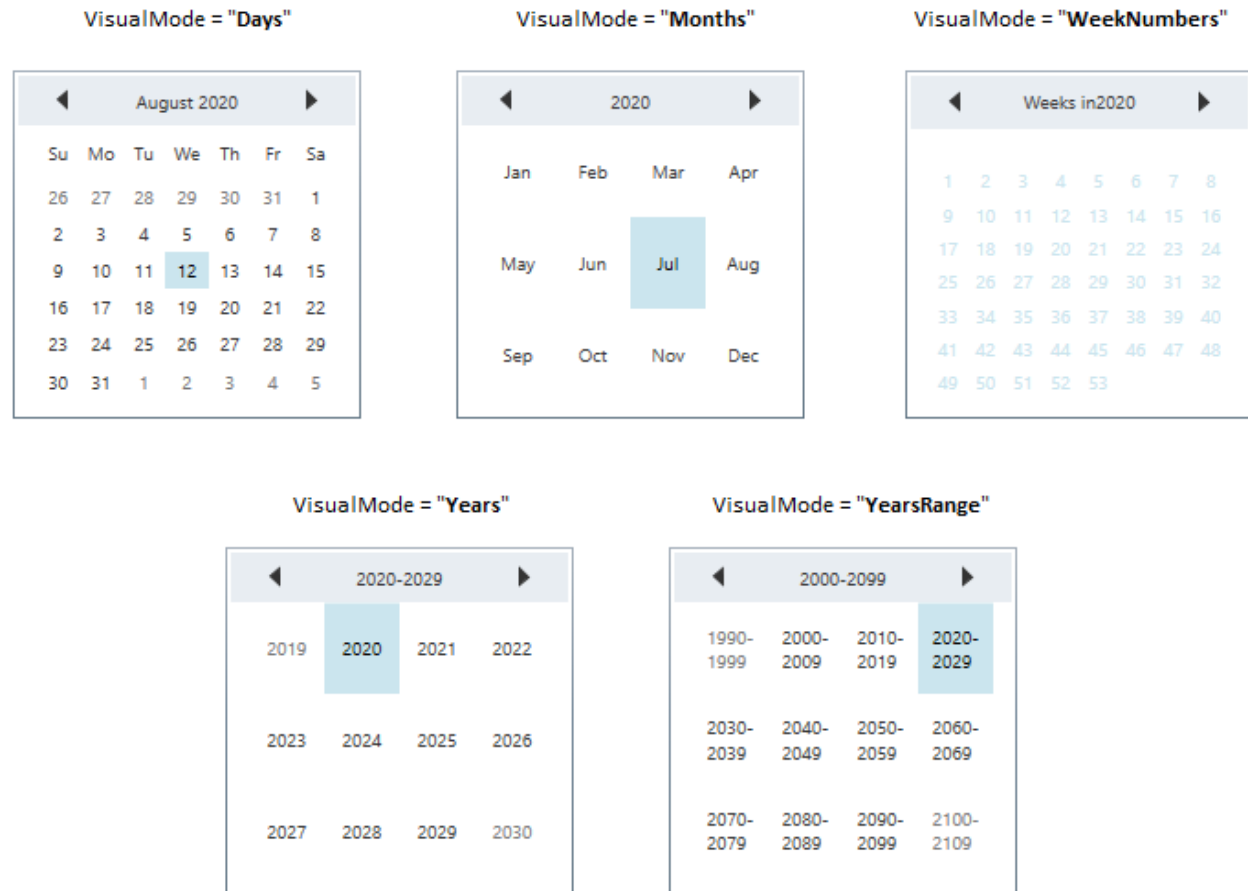
**Note:** [View Sample in GitHub](#)



Change default view (Month, Year, Decade)

By default, the days are displayed in the **CalendarEdit**.

You can change the default calendar view as week numbers, month, years or years range mode by setting the respective value to the [VisualMode](#) property. The default value of **VisualMode** property is **Days**.

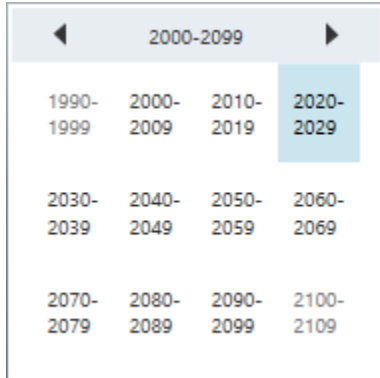


### XML

```
<syncfusion:CalendarEdit Name="calendarEdit"
    VisualMode="YearsRange"/>
```

### C#

```
calendarEdit.VisualMode = CalendarVisualMode.YearsRange;
```



**Note:** [View Sample in GitHub](#)

### Display week numbers

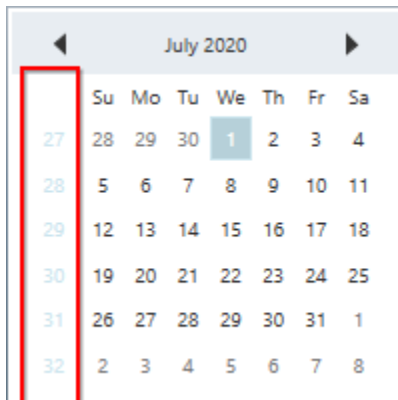
If you want to know the week number of the currently displayed dates, use the [ShowWeekNumbers](#) property as `true`. It will display the respective week numbers in the left side of the `CalendarEdit` control. The default value of `ShowWeekNumbers` property is `false`.

### XML

```
<!--Showing Week number-->
<syncfusion:CalendarEdit Name="calendarEdit"
ShowWeekNumbers="True"/>
```

### C#

```
//Shows week numbers
calendarEdit.ShowWeekNumbers = true;
```



**Note:** [View Sample in GitHub](#)

### Highlight week numbers

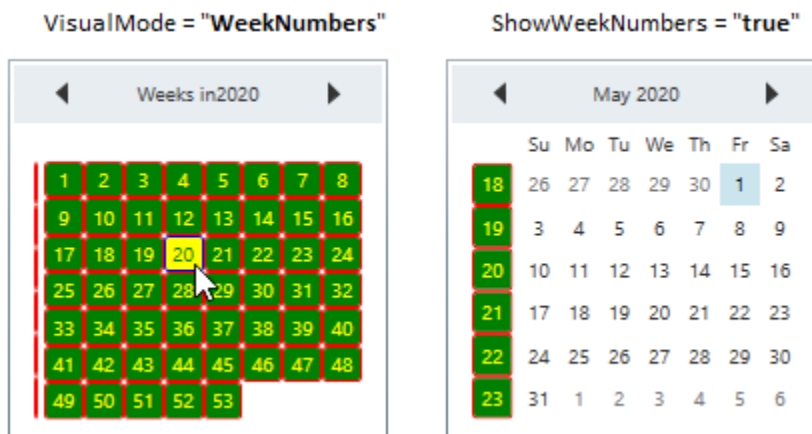
If you want to highlight the week numbers, change its foreground, background or border brush by using the [WeekNumberBackground](#), [WeekNumberForeground](#) and [WeekNumberBorderBrush](#) properties. You can also change the mouse hover background, foreground and border brush for the selected day cell by using the [WeekNumberHoverBackground](#), [WeekNumberHoverForeground](#) and [WeekNumberHoverBorderBrush](#) properties.

### XML

```
<syncfusion:CalendarEdit WeekNumberBackground="Green"
WeekNumberBorderBrush="Red"
WeekNumberForeground="Yellow"
WeekNumberHoverBackground="Yellow"
WeekNumberHoverBorderBrush="Blue"
WeekNumberHoverForeground="Green"
ShowWeekNumbers="True"
VisualMode="WeekNumbers" />
```

**C#**

```
calendarEdit.WeekNumberBackground = Brushes.Green;
calendarEdit.WeekNumberBorderBrush = Brushes.Red;
calendarEdit.WeekNumberForeground = Brushes.Yellow;
calendarEdit.WeekNumberHoverBackground = Brushes.Yellow;
calendarEdit.WeekNumberHoverBorderBrush = Brushes.Blue;
calendarEdit.WeekNumberHoverForeground = Brushes.Green;
calendarEdit.ShowWeekNumbers = true;
calendarEdit.VisualMode = CalendarVisualMode.WeekNumbers;
```



**Note:** [View Sample in GitHub](#)

**Special days**

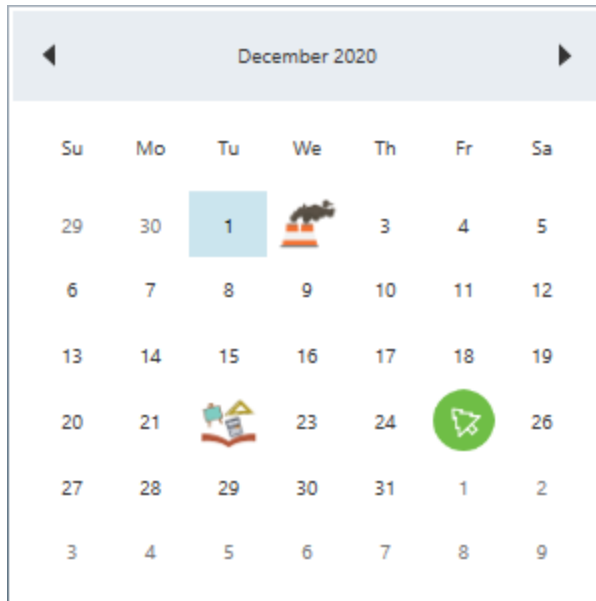
You can differentiate the special day from other days by setting that date value to the [SpecialDate.Date](#) property and adding [SpecialDate.Date](#) into the [SpecialDates](#) collection. You can use the [SpecialDate.CellTemplate](#) property to customize the [SpecialDate](#) day cell appearance.

**C#**

```
//ViewModel.cs
public class ViewModel {
    private SpecialDatesCollection specialDates;
    public SpecialDatesCollection SpecialDates {
        get { return specialDates; }
        set { specialDates = value; }
    }
    public ViewModel() {
        SpecialDates = new SpecialDatesCollection();
    }
}
```

**XML**

```
<Window.Resources>
<DataTemplate x:Key="WorldEnvironmentDay" >
<Image Source="Resources\Icon_Environmental day.png" />
</DataTemplate>
<DataTemplate x:Key="EngineersDay" >
<Image Source="Resources\Icon_Engineer day.png" />
</DataTemplate>
<DataTemplate x:Key="PollutionPreventionDay" >
<Image Source="Resources\Icon_Pollution day.png" />
</DataTemplate>
<DataTemplate x:Key="NationalMathematicsDay" >
<Image Source="Resources\Icon_Mathematics day.png" />
</DataTemplate>
<DataTemplate x:Key="Christmas" >
<Image Source="Resources\Christmas.png" />
</DataTemplate>
<local:ViewModel x:Key="viewModel">
<local:ViewModel.SpecialDates>
<syncfusion:SpecialDate Date="06/05/2020" CellTemplate="{StaticResource
WorldEnvironmentDay }"/>
<syncfusion:SpecialDate Date="09/15/2020" CellTemplate="{StaticResource
EngineersDay }"/>
<syncfusion:SpecialDate Date="12/02/2020" CellTemplate="{StaticResource
PollutionPreventionDay }"/>
<syncfusion:SpecialDate Date="12/22/2020" CellTemplate="{StaticResource
NationalMathematicsDay }"/>
<syncfusion:SpecialDate Date="12/25/2020" CellTemplate="{StaticResource
Christmas }"/>
</local:ViewModel.SpecialDates>
</local:ViewModel>
</Window.Resources>
<Grid>
<syncfusion:CalendarEdit DataContext="{StaticResource viewModel}"
SpecialDates="{Binding SpecialDates}"
Name="calendarEdit" />
</Grid>
```



**Note:** [View Sample in GitHub](#)

### Setting Culture

You can change the culture for `CalendarEdit` control by setting the required culture to the `Culture` property.

### XML

```
<!--Setting french culture-->
<syncfusion:CalendarEdit Name="calendarEdit"
Culture="fr-FR"/>
```

### C#

```
//Setting french culture
calendarEdit.Culture = new CultureInfo("fr-FR");
```



### Show full month and week name

You can display full month names and week day names by setting the [ShowAbbreviatedDayNames](#) and [ShowAbbreviatedMonthNames](#) properties as `false`. The default value of `ShowAbbreviatedDayNames` and `ShowAbbreviatedDayNames` property is `true`.

### XML

```
<syncfusion:CalendarEdit ShowAbbreviatedDayNames="False"
ShowAbbreviatedMonthNames="False"
Name="calendarEdit"/>
```

### C#

```
CalendarEdit calendarEdit = new CalendarEdit();
calendarEdit.ShowAbbreviatedDayNames = false;
calendarEdit.ShowAbbreviatedMonthNames = false;
```

Expanded month and week day names

July 2020						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Abbreviated month and week day names

Jul 2020						
Su	Mo	Tu	We	Th	Fr	Sa
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

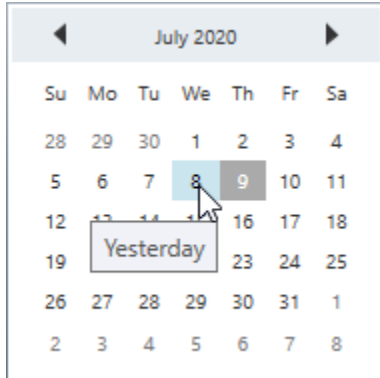
**Note:** [View Sample in GitHub](#)

### Tooltip for particular days

You can set tooltip for particular days in the `CalendarEdit` control by using the [SetToolTip\(Date,ToolTip\)](#) method. You can pass the specific date and tooltip to the `SetToolTip(Date,ToolTip)` method.

### C#

```
Date yesterday = new Date(DateTime.Now.Year, DateTime.Now.Month,
DateTime.Now.Day-1);
Date today = new Date(DateTime.Now.Year, DateTime.Now.Month,
DateTime.Now.Day);
CalendarEdit calendarEdit = new CalendarEdit();
//Setting tooltip for yesterday and today dates
calendarEdit.SetToolTip(yesterday, new ToolTip() { Content = "Yesterday" });
calendarEdit.SetToolTip(today, new ToolTip() { Content = "Today" });
```



### Custom appearance of day cell

You can customize the appearance of day cell by using styles and templates in the `CalendarEdit` control.

#### Custom UI for day cell using style

You can customize the appearance of day cell by using the [DayCellStyle](#) property. The `DataContext` of the `DayCellStyle` is `DayCell`.

#### XML

```
<Window.Resources>
<!-- day cell style -->
<Style x:Key="dayCell"
TargetType="{x:Type syncfusion:DayCell}">
<Setter Property="CornerRadius" Value="0"/>
<Setter Property="Background" Value="Pink"/>
</Style>
</Window.Resources>
<Grid>
<syncfusion:CalendarEdit DayCellStyle="{StaticResource dayCell}"
Name="calendarEdit" />
</Grid>
```



**Note:** [View Sample in GitHub](#)

#### Custom UI for day cell using template

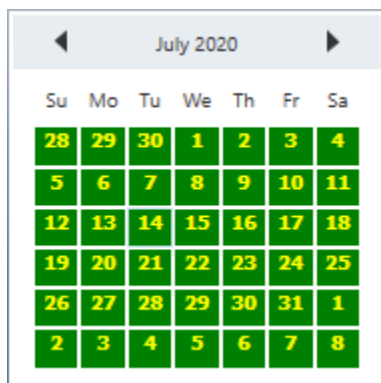
You can customize the appearance of day cell by using the [DayCellsDataTemplate](#) property. The `DataContext` of the `DayCellsDataTemplate` is `DayCell`.

#### XML

```

<Window.Resources>
<!-- day cell template -->
<DataTemplate x:Key="dayCelltemplate"
DataType="syncfusion:DayCell">
<TextBlock TextAlignment="Center"
Foreground="Yellow"
Background="Green"
FontFamily="Tahoma"
FontStyle="Normal"
Text="{Binding Day}"/>
</DataTemplate>
</Window.Resources>
<Grid>
<syncfusion:CalendarEdit DayCellsDataTemplate="{StaticResource
dayCelltemplate}"
Name="calendarEdit" />
</Grid>

```



**Note:** [View Sample in GitHub](#)

Custom UI for day name cell using style

You can customize the appearance of day name cell by using the [DayNameCellStyle](#) property. The `DataContext` of the `DayNameCellStyle` is `DayNameCell`.

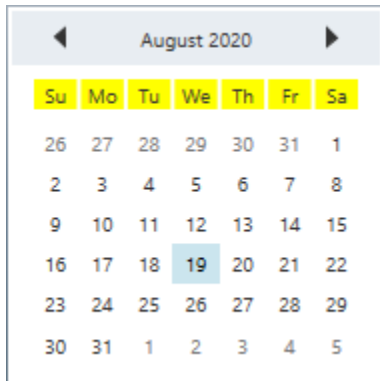
#### XML

```

<Window.Resources>
<!-- day name cell style -->
<Style x:Key="daynameCellStyle"
TargetType="{x:Type syncfusion:DayNameCell}">
<Setter Property="Background" Value="Yellow"/>
<Setter Property="Foreground" Value="Yellow"/>
</Style>
</Window.Resources>
<Grid>
<syncfusion:CalendarEdit DayNameCellStyle="{StaticResource
daynameCellStyle}"
Name="calendarEdit" />
</Grid>

```





**Note:** [View Sample in GitHub](#)

#### Selected date changed notification

The selected date changed in `CalendarEdit` can be examined using `DateChanged` event. The `DateChanged` event contains the old and newly selected date time values in the `OldValue` and `NewValue` properties.

#### XML

```
<syncfusion:CalendarEdit DateChanged="CalendarEdit_DateChanged"
Name="calendarEdit"/>
```

#### C#

```
CalendarEdit calendarEdit = new CalendarEdit();
calendarEdit.DateChanged += CalendarEdit_DateChanged;
```

You can handle the event as follows,

#### C#

```
private void CalendarEdit_DateChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
    //Get old and new selected date values
    var oldValue = e.OldValue;
    var newValue = e.NewValue;
}
```

### Restrict Date Selection in WPF Calendar (CalendarEdit)

This section explains how to restrict a date within a particular range by using `CalendarEdit` control.

Restrict date selection within minimum and maximum date

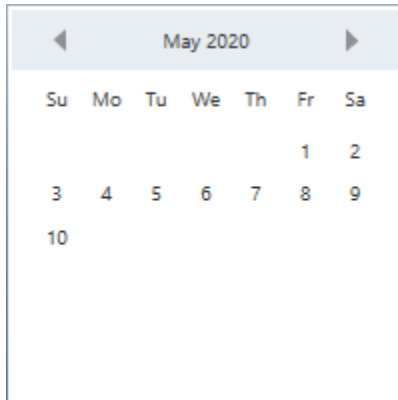
You can restrict the users from selecting a date within the particular range by specifying `MinDate` and `MaxDate` in `CalendarEdit` control.

#### XML

```
<!--Setting date range -->
<syncfusion:CalendarEdit MinDate="05/1/2020"
MaxDate="05/10/2020"
Name="calendarEdit"/>
```

**C#**

```
CalendarEdit calendarEdit = new CalendarEdit();  
calendarEdit.MinDate = new DateTime(2020, 05, 01);  
calendarEdit.MaxDate = new DateTime(2020, 05, 10);
```



**Note:** The `MaxDate` should be greater than `MinDate` of the `CalendarEdit`. If the `MinDate` property is greater than the new `MaxDate`, then the `MinDate` will be reset to the `MaxDate`.

**Note:** [View Sample in GitHub](#)

*Show disabled dates*

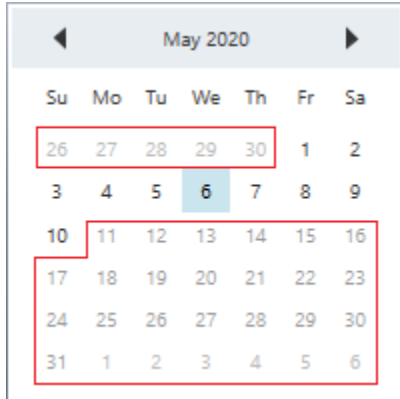
If you want to show the disabled dates which are not within the range of minimum and maximum date limits, use the `MinMaxHidden` property value as `false`. The default value of `MinMaxHidden` property is `true`.

**XML**

```
<syncfusion:CalendarEdit MinMaxHidden="False"  
MinDate="05/1/2020"  
MaxDate="05/10/2020"  
Name="calendarEdit"/>
```

**C#**

```
CalendarEdit calendarEdit = new CalendarEdit();  
calendarEdit.MinDate = new DateTime(2020, 05, 01);  
calendarEdit.MaxDate = new DateTime(2020, 05, 10);  
calendarEdit.MinMaxHidden = false;
```



**Note:** [View Sample in GitHub](#)

### Restrict date selection

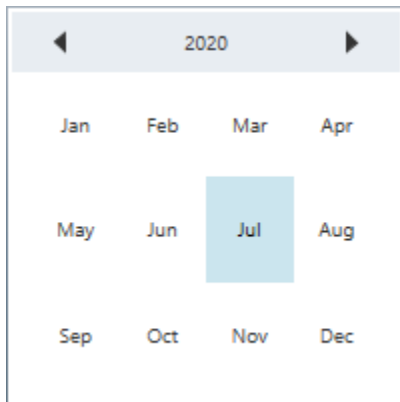
If you want to restrict the user to select the date, use the [DisableDateSelection](#) property value as `true`. However, you can select any month or year. The default value of `DisableDateSelection` property is `false`.

### XML

```
<syncfusion:CalendarEdit Name="calendarEdit"
    DisableDateSelection="true"/>
```

### C#

```
calendarEdit.DisableDateSelection = true;
```



**Note:** [View Sample in GitHub](#)

### Block dates

If you want to block particular dates from the date selection, add that date ranges to the [BlackoutDates](#) collection. You can add more block out date ranges to the `BlackoutDates` collection.

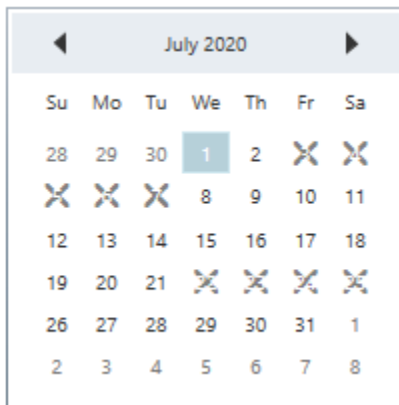
### XML

```
<syncfusion:CalendarEdit Name="calendarEdit">
    <syncfusion:CalendarEdit.BlackoutDates>
        <syncfusion:BlackoutDatesRange StartDate="07/03/2020"
            EndDate="07/07/2020" />
    </syncfusion:CalendarEdit.BlackoutDates>
</syncfusion:CalendarEdit>
```

```
<syncfusion:BlackoutDatesRange StartDate="07/22/2020"
EndDate="07/25/2020" />
</syncfusion:CalendarEdit.BlackoutDates>
</syncfusion:CalendarEdit>
```

**C#**

```
calendarEdit.BlackoutDates.Add(new BlackoutDatesRange() {
StartDate = new DateTime(2020, 07, 03),
EndDate = new DateTime(2020, 07, 07)});
calendarEdit.BlackoutDates.Add(new BlackoutDatesRange() {
StartDate = new DateTime(2020, 07, 22),
EndDate = new DateTime(2020, 07, 25)});
```

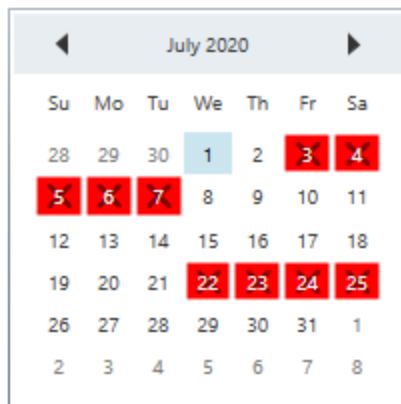


**Note:** [View Sample in GitHub](#)

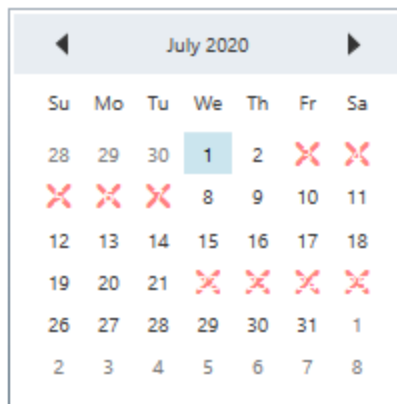
## Custom appearance of blocked days

You can change the background and cross-mark color of the block out days by using the [BlackoutDatesBackground](#) and [BlackoutDatesCrossBrush](#) properties. You can also change the foreground and border color of the block out dates by using the [BlackoutDatesForeground](#) and [BlackoutDatesBorderBrush](#) properties. The default value of [BlackoutDatesBackground](#) is [Transparent](#) and [BlackoutDatesCrossBrush](#) is [Black](#). The default value of [BlackoutDatesForeground](#) is [White](#) and [BlackoutDatesBorderBrush](#) is [Transparent](#).

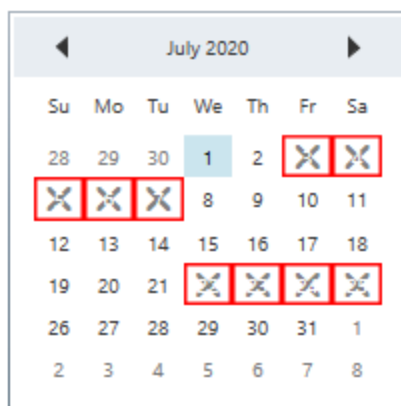
BlackoutDates Background



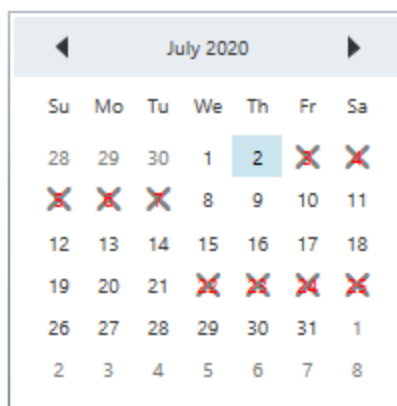
BlackoutDates CrossBrush



BlackoutDates BorderBrush



BlackoutDates Foreground

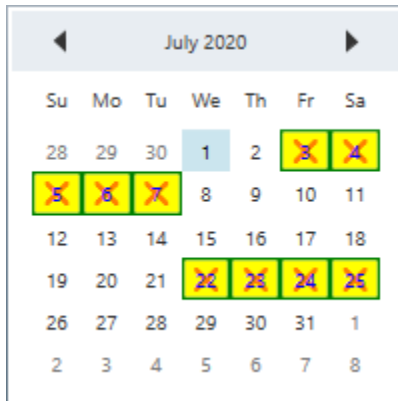
**XML**

```
<syncfusion:CalendarEdit BlackoutDatesBackground="Yellow"
BlackoutDatesBorderBrush="Green"
BlackoutDatesCrossBrush="Red"
BlackoutDatesForeground="Blue"
Name="calendarEdit">
  <syncfusion:CalendarEdit.BlackoutDates>
    <syncfusion:BlackoutDatesRange StartDate="07/03/2020"
    EndDate="07/07/2020" />
    <syncfusion:BlackoutDatesRange StartDate="07/22/2020"
    EndDate="07/25/2020" />
  </syncfusion:CalendarEdit.BlackoutDates>
</syncfusion:CalendarEdit>
```

**C#**

```
calendarEdit.BlackoutDatesBackground = Brushes.Yellow;
calendarEdit.BlackoutDatesBorderBrush = Brushes.Green;
calendarEdit.BlackoutDatesCrossBrush = Brushes.Red;
calendarEdit.BlackoutDatesForeground = Brushes.Blue;
calendarEdit.BlackoutDates.Add(new BlackoutDatesRange() {
    StartDate = new DateTime(2020, 07, 03),
    EndDate = new DateTime(2020, 07, 07)});
```

```
calendarEdit.BlackoutDates.Add(new BlackoutDatesRange() {
    StartDate = new DateTime(2020, 07, 22),
    EndDate = new DateTime(2020, 07, 25)});
```



**Note:** [View Sample in GitHub](#)

Hide previous and next month days

If you want show only the currently displaying month's days without displaying previous and next month's days, use the [ShowPreviousMonthDays](#) and [ShowNextMonthDays](#) property value as `false`. The default value of `ShowPreviousMonthDays` and `ShowNextMonthDays` property is `true`.

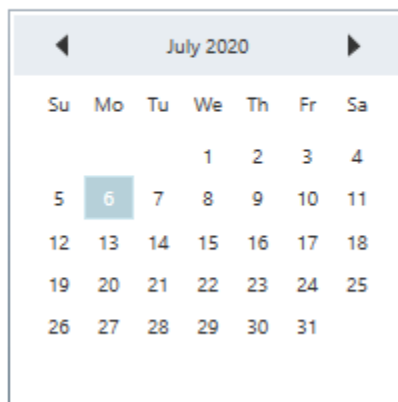
#### XML

```
<syncfusion:CalendarEdit ShowPreviousMonthDays="False"
    ShowNextMonthDays="False"
    Name="calendarEdit" />
```

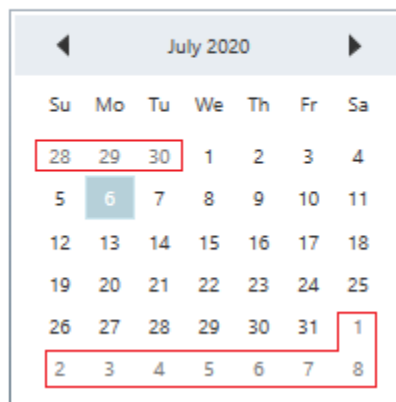
#### C#

```
calendarEdit.ShowNextMonthDays = false;
calendarEdit.ShowPreviousMonthDays = false;
```

Show only current month days



Show current month days  
with previous and next month days



**Note:** [View Sample in GitHub](#)

### Readonly support

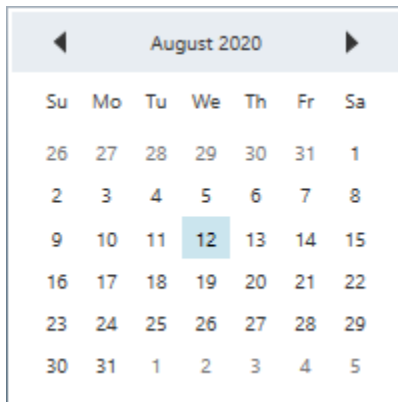
If you want to restrict the user to select the date or month or year, use the [AllowSelection](#) property value as `false`. However, selected date can be changed programmatically in readonly mode and the user can only navigate to any month or year. The default value of `AllowSelection` property is `true`.

### XML

```
<syncfusion:CalendarEdit Date="08/12/2020"
Name="calendarEdit"
AllowSelection="False"/>
```

### C#

```
calendarEdit.AllowSelection = false;
calendarEdit.Date = new DateTime(2020, 08, 12);
```



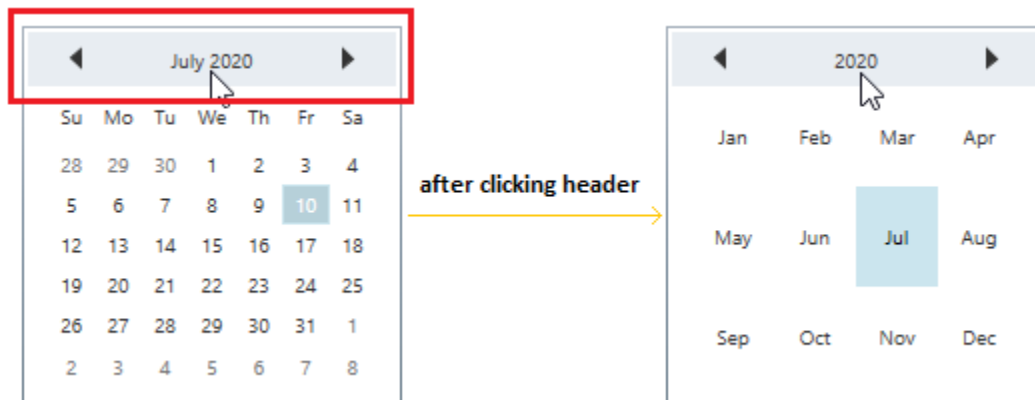
**Note:** [View Sample in GitHub](#)

### Date Navigation in WPF Calendar (CalendarEdit)

This section explains navigation between day, month or year mode in [CalendarEdit](#) control.

Navigate to the day, month or year modes using header

You can easily navigate to the day, month or year or decade modes by clicking the `CalendarEdit` header.

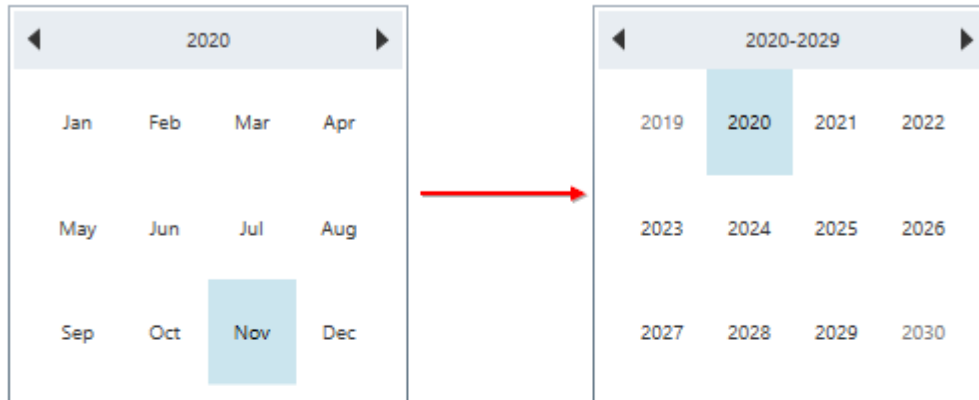


### Navigate to the day, month or year modes using key navigation

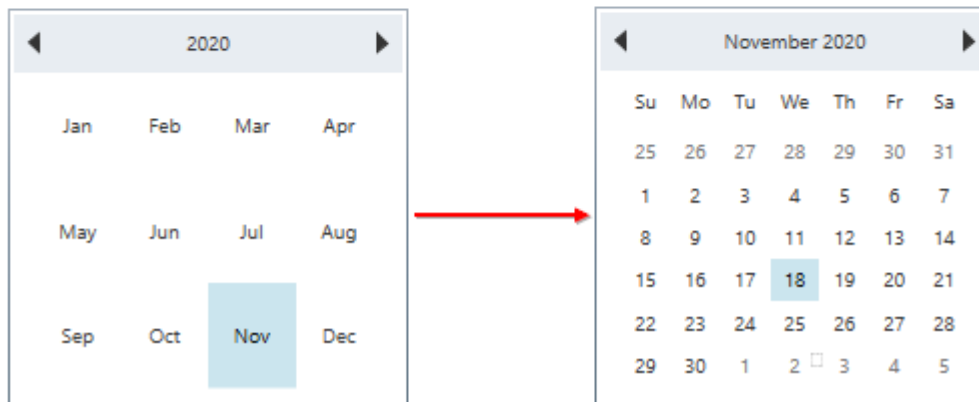
You can easily navigate to the day, month or year or decade modes by pressing the **Alt** with **Up** or **Down** arrow keys. If you will press the **Alt + Up** arrow key combination, it will navigate from dates to month, year and then decade mode. If you will press **Alt + Down** arrow key combination, it will navigate from decade to year, month and then dates mode.

For example, if you will start navigation from month mode, it will be move to next or previous mode as follows,

#### Alt + Up arrow



#### Alt + Down arrow



### Change animation time for calendar mode navigation

If you want to change the animation time for navigate to the day, month or year mode by clicking the **CalendarEdit** header, set value to **ChangeModeTime** property. The default value of **ChangeModeTime** property is 300.

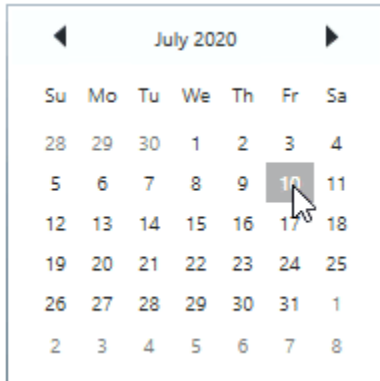
#### XML

```
<syncfusion:CalendarEdit ChangeModeTime="0"
Name="calendarEdit" />
```

#### C#



```
calendarEdit.ChangeModeTime = 0;
```



Here, `ChangeModeTime` property value is 0. So, navigation between day, month or year mode is done without any animation delay.

**Note:** [View Sample in GitHub](#)

#### *Custom appearance of header*

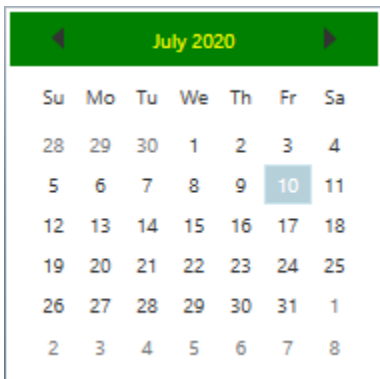
You can change the background and foreground of the `CalendarEdit` header by using the `HeaderBackground` and `HeaderForeground` properties. The default value of `HeaderBackground` is Lavender and `HeaderForeground` is Dark SlateGray.

#### **XML**

```
<syncfusion:CalendarEdit HeaderBackground="Green"
HeaderForeground="Yellow"
Name="calendarEdit" />
```

#### **C#**

```
calendarEdit.HeaderBackground = Brushes.Green;
calendarEdit.HeaderForeground = Brushes.Yellow;
```



**Note:** [View Sample in GitHub](#)

#### Navigate to previous or next month

You can navigate to the previous or next month by clicking on the `Previous-Next` navigation buttons in the header. You can also navigate to the previous or next month by scrolling the mouse or pressing the

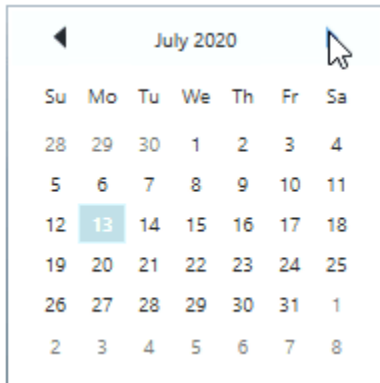
Alt + (Left or Right) key combination. If you want to change the animation time for navigate to the previous or next month, set value to [FrameMovingTime](#) property. The default value of [FrameMovingTime](#) property is 300.

### XML

```
<syncfusion:CalendarEdit FrameMovingTime="0"
Name="calendarEdit" />
```

### C#

```
calendarEdit.FrameMovingTime = 0;
```



Here, [FrameMovingTime](#) property value is 0. So. navigation between next and previous month is done without any animation delay.

**Note:** [View Sample in GitHub](#)

Custom UI for previous and next navigation buttons

You can customize the previous and next navigation buttons by using the [PreviousScrollButtonTemplate](#) an [NextScrollButtonTemplate](#) properties.

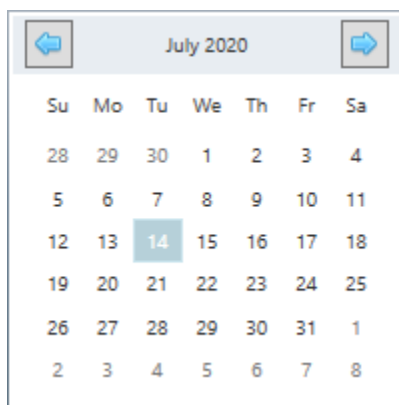
### XML

```
<Window.Resources>
<!--Template for Next Buttons in Calendar Control -->
<ControlTemplate x:Key="nextScrollButtonTemplate">
<Border>
<Button Command="syncfusion:CalendarEdit.NextCommand" >
<Button.Content>
<Image Source="Images/1.png"
Height="16" Width="16"
VerticalAlignment="Center" />
</Button.Content>
</Button>
</Border>
</ControlTemplate>
<!--Template for Previous Buttons in Calendar Control -->
<ControlTemplate x:Key="previousScrollButtonTemplate">
<Border>
<Button Command="syncfusion:CalendarEdit.PrevCommand">
<Button.Content>
```

```

<Image Source="Images/2.png"
Height="16" Width="16"
VerticalAlignment="Center" />
</Button.Content>
</Button>
</Border>
</ControlTemplate>
</Window.Resources>
<Grid>
<syncfusion:CalendarEdit PreviousScrollButtonTemplate="{StaticResource
previousScrollButtonTemplate}"
NextScrollButtonTemplate="{StaticResource nextScrollButtonTemplate}"
Name="calendarEdit"
Width="200" Height="200"/>
</Grid>

```



**Note:** [View Sample in GitHub](#)

### Change navigation direction

You can change previous or next month navigation direction to either **Horizontal** or **Vertical** by using the [MonthChangeDirection](#) property. The default value of **MonthChangeDirection** property is **Horizontal**.

### XML

```

<syncfusion:CalendarEdit MonthChangeDirection="Vertical"
Name="calendarEdit" />

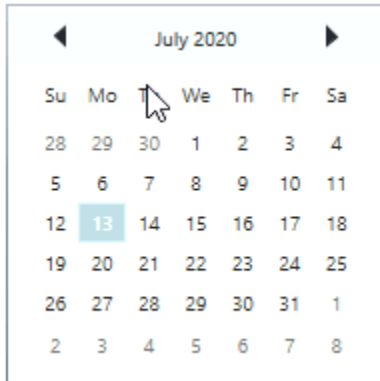
```

### C#

```

calendarEdit.MonthChangeDirection = AnimationDirection.Vertical;

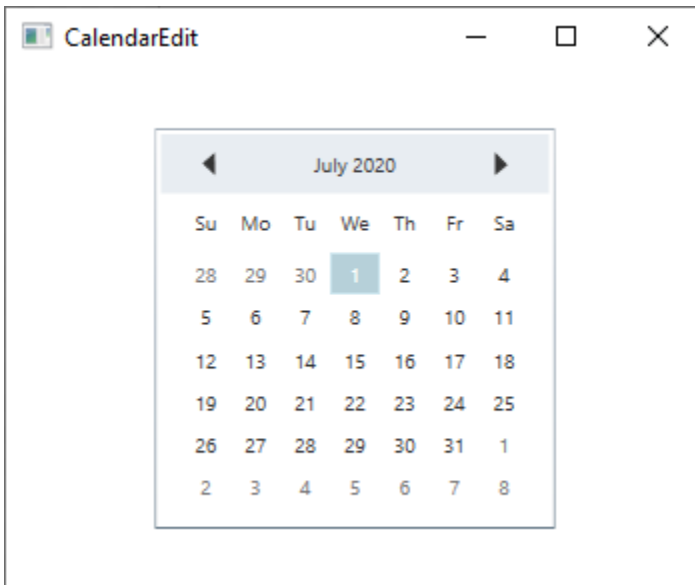
```



**Note:** [View Sample in GitHub](#)

#### Touch support for CalendarEdit

Swipe support is provided for the calendar. When you swipe from right to left over the calendar, it moves to the next month. When you swipe from left to right over the calendar, it moves to the previous month.



**Note:** [View Sample in GitHub](#)

#### Using CalendarEdit Object in WPF Calendar (CalendarEdit)

You can get the object of the CalendarEdit control by using the Calendar property. When you want to see the content after calling the methods, you can store the date in one variable to display or use a MessageBox. The description of each calendar option and code is described as follows.

**Note:** In the following code examples, calendarEdit is used as the instance of the CalendarEdit control.

#### AddDays

Returns the system datetime, that is, the specified number of days away from the specified system datetime. Use MessageBox to see the content of date after this method is called.

#### C#

```
calendarEdit.Calendar.AddDays(calendarEdit.Date, 5);
```

```
MessageBox.Show(calendarEdit.Calendar.AddDays(calendarEdit.Date,  
5).ToString());
```

### *AddHours*

Returns the system datetime, that is, the specified number of hours away from the specified system datetime. Use Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.AddHours(calendarEdit.Date, 2);  
MessageBox.Show(calendarEdit.Calendar.AddHours(calendarEdit.Date,  
2).ToString());
```

### *AddMonths*

Returns the system datetime, that is, the specified number of months away from the specified system datetime. Use Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.AddMonths(calendarEdit.Date, 3);  
MessageBox.Show(calendarEdit.Calendar.AddMonths(calendarEdit.Date,  
3).ToString());
```

### *AddMilliseconds*

Returns the system datetime, that is, the specified number of milliseconds away from the specified system datetime. Use Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.AddMilliseconds(calendarEdit.Date, 200);  
MessageBox.Show(calendarEdit.Calendar.AddMilliseconds(calendarEdit.Date,  
200).ToString());
```

### *AddMinutes*

Returns the system datetime, that is, the specified number of milliseconds away from the specified system datetime. Use Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.AddMinutes(calendarEdit.Date, 5);  
MessageBox.Show(calendarEdit.Calendar.AddMinutes(calendarEdit.Date,  
5).ToString());
```

### *AddSeconds*

Returns the system datetime, that is, specified number of seconds away from the specified system datetime. Use Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.AddSeconds(calendarEdit.Date, 30);  
MessageBox.Show(calendarEdit.Calendar.AddSeconds(calendarEdit.Date,  
30).ToString());
```

### AddWeeks

Returns the system datetime, that is, specified number of weeks away from the specified system datetime. Use Message Box to see the content of date after this method is called.

#### C#

```
calendarEdit.Calendar.AddWeeks(calendarEdit.Date, 2);  
MessageBox.Show(calendarEdit.Calendar.AddWeeks(calendarEdit.Date,  
2).ToString());
```

### AddYears

Returns the system datetime, that is, the specified number of years away from the specified system datetime. Use Message Box to see the content of date after this method is called.

#### C#

```
calendarEdit.Calendar.AddYears(calendarEdit.Date, 1);  
MessageBox.Show(calendarEdit.Calendar.AddYears(calendarEdit.Date,  
1).ToString());
```

### GetDayOfMonth

Returns the day of the month, in the specified System datetime. You can use a Message Box to see the content of date after this method is called.

#### C#

```
calendarEdit.Calendar.GetDayOfMonth(calendarEdit.Date);  
MessageBox.Show(calendarEdit.Calendar.GetDayOfMonth(calendarEdit.Date).ToString());
```

### GetDayOfWeek

Returns the day of the week, in the specified System datetime. You can use a Message Box to see the content of date after this method is called.

#### C#

```
calendarEdit.Calendar.GetDayOfWeek(calendarEdit.Date);  
MessageBox.Show(calendarEdit.Calendar.GetDayOfWeek(calendarEdit.Date).ToString());
```

### GetDayOfYear

Returns the day of the year, in the specified System datetime. You can use a Message Box to see the content of date after this method is called.

#### C#

```
calendarEdit.Calendar.GetDayOfYear(calendarEdit.Date);  
MessageBox.Show(calendarEdit.Calendar.GetDayOfYear(calendarEdit.Date).ToString());
```

### GetDayOfYear

Returns the number of days, in the specified month and year of the current era. You can use a Message Box to see the content of date after this method is called.

**C#**

```
calendarEdit.Calendar.GetDaysInMonth(2009, 2);  
MessageBox.Show(calendarEdit.Calendar.GetDaysInMonth(2009, 2).ToString());
```

*GetDaysInYear*

Returns the number of days, in the specified year of the current era. You can use a Message Box to see the content of date after this method is called.

**C#**

```
calendarEdit.Calendar.GetDaysInYear(2009);  
MessageBox.Show(calendarEdit.Calendar.GetDaysInYear(2009).ToString());
```

*GetEra*

Returns the era, in the specified date time. You can use a Message Box to see the content of date after this method is called.

**C#**

```
calendarEdit.Calendar.GetEra(calendarEdit.Date);  
MessageBox.Show(calendarEdit.Calendar.GetEra(calendarEdit.Date).ToString());
```

*GetHour*

Returns the hour, in the specified date time. You can use a Message Box to see the content of date after this method is called.

**C#**

```
calendarEdit.Calendar.GetHour(calendarEdit.Date);  
MessageBox.Show(calendarEdit.Calendar.GetHour(calendarEdit.Date).ToString());  
;
```

*GetLeapMonth*

Returns the leap month, in the specified year. You can use a Message Box to see the content of date after this method is called.

**C#**

```
calendarEdit.Calendar.GetLeapMonth(2009);  
MessageBox.Show(calendarEdit.Calendar.GetLeapMonth(2009).ToString());
```

*GetMilliseconds*

Returns the milliseconds, in the specified date time. You can use a Message Box to see the content of date after this method is called.

**C#**

```
calendarEdit.Calendar.GetMilliseconds(calendarEdit.Date);  
MessageBox.Show(calendarEdit.Calendar.GetMilliseconds(calendarEdit.Date).ToString());
```

### *GetMinute*

Returns the minute, in the specified datetime. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.GetMinute(calendarEdit.Date);  
MessageBox.Show(calendarEdit.Calendar.GetMinute(calendarEdit.Date).ToString()  
);
```

### *GetMonth*

Returns the month, in the specified datetime. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.GetMonth(calendarEdit.Date);  
MessageBox.Show(calendarEdit.Calendar.GetMonth(calendarEdit.Date).ToString()  
);
```

### *GetMonthsInYear*

Returns the month in the specified year in the current era. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.GetMonthsInYear(2009);  
MessageBox.Show(calendarEdit.Calendar.GetMonthsInYear(2009).ToString());
```

### *GetSecond*

Returns the seconds, in the specified date time. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.GetSecond(calendarEdit.Date);  
MessageBox.Show(calendarEdit.Calendar.GetSecond(calendarEdit.Date).ToString()  
);
```

### *GetWeekOfYear*

Returns the week of the year that includes the date in the specified date time. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.GetWeekOfYear(calendarEdit.Date,  
System.Globalization.CalendarWeekRule.FirstDay, DayOfWeek.Friday);  
MessageBox.Show(calendarEdit.Calendar.GetWeekOfYear(calendarEdit.Date,  
System.Globalization.CalendarWeekRule.FirstDay,  
DayOfWeek.Friday).ToString());
```



### *GetYear*

Returns the week of the year that includes the date in the specified date time. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.GetYear(calendarEdit.Date);  
MessageBox.Show(calendarEdit.Calendar.GetYear(calendarEdit.Date).ToString());  
;
```

### *IsLeapDay*

Determines whether the specified date in the current era is a leap day. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.IsLeapDay(2009, 2, 2);  
MessageBox.Show(calendarEdit.Calendar.IsLeapDay(2009, 2, 2).ToString());
```

### *IsLeapMonth*

Determines whether the specified month, in the specified year, in the current era, is a leap month. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.IsLeapMonth(2009, 3);  
MessageBox.Show(calendarEdit.Calendar.IsLeapMonth(2009, 3).ToString());
```

### *IsLeapYear*

Determines whether the specified year, in the current era, is a leap year. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.IsLeapYear(2009);  
MessageBox.Show(calendarEdit.Calendar.IsLeapYear(2009).ToString());
```

### *MaxSupportedDateTime*

Gets the latest date and time, supported by the calendar object. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.MaxSupportedDateTime;  
MessageBox.Show(calendarEdit.Calendar.MaxSupportedDateTime.ToString());
```

### *MinSupportedDateTime*

Gets the earliest date and time, supported by the calendar object. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.MinSupportedDateTime;  
MessageBox.Show(calendarEdit.Calendar.MinSupportedDateTime.ToString());
```

### *TwoDigitYearMax*

Gets or sets the last year of a 100-year range that can be represented as a 2 digit year. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.TwoDigitYearMax;  
MessageBox.Show(calendarEdit.Calendar.TwoDigitYearMax.ToString());
```

### *ToDateTime*

Returns the datetime that is set to the specified date and time in the current era. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.ToDateTime(2009, 4, 5, 2, 30, 5, 200);  
MessageBox.Show(calendarEdit.Calendar.ToDateTime(2009, 4, 5, 2, 30, 5,  
200).ToString());
```

### *ToFourDigitYear*

Convert specified year to a 4 digit year. You can use a Message Box to see the content of date after this method is called.

#### **C#**

```
calendarEdit.Calendar.ToFourDigitYear(2009);  
MessageBox.Show(calendarEdit.Calendar.ToFourDigitYear(2009).ToString());
```

## Appearance in WPF Calendar (CalendarEdit)

This section explains different styling, theming options available in [CalendarEdit](#) control.

### Setting the foreground

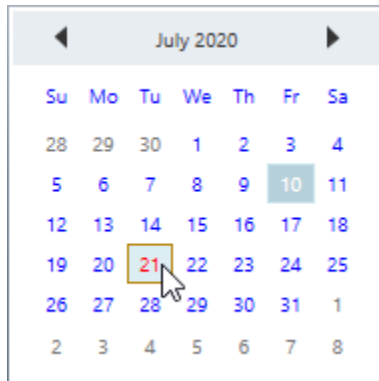
You can change the default foreground, mouse hover foreground and border brush of the **CalendarEdit** by using the **Foreground**, **MouseOverForeground** and **MouseOverBorderBrush** properties. The default value of **Foreground** is **Dark SlateGray** and **MouseOverForeground** is **Black**.

#### **XML**

```
<syncfusion:CalendarEdit Foreground="Blue"  
MouseOverForeground="Red"  
MouseOverBorderBrush="DarkGoldenrod"  
Name="calendarEdit" />
```

#### **C#**

```
calendarEdit.Foreground = Brushes.Blue;  
calendarEdit.MouseOverForeground = Brushes.Red;  
calendarEdit.MouseOverBorderBrush = Brushes.DarkGoldenrod;
```



**Note:** [View Sample in GitHub](#)

### Setting the background

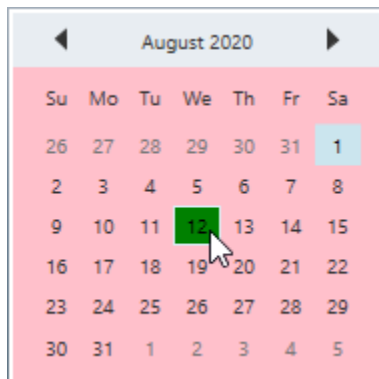
You can change the default background and mouse hover background color of the `CalendarEdit` by using the `Background` and `MouseOverBackground` properties. The default value of `Background` is White and `MouseOverBackground` is Lavender.

### XML

```
<syncfusion:CalendarEdit Background="Pink"
    MouseOverBackground="Green"
    Name="calendarEdit" />
```

### C#

```
calendarEdit.Background = Brushes.Pink;
calendarEdit.MouseOverBackground = Brushes.Green;
```



**Note:** [View Sample in GitHub](#)

### Change flow direction

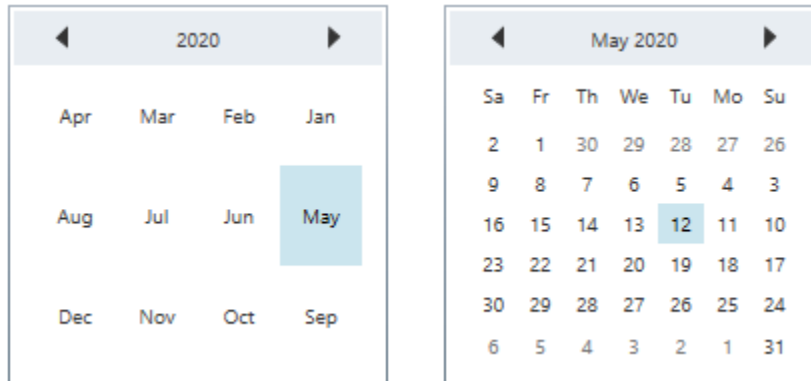
You can change the flow direction of the `CalendarEdit` layout from right to left by setting the `FlowDirection` property value as `RightToLeft`. The default value of `FlowDirection` property is `LeftToRight`.

### XML

```
<syncfusion:CalendarEdit FlowDirection="RightToLeft"
    Name="calendarEdit" />
```

**C#**

```
calendarEdit.FlowDirection = FlowDirection.RightToLeft;
```

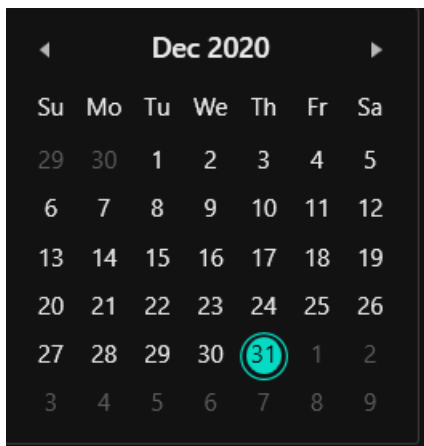


**Note:** [View Sample in GitHub](#)

## Theme

CalendarEdit supports various built-in themes. Refer to the below links to apply themes for the CalendarEdit,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

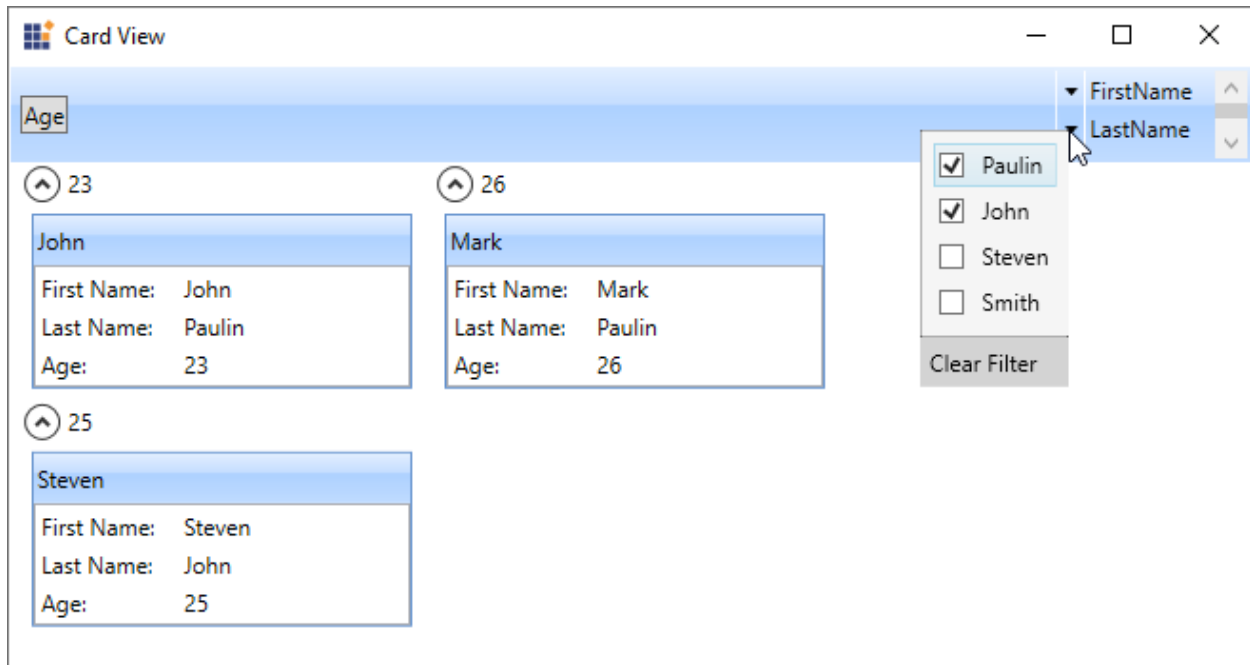


## CardView

## WPF Card View Overview

The WPF [CardView](#) control is a panel that helps organize a list of items in cards. It supports grouping, sorting, filtering, and editing options. Also, supports listing the grouped items in a tree structure.

## Control Structure



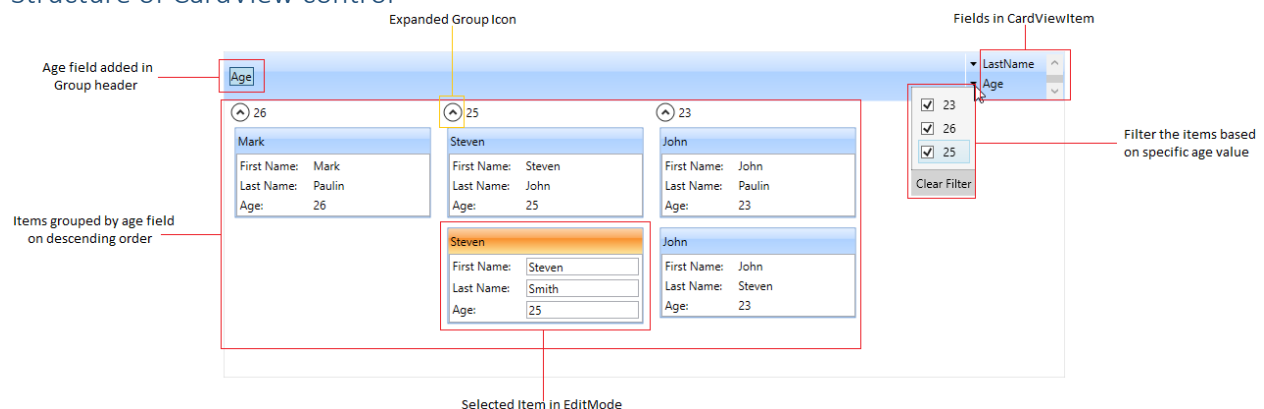
## Key features

- Editing - Allows to editing the fields in the GroupViewItems.
- Grouping - Allows to Grouping between the CardViewItems.
- Filtering - Allows to filtering based on the field's values in the GroupViewItems.
- Sorting - Allows Sorting between the CardViewItems.
- Header - Shows or hides the CardView Header panel.
- Custom UI - Allows to customize the CardViewItems.

## Getting Started with WPF Card View

This section describes how to create a [CardView](#) control in a WPF application and overview of its basic functionalities.

## Structure of CardView control



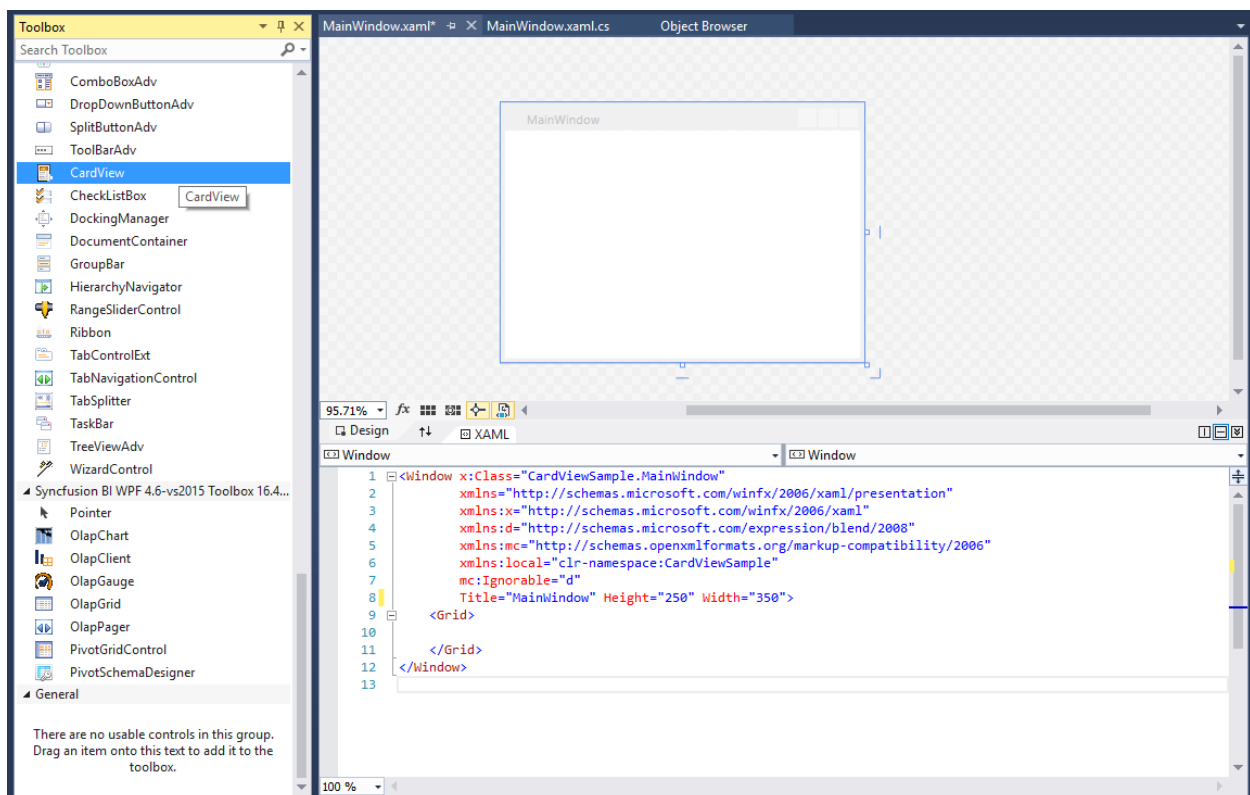
### Assembly deployment

Refer to the [Control Dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

Refer to the [NuGet Packages Documentation](#) to find more details about installing nuget packages in a WPF application.

### Adding WPF CardView control via designer

1. The **CardView** control can be added to an application by dragging it from the toolbox to a designer view. The following dependent assemblies will be added automatically:
  - Syncfusion.Shared.WPF
  - Syncfusion.Tools.WPF



2. Set the properties for **CardView** control in design mode using the SmartTag feature.

### Adding WPF CardView control via XAML

To add the **CardView** control manually in XAML, follow these steps:

1. Create a new WPF project in Visual Studio.
2. Add the following required assembly references to the project:
  - Syncfusion.Shared.WPF
  - Syncfusion.Tools.WPF
3. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf>, and declare the **CardView** control in XAML page.

## XML

```
<Window x:Class="CardViewSample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:CardViewSample"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="Card View" Height="450" Width="800">
<Grid Name="grid">
<syncfusion:CardView Name="cardView"/>
</Grid>
</Window>
```

### Adding WPF CardView control via C#

To add the **CardView** control manually in C#, follow these steps:

1. Create a new WPF application via Visual Studio.
2. Add the following required assembly references to the project:
  - Syncfusion.Shared.WPF
  - Syncfusion.Tools.WPF 3) Include the required namespace.

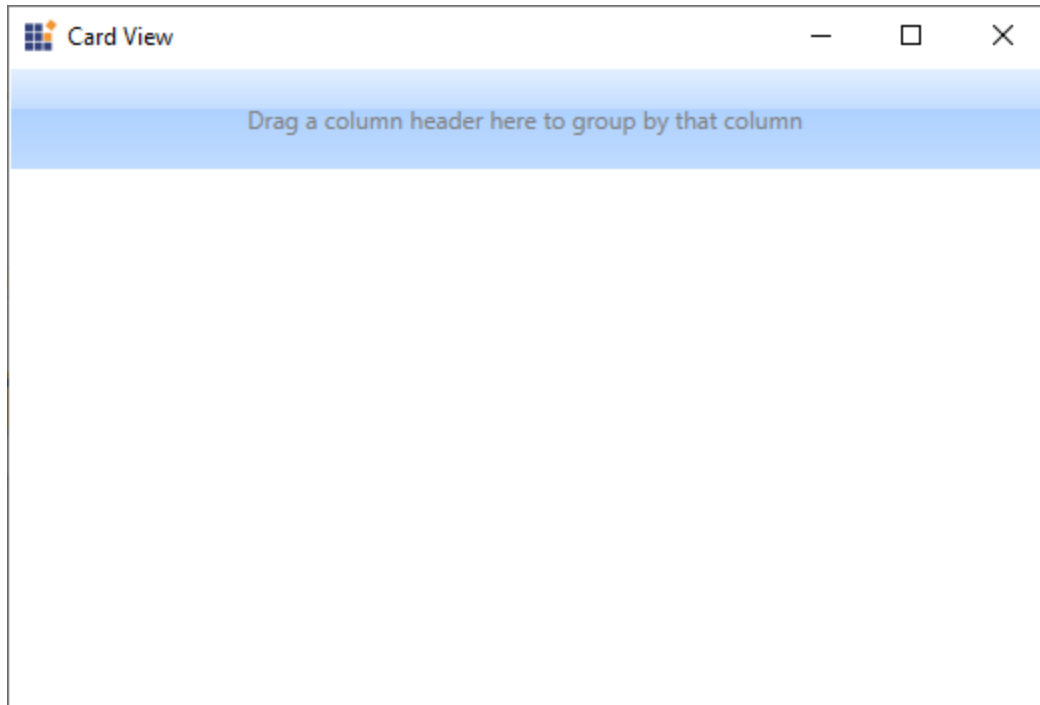
## C#

```
using Syncfusion.Windows.Tools.Controls;
```

- 4) Create an instance of **CardView** control, and add it to the window.

## C#

```
namespace CardViewSample
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            //Creating an instance of CardView control
            CardView cardView = new CardView ();
            //Adding CardView as window content
            this.Content = cardView;
        }
    }
}
```



**Note:** [View Sample in GitHub](#)

Populating items using CardViewItem

You can add the card items inside the control by adding the [CardViewItem](#) into the `CardView.Items` collection property.

#### XML

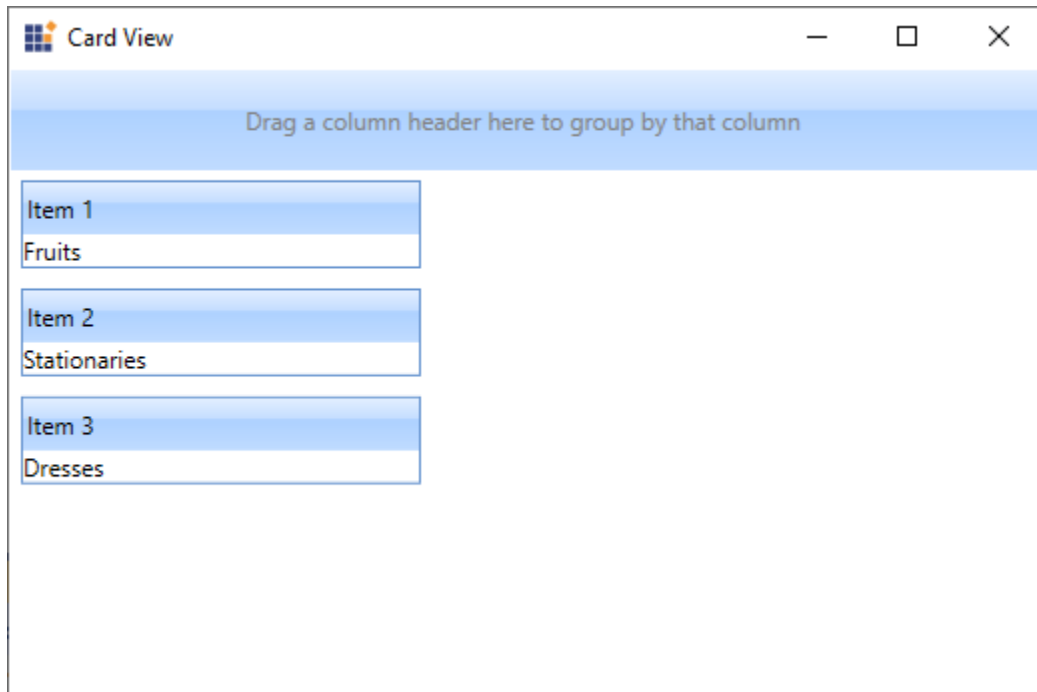
```
<syncfusion:CardView Name="cardView">
  <syncfusion:CardViewItem Header="Item 1">
    <TextBlock Text="Fruits"/>
  </syncfusion:CardViewItem>
  <syncfusion:CardViewItem Header="Item 2">
    <TextBlock Text="Stationaries"/>
  </syncfusion:CardViewItem>
  <syncfusion:CardViewItem Header="Item 3">
    <TextBlock Text="Dresses"/>
  </syncfusion:CardViewItem>
</syncfusion:CardView>
```

#### C#

```
CardViewItem cardViewItem1 = new CardViewItem()
{
    Header = "Item 1",
    Content = new TextBlock() { Text = "Fruits" }
};
CardViewItem cardViewItem2 = new CardViewItem()
{
    Header = "Item 2",
    Content = new TextBlock() { Text = "Stationaries" }
};
```



```
CardViewItem cardViewItem3 = new CardViewItem()
{
    Header = "Item 3",
    Content = new TextBlock() { Text = "Dresses" }
};
CardView cardView= new CardView();
cardView.Items.Add(cardViewItem1);
cardView.Items.Add(cardViewItem2);
cardView.Items.Add(cardViewItem3);
```



**Note:** [View Sample in GitHub](#)

### Populating items using ItemsSource

You can populate the card items to the **CardView** control by using the **ItemsSource** property. You need to use [HeaderTemplate](#) and [ItemTemplate](#) to populate the items into the view.

**Note:** You can use the grouping, sorting, filtering and editing functionalities only by populating the card items through the **ItemsSource** property.

### C#

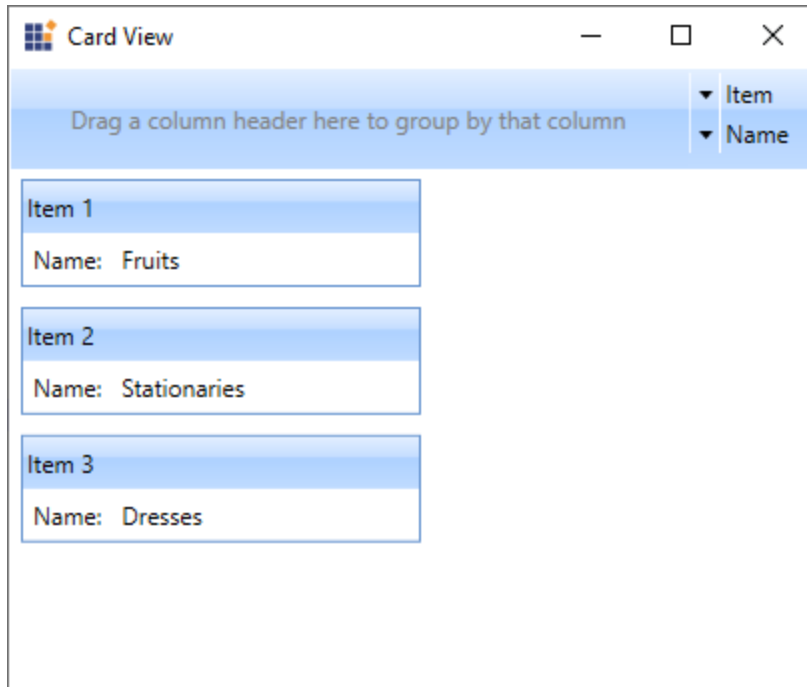
```
//Model.cs
public class CardViewModel
{
    public string Item { get; set; }
    public string Name { get; set; }
}

//ViewModel.cs
public class ViewModel : NotificationObject
{
    private ObservableCollection<CardViewModel> cardViewItems;
    public ObservableCollection<CardViewModel> CardViewItems
```

```
{
get { return cardViewItems; }
set { cardViewItems = value;
this.RaisePropertyChanged(nameof(CardViewItems)); }
}
public ViewModel()
{
CardViewItems = new ObservableCollection<CardViewModel>();
populateItems();
}
private void populateItems()
{
CardViewItems.Add(new CardViewModel() { Item="Item 1", Name = "Fruits" });
CardViewItems.Add(new CardViewModel() { Item="Item 2", Name = "Stationaries"
});
CardViewItems.Add(new CardViewModel() { Item = "Item 3", Name = "Dresses"
});
}
}
```

## XML

```
<syncfusion:CardView ItemsSource="{Binding CardViewItems}"
Name="cardView">
<syncfusion:CardView.HeaderTemplate>
<DataTemplate>
<TextBlock Text="{Binding Item}"/>
</DataTemplate>
</syncfusion:CardView.HeaderTemplate>
<syncfusion:CardView.ItemTemplate>
<DataTemplate>
<StackPanel Orientation="Horizontal">
<TextBlock Text="Name:"
Margin="5" />
<TextBlock Grid.Column="1"
Margin="5"
Text="{Binding Name, UpdateSourceTrigger=PropertyChanged}" />
</StackPanel>
</DataTemplate>
</syncfusion:CardView.ItemTemplate>
</syncfusion:CardView>
```



**Note:** [View Sample in GitHub](#)

#### Select a CardViewItem

You can select any card item by mouse click on the specific item. You can get the selected item by using the [SelectedItem](#) property. The default value of `SelectedItem` property is `null`.

**Note:** You can select only one item at a time.

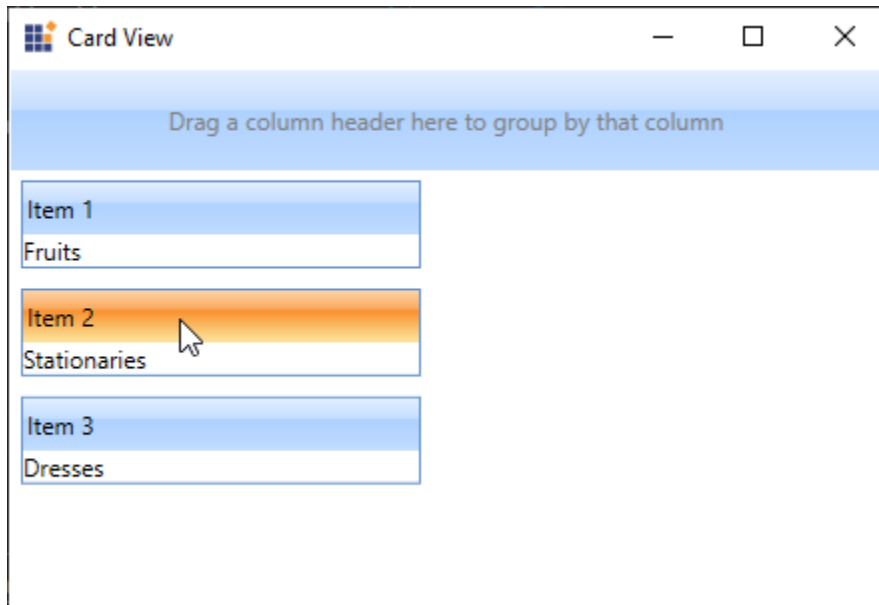
#### XML

```
<syncfusion:CardView Name="cardView">
  <syncfusion:CardViewItem Header="Item 1">
    <TextBlock Text="Fruits"/>
  </syncfusion:CardViewItem>
  <syncfusion:CardViewItem Header="Item 2">
    <TextBlock Text="Stationaries"/>
  </syncfusion:CardViewItem>
  <syncfusion:CardViewItem Header="Item 3">
    <TextBlock Text="Dresses"/>
  </syncfusion:CardViewItem>
</syncfusion:CardView>
```

#### C#

```
CardViewItem cardViewItem1 = new CardViewItem()
{
    Header = "Item 1",
    Content = new TextBlock() { Text = "Fruits" };
};
CardViewItem cardViewItem2 = new CardViewItem()
{
    Header = "Item 2",
    Content = new TextBlock() { Text = "Stationaries" };
};
```

```
};  
CardViewItem cardViewItem3 = new CardViewItem()  
{  
    Header = "Item 3",  
    Content = new TextBlock() { Text = "Dresses" }  
};  
CardView cardView= new CardView();  
cardView.Items.Add(cardViewItem1);  
cardView.Items.Add(cardViewItem2);  
cardView.Items.Add(cardViewItem3);
```



**Note:** [View Sample in GitHub](#)

Select CardViewItem programmatically using property

You can select a particular card item programmatically by using the [CardViewItem.IsSelected](#) property.

The default value of CardViewItem.IsSelected property is false.

### XML

```
<syncfusion:CardView Name="cardView">  
    <syncfusion:CardViewItem Header="Item 1" >  
        <TextBlock Text="Fruits"/>  
    </syncfusion:CardViewItem>  
    <syncfusion:CardViewItem Header="Item 2"  
        IsSelected="True">  
        <TextBlock Text="Stationaries"/>  
    </syncfusion:CardViewItem>  
    <syncfusion:CardViewItem Header="Item 3">  
        <TextBlock Text="Dresses"/>  
    </syncfusion:CardViewItem>  
</syncfusion:CardView>
```

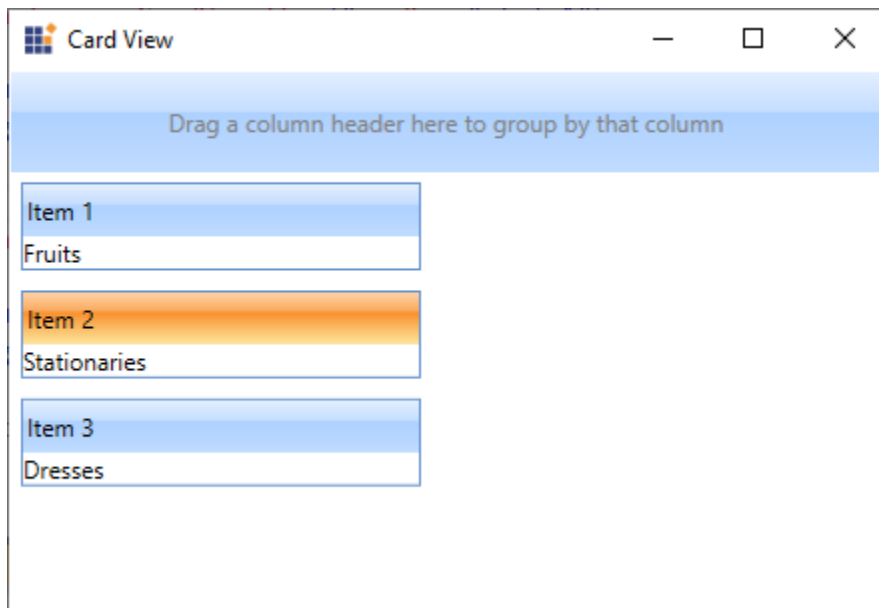
### C#

```
CardViewItem cardViewItem1 = new CardViewItem()
```

```

{
    Header = "Item 1",
    Content = new TextBlock() { Text = "Fruits" }
};
CardViewItem cardViewItem2 = new CardViewItem()
{
    Header = "Item 2",
    IsSelected = true,
    Content = new TextBlock() { Text = "Stationaries" }
};
CardViewItem cardViewItem3 = new CardViewItem()
{
    Header = "Item 3",
    Content = new TextBlock() { Text = "Dresses" }
};
CardView cardView= new CardView();
cardView.Items.Add(cardViewItem1);
cardView.Items.Add(cardViewItem2);
cardView.Items.Add(cardViewItem3);

```



**Note:** [View Sample in GitHub](#)

### Group the CardViewItems

You can group the cards inside the **CardView** control by dragging the required fields from the list and drop it into the dropping region of the **CardView** control header. If you want to disable the grouping, use the [CanGroup](#) property value as **false**.

### C#

```

//Model.cs
public class CardViewModel
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}

```

```

}
//ViewModel.cs
public class ViewModel : NotificationObject
{
    private ObservableCollection<CardViewModel> cardViewItems;
    public ObservableCollection<CardViewModel> CardViewItems
    {
        get { return cardViewItems; }
        set { cardViewItems = value;
            this.RaisePropertyChanged(nameof(CardViewItems)); }
    }
    public ViewModel()
    {
        CardViewItems = new ObservableCollection<CardViewModel>();
        populateItems();
    }
    private void populateItems()
    {
        CardViewItems.Add(new CardViewModel() { FirstName = "John", LastName =
            "Paulin", Age = 23 });
        CardViewItems.Add(new CardViewModel() { FirstName = "Mark", LastName =
            "Paulin", Age = 26 });
        CardViewItems.Add(new CardViewModel() { FirstName = "Steven", LastName =
            "John", Age = 25 });
        CardViewItems.Add(new CardViewModel() { FirstName = "John", LastName =
            "Steven", Age = 23 });
        CardViewItems.Add(new CardViewModel() { FirstName = "Steven", LastName =
            "Smith", Age = 25 });
    }
}

```

## XML

```

<syncfusion:CardView CanGroup="True"
ItemsSource="{Binding CardViewItems}"
Name="cardView">
<syncfusion:CardView.HeaderTemplate>
<DataTemplate>
<TextBlock Text="{Binding FirstName}"/>
</DataTemplate>
</syncfusion:CardView.HeaderTemplate>
<syncfusion:CardView.ItemTemplate>
<DataTemplate >
<ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBlock Text="First Name:" />
<TextBlock Grid.Column="1"
Text="{Binding FirstName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>

```

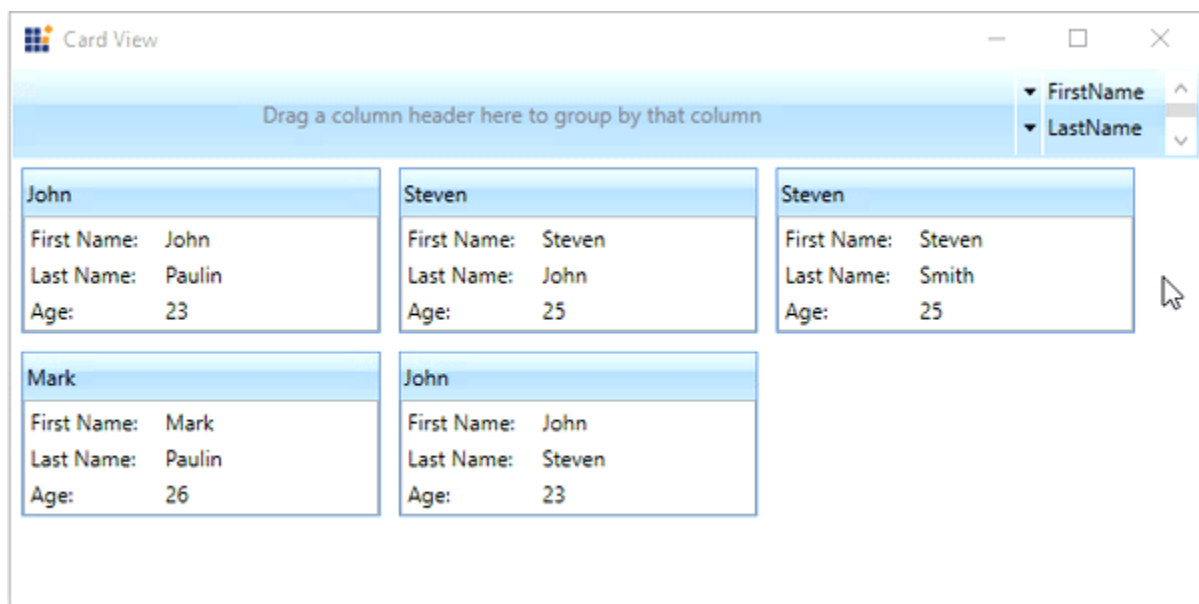
```

<ListBoxItem Padding="1">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="75" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <TextBlock Text="Last Name:" />
    <TextBlock Grid.Column="1"
      Text="{Binding LastName,
        UpdateSourceTrigger=PropertyChanged}" />
    </Grid>
  </ListBoxItem>
  <ListBoxItem Padding="1">
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="75" />
        <ColumnDefinition Width="*" />
      </Grid.ColumnDefinitions>
      <TextBlock Text="Age:" />
      <TextBlock Grid.Column="1"
        Text="{Binding Age,
          UpdateSourceTrigger=PropertyChanged}" />
      </Grid>
    </ListBoxItem>
  </ListBox>
</DataTemplate>
</syncfusion:CardView.ItemTemplate>
</syncfusion:CardView>

```

## C#

```
cardView.CanGroup = true;
```



Here, `CardViewItems` grouped based on `Age` field.

**Note:** [View Sample in GitHub](#)

### Sort the CardViewItems

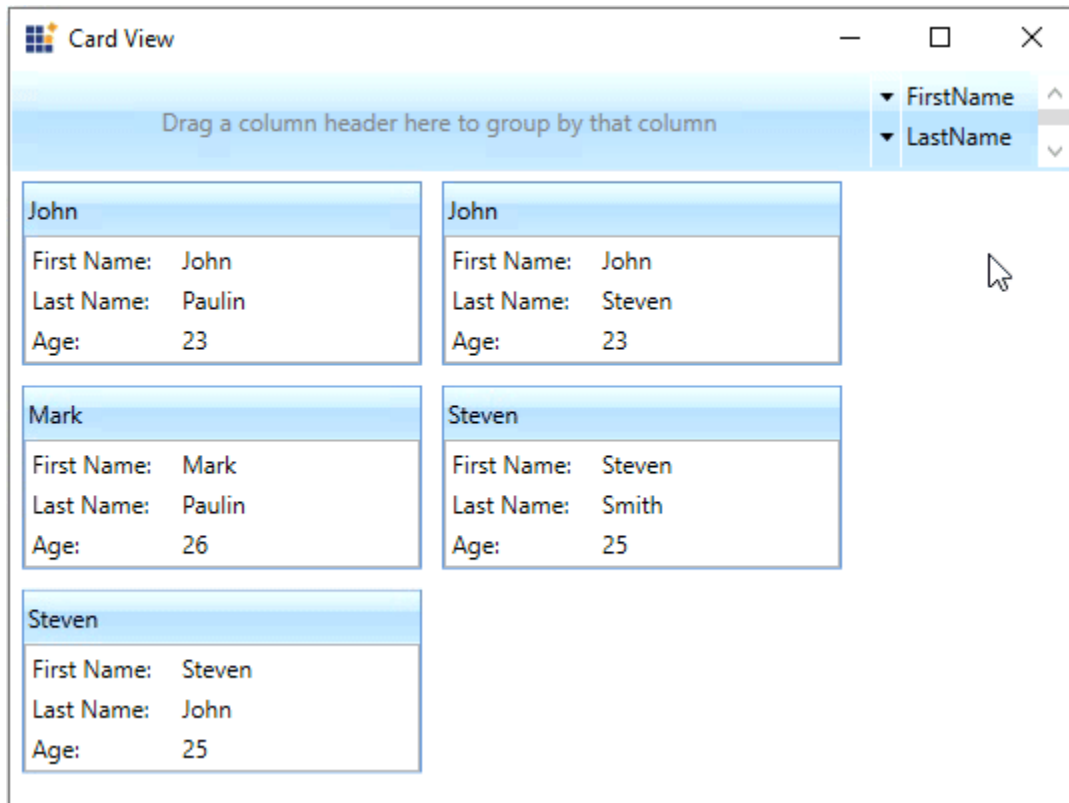
You can sort the cards inside the **CardView** control either default, ascending or descending order by clicking the field names listed in the header. If you want to disable the sorting, use the [CanSort](#) property value as **false**.

### XML

```
<syncfusion:CardView CanSort="True"
ItemsSource="{Binding CardViewItems}"
Name="cardView"/>
```

### C#

```
cardView.CanSort = true;
```



Here, **CardViewItems** sorted based on **FirstName** field.

**Note:** [View Sample in GitHub](#)

### Edit the CardViewItems

You can edit the selected **CardViewItem** value by double-clicking on that item or by pressing the **F2** key. To get out from the editing mode, you need to press the **Esc** or **Enter** key. You can enable the editing mode of card items by setting the [CanEdit](#) property value as **true**.



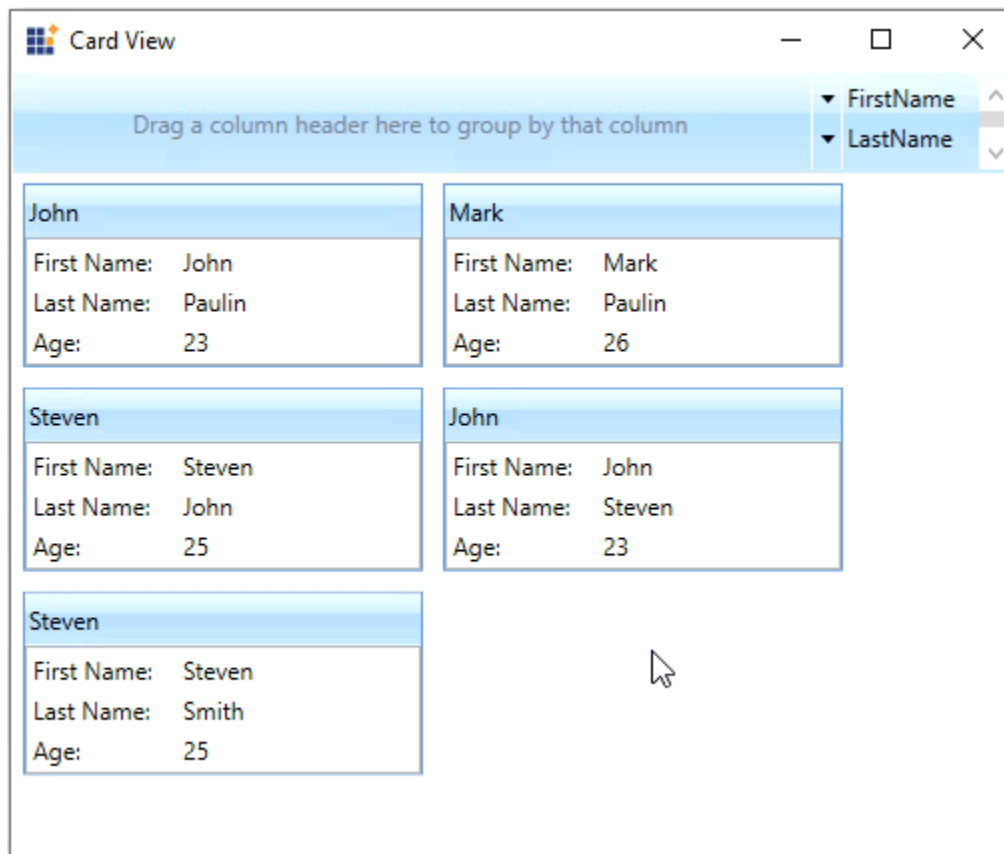
**Note:** To perform an edit operation on selected `CardViewItem`, you need to define the `CardViewItem` edit mode UI with editable functionalities by using [EditItemTemplate](#).

### XML

```
<syncfusion:CardView CanEdit="True"
ItemsSource="{Binding CardViewItems}"
Name="cardView">
<syncfusion:CardView.EditItemTemplate>
<DataTemplate>
<ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBlock Text="First Name:" />
<TextBox
Grid.Column="1"
Text="{Binding FirstName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Last Name:" />
<TextBox Grid.Column="1"
Text="{Binding LastName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Age:" />
<TextBox Grid.Column="1"
Text="{Binding Age,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.EditItemTemplate>
</syncfusion:CardView>
```

### C#

```
cardView.CanEdit = true;
```



**Note:** [View Sample in GitHub](#)

### Orientation of CardViewItems

You can arrange the cards either vertically or horizontally by using the [Orientation](#) property. The default value of [Orientation](#) property is [Vertical](#).

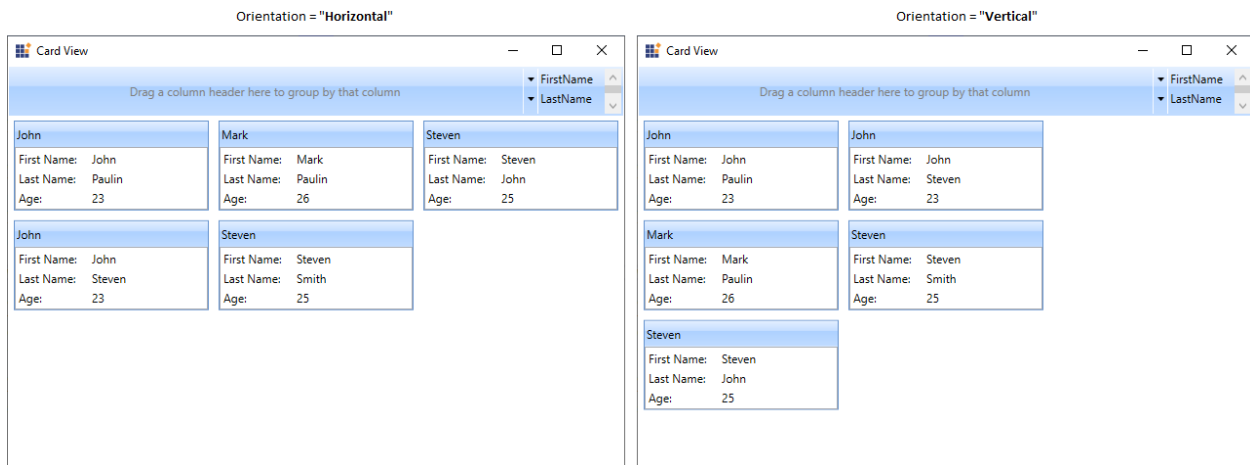
**Note:** If there is not enough space to arrange the cards either vertically or horizontally that is assigned by [Orientation](#) property, then [CardView](#) will automatically arrange the cards at available position.

### XML

```
<syncfusion:CardView Orientation="Horizontal"
ItemsSource="{Binding CardViewItems}"
Name="cardView"/>
```

### C#

```
cardView.Orientation = Orientation.Horizontal;
```



**Note:** [View Sample in GitHub](#)

### Selected item changed notification

You will be notified when selected card item changed in [CardView](#) by using [SelectedItemChanged](#) event. The [SelectedItemChanged](#) event contains the old and newly selected card item in the [OldValue](#) and [NewValue](#) properties.

### XML

```
<syncfusion:CardView SelectedItemChanged="CardView_SelectedItemChanged"
ItemsSource="{Binding CardViewItems}"
Name="cardView"/>
```

### C#

```
cardView.SelectedItemChanged += CardView_SelectedItemChanged;
```

You can handle the event as follows:

### C#

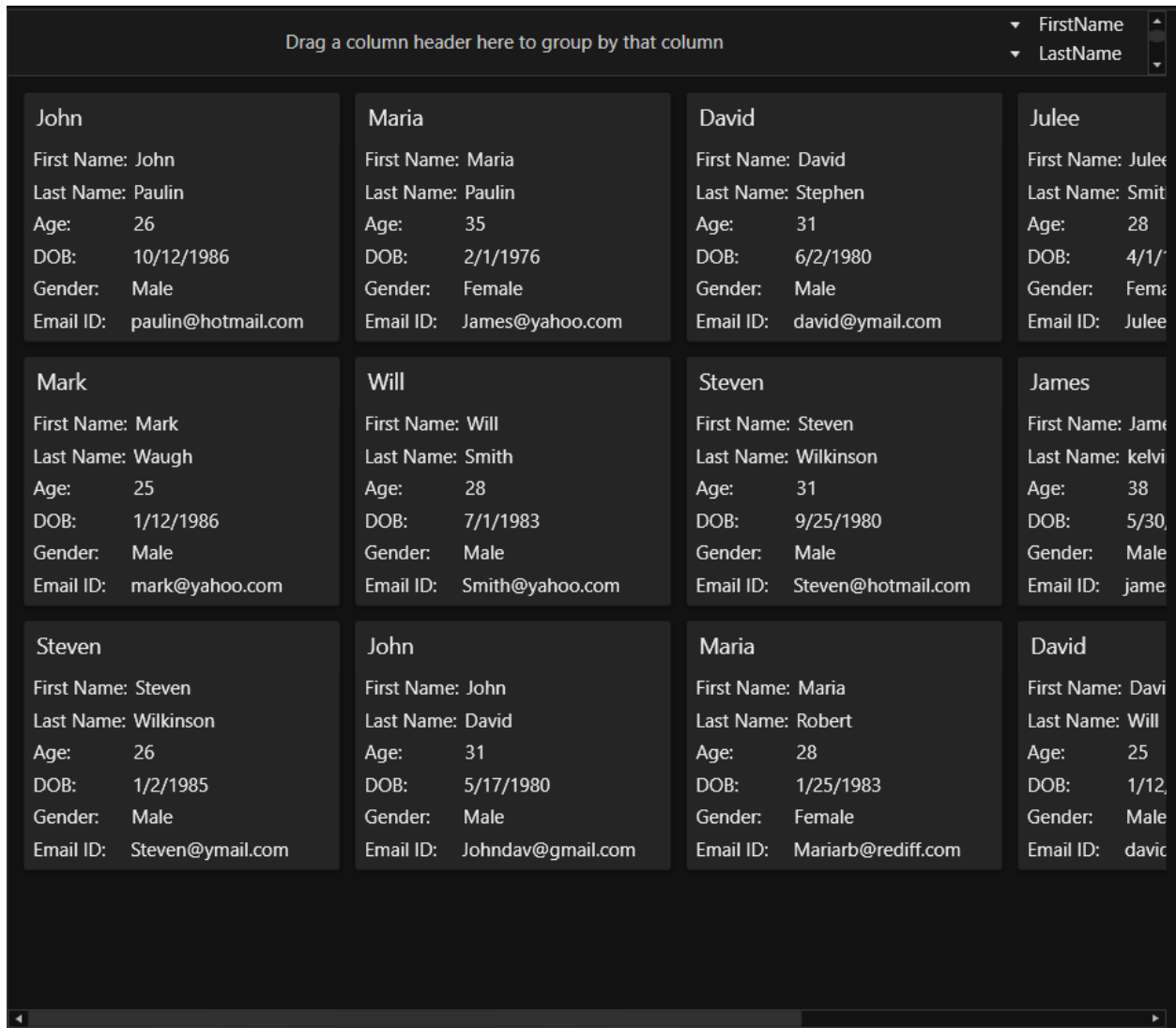
```
private void CardView_SelectedItemChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    var newItem = e.NewValue;
    var oldItem = e.OldValue;
}
```

### Theme

CardView supports various built-in themes. Refer to the below links to apply themes for the CardView,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

**Note:** [View Sample in GitHub](#)



## Data Binding and Customization in WPF Card View

This section describes the data binding support and control customization supports available in the [CardView](#) control.

### Data binding to objects

The [CardView](#) control can bound to an external source to auto create [CardViewItem](#) and display the data using [ItemsSource](#) property. When you are auto generating the [CardViewItem](#) using [ItemsSource](#), you need to use [HeaderTemplate](#) or [ItemContainerStyle](#) to define header and use the [ItemTemplate](#) or [ItemContainerStyle](#) to display the content of the [CardViewItem](#) item.

**Note:** You can use the grouping, sorting, filtering and editing functionalities only by populating the card items through the [ItemsSource](#) property.

### C#

```
//Model.cs
public class CardViewModel
{
    public string FirstName { get; set; }
}
```

```

public string LastName { get; set; }
public int Age { get; set; }
}
//ViewModel.cs
public class ViewModel : NotificationObject
{
    private ObservableCollection<CardViewModel> cardViewItems;
    public ObservableCollection<CardViewModel> CardViewItems
    {
        get { return cardViewItems; }
        set { cardViewItems = value;
            this.RaisePropertyChanged(nameof(CardViewItems)); }
    }
    public ViewModel()
    {
        CardViewItems = new ObservableCollection<CardViewModel>();
        populateItems();
    }
    private void populateItems()
    {
        CardViewItems.Add(new CardViewModel() { FirstName = "John", LastName =
            "Paulin", Age = 23});
        CardViewItems.Add(new CardViewModel() { FirstName = "Mark", LastName =
            "Paulin", Age = 26 });
        CardViewItems.Add(new CardViewModel() { FirstName = "Steven", LastName =
            "John", Age = 25 });
        CardViewItems.Add(new CardViewModel() { FirstName = "John", LastName =
            "Steven", Age = 23 });
        CardViewItems.Add(new CardViewModel() { FirstName = "Steven", LastName =
            "Smith", Age = 25 });
    }
}

```

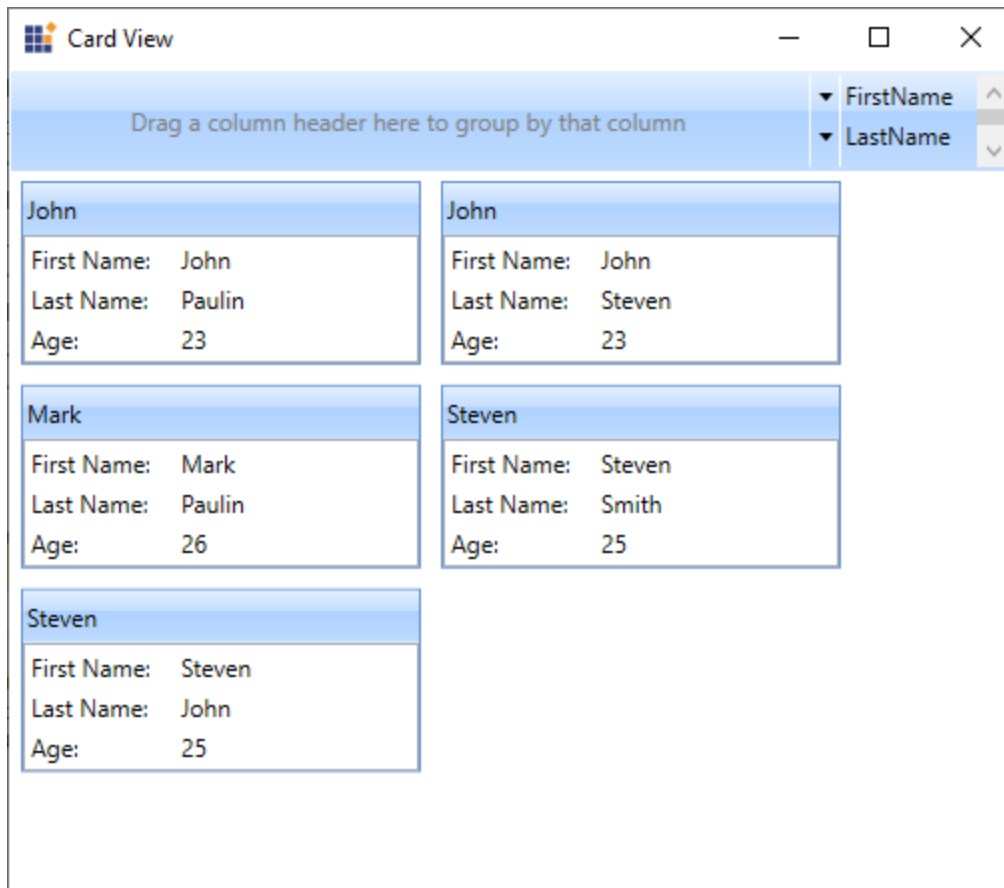
## XML

```

<syncfusion:CardView ItemsSource="{Binding CardViewItems}"
Name="cardView">
    <syncfusion:CardView.HeaderTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding FirstName}"/>
        </DataTemplate>
    </syncfusion:CardView.HeaderTemplate>
    <syncfusion:CardView.ItemTemplate>
        <DataTemplate >
            <ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
                <ListBoxItem Padding="1">
                    <Grid>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="75" />
                            <ColumnDefinition />
                        </Grid.ColumnDefinitions>
                        <TextBlock Text="First Name:" />
                        <TextBlock Grid.Column="1"
                            Text="{Binding FirstName,
                                UpdateSourceTrigger=PropertyChanged}" />
                    </Grid>
                </ListBoxItem>
            </ListBox>
        </DataTemplate>
    </syncfusion:CardView.ItemTemplate>
</syncfusion:CardView>

```

```
</ListBoxItem>
<ListBoxItem Padding="1">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="75" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <TextBlock Text="Last Name:" />
    <TextBlock Grid.Column="1"
      Text="{Binding LastName,
        UpdateSourceTrigger=PropertyChanged}" />
  </Grid>
</ListBoxItem>
<ListBoxItem Padding="1">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="75" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <TextBlock Text="Age:" />
    <TextBlock Grid.Column="1"
      Text="{Binding Age,
        UpdateSourceTrigger=PropertyChanged}" />
  </Grid>
</ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.ItemTemplate>
</syncfusion:CardView>
```



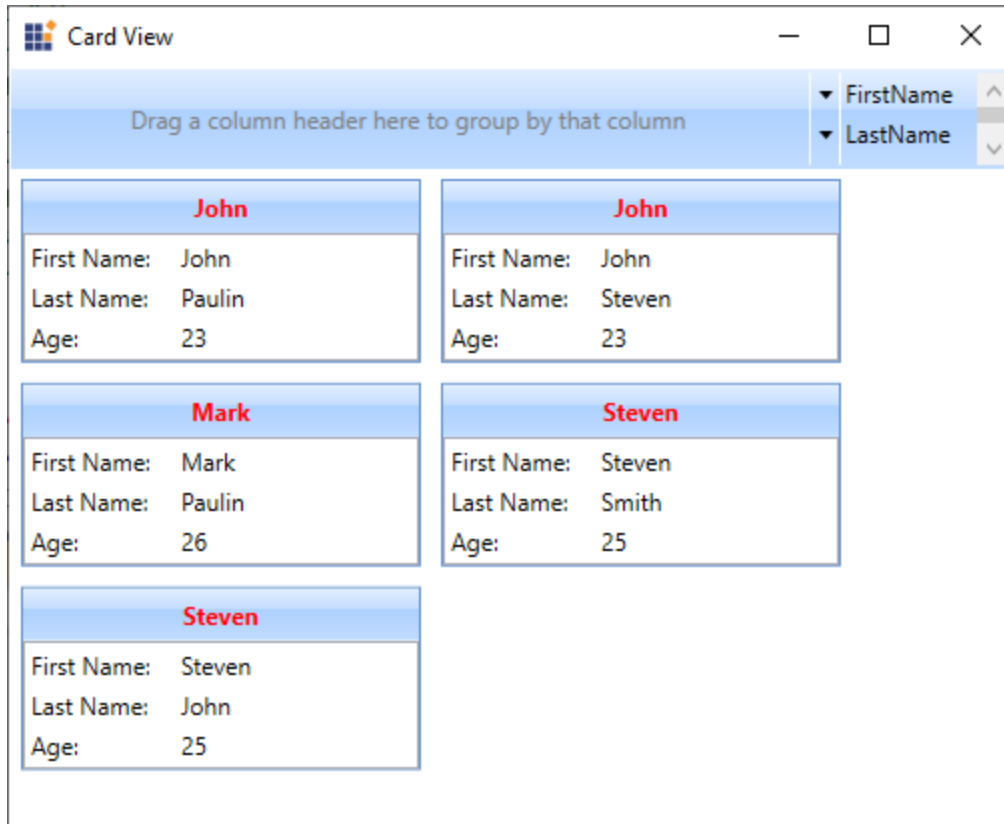
**Note:** [View Sample in GitHub](#)

Custom UI for CardViewItem header

You can change the appearance of card item's header by using the [HeaderTemplate](#) property. The DataContext of the [HeaderTemplate](#) property is [CardViewItem](#).

### XML

```
<syncfusion:CardView ItemsSource="{Binding CardViewItems}"
Name="cardView">
<syncfusion:CardView.HeaderTemplate>
<DataTemplate>
<TextBlock
Text="{Binding FirstName}"
TextAlignment="Center"
FontWeight="Bold"
Foreground="Red"/>
</DataTemplate>
</syncfusion:CardView.HeaderTemplate>
</syncfusion:CardView>
```



**Note:** [View Sample in GitHub](#)

Custom UI for CardViewItem content

You can change the appearance of card item's content by using the `ItemTemplate` property. The `DataContext` of the `ItemTemplate` property is `CardViewItem`.

### XML

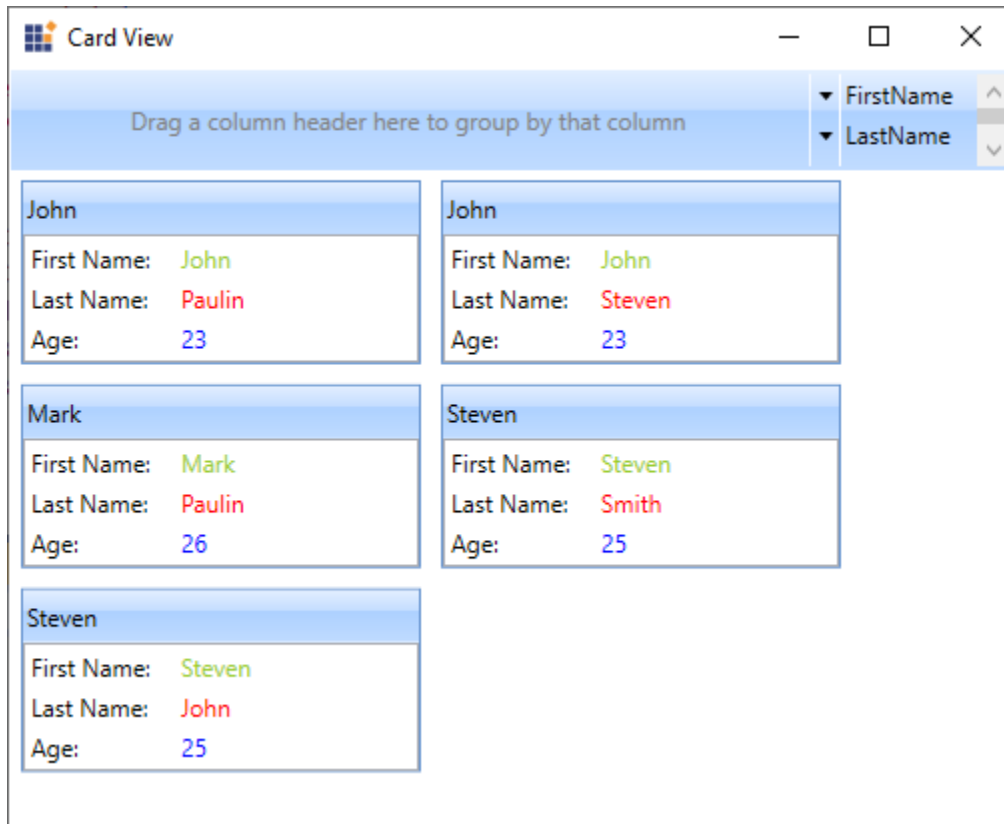
```
<syncfusion:CardView ItemsSource="{Binding CardViewItems}"
Name="cardView">
  <syncfusion:CardView.ItemTemplate>
    <DataTemplate >
      <ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
        <ListBoxItem Padding="1">
          <Grid>
            <Grid.ColumnDefinitions>
              <ColumnDefinition Width="75" />
              <ColumnDefinition />
            </Grid.ColumnDefinitions>
            <TextBlock Text="First Name:" />
            <TextBlock Grid.Column="1"
              Foreground="YellowGreen"
              Text="{Binding FirstName,
                UpdateSourceTrigger=PropertyChanged}" />
          </Grid>
        </ListBoxItem>
        <ListBoxItem Padding="1">
          <Grid>
            <Grid.ColumnDefinitions>
```



```

<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Last Name:" />
<TextBlock Grid.Column="1"
Foreground="Red"
Text="{Binding LastName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Age:" />
<TextBlock Grid.Column="1"
Foreground="Blue"
Text="{Binding Age,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.ItemTemplate>
</syncfusion:CardView>

```



**Note:** [View Sample in GitHub](#)

Custom UI for edit mode CardViewItem

You can change the appearance of card items in edit mode by using `EditItemTemplate` property. The `DataContext` of `EditItemTemplate` property is `CardViewItem`.

### XML

```
<syncfusion:CardView CanEdit="True"
ItemsSource="{Binding CardViewItems}" Name="cardView">
<syncfusion:CardView.EditItemTemplate>
<DataTemplate>
<ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
<ListBoxItem Padding="1" HorizontalContentAlignment="Stretch">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBlock Text="First Name:" />
<TextBox
Background="Black"
Foreground="White"
Grid.Column="1"
Text="{Binding FirstName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1" HorizontalContentAlignment="Stretch">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Last Name:" />
<TextBox
Background="LightGreen"
Foreground="Red"
Grid.Column="1"
Text="{Binding LastName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1" HorizontalContentAlignment="Stretch">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Age:" />
<TextBox
Background="Pink"
Foreground="Blue"
Grid.Column="1"
Text="{Binding Age,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.EditItemTemplate>
</syncfusion:CardView>
```

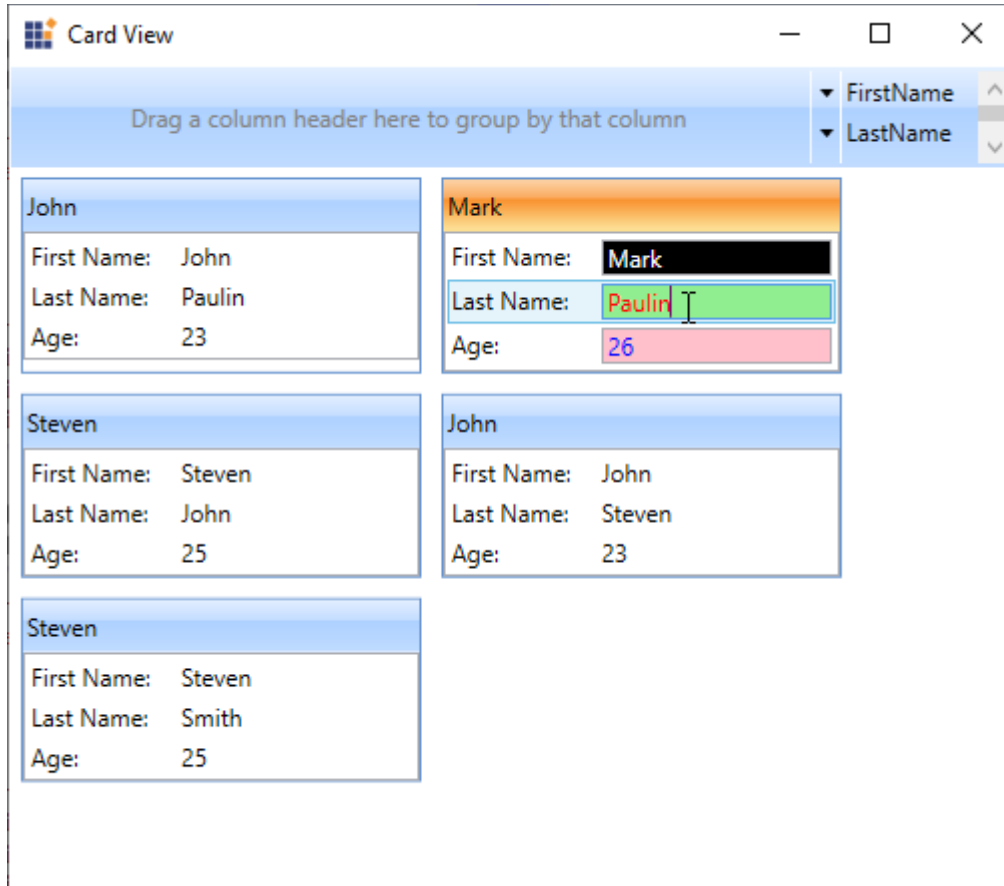
```

</ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.EditItemTemplate>
</syncfusion:CardView>

```

**C#**

```
cardView.CanEdit = true;
```



**Note:** [View Sample in GitHub](#)

### Different UI styles for specific CardViewItem

You can change the UI style of specific card item based on the field values or any other logics by using the `ItemContainerStyleSelector` property.

**C#**

```

public class CardViewItemContainerStyleSelector : StyleSelector
{
    public Style Style1 { get; set; }
    public Style Style2 { get; set; }
    public override Style SelectStyle(object item, DependencyObject container) {
        string LastName = (item as CardViewModel).LastName;
        if (LastName == "Paulin") {
            return Style1;
        }
    }
}

```

```

}
else {
return Style2;
}
}
}

```

**XML**

```

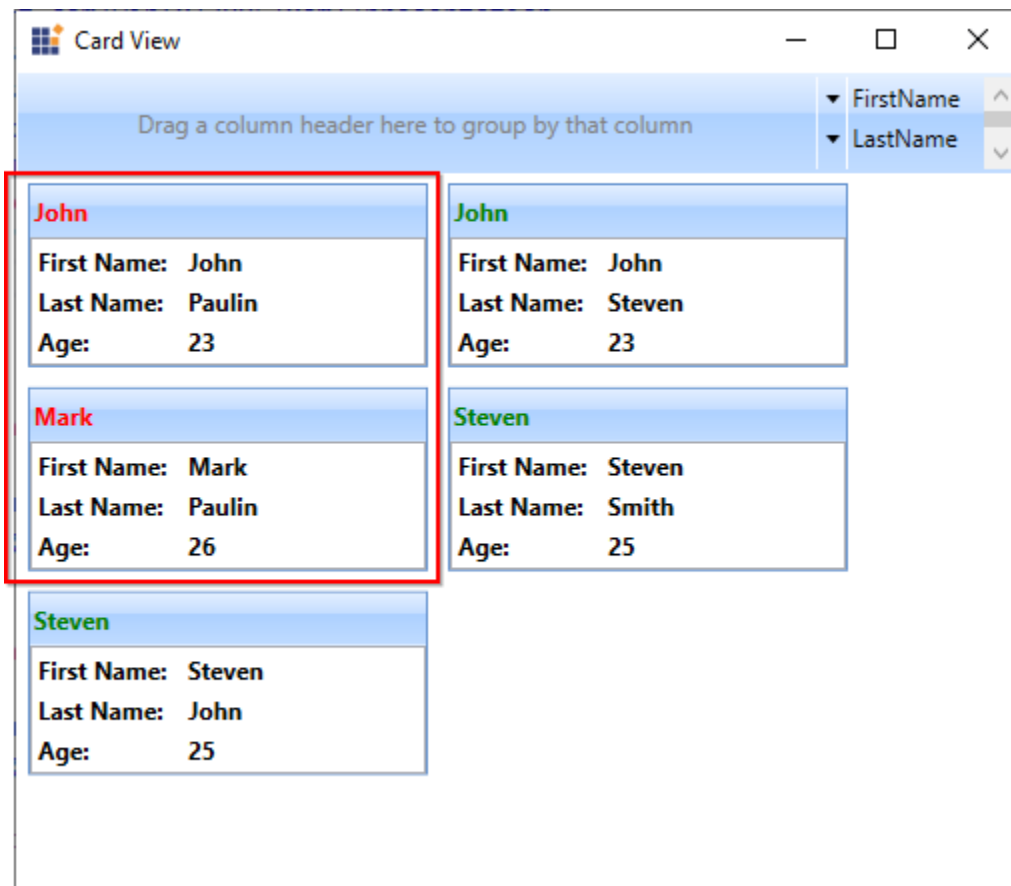
<Window.Resources>
<local:ViewModel x:Key="viewModel"/>
<Style TargetType="{x:Type syncfusion:CardViewItem}" x:Key="style1">
<Setter Property="Header" Value="{Binding FirstName}"/>
<Setter Property="Foreground" Value="Red"/>
<Setter Property="FontWeight" Value="Bold"/>
</Style>
<Style TargetType="{x:Type syncfusion:CardViewItem}" x:Key="style2">
<Setter Property="Header" Value="{Binding FirstName}"/>
<Setter Property="Foreground" Value="Green"/>
<Setter Property="FontWeight" Value="Bold"/>
</Style>
<local:CardViewItemContainerStyleSelector
x:Key="cardViewItemContainerStyleSelector"
Style1="{StaticResource style1}"
Style2="{StaticResource style2}"/>
</Window.Resources>
<Grid Margin="1">
<syncfusion:CardView ItemContainerStyleSelector="{StaticResource
cardViewItemContainerStyleSelector}"
ItemsSource="{Binding CardViewItems}"
DataContext="{StaticResource viewModel}"
Name="cardView">
<syncfusion:CardView.ItemTemplate>
<DataTemplate >
<ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBlock Text="First Name:" />
<TextBlock Grid.Column="1"
Text="{Binding FirstName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Last Name:" />
<TextBlock Grid.Column="1"
Text="{Binding LastName,

```

```

UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Age:" />
<TextBlock Grid.Column="1"
Text="{Binding Age,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.ItemTemplate>
</syncfusion:CardView>
</Grid>

```



Here, the different style are applied to the card items based on the `LastName` field.

**Note:** [View Sample in GitHub](#)

### Change flow direction

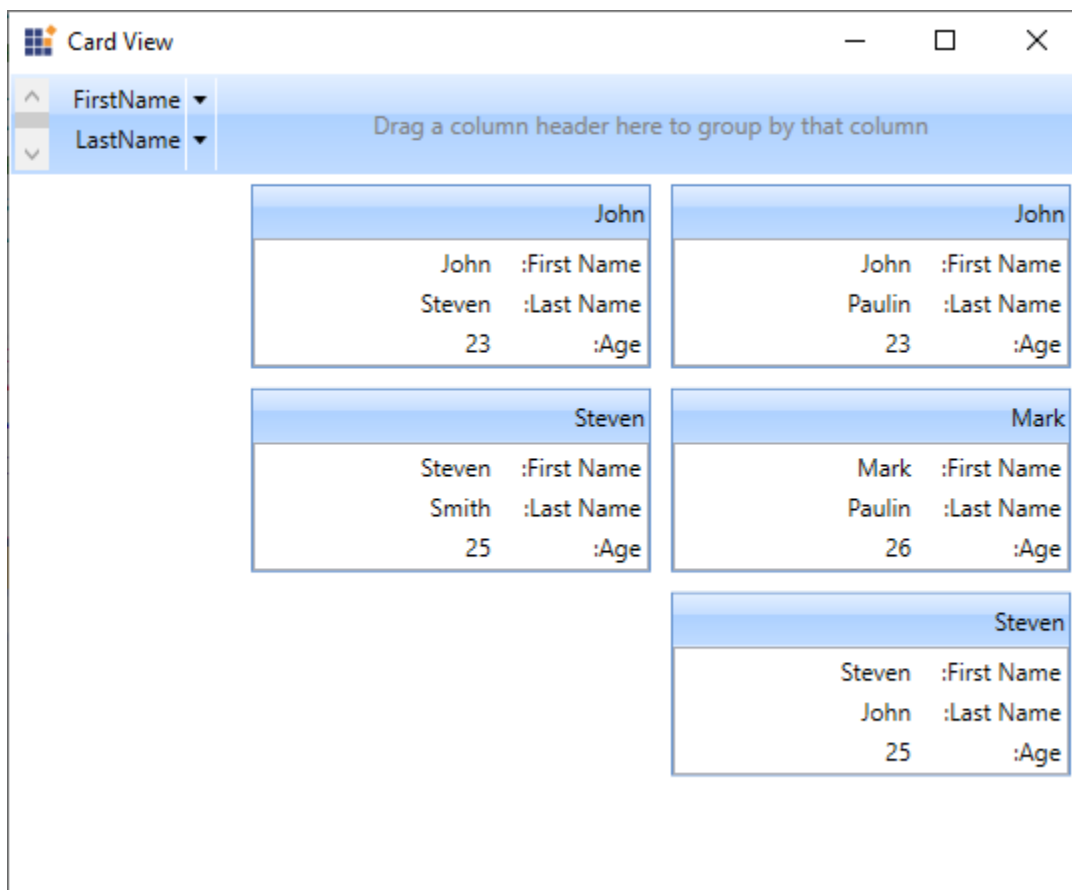
You can change the flow direction of the **CardView** control layout from right to left by setting the **FlowDirection** property value as **RightToLeft**. The default value of **FlowDirection** property is **LeftToRight**.

### XML

```
<syncfusion:CardView FlowDirection="RightToLeft"
ItemsSource="{Binding CardViewItems}"
Name="cardView"/>
```

### C#

```
cardView.FlowDirection = FlowDirection.RightToLeft;
```



**Note:** [View Sample in GitHub](#)

### Theme

You can customize the appearance of the **CardView** control by using the [SfSkinManager.SetVisualStyle](#) method. The following are the various built-in visual styles for **CardView** control.

- Blend
- Lime
- MaterialDark

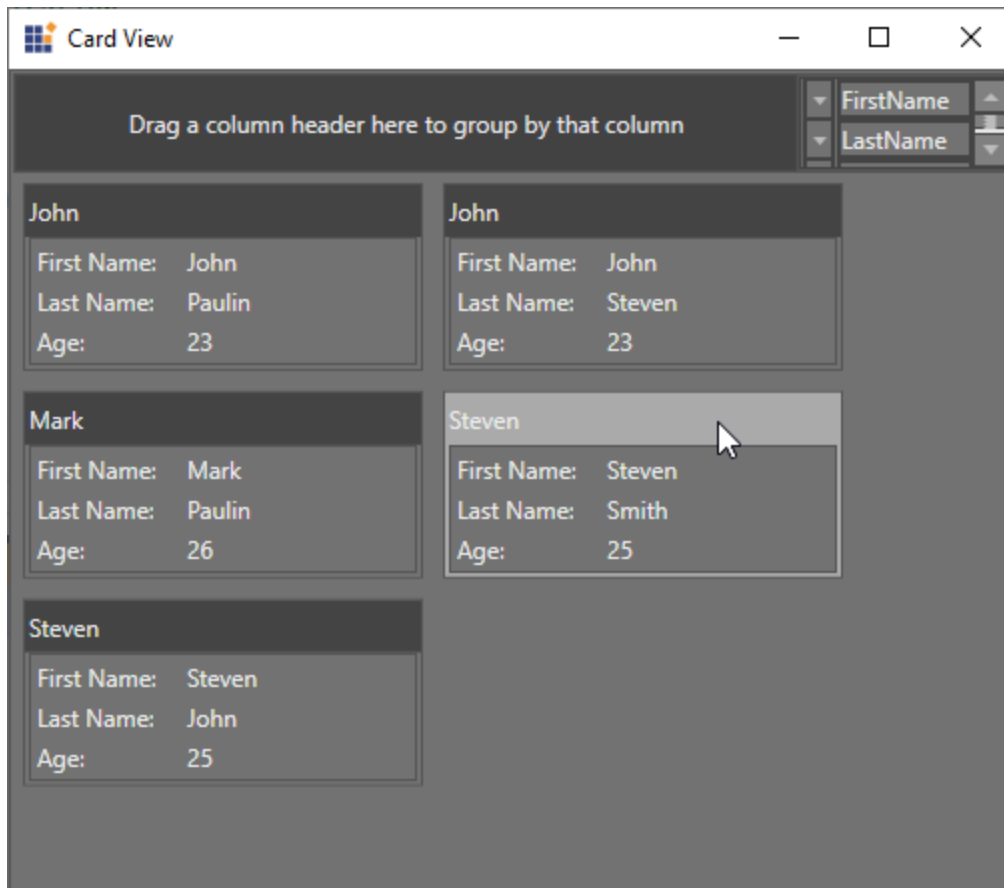
- MaterialDarkBlue
- MaterialLight
- MaterialLightBlue
- Metro
- Office2010Black
- Office2010Blue
- Office2010Silver
- Office2013DarkGray
- Office2013LightGray
- Office2013White
- Office2016Colorful
- Office2016DarkGray
- Office2016White
- Office2019Black
- Office2019Colorful
- Office365
- Saffron
- VisualStudio2013
- VisualStudio2015

### XML

```
<Window
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF">
<Grid>
<syncfusion:CardView ItemsSource="{Binding CardViewItems}"
Name="cardView"/>
</Grid>
</Window>
```

### C#

```
//Namespace for the SfSkinManager.
using Syncfusion.SfSkinManager;
SfSkinManager.SetVisualStyle(cardView, VisualStyles.Blend);
```



Here, the `Blend` style is applied to the `CardView` control.

**Note:** [View Sample in GitHub](#)

### Editing Mode in WPF Card View

This section describes how to enable or disable the edit mode and perform edit operation in `CardView` control.

#### Enable/disable the editing mode

You can enable or disable the editing mode of card items by setting the `CanEdit` property value as `true` or `false`. The default value of `CanEdit` property is `false`.

**Note:** If you want edit the card items, you must set the `CanEdit` property value as `true`. Otherwise, you will not be able to perform the edit operation.

#### XML

```
<syncfusion:CardView CanEdit="True"
    Name="cardView"/>
```

#### C#

```
cardView.CanEdit = true;
```

**Note:** [View Sample in GitHub](#)



### Card editing using keyboard and mouse interaction

You can edit the selected `CardViewItem` value by double-clicking on that item or by pressing the **F2** key. To get out from the editing mode, you need to press the **Esc** or **Enter** key.

**Note:** To perform an edit operation on selected `CardViewItem`, you need to define the `CardViewItem` edit mode UI with editable functionalities by using [EditItemTemplate](#). `EditItemTemplate` is applied to the selected item in the edit mode, and `ItemTemplate` is applied to the selected item in the view mode.

### C#

```
//Model.cs
public class CardViewModel
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}

//ViewModel.cs
public class ViewModel : NotificationObject
{
    private ObservableCollection<CardViewModel> cardViewItems;
    public ObservableCollection<CardViewModel> CardViewItems
    {
        get { return cardViewItems; }
        set { cardViewItems = value;
            this.RaisePropertyChanged(nameof(CardViewItems)); }
    }
    public ViewModel()
    {
        CardViewItems = new ObservableCollection<CardViewModel>();
        populateItems();
    }
    private void populateItems()
    {
        CardViewItems.Add(new CardViewModel() { FirstName = "John", LastName = "Paulin", Age = 23 });
        CardViewItems.Add(new CardViewModel() { FirstName = "Mark", LastName = "Paulin", Age = 26 });
        CardViewItems.Add(new CardViewModel() { FirstName = "Steven", LastName = "John", Age = 25 });
        CardViewItems.Add(new CardViewModel() { FirstName = "John", LastName = "Steven", Age = 23 });
        CardViewItems.Add(new CardViewModel() { FirstName = "Steven", LastName = "Smith", Age = 25 });
    }
}
```

### XML

```
<syncfusion:CardView CanEdit="True"
ItemsSource="{Binding CardViewItems}"
Name="cardView">
    <syncfusion:CardView.EditItemTemplate>
        <DataTemplate>
            <ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
```

```

<ListBoxItem Padding="1">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="75" />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <TextBlock Text="First Name:" />
    <TextBox
      Grid.Column="1"
      Text="{Binding FirstName,
        UpdateSourceTrigger=PropertyChanged}" />
    </Grid>
  </ListBoxItem>
  <ListBoxItem Padding="1">
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="75" />
        <ColumnDefinition Width="*" />
      </Grid.ColumnDefinitions>
      <TextBlock Text="Last Name:" />
      <TextBox Grid.Column="1"
        Text="{Binding LastName,
          UpdateSourceTrigger=PropertyChanged}" />
    </Grid>
  </ListBoxItem>
  <ListBoxItem Padding="1">
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="75" />
        <ColumnDefinition Width="*" />
      </Grid.ColumnDefinitions>
      <TextBlock Text="Age:" />
      <TextBox Grid.Column="1"
        Text="{Binding Age,
          UpdateSourceTrigger=PropertyChanged}" />
    </Grid>
  </ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.EditItemTemplate>
<syncfusion:CardView.HeaderTemplate>
  <DataTemplate>
    <TextBlock Text="{Binding FirstName}"/>
  </DataTemplate>
</syncfusion:CardView.HeaderTemplate>
<syncfusion:CardView.ItemTemplate>
  <DataTemplate >
    <ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
      <ListBoxItem Padding="1">
        <Grid>
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="75" />
            <ColumnDefinition />
          </Grid.ColumnDefinitions>
          <TextBlock Text="First Name:" />
          <TextBlock Grid.Column="1"
            Text="{Binding FirstName,

```

```

UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Last Name:" />
<TextBlock Grid.Column="1"
Text="{Binding LastName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Age:" />
<TextBlock Grid.Column="1"
Text="{Binding Age,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.ItemTemplate>
</syncfusion:CardView>

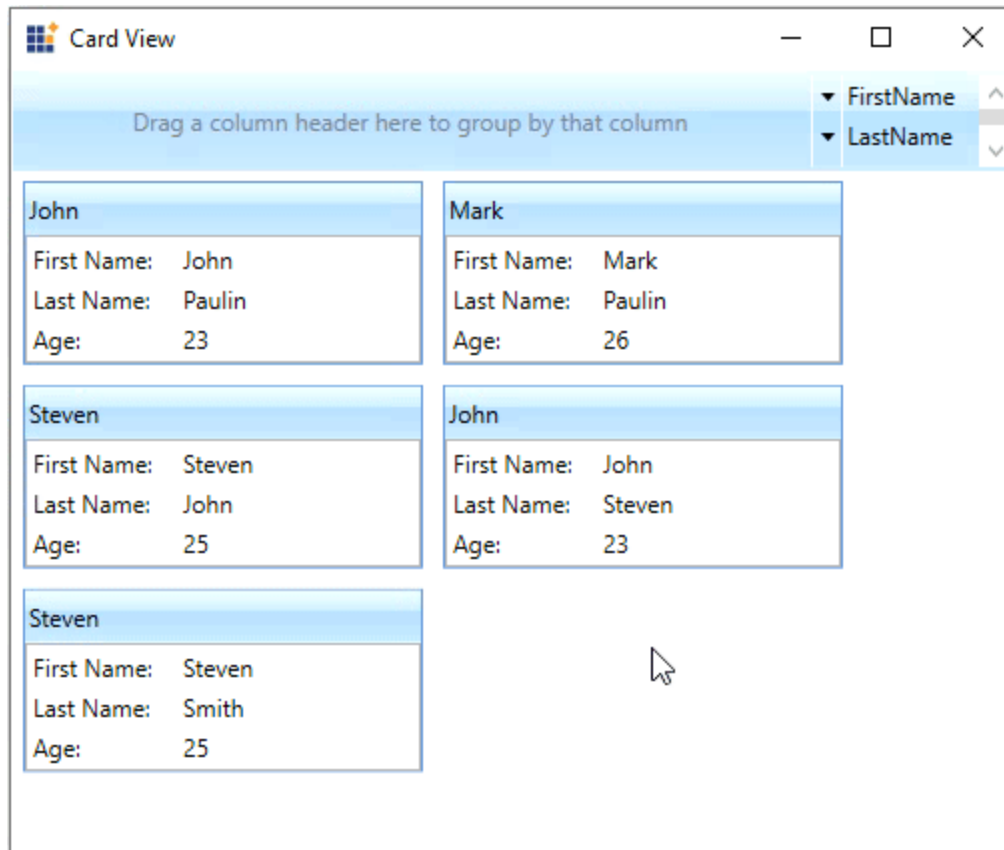
```

## C#

```

cardView.CanEdit = true;

```



**Note:** [View Sample in GitHub](#)

Start card editing programmatically

If you want to programmatically start the edit mode of selected `CardViewItem`, use the [BeginEdit](#) method. You can also programmatically change the edit mode to view mode by using the [EndEdit](#) method.

**Note:** You must set the `CanEdit` property as `true` to perform `BeginEdit` or `EndEdit`.

### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid Margin="1">
<Grid.RowDefinitions>
<RowDefinition Height="auto"/>
<RowDefinition/>
</Grid.RowDefinitions>
<StackPanel Grid.Row="0" Orientation="Horizontal">
<Button Click="BeginEdit_Click" Content="Begin Edit"/>
<Button Click="EndEdit_Click" Content="End Edit"/>
</StackPanel>
<syncfusion:CardView CanEdit="True" Grid.Row="1"
ItemsSource="{Binding CardViewItems}"
Name="cardView">
<syncfusion:CardView.EditItemTemplate>
<DataTemplate>
```

```

<ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
<ListBoxItem Padding="1" HorizontalContentAlignment="Stretch">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBlock Text="First Name:" />
<TextBox
Grid.Column="1"
Text="{Binding FirstName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1" HorizontalContentAlignment="Stretch">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Last Name:" />
<TextBox Grid.Column="1"
Text="{Binding LastName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1" HorizontalContentAlignment="Stretch">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Age:" />
<TextBox Grid.Column="1"
Text="{Binding Age,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.EditItemTemplate>
</syncfusion:CardView>
</Grid>

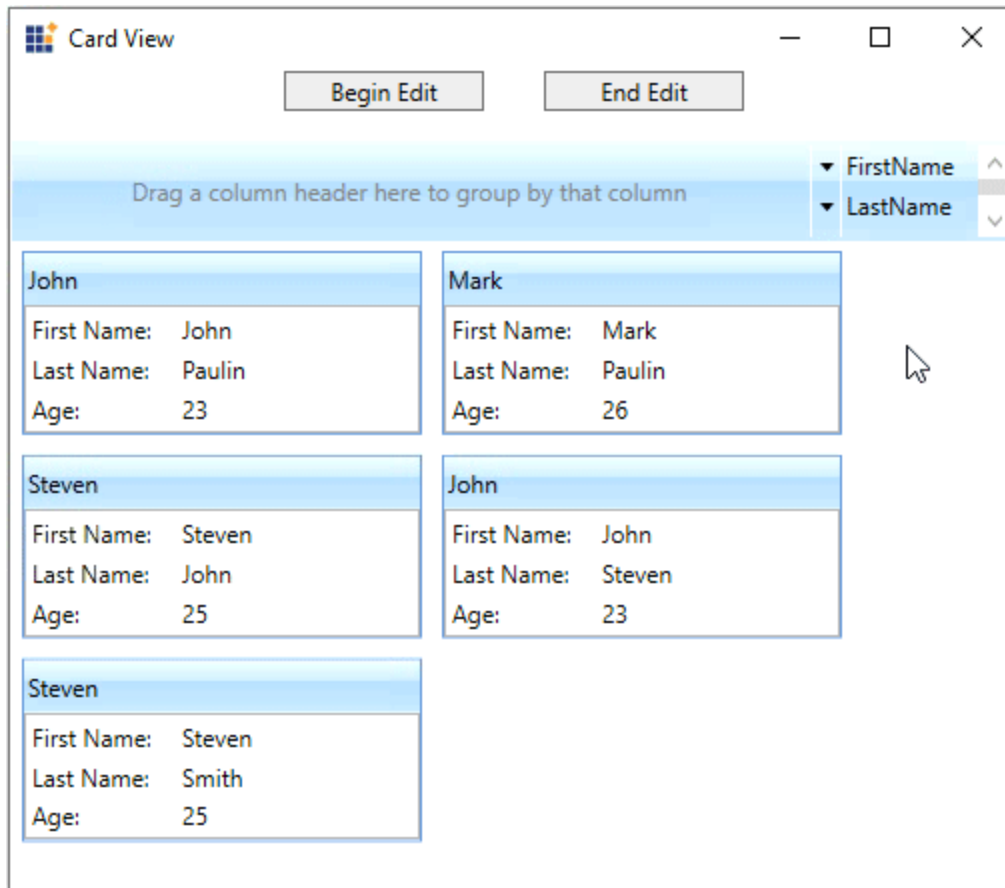
```

**C#**

```

private void BeginEdit_Click(object sender, RoutedEventArgs e) {
//Turn on the edit mode
cardView.BeginEdit();
}
private void EndEdit_Click(object sender, RoutedEventArgs e) {
//Turn off the edit mode
cardView.EndEdit();
}

```



**Note:** [View Sample in GitHub](#)

Custom UI for edit mode CardViewItem

You can change the appearance of card items in edit mode by using `EditItemTemplate` property. The `DataContext` of `EditItemTemplate` property is `CardViewItem`.

#### XML

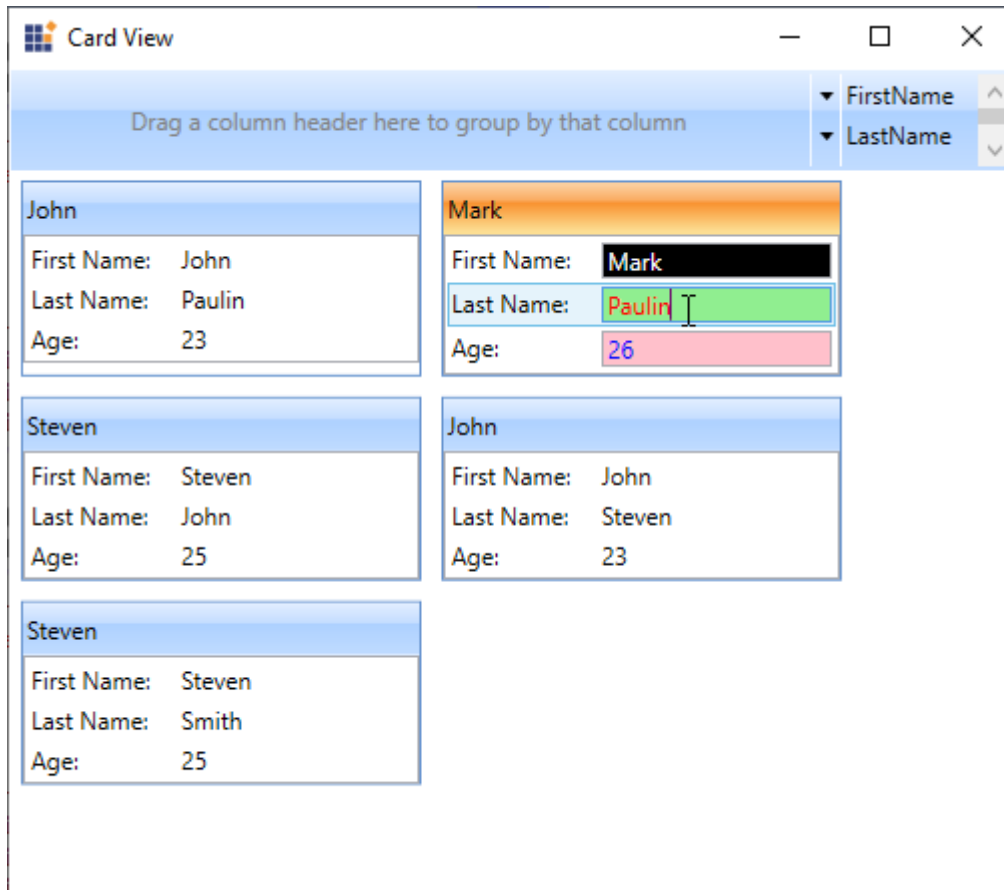
```
<syncfusion:CardView CanEdit="True"
ItemsSource="{Binding CardViewItems}" Name="cardView">
<syncfusion:CardView.EditItemTemplate>
<DataTemplate>
<ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
<ListBoxItem Padding="1" HorizontalContentAlignment="Stretch">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBlock Text="First Name:" />
<TextBox
Background="Black"
Foreground="White"
Grid.Column="1"
Text="{Binding FirstName,
UpdateSourceTrigger=PropertyChanged}" />

```

```
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1" HorizontalContentAlignment="Stretch">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="75" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <TextBlock Text="Last Name:" />
    <TextBox
      Background="LightGreen"
      Foreground="Red"
      Grid.Column="1"
      Text="{Binding LastName,
        UpdateSourceTrigger=PropertyChanged}" />
    </Grid>
  </ListBoxItem>
  <ListBoxItem Padding="1" HorizontalContentAlignment="Stretch">
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="75" />
        <ColumnDefinition Width="*" />
      </Grid.ColumnDefinitions>
      <TextBlock Text="Age:" />
      <TextBox
        Background="Pink"
        Foreground="Blue"
        Grid.Column="1"
        Text="{Binding Age,
          UpdateSourceTrigger=PropertyChanged}" />
    </Grid>
  </ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.EditItemTemplate>
```

**C#**

```
cardView.CanEdit = true;
```



**Note:** [View Sample in GitHub](#)

## Grouping Mode in WPF Card View

This section describes how to enable or disable the grouping mode and perform group operations in [CardView](#) control.

### Enable/disable the grouping mode

You can enable or disable the grouping mode of card items by setting the [CanGroup](#) property value as `true` or `false`. The default value of [CanGroup](#) property is `true`.

### XML

```
<syncfusion:CardView CanGroup="True"
    Name="cardView"/>
```

### C#

```
cardView.CanGroup = true;
```

**Note:** [View Sample in GitHub](#)



### Group the CardViewItems

You can group the cards inside the **CardView** control by dragging the available field from the list and drop it into the dropping region of the **CardView** control header. Based on the dropped field, the cards are grouped.

#### C#

```
//Model.cs
public class CardViewModel
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}

//ViewModel.cs
public class ViewModel : NotificationObject
{
    private ObservableCollection<CardViewModel> cardViewItems;
    public ObservableCollection<CardViewModel> CardViewItems
    {
        get { return cardViewItems; }
        set { cardViewItems = value;
            this.RaisePropertyChanged(nameof(CardViewItems)); }
    }
    public ViewModel()
    {
        CardViewItems = new ObservableCollection<CardViewModel>();
        populateItems();
    }
    private void populateItems()
    {
        CardViewItems.Add(new CardViewModel() { FirstName = "John", LastName=
        "Paulin", Age = 23});
        CardViewItems.Add(new CardViewModel() { FirstName = "Mark", LastName =
        "Paulin",Age = 26 });
        CardViewItems.Add(new CardViewModel() { FirstName = "Steven", LastName =
        "John", Age = 25 });
        CardViewItems.Add(new CardViewModel() { FirstName = "John", LastName =
        "Steven", Age = 23 });
        CardViewItems.Add(new CardViewModel() { FirstName = "Steven", LastName =
        "Smith", Age = 25 });
    }
}
```

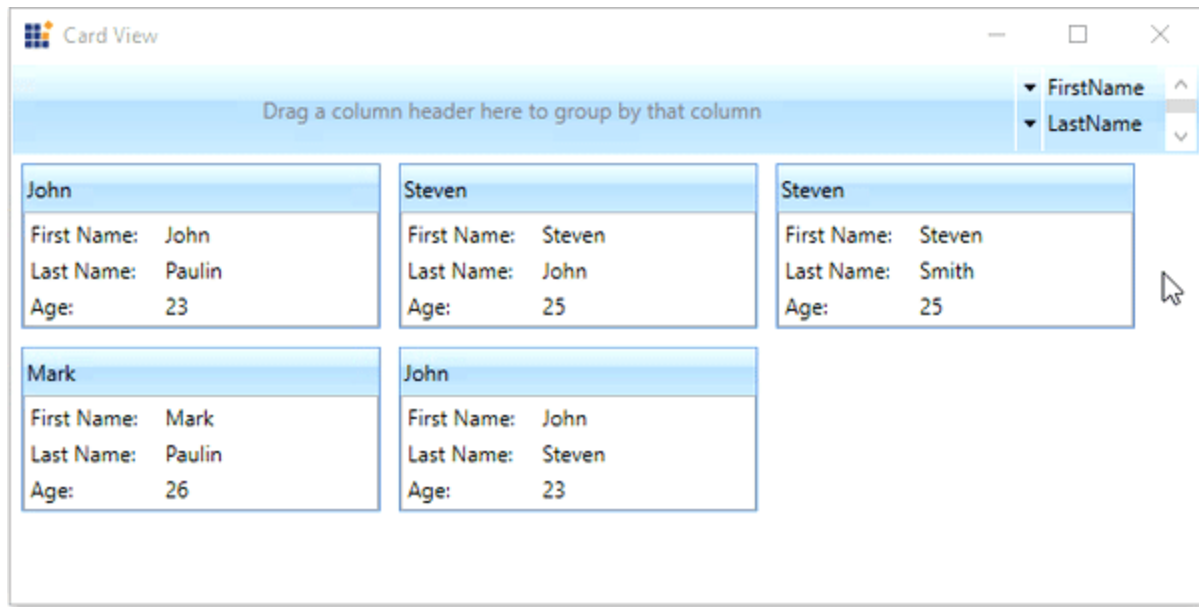
#### XML

```
<syncfusion:CardView CanGroup="True"
ItemsSource="{Binding CardViewItems}"
Name="cardView">
    <syncfusion:CardView.HeaderTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding FirstName}"/>
        </DataTemplate>
    </syncfusion:CardView.HeaderTemplate>
    <syncfusion:CardView.ItemTemplate>
        <DataTemplate >
```

```
<ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBlock Text="First Name:" />
<TextBlock Grid.Column="1"
Text="{Binding FirstName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Last Name:" />
<TextBlock Grid.Column="1"
Text="{Binding LastName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Age:" />
<TextBlock Grid.Column="1"
Text="{Binding Age,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.ItemTemplate>
</syncfusion:CardView>
```

## C#

```
cardView.CanGroup = true;
```



Here, `CardViewItems` grouped based on `Age` field.

**Note:** [View Sample in GitHub](#)

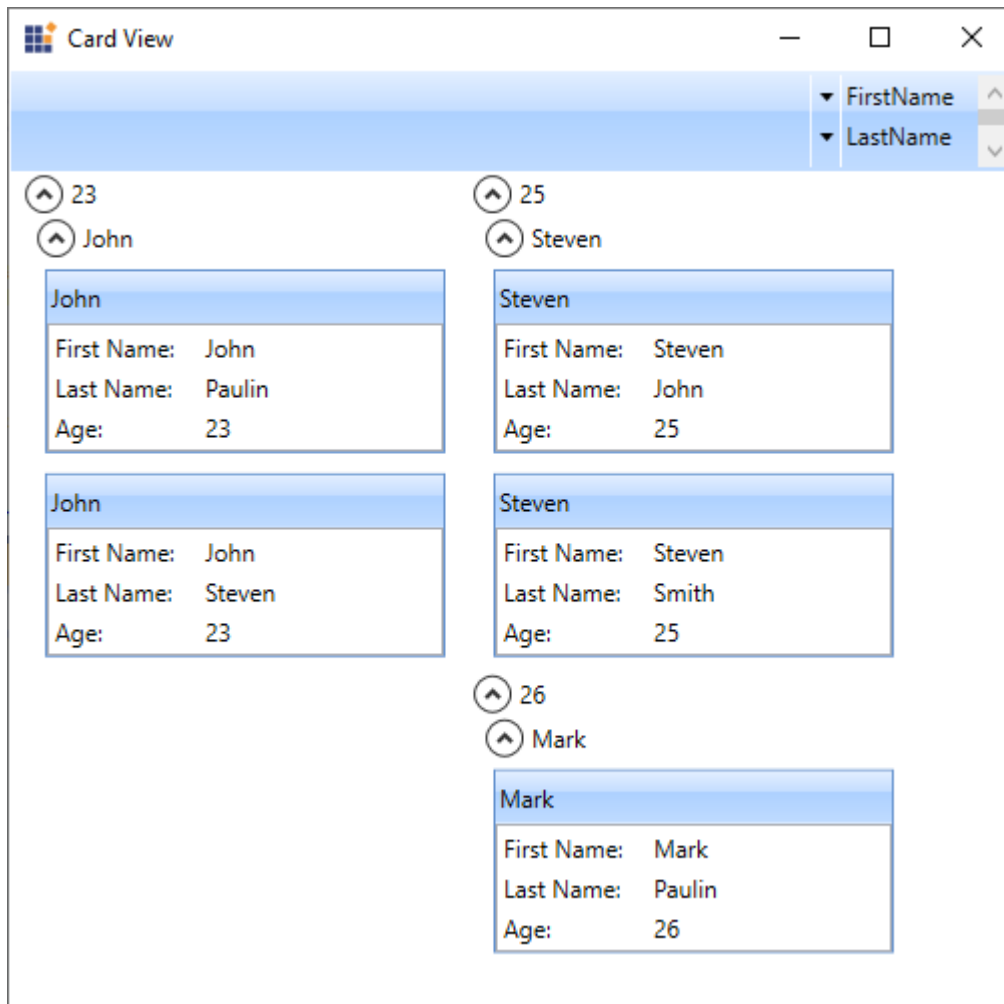
#### Group the cards programmatically

You can group the cards programmatically by passing the specific field name into the [GroupCards\(String\)](#) method. Based on the field name, cards are grouped programmatically. You can also make the multi-level grouping programmatically.

#### C#

```
cardView.CanGroup = true;  
cardView.GroupCards("Age");  
cardView.GroupCards("FirstName");
```

Here, the cards are grouped programmatically based on the `Age` and `FirstName` fields.



**Note:** [View Sample in GitHub](#)

### Nested grouping of CardViewItems

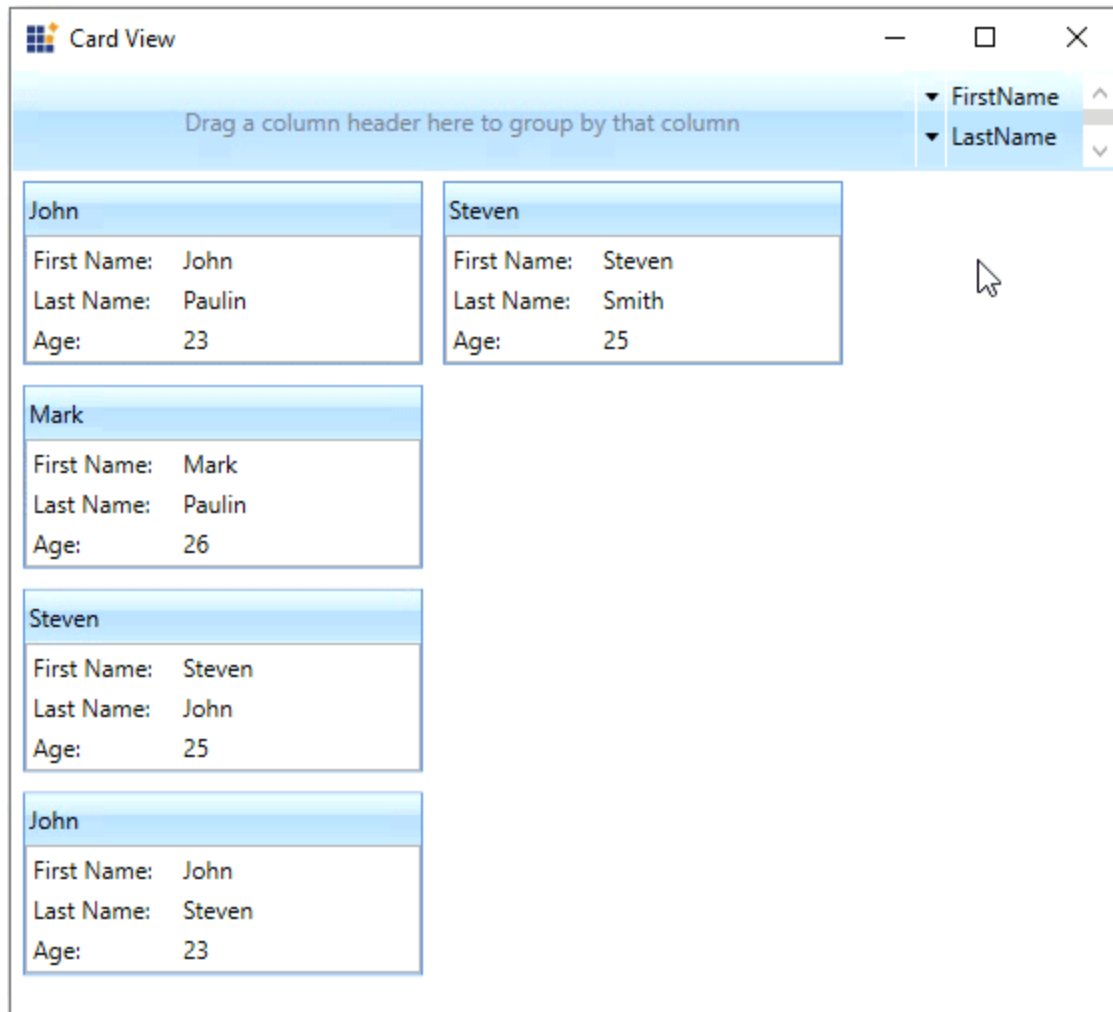
You can group the card items in multiple nested level by dragging the required fields from the list and drop it into the dropping region of the **CardView** control header. Based on the fields add into the dropping region, the card items will be grouped in nested level.

### XML

```
<syncfusion:CardView CanGroup="True"
  ItemsSource="{Binding CardViewItems}"
  Name="cardView"/>
```

### C#

```
cardView.CanGroup = true;
```



**Note:** [View Sample in GitHub](#)

Hide the grouping header

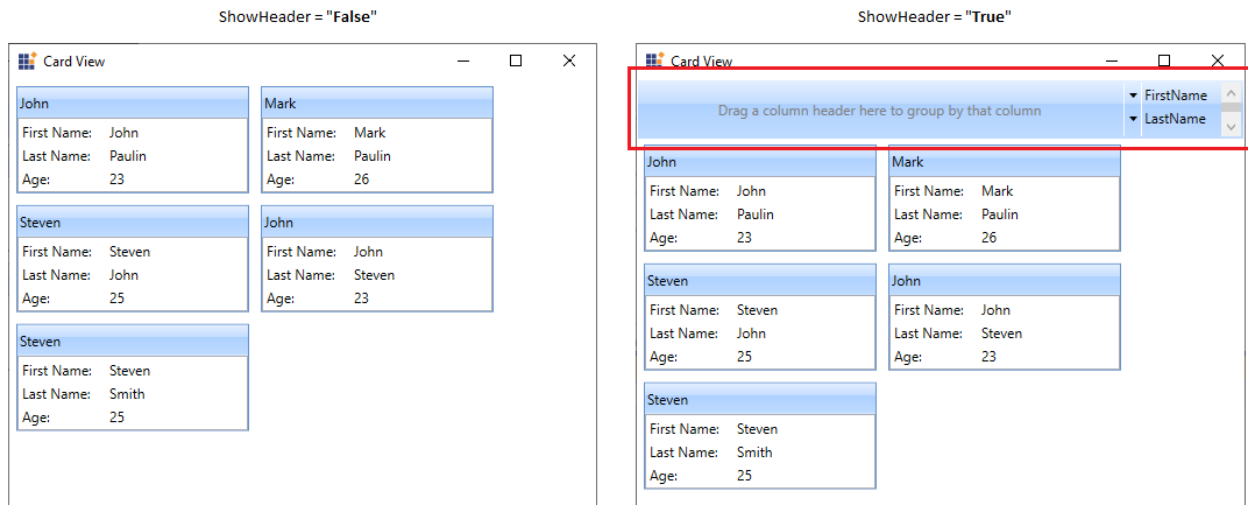
You can hide the grouping header by using the [ShowHeader](#) property value as `false`. The default value of `ShowHeader` property is `true`.

#### XML

```
<syncfusion:CardView ShowHeader="False"
ItemsSource="{Binding CardViewItems}"
Name="cardView"/>
```

#### C#

```
cardView.ShowHeader = false;
```



**Note:** [View Sample in GitHub](#)

## Sorting Mode in WPF Card View

This section describes how to enable or disable the sorting mode and perform sort operations in [CardView](#) control.

### Enable/disable the sorting mode

You can enable or disable the sorting mode of card items by setting the [CanSort](#) property value as `true` or `false`. The default value of [CanSort](#) property is `true`.

### XML

```
<syncfusion:CardView CanSort="True"
Name="cardView"/>
```

### C#

```
cardView.CanSort = true;
```

**Note:** [View Sample in GitHub](#)

### Sort the CardViewItems

You can sort the cards inside the [CardView](#) control either default order, ascending or descending order by clicking the field name listed in the header. If you want to disable the sorting, use the [CanSort](#) property value as `false`.

### C#

```
//Model.cs
public class CardViewModel
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}

//ViewModel.cs
public class ViewModel : NotificationObject
{
}
```

```

private ObservableCollection<CardViewModel> cardViewItems;
public ObservableCollection<CardViewModel> CardViewItems
{
    get { return cardViewItems; }
    set { cardViewItems = value;
        this.RaisePropertyChanged(nameof(CardViewItems)); }
}
public ViewModel()
{
    CardViewItems = new ObservableCollection<CardViewModel>();
    populateItems();
}
private void populateItems()
{
    CardViewItems.Add(new CardViewModel() { FirstName = "John", LastName =
    "Paulin", Age = 23 });
    CardViewItems.Add(new CardViewModel() { FirstName = "Mark", LastName =
    "Paulin", Age = 26 });
    CardViewItems.Add(new CardViewModel() { FirstName = "Steven", LastName =
    "John", Age = 25 });
    CardViewItems.Add(new CardViewModel() { FirstName = "John", LastName =
    "Steven", Age = 23 });
    CardViewItems.Add(new CardViewModel() { FirstName = "Steven", LastName =
    "Smith", Age = 25 });
}
}

```

### XML

```

<syncfusion:CardView CanSort="True"
ItemsSource="{Binding CardViewItems}"
Name="cardView">
    <syncfusion:CardView.HeaderTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding FirstName}"/>
        </DataTemplate>
    </syncfusion:CardView.HeaderTemplate>
    <syncfusion:CardView.ItemTemplate>
        <DataTemplate >
            <ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
                <ListBoxItem Padding="1">
                    <Grid>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="75" />
                            <ColumnDefinition />
                        </Grid.ColumnDefinitions>
                        <TextBlock Text="First Name:" />
                        <TextBlock Grid.Column="1"
Text="{Binding FirstName,
UpdateSourceTrigger=PropertyChanged}" />
                    </Grid>
                </ListBoxItem>
                <ListBoxItem Padding="1">
                    <Grid>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="75" />

```

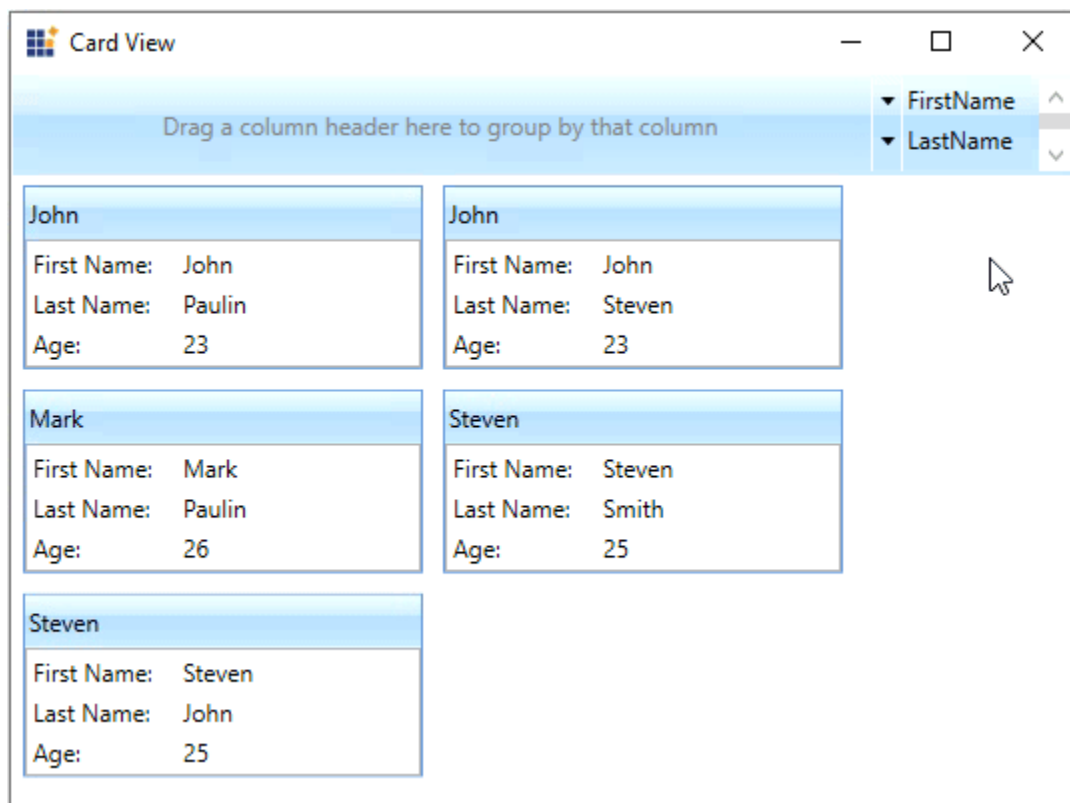
```

<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Last Name:" />
<TextBlock Grid.Column="1"
Text="{Binding LastName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Age:" />
<TextBlock Grid.Column="1"
Text="{Binding Age,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.ItemTemplate>
</syncfusion:CardView>

```

## C#

```
cardView.CanSort = true;
```





Here, `CardViewItems` sorted based on `FirstName` field.

**Note:** [View Sample in GitHub](#)

Sort the grouped `CardViewItems`

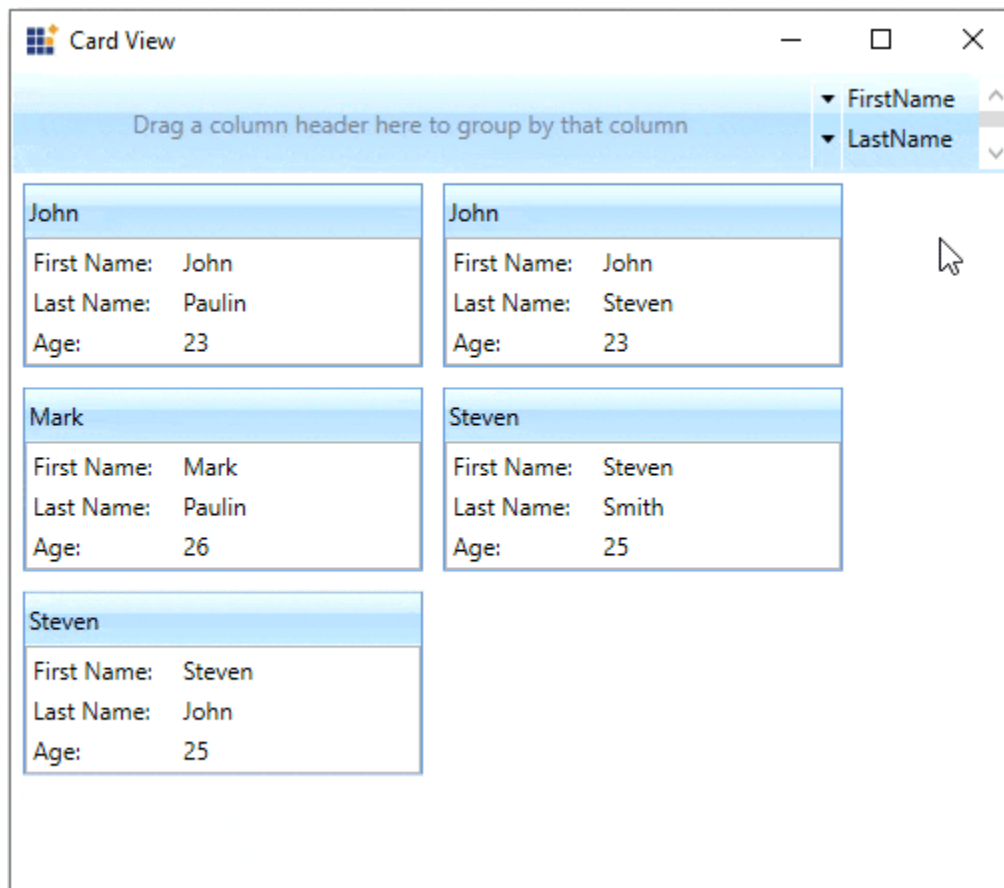
You can sort the grouped items either default order, ascending or descending order by directly clicking the grouped fields that is available in the dropping region of the `CardView` header.

#### XML

```
<syncfusion:CardView CanSort="True"
CanGroup="True"
ItemsSource="{Binding CardViewItems}"
Name="cardView"/>
```

#### C#

```
cardView.CanGroup = true;
cardView.CanSort = true;
```



**Note:** [View Sample in GitHub](#)

Hide the sorting header

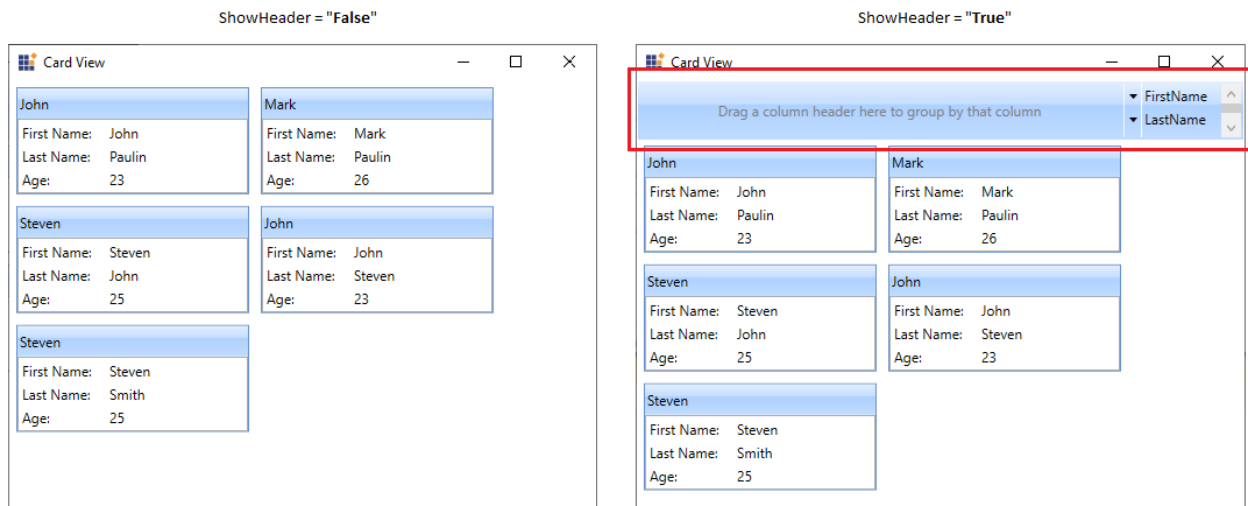
You can hide the sorting header by using the `ShowHeader` property value as `false`. The default value of `ShowHeader` property is `true`.

**XML**

```
<syncfusion:CardView ShowHeader="False"
ItemsSource="{Binding CardViewItems}"
Name="cardView"/>
```

**C#**

```
cardView.ShowHeader = false;
```



**Note:** [View Sample in GitHub](#)

## Filtering Mode in WPF Card View

This section describes how to enable or disable the filtering mode and perform filter operations in [CardView](#) control.

## Filter the CardViewItems

Cards can be filtered by the values which are given for the fields. To filter the cards, just check the values that is listed in the popup.

**C#**

```
//Model.cs
public class CardViewModel
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}

//ViewModel.cs
public class ViewModel : NotificationObject
{
    private ObservableCollection<CardViewModel> cardViewItems;
    public ObservableCollection<CardViewModel> CardViewItems
    {
        get { return cardViewItems; }
        set { cardViewItems = value;
            this.RaisePropertyChanged(nameof(CardViewItems)); }
    }
}
```

```

}
public ViewModel()
{
    CardViewItems = new ObservableCollection<CardViewModel>();
    populateItems();
}
private void populateItems()
{
    CardViewItems.Add(new CardViewModel() { FirstName = "John", LastName=
    "Paulin", Age = 23 });
    CardViewItems.Add(new CardViewModel() { FirstName = "Mark", LastName =
    "Paulin", Age = 26 });
    CardViewItems.Add(new CardViewModel() { FirstName = "Steven", LastName =
    "John", Age = 25 });
    CardViewItems.Add(new CardViewModel() { FirstName = "John", LastName =
    "Steven", Age = 23 });
    CardViewItems.Add(new CardViewModel() { FirstName = "Steven", LastName =
    "Smith", Age = 25 });
}
}

```

**XML**

```

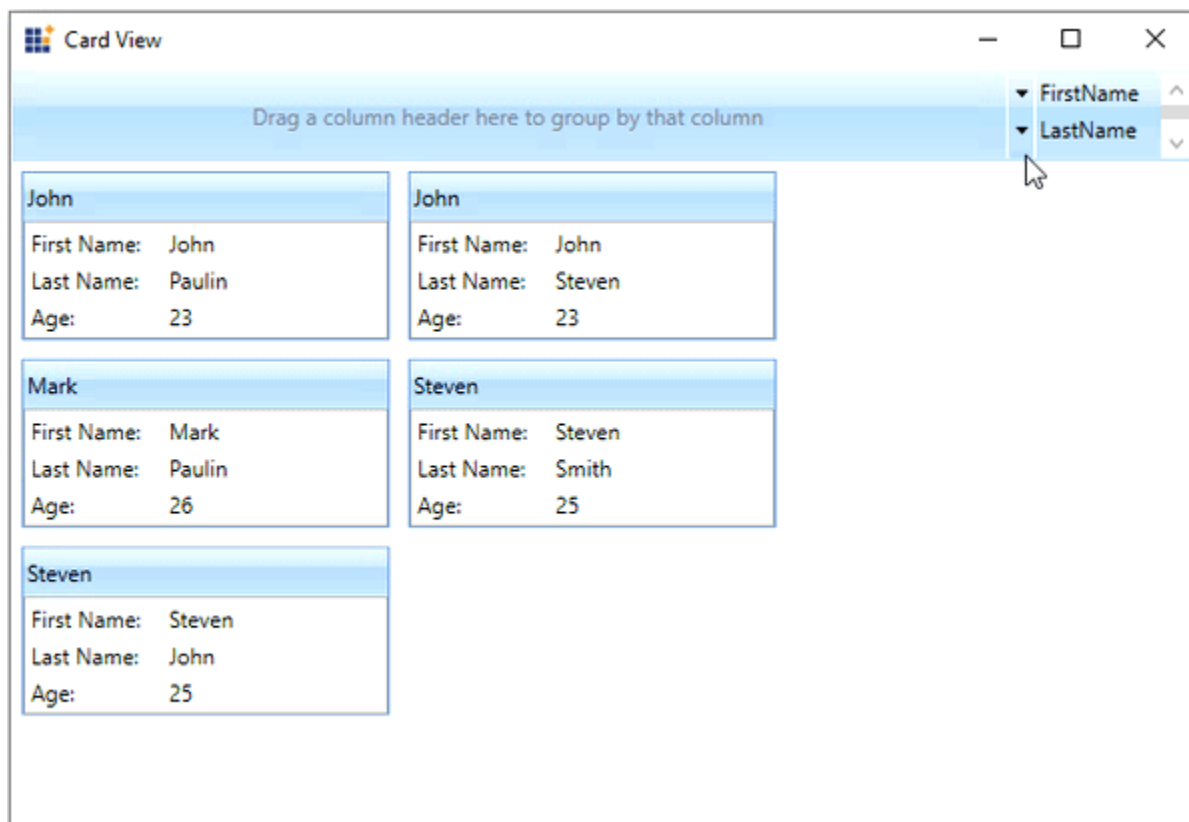
<syncfusion:CardView ItemsSource="{Binding CardViewItems}"
Name="cardView">
<syncfusion:CardView.HeaderTemplate>
<DataTemplate>
<TextBlock Text="{Binding FirstName}"/>
</DataTemplate>
</syncfusion:CardView.HeaderTemplate>
<syncfusion:CardView.ItemTemplate>
<DataTemplate >
<ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled">
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBlock Text="First Name:" />
<TextBlock Grid.Column="1"
Text="{Binding FirstName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Last Name:" />
<TextBlock Grid.Column="1"
Text="{Binding LastName,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>

```

```

</ListBoxItem>
<ListBoxItem Padding="1">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="75" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="Age:" />
<TextBlock Grid.Column="1"
Text="{Binding Age,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</ListBoxItem>
</ListBox>
</DataTemplate>
</syncfusion:CardView.ItemTemplate>
</syncfusion:CardView>

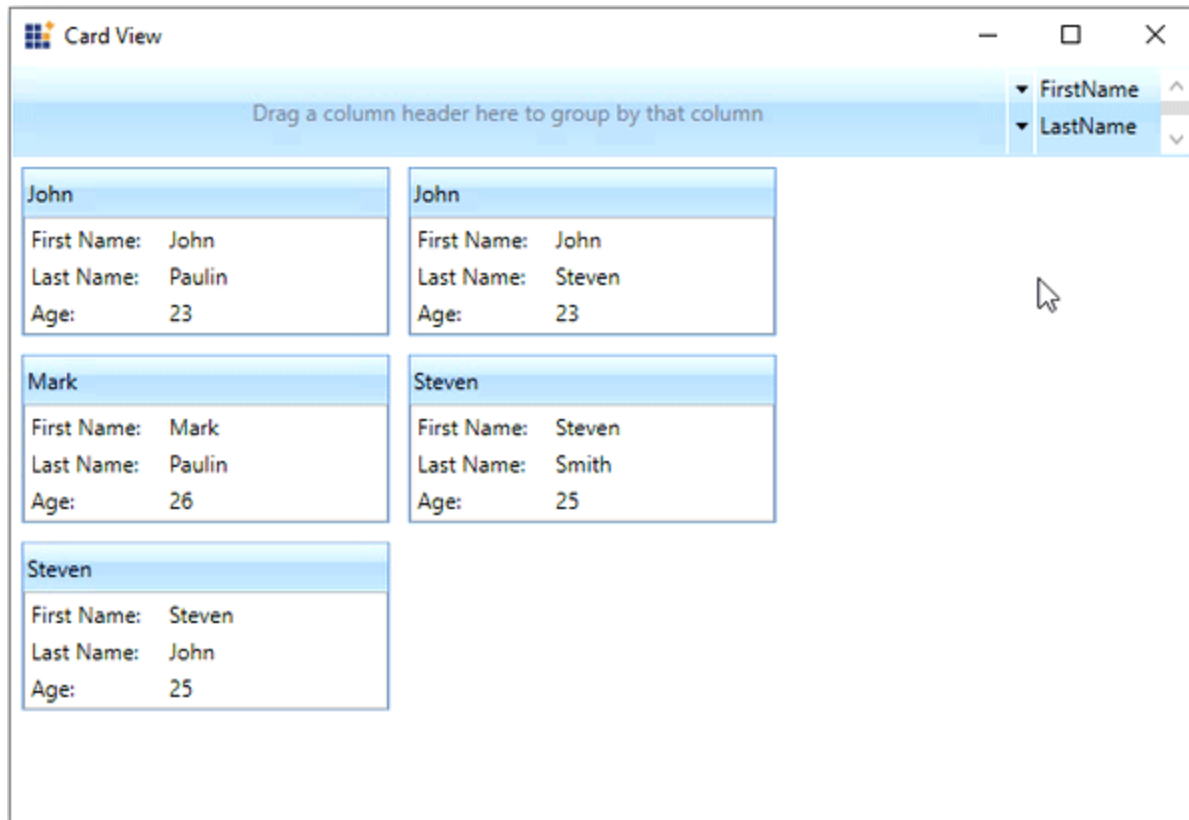
```



**Note:** [View Sample in GitHub](#)

Reset the filter

If you want to reset the filtered card selection, click the **Clear Filter** button that is placed in the bottom of field's value list popup. Then, it will remove the filtered items and shows all card items



**Note:** [View Sample in GitHub](#)

Hide the filtering header

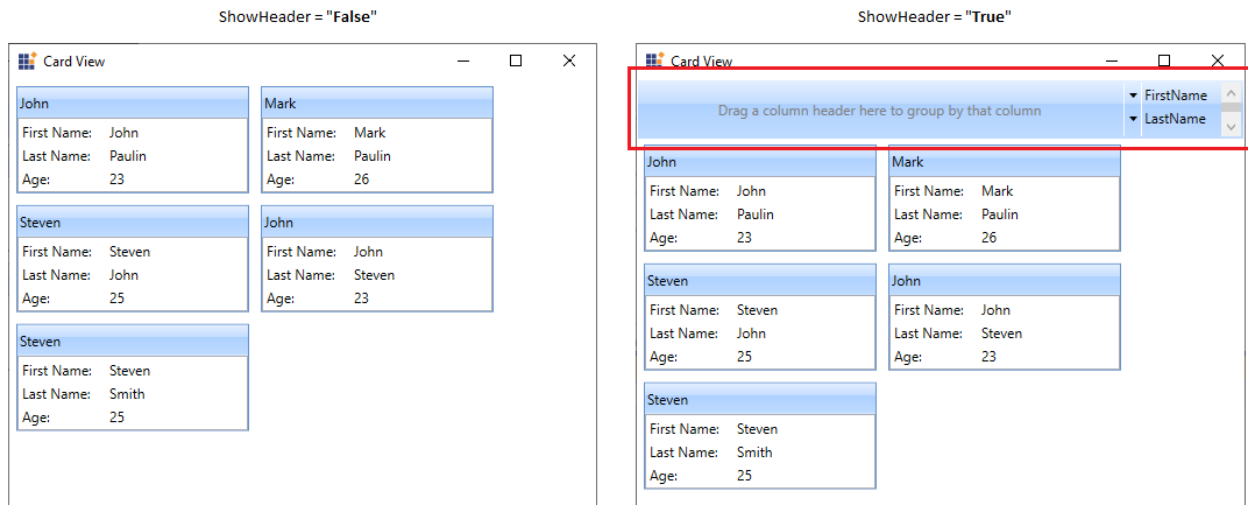
You can hide the filtering header by using the [ShowHeader](#) property value as `false`. The default value of `ShowHeader` property is `true`.

#### XML

```
<syncfusion:CardView ShowHeader="False"
ItemsSource="{Binding CardViewItems}"
Name="cardView"/>
```

#### C#

```
cardView.ShowHeader = false;
```



**Note:** [View Sample in GitHub](#)

### Localization in WPF Card View

Localization is the process of translating the application resources into different language for the specific cultures. You can localize the CardView by [adding resource file](#). Application culture can be changed by setting [CurrentUICulture](#) after `InitializeComponent` method.

Below application culture changed to French.

#### C#

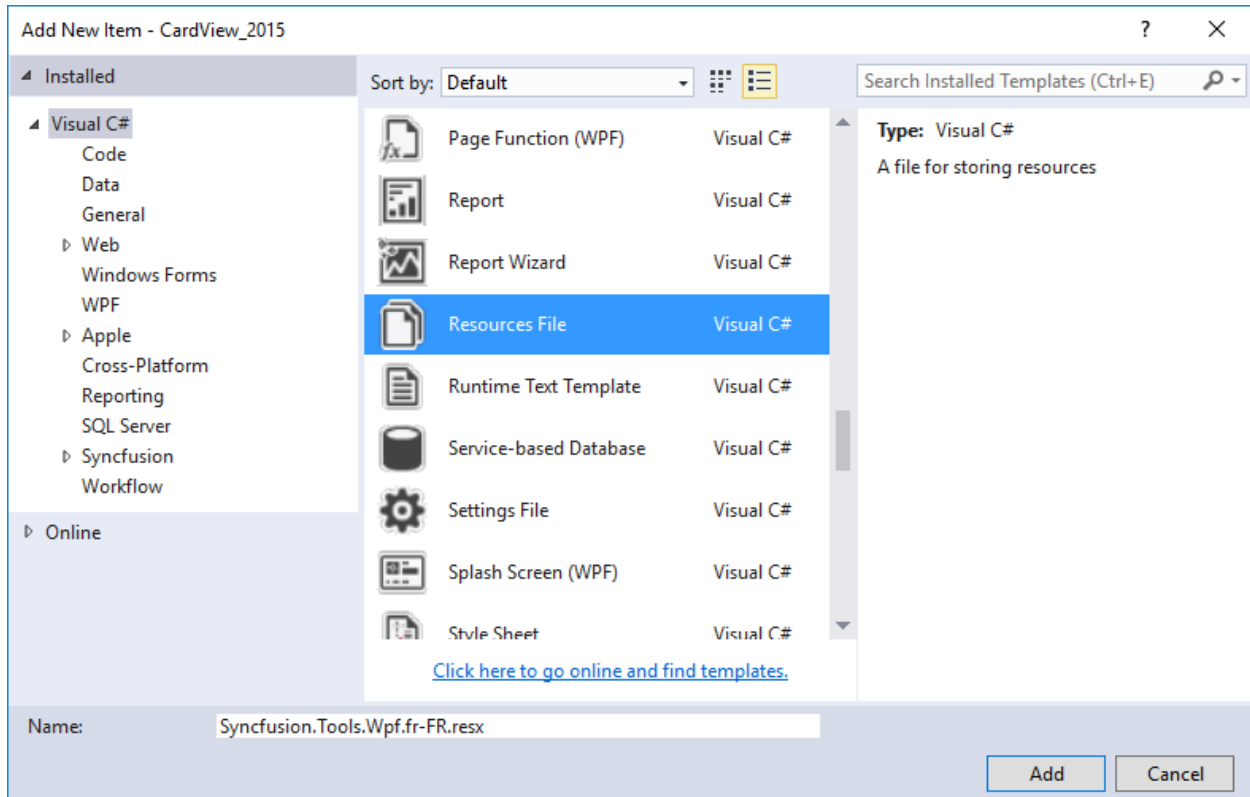
```
public MainWindow ()
{
    InitializeComponent();
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("fr-FR");
}
```

To localize the CardView based on [CurrentUICulture](#) using resource files, follow the below steps.

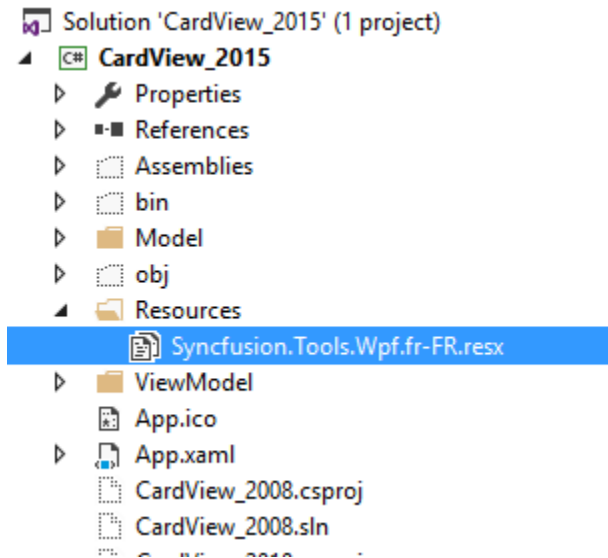
**Step 1:** Create new folder and named as **Resources** in your application.

**Step 2:** Right-click on the Resources folder, select **Add** and then **NewItem**.

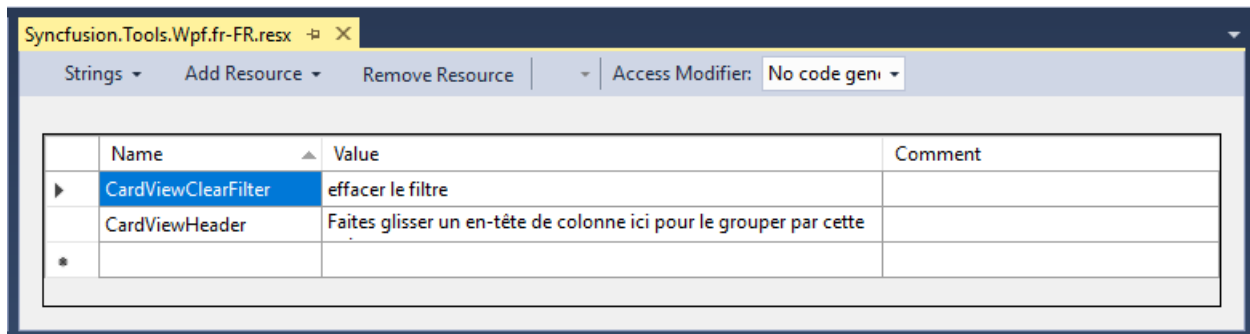
**Step 3:** In **Add New Item** wizard, select the **Resource File** option and name the filename as **Syncfusion.Tools.Wpf.<culture name>.resx**. For example, you have to give name as **Syncfusion.Tools.Wpf.fr-FR.resx** for French culture.



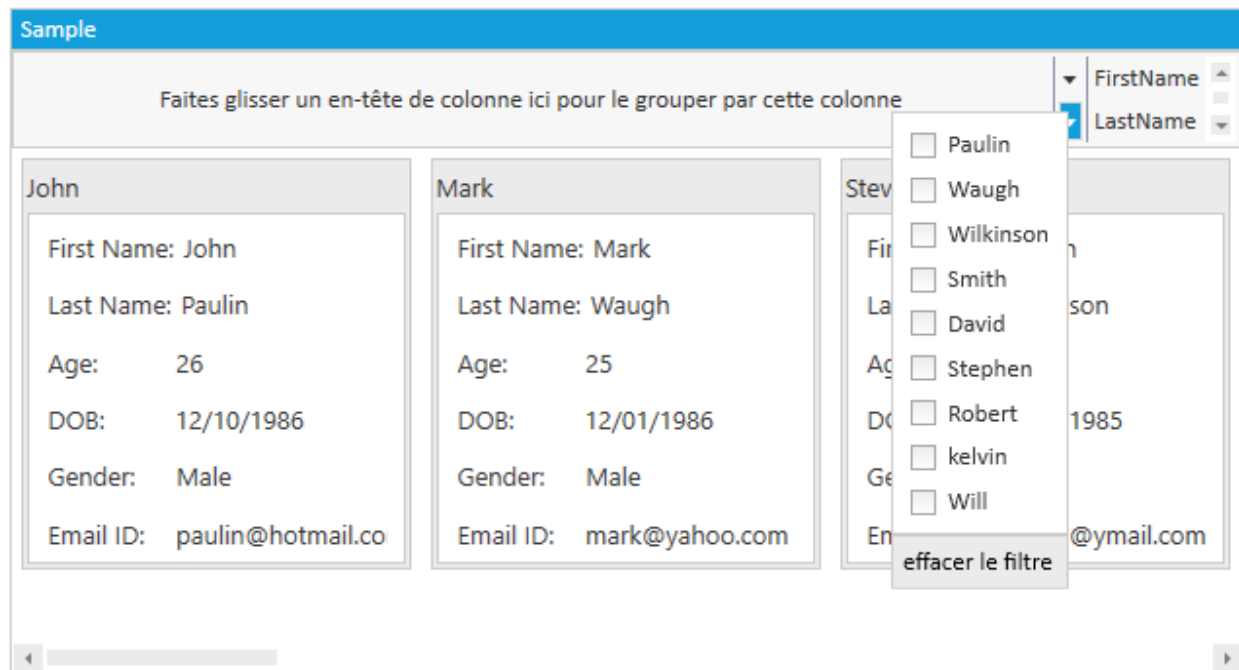
**Step 4:** Now, select **Add** option to add the resource file in **Resources** folder.



**Step 5:** Add the Name/Value pair in Resource Designer of **Syncfusion.Tools.Wpf.fr-FR.resx** file and change its corresponding value to corresponding culture.



The following screenshot shows the localized CardView control.



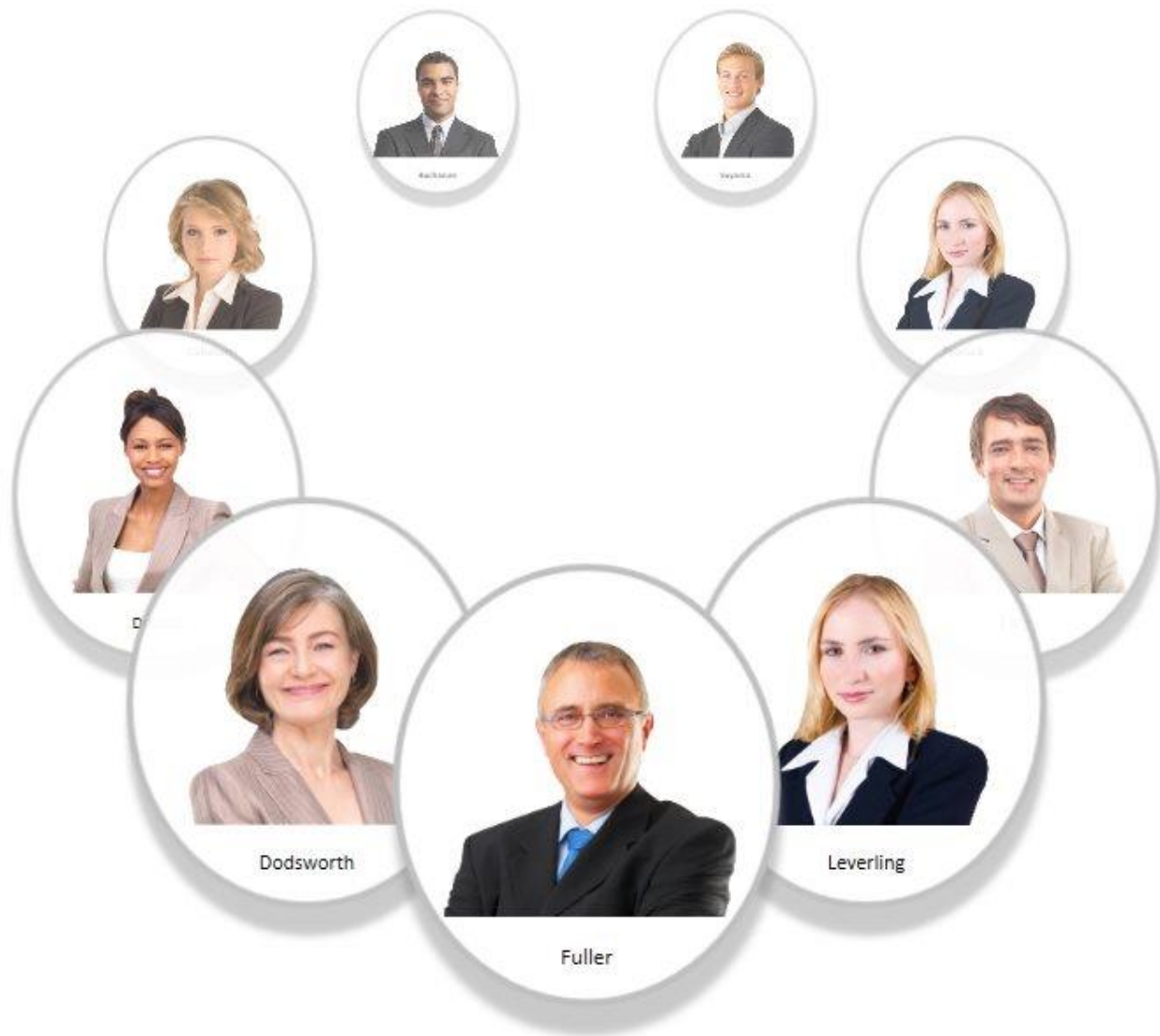
## Carousel

### WPF Carousel Overview

The [Carousel](#) is a circular conveyor used on which objects are displayed and rotated. The [Carousel](#) control provides a 3D interface for displaying objects with interactive navigation, Data Binding Path, ItemsPerPage, Scaling and Skewing.



### Structure of the Carousel Control



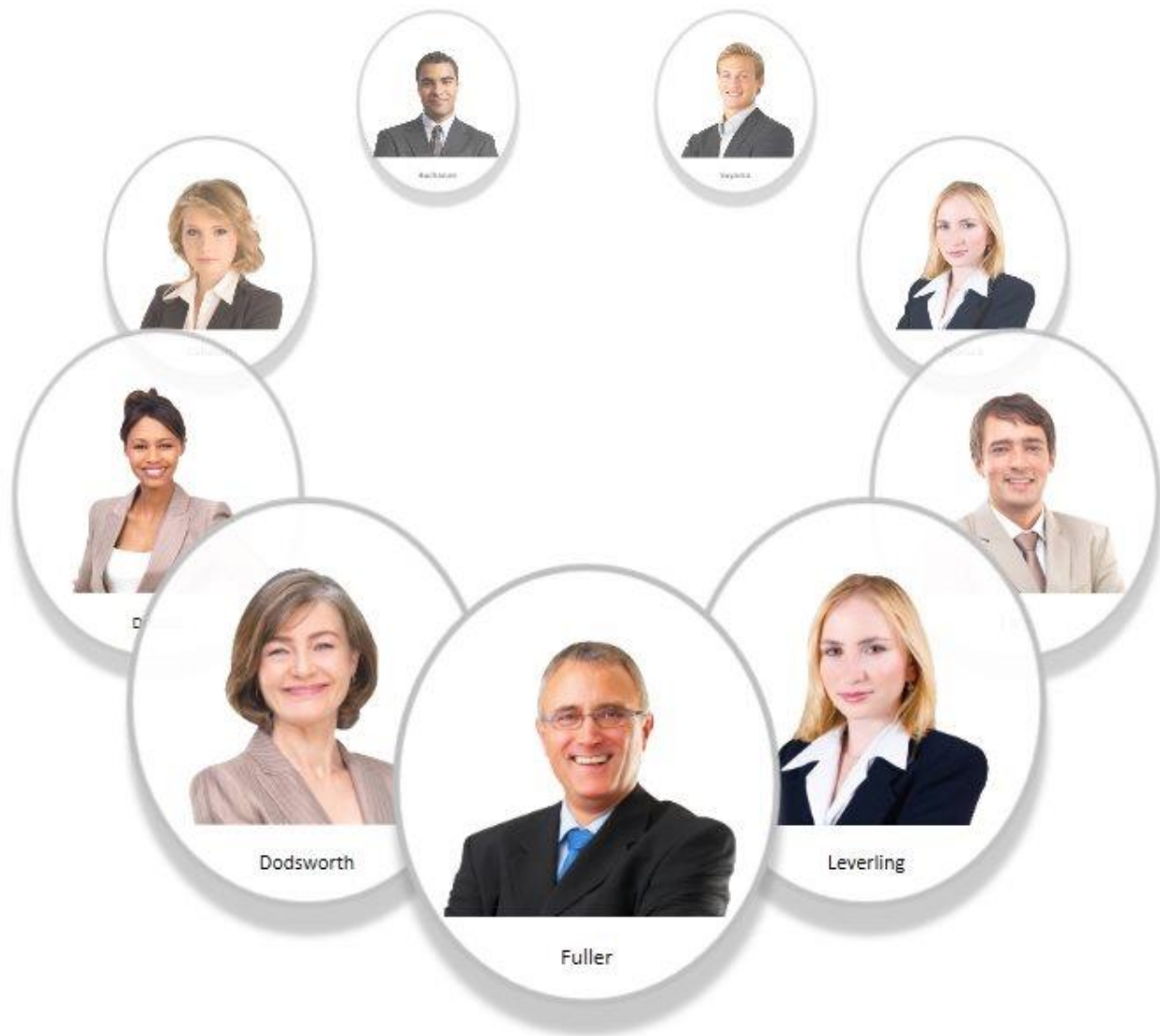
### Features

- Data Binding support
- Path support
- ItemsPerPage support
- Scaling and Skewing support
- Rotation support
- Opacity support

### Getting Started with WPF Carousel

This section explains how to create a [WPF Carousel](#) and explains about its structure.

## Structure of Carousel



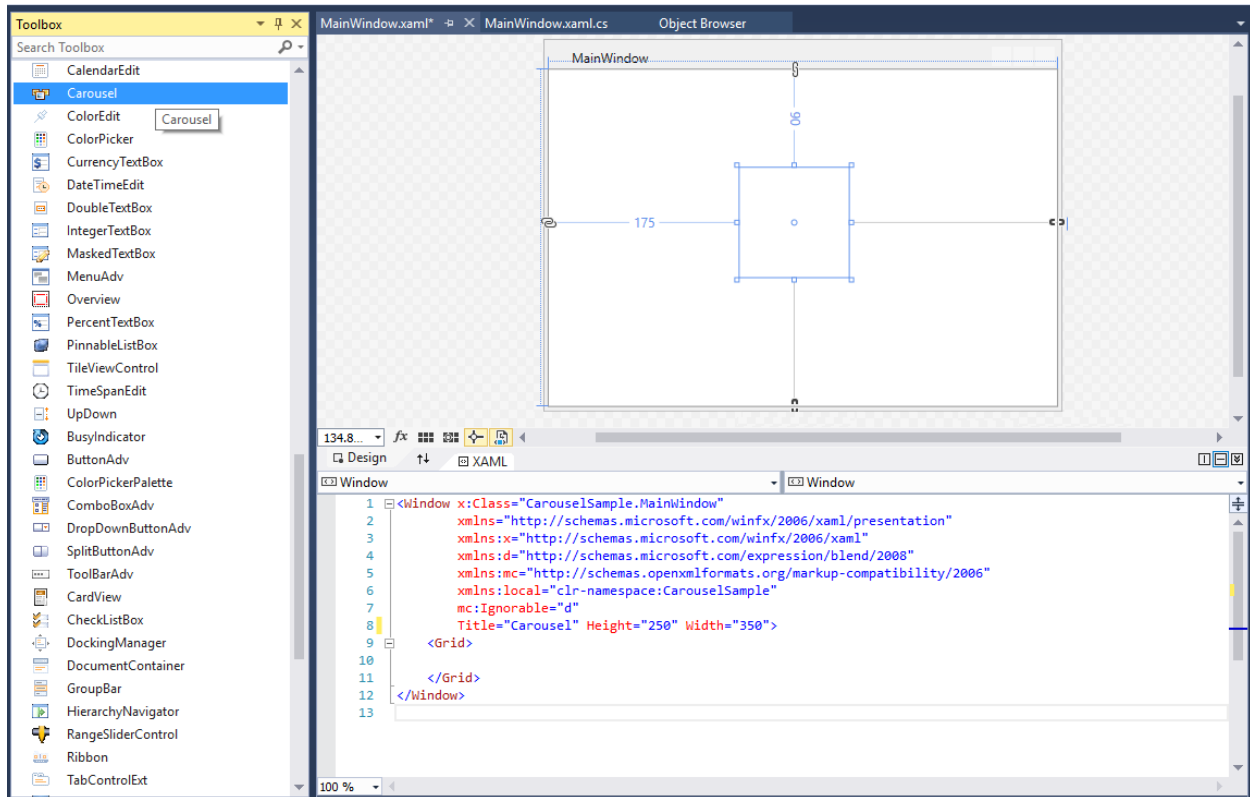
### Assembly deployment

Refer to the [Control Dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

Refer to this [documentation](#) to find more details about installing nuget packages in a WPF application.

### Adding WPF Carousel via designer

1) The WPF **Carousel** can be added to an application by dragging it from the toolbox to a designer view. The following dependent assemblies will be added automatically: \* Syncfusion.Shared.WPF



2) Set the properties for **Carousel** in design mode using the SmartTag feature.

### Adding WPF Carousel via XAML

To add the **Carousel** manually in XAML, follow these steps:

1) Create a new WPF project in Visual Studio. 2) Add the following required assembly references to the project: \* Syncfusion.Shared.WPF 3) Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf>, and declare the Carousel in XAML page.

### XML

```
<?xml version='1.0' encoding='utf-8'/?>
<Window x:Class="Carousel_sample.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:local="clr-namespace:Carousel_sample"
  xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
  mc:Ignorable="d"
  Title="MainWindow" Height="450" Width="800">
  <Grid Name="grid">
    <syncfusion:Carousel Name="carousel"
      Height="200"
      Width="200"/>
  </Grid>
</Window>
```

### Adding WPF Carousel via C#

To add the **Carousel** manually in C#, follow these steps:

1) Create a new WPF application via Visual Studio. 2) Add the following required assembly references to the project: \* Syncfusion.Shared.WPF 3) Include the required namespace.

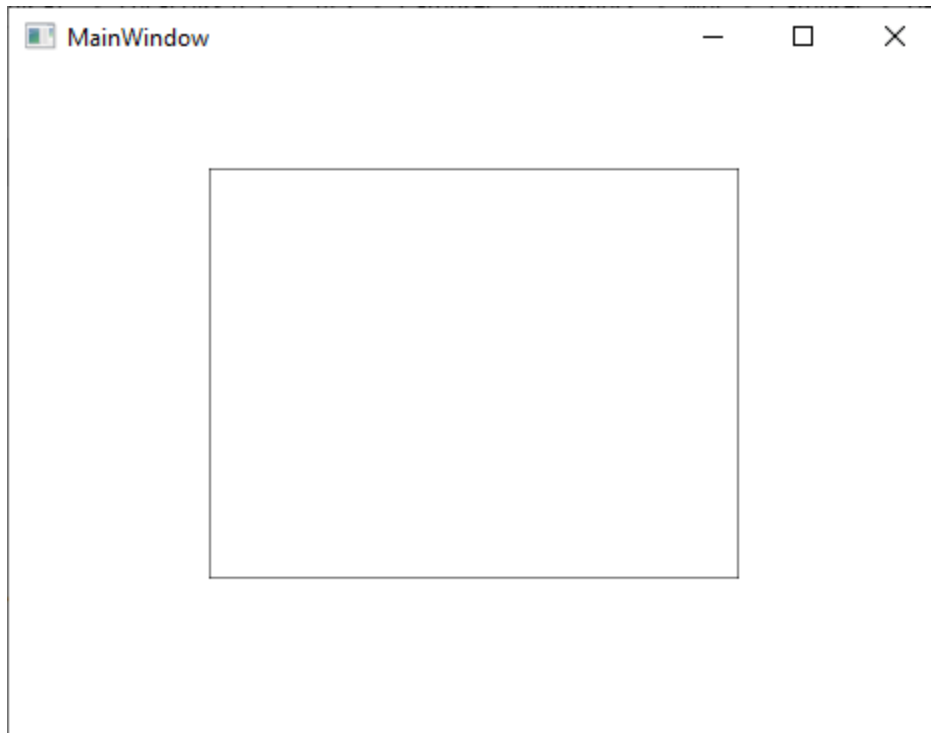
**C#**

```
using Syncfusion.Windows.Shared;
```

4) Create an instance of `Carousel`, and add it to the window.

**C#**

```
// Creating an instance of the Carousel  
Carousel carousel = new Carousel();  
// Setting height and width to Carousel  
carousel.Height = 200;  
carousel.Width = 260;
```



**Note:** [View Sample in GitHub](#)

Populating items using `CarouselItem`

You can add the carousel items inside the control using the [CarouselItem](#) property.

**XML**

```
<syncfusion:Carousel x:Name="carousel"  
Height="700" Width="500">  
  <syncfusion:CarouselItem>  
    <syncfusion:CarouselItem.Content>  
      <Viewbox Height="100" Width="100">  
        <Image Source="Images/Buchanan.png" />  
      </Viewbox>  
    </syncfusion:CarouselItem.Content>  
  </syncfusion:CarouselItem>  
</syncfusion:Carousel>
```

```

</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/Callahan.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/Davolio-1.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/Callahan.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/dodsworth.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/Fuller.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/King.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/Leverling.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
</syncfusion:Carousel>

```

## C#

```

Carousel carousel = new Carousel()
{

```

```
Width=700,
Height = 500
};
Image image = new Image();
Image image1 = new Image();
Image image2 = new Image();
Image image3 = new Image();
Image image4 = new Image();
Image image5 = new Image();
Image image6 = new Image();
Image image7 = new Image();
BitmapImage bitimg1 = new BitmapImage(new
Uri("/Sample;component/Images/1.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg2 = new BitmapImage(new
Uri("/Sample;component/Images/2.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg3 = new BitmapImage(new
Uri("/Sample;component/Images/3.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg4 = new BitmapImage(new
Uri("/Sample;component/Images/4.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg5 = new BitmapImage(new
Uri("/Sample;component/Images/5.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg6 = new BitmapImage(new
Uri("/Sample;component/Images/6.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg7 = new BitmapImage(new
Uri("/Sample;component/Images/7.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg8 = new BitmapImage(new
Uri("/Sample;component/Images/8.png", UriKind.RelativeOrAbsolute));
image.Source = bitimg1 as ImageSource;
image1.Source = bitimg2 as ImageSource;
image2.Source = bitimg3 as ImageSource;
image3.Source = bitimg4 as ImageSource;
image4.Source = bitimg5 as ImageSource;
image5.Source = bitimg6 as ImageSource;
image6.Source = bitimg7 as ImageSource;
image7.Source = bitimg8 as ImageSource;
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image1 } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image2 } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image3 } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image4 } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image5 } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image6 } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image7 } });
```



---

**Note:** [View Sample in GitHub](#)

---

Populating items using collection binding

You can populate items to the [Carousel](#) control by setting the collection value to the `ItemsSource` property.

#### **C#**

```
//Model.cs
public class CarouselModel {
    public string Header { get; set; }
}

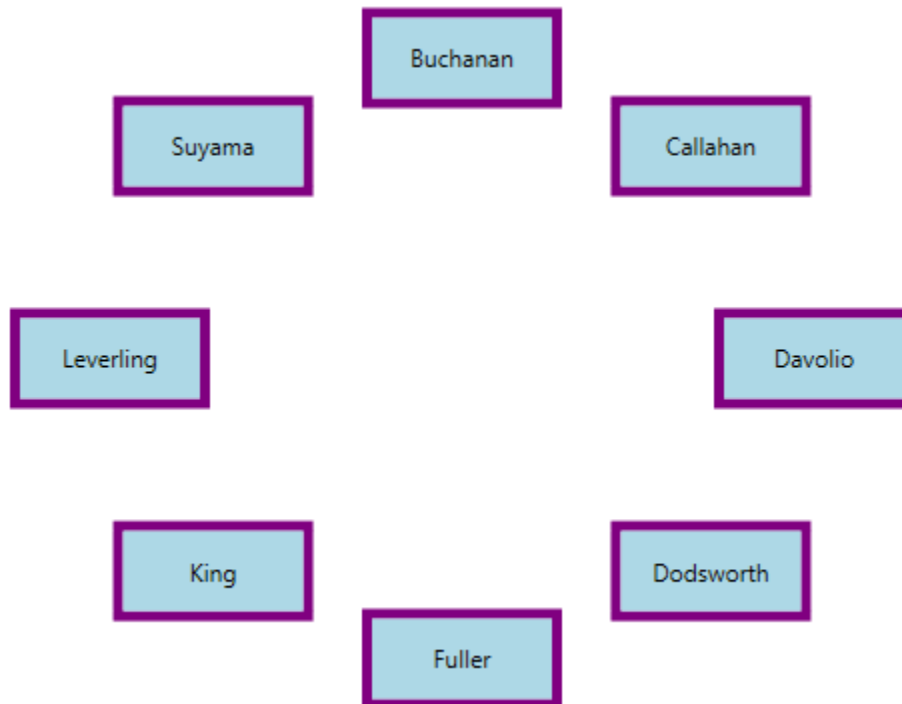
//ViewModel.cs
public class ViewModel {
    private ObservableCollection<CarouselModel> collection;
    public ObservableCollection<CarouselModel> HeaderCollection
    {
        get {
            return collection;
        }
        set {
            collection = value;
        }
    }

    public ViewModel() {
        HeaderCollection = new ObservableCollection<CarouselModel>();
        HeaderCollection.Add(new CarouselModel() { Header = "Buchanan" });
        HeaderCollection.Add(new CarouselModel() { Header = "Callahan" });
        HeaderCollection.Add(new CarouselModel() { Header = "Davolio" });
        HeaderCollection.Add(new CarouselModel() { Header = "Dodsworth" });
    }
}
```

```
HeaderCollection.Add(new CarouselModel() { Header = "Fuller" });
HeaderCollection.Add(new CarouselModel() { Header = "King" });
HeaderCollection.Add(new CarouselModel() { Header = "Leverling" });
HeaderCollection.Add(new CarouselModel() { Header = "Suyama" });
}
```

**XML**

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:Carousel Name="Carousel"
ItemsSource="{Binding HeaderCollection}">
<syncfusion:Carousel.ItemTemplate>
<DataTemplate>
<Border Height="50"
Width="100"
BorderBrush="Purple"
BorderThickness="5"
Background="LightBlue">
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
Text="{Binding Header}"/>
</Border>
</DataTemplate>
</syncfusion:Carousel.ItemTemplate>
</syncfusion:Carousel>
</Grid>
```





---

**Note:** [View Sample in GitHub](#)

---

### Select carousel item

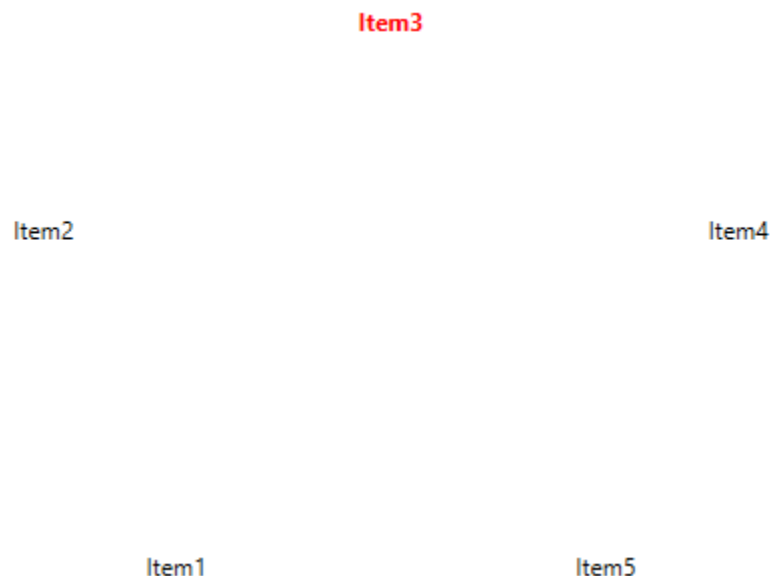
You can select a carousel item by mouse click on the specific item. You can get the selected item and its value by using the [SelectedItem](#) and [SelectedValue](#) properties. You can also get the selected item index by using the [SelectedIndex](#) property. You can only select a single item at a time.

### Select carousel item programmatically using property

You can select a particular carousel item programmatically by using the [CarouselItem.IsSelected](#) property.

### XML

```
<Window.Resources>
<Style x:Key="selecteditemStyle" TargetType="syncfusion:CarouselItem" >
<Style.Triggers>
<Trigger Property="IsSelected" Value="True">
<Setter Property="Foreground" Value="Red"/>
<Setter Property="FontWeight" Value="Bold"/>
</Trigger>
</Style.Triggers>
</Style>
</Window.Resources>
<Grid>
<syncfusion:Carousel ItemContainerStyle="{StaticResource selecteditemStyle}"
x:Name="carousel">
<syncfusion:CarouselItem Content="Item1"/>
<syncfusion:CarouselItem Content="Item2"/>
<syncfusion:CarouselItem Content="Item3" IsSelected="True"/>
<syncfusion:CarouselItem Content="Item4"/>
<syncfusion:CarouselItem Content="Item5"/>
</syncfusion:Carousel>
```




---

**Note:** [View Sample in GitHub](#)

---

*Select carousel item programmatically using command and methods*

You can select a previous, next, first or last carousel items programmatically by using the commands and methods.

- [SelectFirstItemCommand](#) or [SelectFirstItem\(\)](#) - To select the first item.
- [SelectLastItemCommand](#) or [SelectLastItem\(\)](#) - To select the last item.
- [SelectPreviousItemCommand](#) or [SelectPreviousItem](#) - To select the previous item from the currently selected item
- [SelectNextItemCommand](#) or [SelectNextItem\(\)](#) - To select the next item from the currently selected item.
- [SelectPreviousPageCommand](#) or [SelectPreviousPage\(\)](#) - To select the previous page item.
- [SelectNextPageCommand](#) or [SelectNextPage\(\)](#) - To select the next page item.

---

**Note:** [View Sample in GitHub](#)

---

#### Rotate carousel item

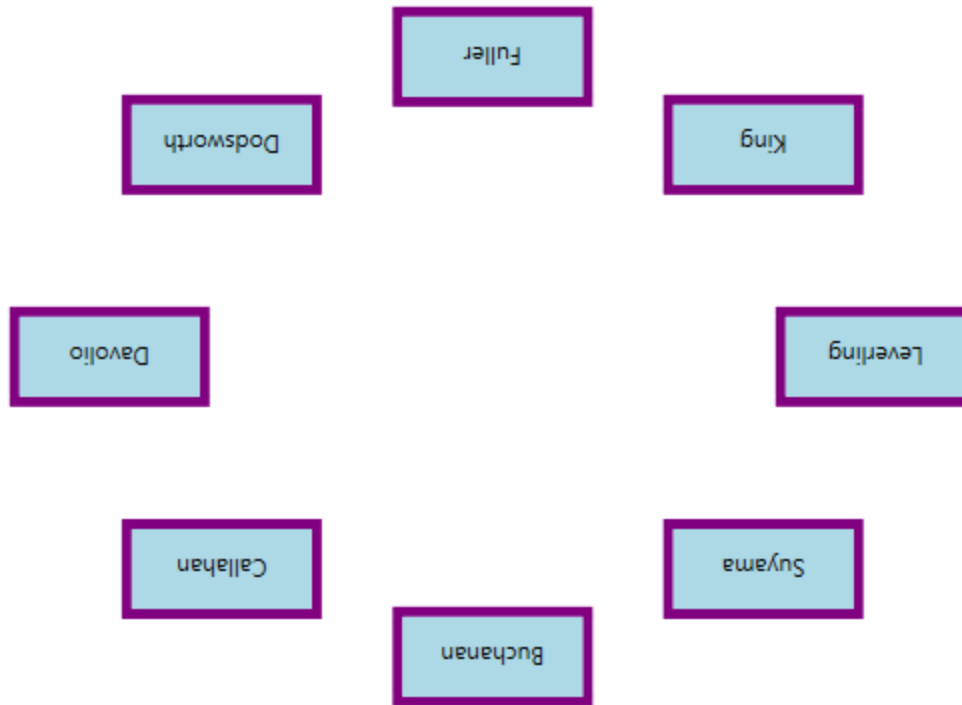
You can place the carousel item with a specific rotation angle by using the [RotationAngle](#) property. You can change the rotation speed of the carousel items by using the [RotationSpeed](#) property. You can also disable the rotate animation by using the [EnableRotationAnimation](#) property value as `false`. The default value of `RotationAngle` property is `0` and `RotationSpeed` property is `200` and `EnableRotationAnimation` property is `true`.

#### XML

```
<syncfusion:Carousel RotationAngle="180"
RotationSpeed="500"
EnableRotationAnimation="True"
ItemsSource="{Binding HeaderComponent}"
Name="carousel" />
```

#### C#

```
carousel.RotationAngle = 180;
carousel.RotationSpeed = 500;
carousel.EnableRotationAnimation = true;
```



---

**Note:** [View Sample in GitHub](#)

---

#### Resize the carousel items

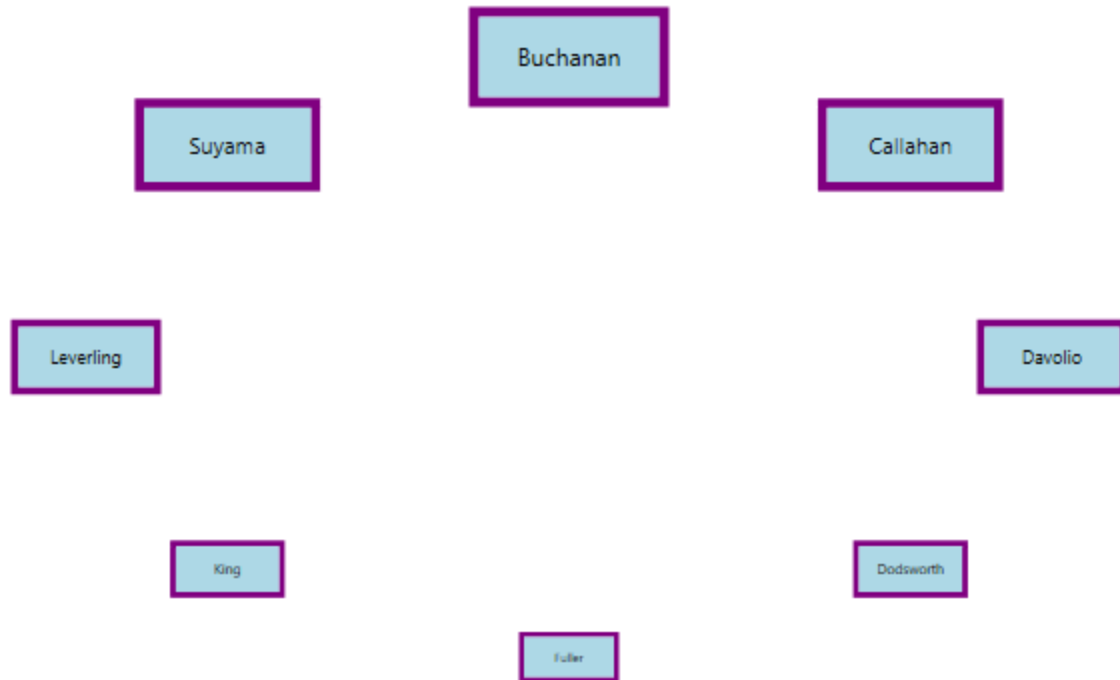
If you want to change the size of the carousel items except the selected item, use the [ScaleFraction](#) property. You can disable it by setting the [ScalingEnabled](#) property value as `false`. Value range of `ScaleFraction` property is 0 to 1. The default value `ScaleFraction` property is 0 and `ScalingEnabled` property is `true`.

#### XML

```
<syncfusion:Carousel ScaleFraction="0.50"
ItemsSource="{Binding HeaderComponent}"
Name="carousel"/>
```

#### C#

```
carousel. ScaleFraction = 0.50;
```



---

**Note:** [View Sample in GitHub](#)

---

#### Change radius of carousel item

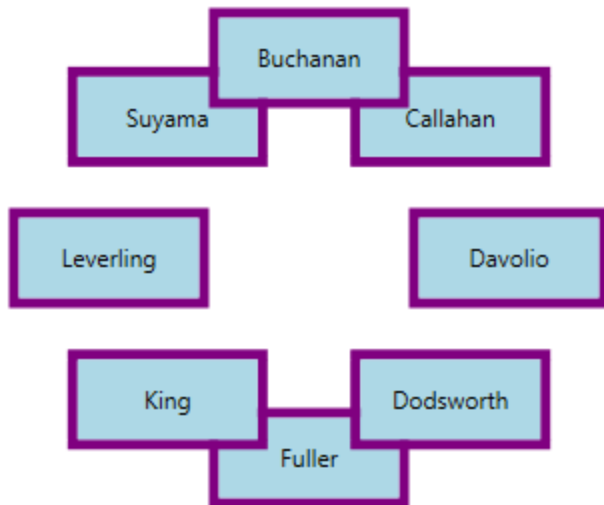
You can change the radius of the `Carousel` control by setting the value to the [RadiusX](#) and [RadiusY](#) properties. Based on the radius points, items are arranged. The default value of `RadiusX` property is 250 and `RadiusY` property is 150.

#### XML

```
<syncfusion:Carousel RadiusX="100"  
RadiusY="100"  
ItemsSource="{Binding HeaderCollection}"  
Name="carousel"/>
```

#### C#

```
carousel.RadiusX = 100;  
carousel.RadiusY = 100;
```



**Note:** [View Sample in GitHub](#)

#### Custom UI of carousel item

You can customize the appearance of each carousel item by using the `ItemTemplate` property. If you want to change the appearance of particular carousel item appearance, use `ItemTemplateSelector` property. The `DataContext` of the `ItemTemplate` property is `CarouselItem`.

#### C#

```
//Model.cs
public class CarouselModel {
    public string Header { get; set; }
}

//ViewModel.cs
public class ViewModel {
    private ObservableCollection<CarouselModel> collection;
    public ObservableCollection<CarouselModel> HeaderCollection {
        get {
            return collection;
        }
        set {
            collection = value;
        }
    }

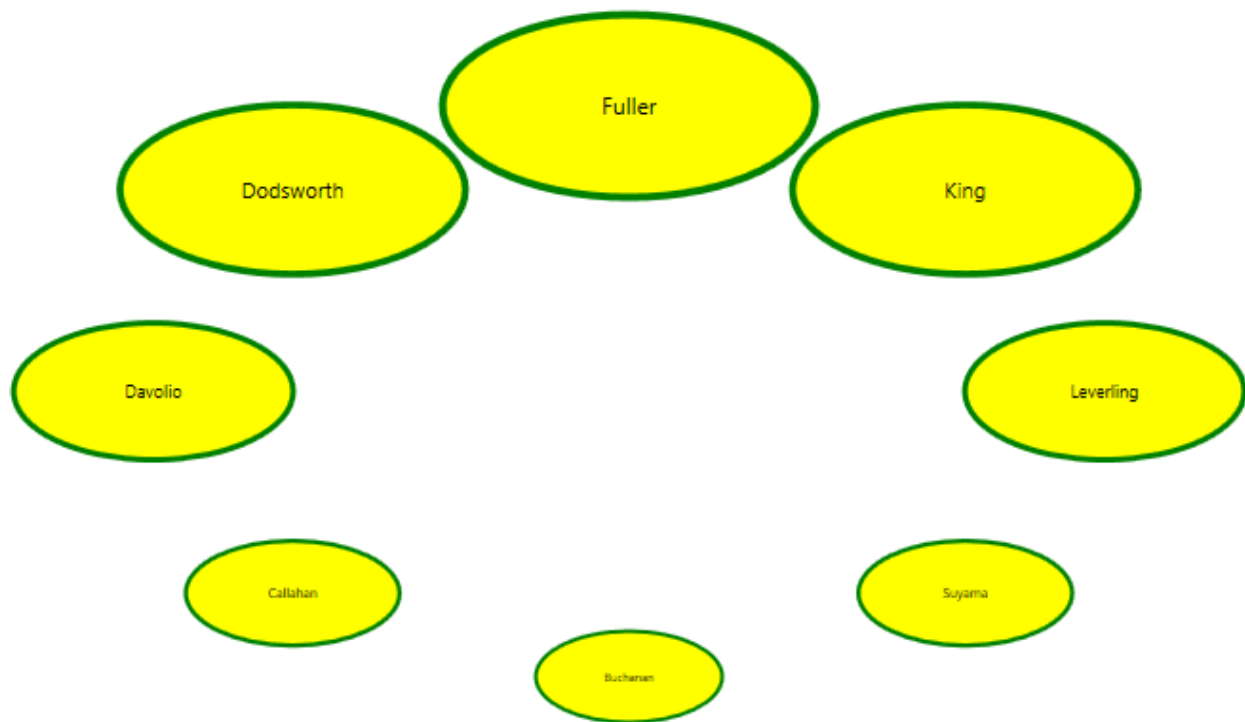
    public ViewModel() {
        HeaderCollection = new ObservableCollection<CarouselModel>();
        HeaderCollection.Add(new CarouselModel() { Header = "Buchanan" });
        HeaderCollection.Add(new CarouselModel() { Header = "Callahan" });
        HeaderCollection.Add(new CarouselModel() { Header = "Davolio" });
        HeaderCollection.Add(new CarouselModel() { Header = "Dodsworth" });
        HeaderCollection.Add(new CarouselModel() { Header = "Fuller" });
        HeaderCollection.Add(new CarouselModel() { Header = "King" });
        HeaderCollection.Add(new CarouselModel() { Header = "Leverling" });
        HeaderCollection.Add(new CarouselModel() { Header = "Suyama" });
    }
}
```

#### XML

```

<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:Carousel ItemsSource="{Binding HeaderComponent}"
ScaleFraction="0.5"
Name="carousel" >
<syncfusion:Carousel.ItemTemplate>
<DataTemplate>
<Grid>
<Ellipse Width="200" Height="100"
Stroke="Green"
StrokeThickness="4"
Fill="Yellow"/>
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
Text="{Binding Header}" />
</Grid>
</DataTemplate>
</syncfusion:Carousel.ItemTemplate>
</syncfusion:Carousel>
</Grid>

```



**Note:** [View Sample in GitHub](#)

#### Custom display path for carousel items

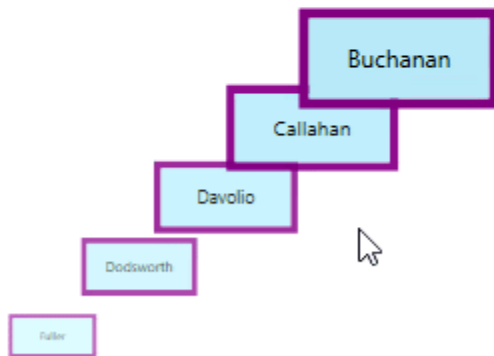
You can change the custom display path of the carousel items by using the [Carousel.Path](#) property. You can enable it by setting the [VisualMode](#) property as `VisualMode.CustomPath`. The default value of `Carousel.Path` property is `null` and `VisualMode` property is `Standard`.

#### XML

```
<syncfusion:Carousel VisualMode="CustomPath"
ItemsSource="{Binding HeaderComponent}"
Name="carousel">
<syncfusion:Carousel.Path>
<Path Data="M0,100 L100,20"
Stroke="Blue"
StrokeThickness="2"
HorizontalAlignment="Stretch"
VerticalAlignment="Stretch"/>
</syncfusion:Carousel.Path>
</syncfusion:Carousel>
```

**C#**

```
carousel.VisualMode = VisualMode.CustomPath;
```



**Note:** [View Sample in GitHub](#)

Number of items to be visible in Page

By default, all the items are displayed in the **Carousel** control. If you will be added more items and wants to display less number of items at a time, use the [ItemsPerPage](#) property. **ItemsPerPage** is effective only on **VisualMode.CustomPath** view mode. The default value of **ItemsPerPage** property is -1.

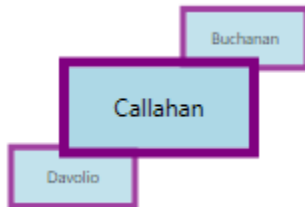
**XML**

```
<syncfusion:Carousel ItemsPerPage="3"
VisualMode="CustomPath"
ItemsSource="{Binding HeaderComponent}"
```

```
Name="carousel"/>
```

### C#

```
carousel.ItemsPerPage = 3;  
carousel.VisualMode = VisualMode.CustomPath;
```



**Note:** [View Sample in GitHub](#)

Selected item changed notification

The selected item changed in `Carousel` can be examined using [SelectionChanged](#) event. The `SelectionChanged` event contains the old and newly selected item in the `OldValue` and `NewValue` properties.

### XML

```
<syncfusion:Carousel SelectionChanged="Carousel_SelectionChanged"  
ItemsSource="{Binding HeaderCollection}"  
Name="carousel"/>
```

### C#

```
Carousel carousel = new Carousel();  
carousel.SelectionChanged += Carousel_SelectionChanged;
```

You can handle the event as follows,

### C#

```
private void Carousel_SelectionChanged(DependencyObject d,  
DependencyPropertyChangedEventArgs e) {  
    //Get old and new selected carousel item  
    var oldValue = e.OldValue;  
    var newValue = e.NewValue;  
}
```

### Theme

Carousel supports various built-in themes. Refer to the below links to apply themes for the Carousel,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)





## Populating Items in WPF Carousel

This section explains how to populate items and different UI customization features available in [Carousel](#) control.

### Populating items using CarouselItem

You can add the carousel items inside the control using the [CarouselItem](#) property.

#### XML

```
<syncfusion:Carousel x:Name="carousel"
Height="700" Width="500">
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/Buchanan.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/Callahan.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/Davolio-1.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
```

```

<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/Callahan.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/dodsworth.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/Fuller.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/King.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
<syncfusion:CarouselItem>
<syncfusion:CarouselItem.Content>
<Viewbox Height="100" Width="100">
<Image Source="Images/Leverling.png"/>
</Viewbox>
</syncfusion:CarouselItem.Content>
</syncfusion:CarouselItem>
</syncfusion:Carousel>

```

**C#**

```

Carousel carousel = new Carousel()
{
    Width=700,
    Height = 500
};
Image image = new Image();
Image image1 = new Image();
Image image2 = new Image();
Image image3 = new Image();
Image image4 = new Image();
Image image5 = new Image();
Image image6 = new Image();
Image image7 = new Image();
BitmapImage bitimg1 = new BitmapImage(new
Uri("/Sample;component/Images/1.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg2 = new BitmapImage(new
Uri("/Sample;component/Images/2.png", UriKind.RelativeOrAbsolute));

```

```
BitmapImage bitimg3 = new BitmapImage(new
Uri("/Sample;component/Images/3.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg4 = new BitmapImage(new
Uri("/Sample;component/Images/4.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg5 = new BitmapImage(new
Uri("/Sample;component/Images/5.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg6 = new BitmapImage(new
Uri("/Sample;component/Images/6.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg7 = new BitmapImage(new
Uri("/Sample;component/Images/7.png", UriKind.RelativeOrAbsolute));
BitmapImage bitimg8 = new BitmapImage(new
Uri("/Sample;component/Images/8.png", UriKind.RelativeOrAbsolute));
image.Source = bitimg1 as ImageSource;
image1.Source = bitimg2 as ImageSource;
image2.Source = bitimg3 as ImageSource;
image3.Source = bitimg4 as ImageSource;
image4.Source = bitimg5 as ImageSource;
image5.Source = bitimg6 as ImageSource;
image6.Source = bitimg7 as ImageSource;
image7.Source = bitimg8 as ImageSource;
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image1 } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image2 } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image3 } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image4 } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image5 } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image6 } });
carousel.Items.Add(new CarouselItem() { Content = new Viewbox() { Child =
image7 } });
```



---

**Note:** [View Sample in GitHub](#)

---

Populating items using collection binding

You can populate items to the `Carousel` control by setting the collection value to the `ItemsSource` property.

#### **C#**

```
//Model.cs
public class CarouselModel {
    public string Header { get; set; }
}

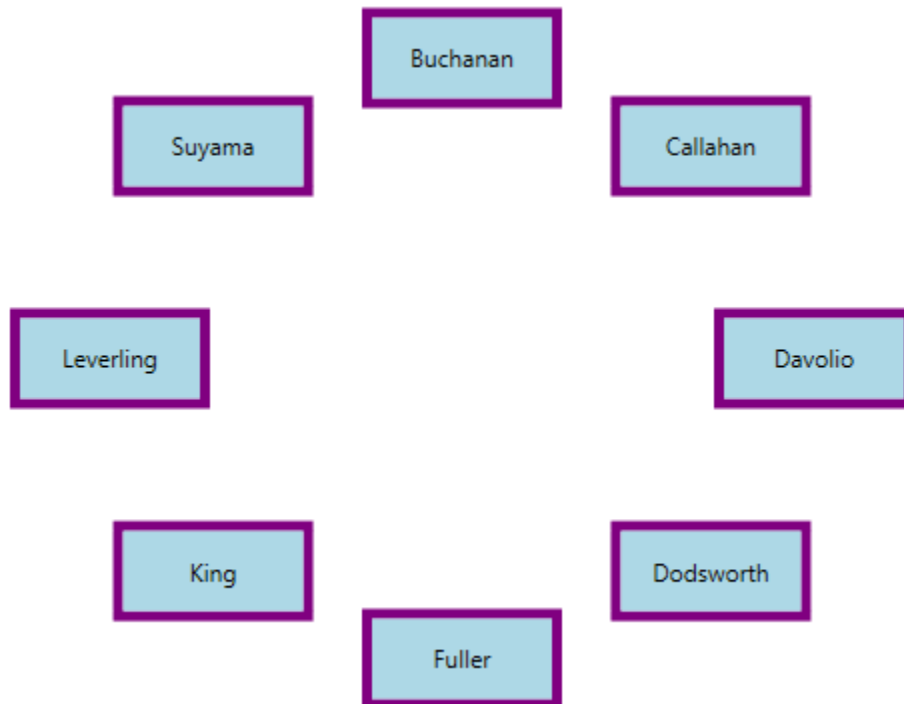
//ViewModel.cs
public class ViewModel {
    private ObservableCollection<CarouselModel> collection;
    public ObservableCollection<CarouselModel> HeaderCollection
    {
        get {
            return collection;
        }
        set {
            collection = value;
        }
    }

    public ViewModel() {
        HeaderCollection = new ObservableCollection<CarouselModel>();
        HeaderCollection.Add(new CarouselModel() { Header = "Buchanan" });
        HeaderCollection.Add(new CarouselModel() { Header = "Callahan" });
        HeaderCollection.Add(new CarouselModel() { Header = "Davolio" });
        HeaderCollection.Add(new CarouselModel() { Header = "Dodsworth" });
    }
}
```

```
HeaderCollection.Add(new CarouselModel() { Header = "Fuller" });
HeaderCollection.Add(new CarouselModel() { Header = "King" });
HeaderCollection.Add(new CarouselModel() { Header = "Leverling" });
HeaderCollection.Add(new CarouselModel() { Header = "Suyama" });
}
```

**XML**

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:Carousel Name="Carousel"
ItemsSource="{Binding HeaderCollection}">
<syncfusion:Carousel.ItemTemplate>
<DataTemplate>
<Border Height="50"
Width="100"
BorderBrush="Purple"
BorderThickness="5"
Background="LightBlue">
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
Text="{Binding Header}"/>
</Border>
</DataTemplate>
</syncfusion:Carousel.ItemTemplate>
</syncfusion:Carousel>
</Grid>
```



**Note:** [View Sample in GitHub](#)

### Custom UI for carousel item using template

You can customize the appearance of each carousel item by using the `ItemTemplate` property. If you want to change the appearance of particular carousel item appearance, use `ItemTemplateSelector` property. The `DataContext` of the `ItemTemplate` property is `CarouselItem`.

### C#

```
//Model.cs
public class CarouselModel {
    public string Header { get; set; }
}

//ViewModel.cs
public class ViewModel {
    private ObservableCollection<CarouselModel> collection;
    public ObservableCollection<CarouselModel> HeaderCollection {
        get {
            return collection;
        }
        set {
            collection = value;
        }
    }
    public ViewModel() {
        HeaderCollection = new ObservableCollection<CarouselModel>();
        HeaderCollection.Add(new CarouselModel() { Header = "Buchanan" });
        HeaderCollection.Add(new CarouselModel() { Header = "Callahan" });
        HeaderCollection.Add(new CarouselModel() { Header = "Davolio" });
        HeaderCollection.Add(new CarouselModel() { Header = "Dodsworth" });
        HeaderCollection.Add(new CarouselModel() { Header = "Fuller" });
        HeaderCollection.Add(new CarouselModel() { Header = "King" });
        HeaderCollection.Add(new CarouselModel() { Header = "Leverling" });
        HeaderCollection.Add(new CarouselModel() { Header = "Suyama" });
    }
}
```

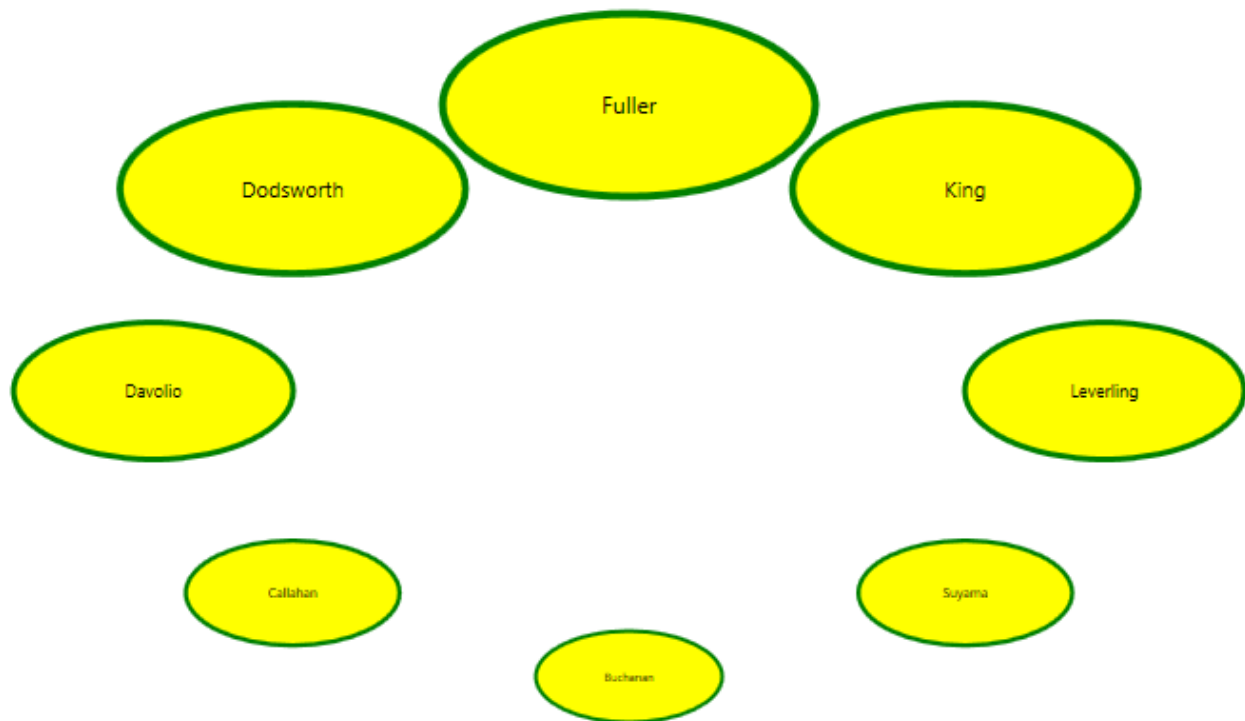
### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:Carousel ItemsSource="{Binding HeaderCollection}"
ScaleFraction="0.5"
Name="carousel" >
<syncfusion:Carousel.ItemTemplate>
<DataTemplate>
<Grid>
<Ellipse Width="200" Height="100"
Stroke="Green"
StrokeThickness="4"
Fill="Yellow"/>
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
Text="{Binding Header}"/>
</Grid>
</DataTemplate>
</syncfusion:Carousel.ItemTemplate>
</syncfusion:Carousel>
</Grid>
```

```

</Grid>
</DataTemplate>
</syncfusion:Carousel.ItemTemplate>
</syncfusion:Carousel>
</Grid>

```



**Note:** [View Sample in GitHub](#)

#### Custom UI for carousel item using style

You can change the each carousel item appearance by using the `ItemContainerStyle` which is applied to the container element that generated for each carousel item. The default value of `ItemContainerStyle` is null. The `DataContext` of the `ItemContainerStyle` property is `CarouselItem`.

#### C#

```

//Model.cs
public class CarouselModel {
    public string Header { get; set; }
    public ImageSource ImageSource { get; set; }
}

//ViewModel.cs
public class ViewModel {
    private ObservableCollection<CarouselModel> persons;
    public ObservableCollection<CarouselModel> Persons
    {
        get {
            return persons;
        }
        set {
            persons = value;
        }
    }
}

```

```

}
public ViewModel() {
    Persons = new ObservableCollection<CarouselModel>();
    Persons.Add(new CarouselModel() {
        Header = "Buchanan",
        ImageSource = new BitmapImage
            (new Uri(@"Images/Buchanan.png", UriKind.RelativeOrAbsolute))
    });
    Persons.Add(new CarouselModel() {
        Header = "Callahan",
        ImageSource = new BitmapImage
            (new Uri(@"Images/Callahan.png", UriKind.RelativeOrAbsolute))
    });
    Persons.Add(new CarouselModel() {
        Header = "Davolio",
        ImageSource = new BitmapImage
            (new Uri(@"Images/Davolio-1.png", UriKind.RelativeOrAbsolute))
    });
    Persons.Add(new CarouselModel() {
        Header = "Dodsworth",
        ImageSource = new BitmapImage
            (new Uri(@"Images/dodsworth.png", UriKind.RelativeOrAbsolute))
    });
    Persons.Add(new CarouselModel() {
        Header = "Fuller",
        ImageSource = new BitmapImage
            (new Uri(@"Images/Fuller.png", UriKind.RelativeOrAbsolute))
    });
    Persons.Add(new CarouselModel() {
        Header = "King",
        ImageSource = new BitmapImage
            (new Uri(@"Images/King.png", UriKind.RelativeOrAbsolute))
    });
    Persons.Add(new CarouselModel() {
        Header = "Leverling",
        ImageSource = new BitmapImage
            (new Uri(@"Images/Leverling.png", UriKind.RelativeOrAbsolute))
    });
    Persons.Add(new CarouselModel() {
        Header = "Suyama",
        ImageSource = new BitmapImage
            (new Uri(@"Images/Suyama.png", UriKind.RelativeOrAbsolute))
    });
}
}

```

**XML**

```

<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:Carousel ScaleFraction="0.5"
    VisualMode="Standard"
    Name="carousel"
    ItemsSource="{Binding Persons}">

```



```
<syncfusion:Carousel.ItemContainerStyle>
<Style TargetType="syncfusion:CarouselItem">
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:CarouselItem">
<Grid>
<Grid.RowDefinitions>
<RowDefinition/>
<RowDefinition/>
</Grid.RowDefinitions>
<Border x:Name="border">
<Grid>
<Ellipse Stroke="Red"
StrokeThickness="4"
Fill="Yellow"/>
<Border Margin="20">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Viewbox Height="150" Width="150">
<Image Source="{Binding ImageSource}" />
</Viewbox>
<TextBlock Text="{Binding Header}"
FontWeight="Bold"
Grid.Row="1"
HorizontalAlignment="Center" />
</Grid>
</Border>
</Grid>
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</syncfusion:Carousel.ItemContainerStyle>
</syncfusion:Carousel>
</Grid>
```



**Note:** [View Sample in GitHub](#)

Custom UI for specific carousel item using style selector

You can select a various custom appearance for the carousel items by using the `ItemContainerStyleSelector` property. The `DataContext` of the `ItemContainerStyleSelector` property is `CarouselItem`.

### C#

```
//Model.cs
public class CarouselModel {
    public string Name { get; set; }
    public int Age { get; set; }
}

//ViewModel.cs
public class ViewModel {
    private ObservableCollection<CarouselModel> persons;
    public ObservableCollection<CarouselModel> Persons {
        get {
            return persons;
        }
        set {
            persons = value;
        }
    }
    public ViewModel() {
        Persons = new ObservableCollection<CarouselModel>();
    }
}
```

```

Persons.Add(new CarouselModel() { Name = "Buchanan", Age = 40 });
Persons.Add(new CarouselModel() { Name = "Callahan", Age = 53 });
Persons.Add(new CarouselModel() { Name = "Davolio", Age = 42 });
Persons.Add(new CarouselModel() { Name = "Dodsworth", Age = 60 });
Persons.Add(new CarouselModel() { Name = "Fuller", Age = 46 });
Persons.Add(new CarouselModel() { Name = "King", Age = 65 });
Persons.Add(new CarouselModel() { Name = "Leverling", Age = 57 });
Persons.Add(new CarouselModel() { Name = "Suyama", Age = 30 });
}
}

// A class that choose style for for the items
public class PersonStyleSelector : StyleSelector {
public Style ElderStyle { get; set; }
public Style YoungerStyle { get; set; }
public override Style SelectStyle(object item, DependencyObject container) {
if ((item as CarouselModel).Age > 50)
return ElderStyle;
else
return YoungerStyle;
}
}

```

## XML

```

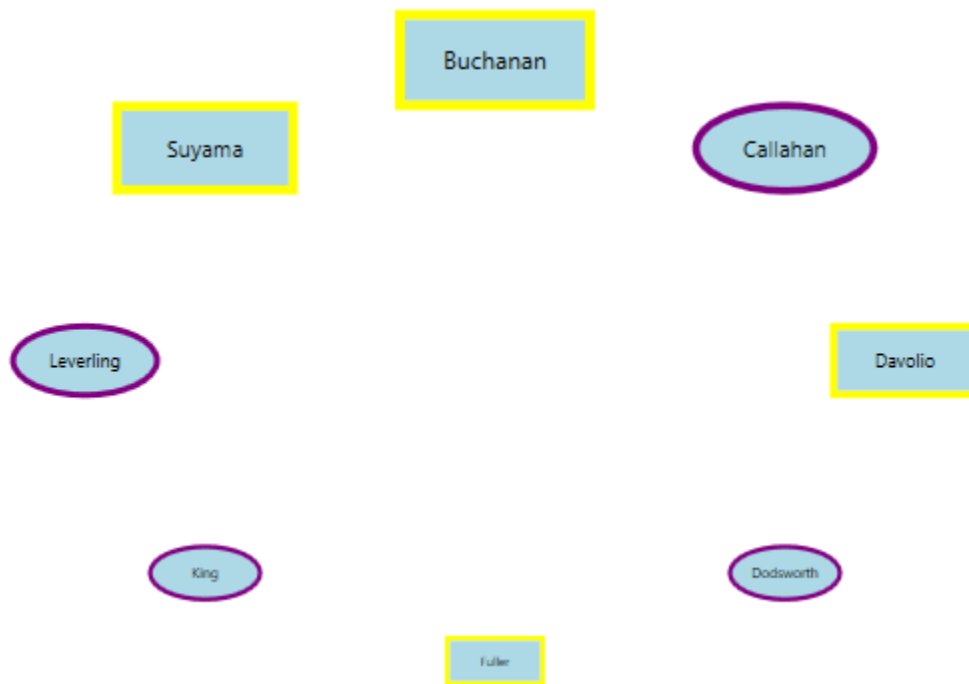
<Window.Resources>
<Style TargetType="syncfusion:CarouselItem" x:Key="elderStyle">
<Setter Property="Template">
<Setter.Value>
<ControlTemplate>
<Grid>
<Ellipse Width="100" Height="50"
Stroke="Purple"
StrokeThickness="4"
Fill="LightBlue"/>
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
Text="{Binding Name}"/>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<Style TargetType="syncfusion:CarouselItem" x:Key="youngerStyle">
<Setter Property="Template">
<Setter.Value>
<ControlTemplate>
<Border Height="50"
Width="100"
BorderBrush="Yellow"
BorderThickness="5"
Background="LightBlue">
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
Text="{Binding Name}"/>
</Border>
</ControlTemplate>

```

```

</Setter.Value>
</Setter>
</Style>
<local:PersonStyleSelector x:Key="personStyleSelector"
ElderStyle="{StaticResource elderStyle}"
YoungerStyle="{StaticResource youngerStyle}"/>
</Window.Resources>
<Grid>
<syncfusion:Carousel ItemContainerStyleSelector="{StaticResource
personStyleSelector}"
ItemsSource="{Binding Persons}"
DisplayMemberPath="Name"
ScaleFraction="0.5"
VisualMode="Standard"
Name="carousel" >
<syncfusion:Carousel.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:Carousel.DataContext>
</syncfusion:Carousel>
</Grid>

```



**Note:** [View Sample in GitHub](#)

### Virtualization support

You can enable the UI virtualization support in `Carousel`, which allows the users to load large sets of data without affecting loading or scrolling performance by setting the `EnableVirtualization` property value as `true`. This feature allows users to reduce the loading time of `Carousel` items regardless of items count. The default value of `EnableVirtualization` property is `false`.

### XML

```

<syncfusion:Carousel EnableVirtualization="True"

```

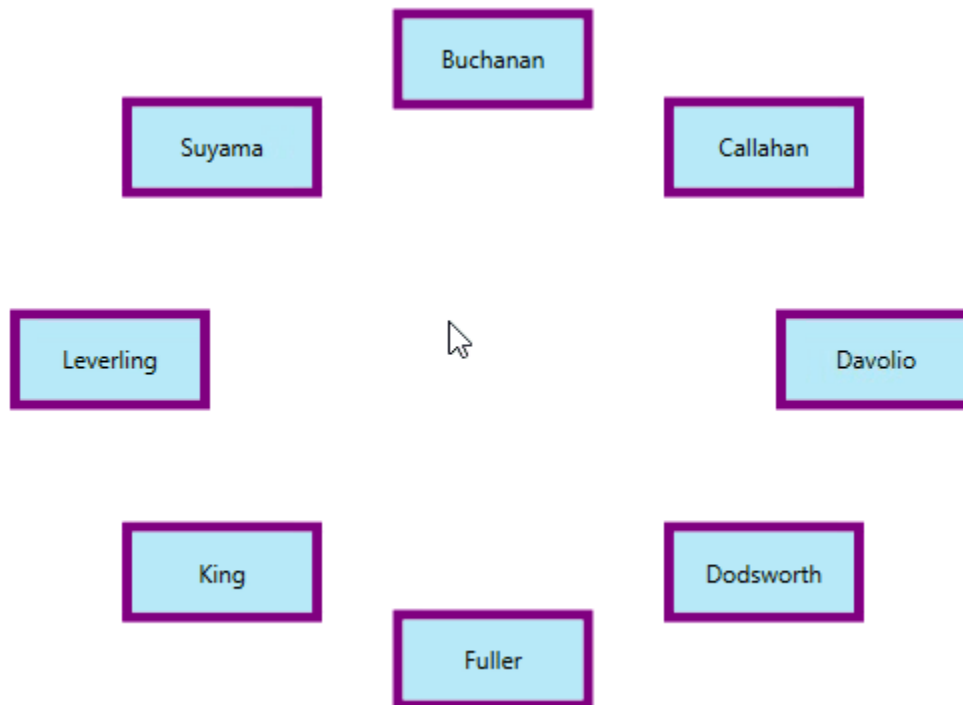
```
Name="carousel"/>
```

### C#

```
Carousel carousel = new Carousel();
carousel.EnableVirtualization = true;
```

### Select carousel item

You can select a carousel item by mouse click on the specific item. You can get the selected item and its value by using the [SelectedItem](#) and [SelectedValue](#) properties. You can also get the selected item index by using the [SelectedIndex](#) property. You can only select a single item at a time.



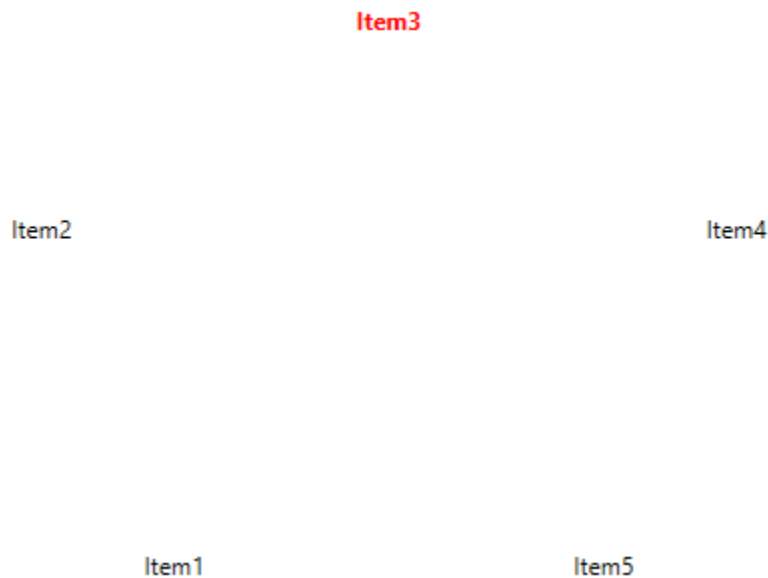
### Select carousel item programmatically using property

You can select a particular carousel item programmatically by using the [CarouselItem.IsSelected](#) property.

### XML

```
<Window.Resources>
<Style x:Key="selecteditemStyle" TargetType="syncfusion:CarouselItem" >
<Style.Triggers>
<Trigger Property="IsSelected" Value="True">
<Setter Property="Foreground" Value="Red"/>
<Setter Property="FontWeight" Value="Bold"/>
</Trigger>
</Style.Triggers>
</Style>
</Window.Resources>
<Grid>
<syncfusion:Carousel ItemContainerStyle="{StaticResource selecteditemStyle}"
x:Name="carousel">
```

```
<syncfusion:CarouselItem Content="Item1"/>
<syncfusion:CarouselItem Content="Item2"/>
<syncfusion:CarouselItem Content="Item3" IsSelected="True"/>
<syncfusion:CarouselItem Content="Item4"/>
<syncfusion:CarouselItem Content="Item5"/>
</syncfusion:Carousel>
</Grid>
```



**Note:** [View Sample in GitHub](#)

Select carousel item programmatically using command and methods

You can select a previous, next, first or last carousel items programmatically by using the commands and methods.

- [SelectFirstItemCommand](#) or [SelectFirstItem\(\)](#) - To select the first item.
- [SelectLastItemCommand](#) or [SelectLastItem\(\)](#) - To select the last item.
- [SelectPreviousItemCommand](#) or [SelectPreviousItem](#) - To select the previous item from the currently selected item.
- [SelectNextItemCommand](#) or [SelectNextItem\(\)](#) - To select the next item from the currently selected item.
- [SelectPreviousPageCommand](#) or [SelectPreviousPage\(\)](#) - To select the previous page item.
- [SelectNextPageCommand](#) or [SelectNextPage\(\)](#) - To select the next page item.

**Note:** [View Sample in GitHub](#)

Selected item changed notification

The selected item changed in `Carousel` can be examined using [SelectionChanged](#) event. The `SelectionChanged` event contains the old and newly selected item in the `OldValue` and `NewValue` properties.

#### XML

```
<syncfusion:Carousel SelectionChanged="Carousel_SelectionChanged"
```

```
Name="carousel"/>
```

**C#**

```
Carousel carousel = new Carousel();
carousel.SelectionChanged += Carousel_SelectionChanged;
```

You can handle the event as follows,

**C#**

```
private void Carousel_SelectionChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
    //Get old and new selected carousel item
    var oldValue = e.OldValue;
    var newValue = e.NewValue;
}
```

## Change flow direction

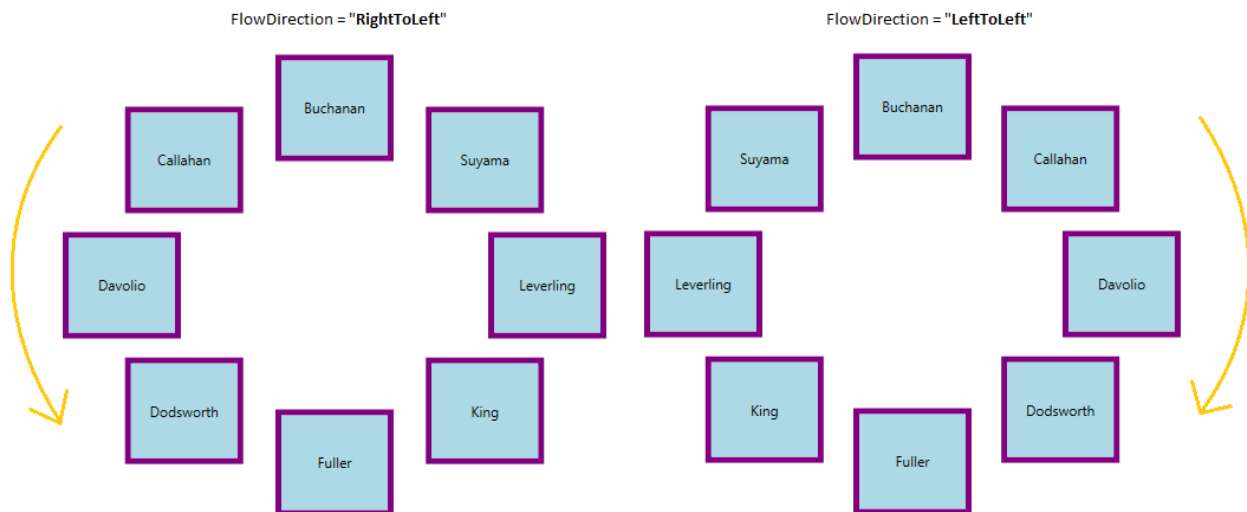
You can change the flow direction of the **Carousel** layout from right to left by setting the **FlowDirection** property value as **RightToLeft**. The default value of **FlowDirection** property is **LeftToRight**.

**XML**

```
<syncfusion:Carousel FlowDirection="RightToLeft"
Name="carousel"/>
```

**C#**

```
Carousel carousel = new Carousel();
carousel.FlowDirection = FlowDirection.RightToLeft;
```



**Note:** [View Sample in GitHub](#)

## Items Navigation in WPF Carousel

You can navigate to the carousel items by using the mouse or key navigation in the [Carousel](#) control.

Navigate carousel item using keyboard navigation

You can navigate to the previous or next carousel items one by one either backward or forward direction by pressing the keys in the keyboard.

### C#

```
//Model.cs
public class Model {
    private string header;
    public string Header {
        get { return header; }
        set { header = value; }
    }
}

//Viewmodel.cs
public class ViewModel {
    private ObservableCollection<Model> carouselItem;
    public ObservableCollection<Model> CarouselItem {
        get { return carouselItem; }
        set { carouselItem = value; }
    }
    public ViewModel() {
        CarouselItem = new ObservableCollection<Model>();
        CarouselItem.Add(new Model() { Header = "Item1" });
        CarouselItem.Add(new Model() { Header = "Item2" });
        CarouselItem.Add(new Model() { Header = "Item3" });
        CarouselItem.Add(new Model() { Header = "Item4" });
        CarouselItem.Add(new Model() { Header = "Item5" });
        CarouselItem.Add(new Model() { Header = "Item6" });
        CarouselItem.Add(new Model() { Header = "Item7" });
        CarouselItem.Add(new Model() { Header = "Item8" });
        CarouselItem.Add(new Model() { Header = "Item9" });
        CarouselItem.Add(new Model() { Header = "Item10" });
    }
}
```

### XML

```
<syncfusion:Carousel Name="Carousel"
ItemsSource="{Binding CarouselItem}">
<syncfusion:Carousel.DataContext>
<local:ViewModel/>
</syncfusion:Carousel.DataContext>
<syncfusion:Carousel.ItemTemplate>
<DataTemplate>
<Border Height="40"
Width="60"
BorderBrush="Purple"
BorderThickness="5"
Background="LightBlue">
<TextBlock Text="{Binding Header}"
HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</Border>
</DataTemplate>
</syncfusion:Carousel.ItemTemplate>
```









---

```
</syncfusion:Carousel>
```

---

The following table explains how to navigating the carousel item using keyboard,

S.No	Key	Description	Image
1	Up, Left	Navigate to the previous item from the currently selected item.	
2	Down, Right	Navigate to the next item from the currently selected item.	
3	Home, Ctrl+Up, Ctrl+Left	Navigate to the first item from the currently selected item.	
4	End, Ctrl+Down, Ctrl+Right	Navigate to the last item from the currently selected item.	
5	Page Up	Navigate to the previous page item from the current page item based on the ItemsPerPage property value. Here, ItemsPerPage value is 5.	
6	Page Down	Navigate to the next page item from the current page item based on the ItemsPerPage property value. Here, ItemsPerPage value is 5.	

---

**Note:** [View Sample in GitHub](#)

---

#### Navigate carousel item using scroll bar

By default, scroll bars are in collapsed state. If you want to navigate to the previous or next item one by one from the currently selected item using scroll bars, you need to enable the visibility of vertical or horizontal scroll bars by setting the `ScrollViewer.VerticalScrollBarVisibility` or `ScrollViewer.HorizontalScrollBarVisibility` properties value as `Visible` or `auto`.

---

**Note:** If you set `ScrollViewer.VerticalScrollBarVisibility` or `ScrollViewer.HorizontalScrollBarVisibility` properties value as `Auto`, the scroll bar is automatically visible, based on the items.

---

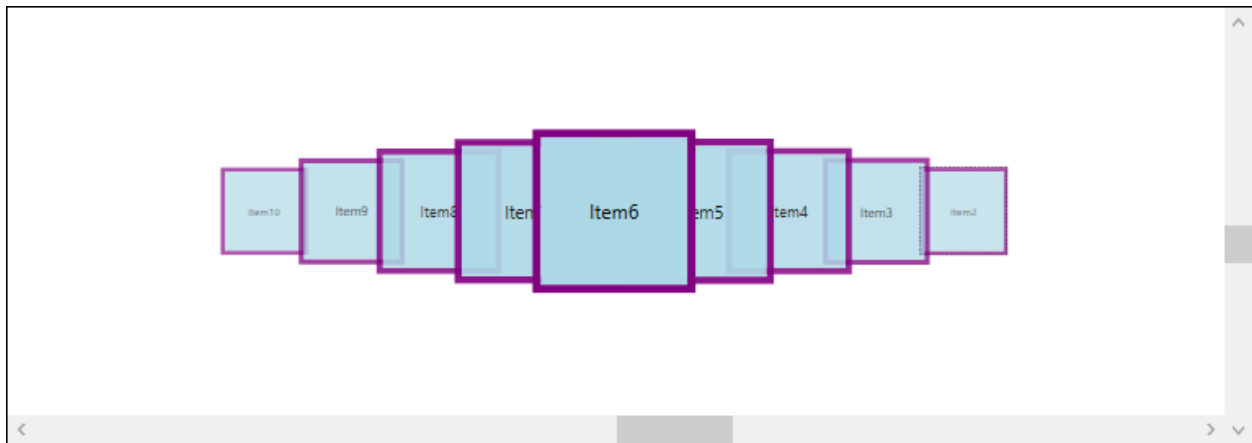
#### XML

```
<syncfusion:Carousel ScrollViewer.VerticalScrollBarVisibility="Visible"
ScrollViewer.HorizontalScrollBarVisibility="Visible"
VisualMode="CustomPath"
Name="carousel"
ItemsSource="{Binding CarouselItem}">
  <syncfusion:Carousel.Path>
    <Path Data="M0,0 L100,0" />
  </syncfusion:Carousel.Path>
  <syncfusion:Carousel.DataContext>
    <local:ViewModel/>
  </syncfusion:Carousel.DataContext>
  <syncfusion:Carousel.ItemTemplate>
    <DataTemplate>
      <Border Height="100"
```

```
Width="100"
BorderBrush="Purple"
BorderThickness="5"
Background="LightBlue">
<TextBlock Text="{Binding Header}"
HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</Border>
</DataTemplate>
</syncfusion:Carousel.ItemTemplate>
</syncfusion:Carousel>
```

## C#

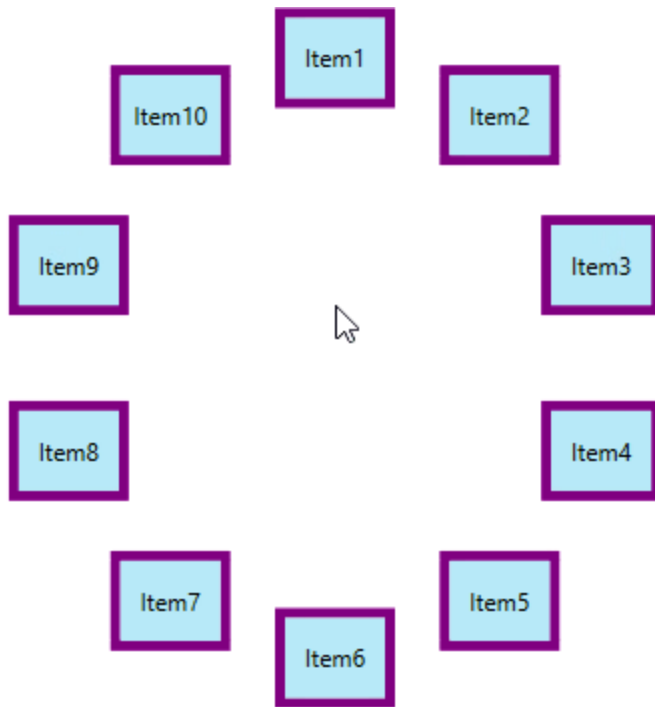
```
ScrollViewer.SetHorizontalScrollBarVisibility(carousel,
ScrollBarVisibility.Visible);
ScrollViewer.SetVerticalScrollBarVisibility(carousel,
ScrollBarVisibility.Visible);
```



**Note:** [View Sample in GitHub](#)

Navigate carousel item using mouse wheel

You can navigate to the previous or next item one by one from the currently selected item by using the mouse wheel on forward and backward direction.



**Note:** [View Sample in GitHub](#)

#### Navigate using commands

You can easily navigate to the first, last, previous or next items and also previous page or next pages by using the built-in commands in the `Carousel` control.

S.No	Command	Description	Keyboard shortcut
1	SelectFirstItemCommand	This command selects the first item in carousel control. It is executed when home key is pressed.	Home
2	SelectLastItemCommand	This command selects the last item in carousel control. It is executed when end key is pressed.	End
3	SelectNextItemCommand	This command selects the next item in carousel control. It is executed when right or down arrow key is pressed.	Right and Down arrow
4	SelectPreviousItemCommand	This command selects the previous item in carousel control. It is executed when left or top arrow key is pressed.	Left and Top arrow
5	SelectNextPageCommand	This command selects the item in next page of carousel control. It is executed when page down key is pressed.	PageDown

6	SelectPreviousPageCommand	This command selects the item in previous page of carousel control. It is executed when page up key is pressed.	PageUp
---	---------------------------	---	--------

**Note:** [View Sample in GitHub](#)

### Looping items in Custom path view

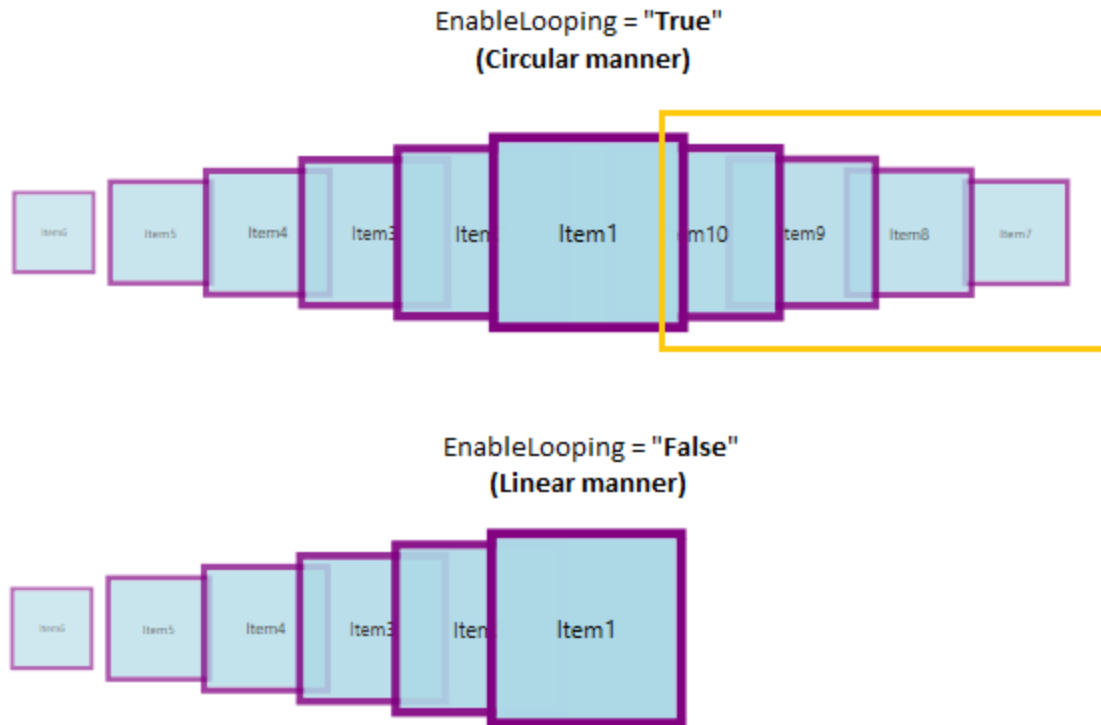
Carousel has looping functionality, it allows users to loop items after reaching the last item. In the Standard visual mode, the carousel items can be scrolled on the circular manner. But, on the CustomPath visual mode, the carousel items scrolled in linear manner and first or last item hidden from the view. If you want to bring the first or last item into view in circular manner, use [EnableLooping](#) property value as `true`. The default value of `EnableLooping` property is `false`.

### XML

```
<syncfusion:Carousel EnableLooping="True"
VisualMode="CustomPath"
Name="Carousel"
ItemsSource="{Binding CarouselItem}">
<syncfusion:Carousel.Path>
<Path Data="M0,0 L100,0" />
</syncfusion:Carousel.Path>
<syncfusion:Carousel.DataContext>
<local:ViewModel/>
</syncfusion:Carousel.DataContext>
<syncfusion:Carousel.ItemTemplate>
<DataTemplate>
<Border Height="40"
Width="60"
BorderBrush="Purple"
BorderThickness="5"
Background="LightBlue">
<TextBlock Text="{Binding Header}"
HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</Border>
</DataTemplate>
</syncfusion:Carousel.ItemTemplate>
</syncfusion:Carousel>
```

### C#

```
Carousel.EnableLooping = true;
Carousel.VisualMode = VisualMode.CustomPath;
```



**Note:** [View Sample in GitHub](#)

### Standard Path in WPF Carousel

This section explains about resizing, skewing, rotation animation and opacity supports available in WPF [Carousel](#) control's standard path mode.

#### Load carousel items in standard path

You can load the carousel items in standard path by using the [VisualMode](#) property as `VisualMode.Standard`. The standard path of carousel items is a circular path. The default value of `VisualMode` property is `VisualMode.Standard`.

#### C#

```
//Model.cs
public class CarouselModel {
    public string Header { get; set; }
}

//ViewModel.cs
public class ViewModel {
    private ObservableCollection<CarouselModel> collection;
    public ObservableCollection<CarouselModel> HeaderCollection
    {
        get {
            return collection;
        }
        set {
            collection = value;
        }
    }
    public ViewModel() {
        HeaderCollection = new ObservableCollection<CarouselModel>();
    }
}
```

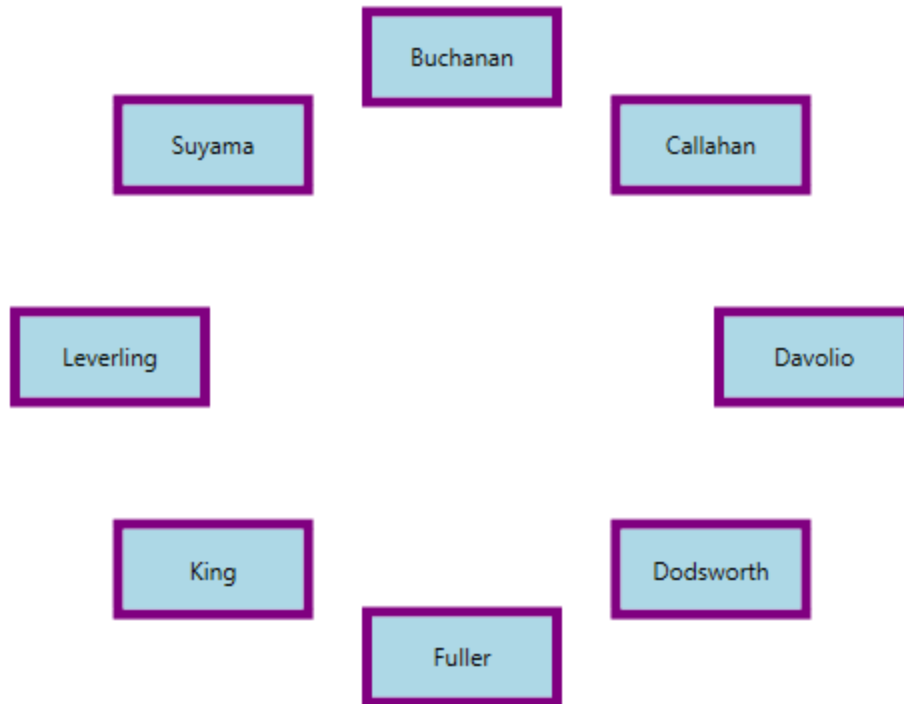
```
HeaderCollection.Add(new CarouselModel() { Header = "Buchanan" });
HeaderCollection.Add(new CarouselModel() { Header = "Callahan" });
HeaderCollection.Add(new CarouselModel() { Header = "Davolio" });
HeaderCollection.Add(new CarouselModel() { Header = "Dodsworth" });
HeaderCollection.Add(new CarouselModel() { Header = "Fuller" });
HeaderCollection.Add(new CarouselModel() { Header = "King" });
HeaderCollection.Add(new CarouselModel() { Header = "Leverling" });
HeaderCollection.Add(new CarouselModel() { Header = "Suyama" });
}
```

## XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:Carousel Name="Carousel"
VisualMode="Standard"
ItemsSource="{Binding HeaderCollection}">
<syncfusion:Carousel.ItemTemplate>
<DataTemplate>
<Border Height="50"
Width="100"
BorderBrush="Purple"
BorderThickness="5"
Background="LightBlue">
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
Text="{Binding Header}"/>
</Border>
</DataTemplate>
</syncfusion:Carousel.ItemTemplate>
</syncfusion:Carousel>
</Grid>
```

## XML

```
carousel.VisualMode = VisualMode.Standard;
```



---

**Note:** [View Sample in GitHub](#)

---

#### Change radius of carousel control

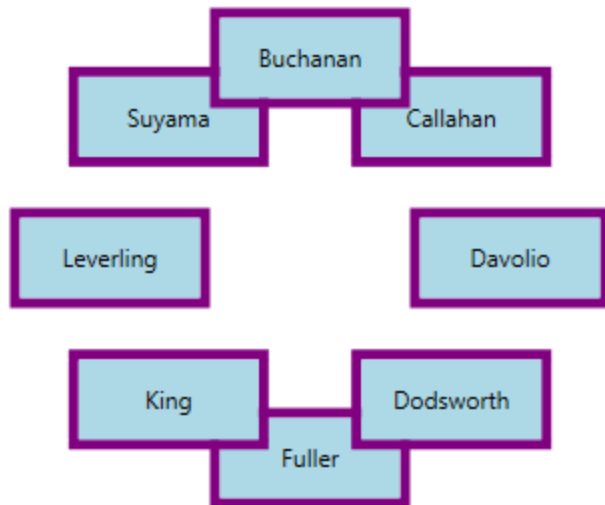
You can change the radius of the `Carousel` control by setting the value to the [RadiusX](#) and [RadiusY](#) properties. Based on the radius points, items are arranged. The default value of `RadiusX` property is 250 and `RadiusY` property is 150.

#### XML

```
<syncfusion:Carousel RadiusX="100"  
RadiusY="100"  
VisualMode="Standard"  
Name="carousel"/>
```

#### C#

```
carousel.RadiusX = 100;  
carousel.RadiusY = 100;  
carousel.VisualMode = VisualMode.Standard;
```



**Note:** [View Sample in GitHub](#)

### Change rotation speed

If you want to change the rotation speed of the carousel items when selecting or navigating from one item to another item, use the [RotationSpeed](#) property. The default value of `RotationSpeed` property is 200.

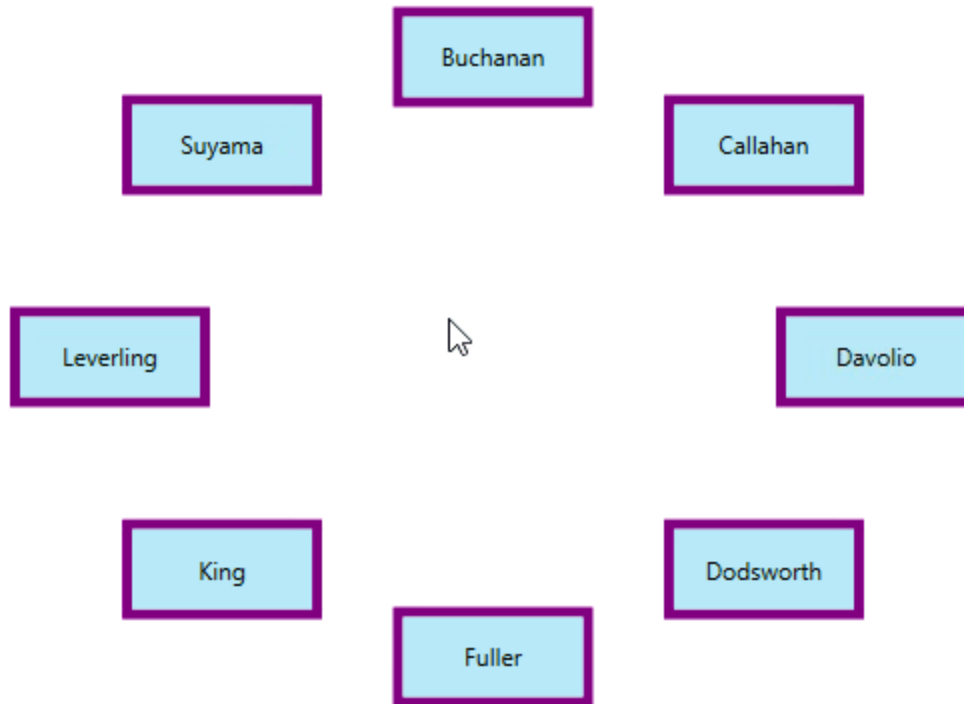
### XML

```
<syncfusion:Carousel RotationSpeed="150"
VisualMode="Standard"
Name="carousel" />
```

### C#

```
carousel.RotationSpeed = 150;
carousel.VisualMode = VisualMode.Standard;
```





---

**Note:** [View Sample in GitHub](#)

---

#### Disable rotation animation

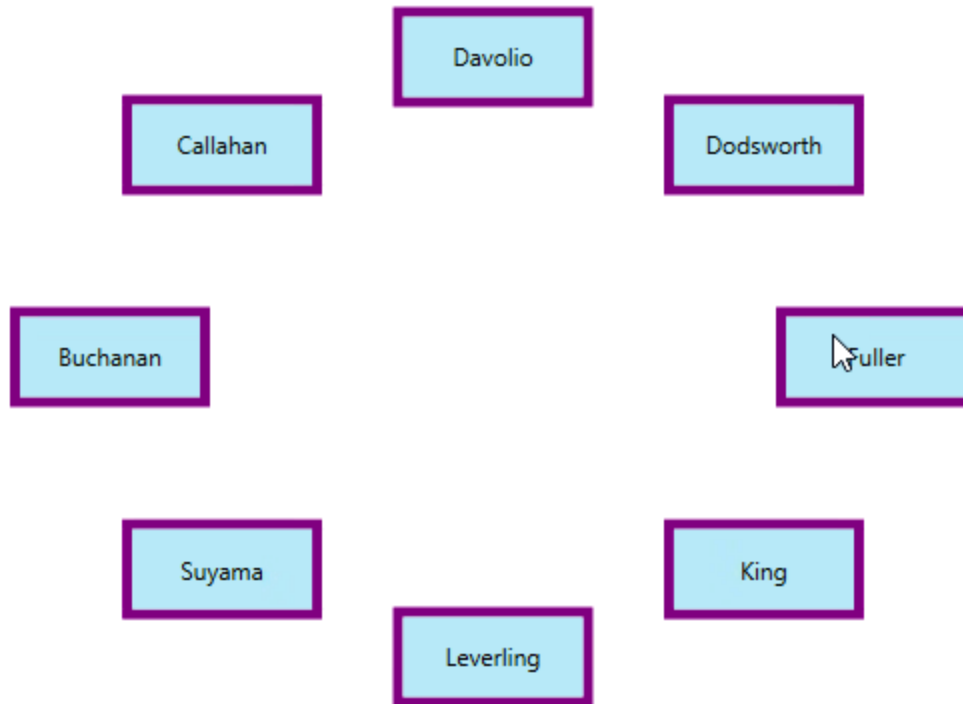
If you want to disable the animated rotation of carousel items when selecting or navigating from one item to another item, use the [EnableRotationAnimation](#) property value as `false`. The default value of `EnableRotationAnimation` property is `true`.

#### XML

```
<syncfusion:Carousel EnableRotationAnimation="False"
    VisualMode="Standard"
    Name="carousel" />
```

#### C#

```
carousel.EnableRotationAnimation = false;
carousel.VisualMode = VisualMode.Standard;
```



---

**Note:** [View Sample in GitHub](#)

---

#### Resize carousel item

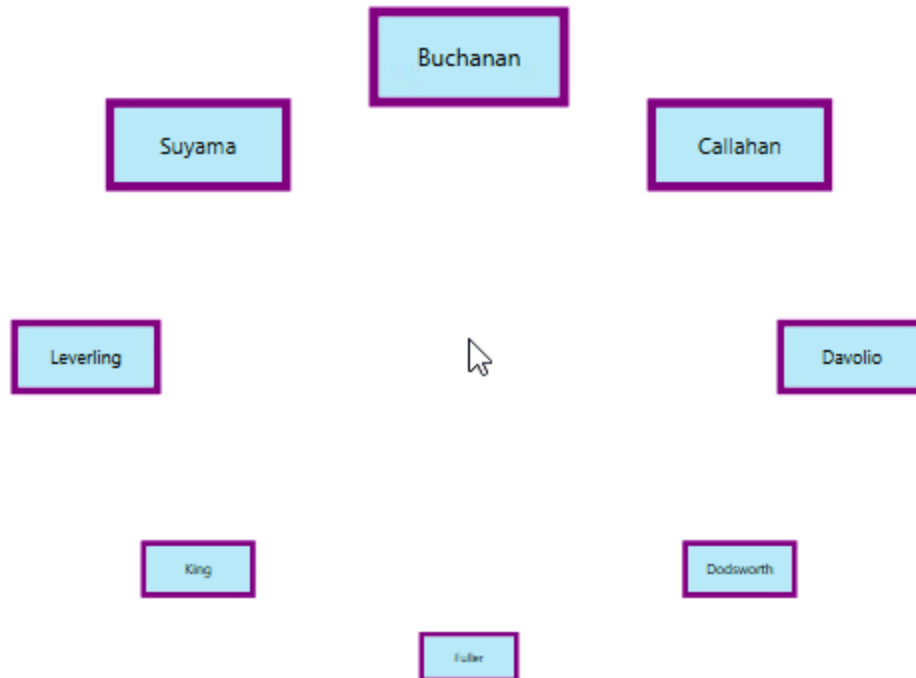
If you want to change the size of the carousel items except the selected item in the `VisualMode.Standard` mode, use the [ScaleFraction](#) property. You can disable it by setting the [ScalingEnabled](#) property value as `false`. Value range of `ScaleFraction` property is 0 to 1. The default value `ScaleFraction` property is 0 and `ScalingEnabled` property is `true`.

#### XML

```
<syncfusion:Carousel ScaleFraction="0.5"
ScalingEnabled="True"
VisualMode="Standard"
Name="carousel" />
```

#### C#

```
carousel.ScaleFraction = 0.5;
carousel.ScalingEnabled = true;
carousel.VisualMode = VisualMode.Standard;
```



---

**Note:** [View Sample in GitHub](#)

---

#### Opacity for carousel item

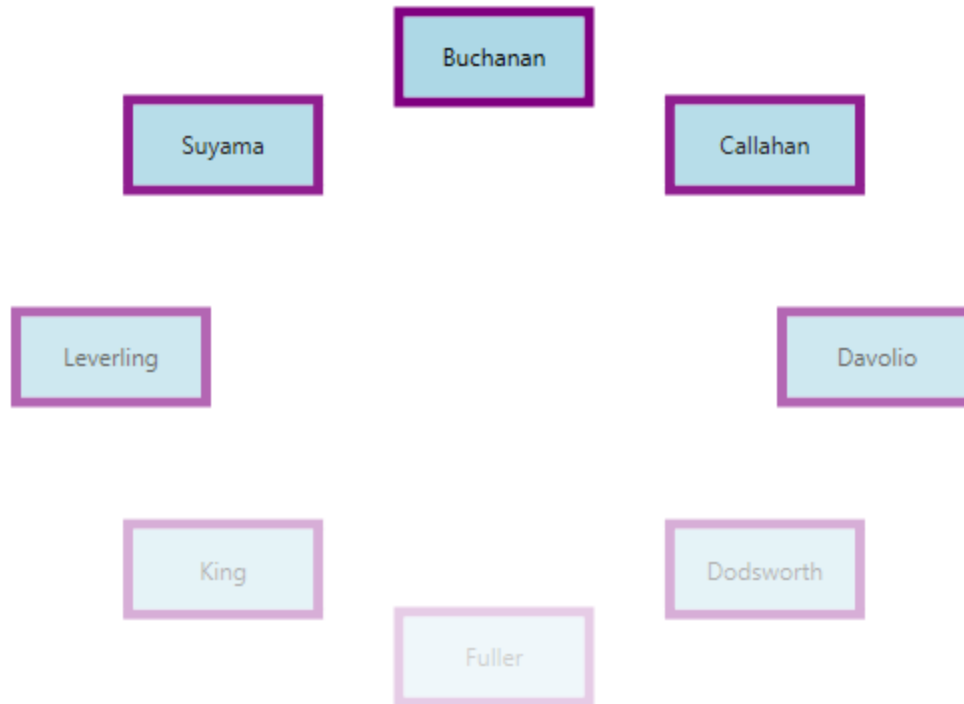
If you want to change the opacity of the carousel items except the selected item in the `VisualMode.Standard` mode, set the fraction value to the [OpacityFraction](#) property. You can disable it by setting the [OpacityEnabled](#) property value as `false`. Value range of `OpacityFraction` property is 0 to 1. The default value of `OpacityFraction` property is 0 and `OpacityEnabled` property is `true`.

#### XML

```
<syncfusion:Carousel OpacityFraction="0.8"
    OpacityEnabled="True"
    VisualMode="Standard"
    Name="carousel" />
```

#### C#

```
carousel.OpacityFraction = 0.8;
carousel.OpacityEnabled = true;
carousel.VisualMode = VisualMode.Standard;
```



**Note:** [View Sample in GitHub](#)

Skewing the carousel item

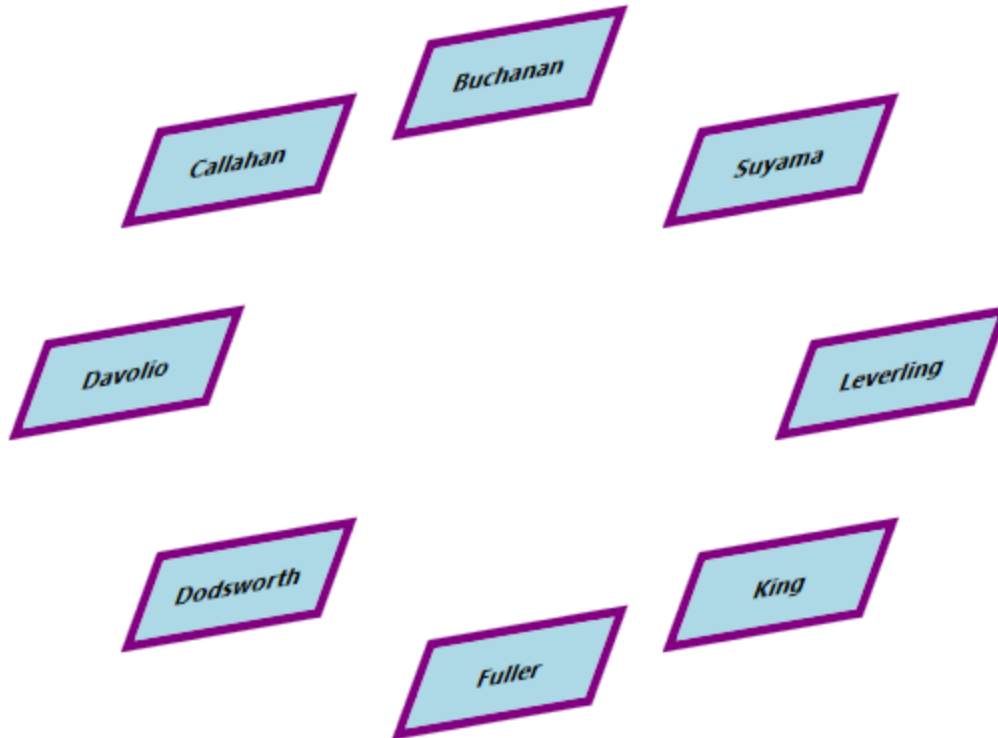
If you want to skewing the carousel items with particular X-Y fraction angle, use the [SkewAngleXFraction](#) and [SkewAngleYFraction](#) properties. You can enable it by setting the [SkewAngleXEnabled](#) and [SkewAngleYEnabled](#) property value as `true`. The default value `SkewAngleXFraction` and `SkewAngleYFraction` properties is `0` and default value of `SkewAngleXEnabled` & `SkewAngleYEnabled` property is `false`.

#### XML

```
<syncfusion:Carousel SkewAngleXFraction="20"
SkewAngleYFraction="10"
SkewAngleXEnabled="True"
SkewAngleYEnabled="True"
VisualMode="Standard"
Name="carousel" />
```

#### C#

```
carousel.SkewAngleXFraction = 20;
carousel.SkewAngleYFraction = 10;
carousel.SkewAngleXEnabled = true;
carousel.SkewAngleYEnabled = true;
carousel.VisualMode = VisualMode.Standard;
```



---

**Note:** [View Sample in GitHub](#)

---

### Custom Path in WPF carousel (Carousel)

This section explains about resizing, skewing, page customization and opacity supports available in WPF [Carousel](#) control's custom path mode.

#### Load carousel items in custom path

If you want to load the carousel items in your custom path, set the path value to the [Carousel.Path](#). You can enable it only by setting the [VisualMode](#) property as `VisualMode.CustomPath`. The default value of `VisualMode` property is `VisualMode.Standard` and `Carousel.Path` property is null.

---

**Note:** If you not set any path value to `Carousel.Path` property on `VisualMode.CustomPath` mode, carousel items will be loaded in the U shaped path.

---

#### C#

```
//Model.cs
public class CarouselModel {
    public string Header { get; set; }
}

//ViewModel.cs
public class ViewModel {
    private ObservableCollection<CarouselModel> collection;
    public ObservableCollection<CarouselModel> HeaderCollection
    {
        get {
            return collection;
        }
        set {
            collection = value;
        }
    }
}
```

```

}
public ViewModel() {
    HeaderCollection = new ObservableCollection<CarouselModel>();
    HeaderCollection.Add(new CarouselModel() { Header = "Buchanan" });
    HeaderCollection.Add(new CarouselModel() { Header = "Callahan" });
    HeaderCollection.Add(new CarouselModel() { Header = "Davolio" });
    HeaderCollection.Add(new CarouselModel() { Header = "Dodsworth" });
    HeaderCollection.Add(new CarouselModel() { Header = "Fuller" });
    HeaderCollection.Add(new CarouselModel() { Header = "King" });
    HeaderCollection.Add(new CarouselModel() { Header = "Leverling" });
    HeaderCollection.Add(new CarouselModel() { Header = "Suyama" });
}
}

```

**XML**

```

<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:Carousel Name="Carousel"
VisualMode="CustomPath"
SelectedIndex="3"
ItemsSource="{Binding HeaderCollection}">
<syncfusion:Carousel.Path>
<Path Data="M0,100 L100,20" />
</syncfusion:Carousel.Path>
<syncfusion:Carousel.ItemTemplate>
<DataTemplate>
<Border Height="50"
Width="100"
BorderBrush="Purple"
BorderThickness="5"
Background="LightBlue">
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
Text="{Binding Header}" />
</Border>
</DataTemplate>
</syncfusion:Carousel.ItemTemplate>
</syncfusion:Carousel>
</Grid>

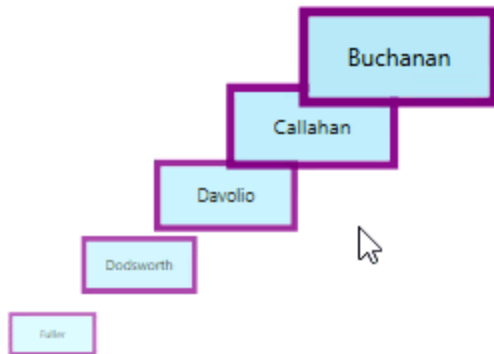
```

**XML**

```

carousel.VisualMode = VisualMode.CustomPath;
carousel.SelectedIndex = 3;

```



---

**Note:** [View Sample in GitHub](#)

---

Number of items to be visible in a Page

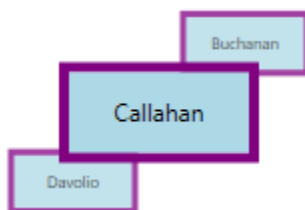
By default, all the items are displayed in the `Carousel` control. If you will be added more items and wants to display less number of items at a time, use the [ItemsPerPage](#) property. `ItemsPerPage` is effective only on `VisualMode.CustomPath` view mode. The default value of `ItemsPerPage` property is `-1`.

#### XML

```
<syncfusion:Carousel ItemsPerPage="3"
    VisualMode="CustomPath"
    ItemsSource="{Binding HeaderComponent}"
    Name="carousel"/>
```

#### C#

```
carousel.ItemsPerPage = 3;
carousel.VisualMode = VisualMode.CustomPath;
```



---

**Note:** [View Sample in GitHub](#)

---

### Resize carousel items

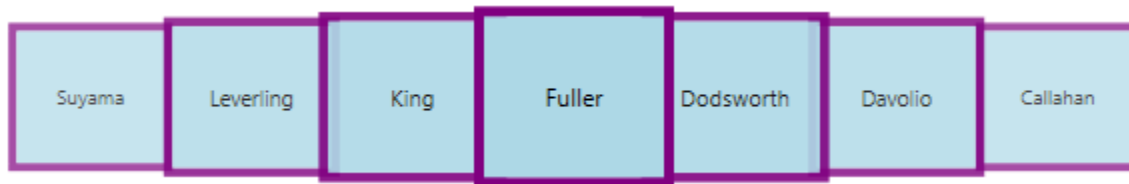
If you want to change the size of the carousel items except the selected item in the `VisualMode.CustomPath` mode, use the [ScaleFraction](#) property. You can disable it by setting the [ScalingEnabled](#) property value as `false`. Value range of `ScaleFraction` property is 0 to 1. The default value `ScaleFraction` property is `Double.NaN` and `ScalingEnabled` property is `true`.

#### XML

```
<syncfusion:Carousel ScaleFraction="0.7"
ScalingEnabled="True"
VisualMode="CustomPath"
ItemsSource="{Binding HeaderCollection}"
Name="carousel">
<syncfusion:Carousel.Path>
<Path Data="M0,0 L100,0" />
</syncfusion:Carousel.Path>
</syncfusion:Carousel>
```

#### C#

```
carousel.ScaleFraction = 0.7;
carousel.ScalingEnabled = true;
carousel.VisualMode = VisualMode.CustomPath;
```



**Note:** [View Sample in GitHub](#)

### Resize specific carousel item

If you want to individually change the size of the next, previous item or selected carousel items in the `VisualMode.CustomPath` mode, set the fraction values to the [ScaleFractions](#) collection property. Value range of resizing is 0 to 1. The default value of `ScaleFractions` property is `null`.

**Note:** This will effective only on when setting the `ScalingEnabled` property value as `true`.

#### XML

```
<syncfusion:Carousel ScalingEnabled="True"
VisualMode="CustomPath"
ItemsSource="{Binding HeaderCollection}"
Name="carousel">
<syncfusion:Carousel.Path>
<Path Data="M0,0 L100,0" />
</syncfusion:Carousel.Path>
<syncfusion:Carousel.ScaleFractions>
<syncfusion:PathFractionCollection>
<!--Resize next items from the selected item-->
<syncfusion:FractionValue Fraction="0" Value="0.6"/>
<!--Resize selected item-->
<syncfusion:FractionValue Fraction="0.5" Value="0.9"/>
</syncfusion:PathFractionCollection>
</syncfusion:Carousel.ScaleFractions>
</syncfusion:Carousel>
```



```

<!--Resize previous items from the selected item-->
<syncfusion:FractionValue Fraction="1" Value="0"/>
</syncfusion:PathFractionCollection>
</syncfusion:Carousel.ScaleFractions>
</syncfusion:Carousel>

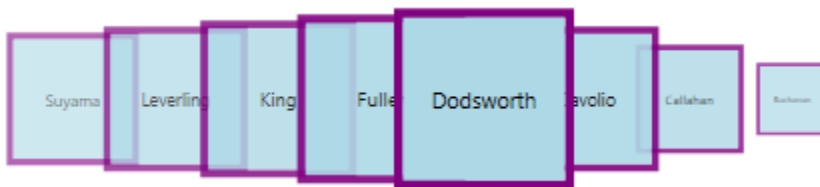
```

**C#**

```

FractionValue NextItemfraction = new FractionValue() { Fraction = 0, Value = 0.6 };
FractionValue selectedItemfraction = new FractionValue() { Fraction = 0.5, Value = 0.9 };
FractionValue PreviousItemfraction = new FractionValue() { Fraction = 1, Value = 0 };
PathFractionCollection pathFraction = new PathFractionCollection();
pathFraction.Add(NextItemfraction);
pathFraction.Add(selectedItemfraction);
pathFraction.Add(PreviousItemfraction);
//Adding scale fractions to the carousel items
carousel.ScaleFractions = pathFraction;
carousel.ScrollingEnabled = true;
carousel.VisualMode = VisualMode.CustomPath;

```



**Note:** [View Sample in GitHub](#)

**Opacity for carousel items**

If you want to change the opacity of the carousel items except the selected item in the VisualMode.CustomPath mode, set the fraction value to the [OpacityFraction](#) property. You can disable it by setting the [OpacityEnabled](#) property value as false. Value range of OpacityFraction property is 0 to 1. The default value of OpacityFraction property is Double.NaN and OpacityEnabled property is true.

**XML**

```

<syncfusion:Carousel OpacityFraction="0.9"
OpacityEnabled="True"
VisualMode="CustomPath"
ItemsSource="{Binding HeaderCollection}"
Name="carousel">
<syncfusion:Carousel.Path>
<Path Data="M0,0 L100,0" />
</syncfusion:Carousel.Path>
</syncfusion:Carousel>

```

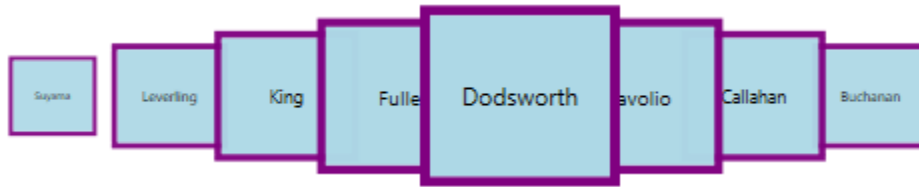
**C#**

```

carousel.OpacityFraction = 0.9;

```

```
carousel.OpacityEnabled = true;
carousel.VisualMode = VisualMode.CustomPath;
```



**Note:** [View Sample in GitHub](#)

### Opacity for specific carousel item

If you want to individually change the opacity of the next, previous items or selected carousel items in the `VisualMode.CustomPath` mode, set the fraction values to the [OpacityFractions](#) collection property. Value range of opacity is 0 to 1. The default value of `OpacityFractions` property is null.

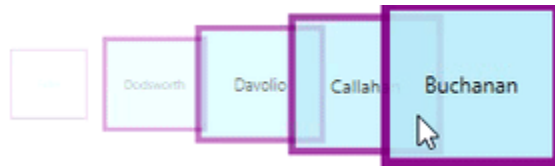
**Note:** This will effective only on when setting the `OpacityEnabled` property value as true.

### XML

```
<syncfusion:Carousel OpacityEnabled="True"
VisualMode="CustomPath"
ItemsSource="{Binding HeaderComponent}"
Name="carousel">
  <syncfusion:Carousel.Path>
    <Path Data="M0,0 L100,0" />
  </syncfusion:Carousel.Path>
  <syncfusion:Carousel.OpacityFractions>
    <syncfusion:PathFractionCollection>
      <!--Opacity for next items from the selected item-->
      <syncfusion:FractionValue Fraction="0" Value="0"/>
      <!--Opacity for selected item-->
      <syncfusion:FractionValue Fraction="0.5" Value="0.9"/>
      <!--Opacity for previous items from the selected item-->
      <syncfusion:FractionValue Fraction="1" Value="0.7"/>
    </syncfusion:PathFractionCollection>
  </syncfusion:Carousel.OpacityFractions>
</syncfusion:Carousel>
```

### C#

```
FractionValue NextItemfraction = new FractionValue() { Fraction = 0, Value = 0 };
FractionValue selectedItemfraction = new FractionValue() { Fraction = 0.5, Value = 0.9 };
FractionValue PreviousItemfraction = new FractionValue() { Fraction = 1, Value = 0.7 };
PathFractionCollection pathFraction = new PathFractionCollection();
pathFraction.Add(NextItemfraction);
pathFraction.Add(selectedItemfraction);
pathFraction.Add(PreviousItemfraction);
//Adding opacity fractions to the carousel items
carousel.OpacityFractions = pathFraction;
carousel.OpacityEnabled = true;
carousel.VisualMode = VisualMode.CustomPath;
```



**Note:** [View Sample in GitHub](#)

### Skewing the carousel items

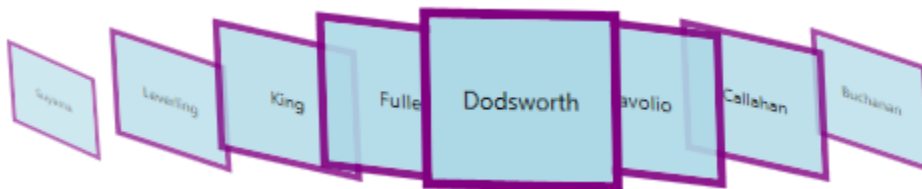
If you want to skewing the carousel items with particular X-Y fraction angle, use the [SkewAngleXFraction](#) and [SkewAngleYFraction](#) properties. You can enable it by setting the [SkewAngleXEnabled](#) and [SkewAngleYEnabled](#) property value as true. The default value [SkewAngleXFraction](#) and [SkewAngleYFraction](#) properties is `Double.NaN` and default value of [SkewAngleXEnabled](#) & [SkewAngleYEnabled](#) property is false.

### XML

```
<syncfusion:Carousel SkewAngleXFraction="5"
SkewAngleYFraction="30"
SkewAngleXEnabled="True"
SkewAngleYEnabled="True"
VisualMode="CustomPath"
ItemsSource="{Binding HeaderCollection}"
Name="carousel">
  <syncfusion:Carousel.Path>
    <Path Data="M0,0 L100,0" />
  </syncfusion:Carousel.Path>
</syncfusion:Carousel>
```

### C#

```
carousel.SkewAngleXFraction = 5;
carousel.SkewAngleYFraction = 30;
carousel.SkewAngleXEnabled = true;
carousel.SkewAngleYEnabled = true;
carousel.VisualMode = VisualMode.CustomPath;
```



**Note:** [View Sample in GitHub](#)

### Skewing the specific carousel item

If you want to individually skewing the next, previous item or selected carousel items in the `VisualMode.CustomPath` mode, set the X-Y fraction angle values to the [SkewAngleXFractions](#) and [SkewAngleYFractions](#) collection property. The default value of [SkewAngleXFractions](#) and [SkewAngleYFractions](#) property is null.

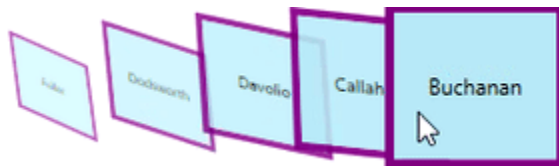
**Note:** This will effective only on when setting the `SkewAngleXEnabled` and `SkewAngleYEnabled` properties value as `true`.

### XML

```
<syncfusion:Carousel SkewAngleXEnabled="True"
SkewAngleYEnabled="True"
VisualMode="CustomPath"
ItemsSource="{Binding HeaderComponent}"
Name="carousel">
  <syncfusion:Carousel.Path>
    <Path Data="M0,0 L100,0" />
  </syncfusion:Carousel.Path>
  <syncfusion:Carousel.SkewAngleXFractions>
    <syncfusion:PathFractionCollection>
      <!--Skewing X angle of next items from the selected item-->
      <syncfusion:FractionValue Fraction="0" Value="10"/>
      <!--Skewing X angle of selected item-->
      <syncfusion:FractionValue Fraction="0.5" Value="0"/>
      <!--Skewing X angle of previous items from the selected item-->
      <syncfusion:FractionValue Fraction="1" Value="40"/>
    </syncfusion:PathFractionCollection>
  </syncfusion:Carousel.SkewAngleXFractions>
  <syncfusion:Carousel.SkewAngleYFractions>
    <syncfusion:PathFractionCollection>
      <!--Skewing Y angle of next items from the selected item-->
      <syncfusion:FractionValue Fraction="0" Value="10"/>
      <!--Skewing Y angle of selected item-->
      <syncfusion:FractionValue Fraction="0.5" Value="0"/>
      <!--Skewing Y angle of previous items from the selected item-->
      <syncfusion:FractionValue Fraction="1" Value="40"/>
    </syncfusion:PathFractionCollection>
  </syncfusion:Carousel.SkewAngleYFractions>
</syncfusion:Carousel>
```

### C#

```
carousel.SkewAngleXEnabled = true;
carousel.SkewAngleYEnabled = true;
carousel.VisualMode = VisualMode.CustomPath;
```



**Note:** [View Sample in GitHub](#)

## Syntax Editor

### WPF Syntax Editor Overview

This section covers information on the Essential Edit for WPF, its key features and prerequisites, compatibility with various operating systems and browsers, and the documentation details to use the control. It comprises the following sub sections.

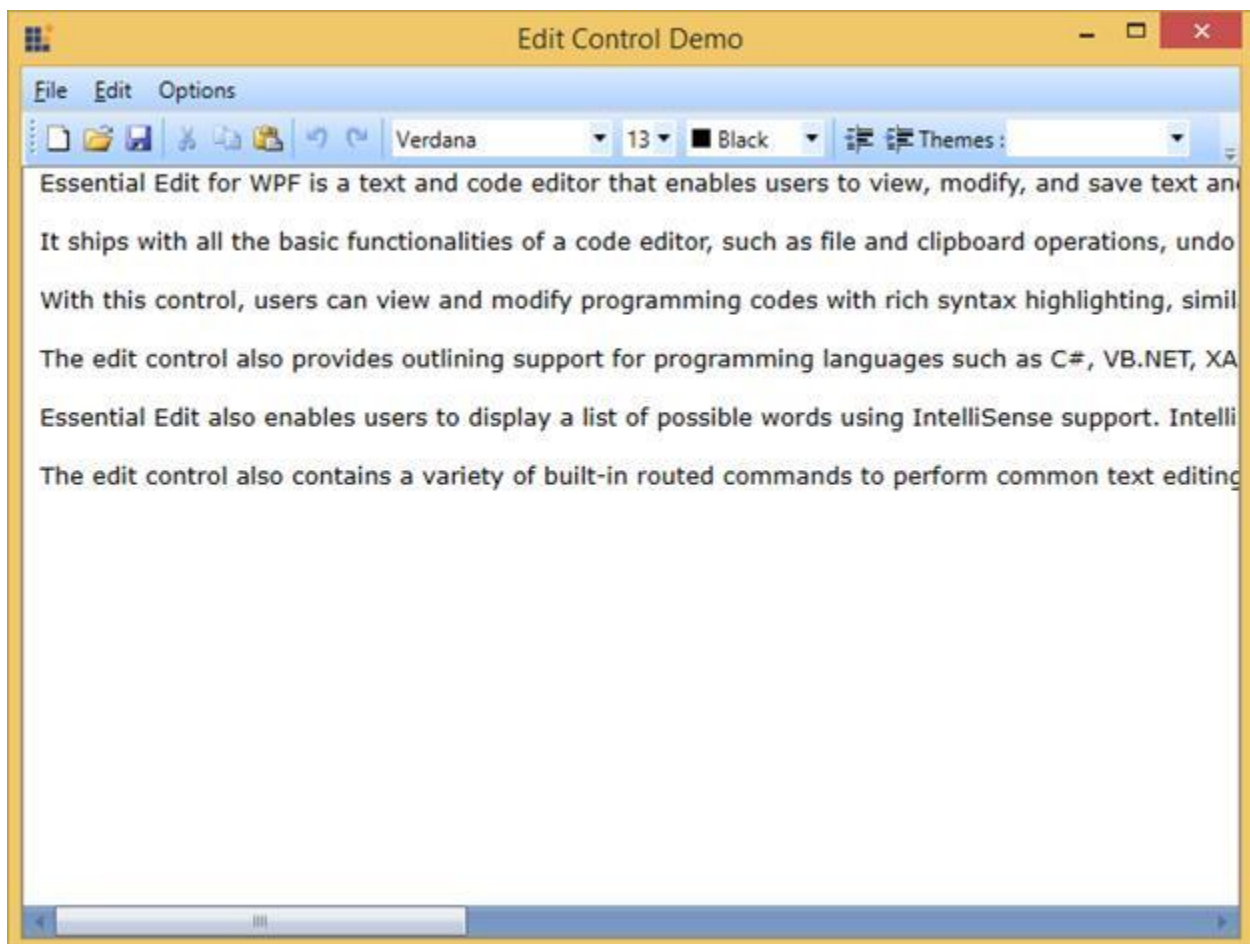
#### Introduction to Essential Edit WPF

Essential Edit WPF is easy to use and extensible control. This can be viewed as an advance notepad with features such as formatting, code editing, and more. It allows users to view, edit, and save the following:

- Large formatted text
- Code
- Data from the database

Essential Edit WPF allows to create interactive code editor applications with its unique feature set. This is a fully functional context menu that can be used to perform all basic editing operations such as select all, cut, copy, paste, undo, redo, and so on.

The following image shows an example of Essential Edit control.



#### Real world scenarios

The Essential Edit WPF control is a high performance control for creating text and language editors. Users can customize their page with the font, font color, background, and size properties. It has more features like context menu, expand and collapse functionality, and syntax highlight functionality for C#, XAML, and XML language patterns. Some of the real world scenarios of Essential Edit control for WPF are as follows:

- Text editor application like notepad.
- Code editor application like Visual Studio 2008 IDE.

### Key features

The following are the key features of Essential Edit WPF:

- Basic editing:** Essential Edit WPF allows users to modify, edit text documents, and source code files. It facilitates to perform all edit operations such as select, cut, copy, paste, select all, delete, and so on.
- File support:** Allows to open supported file types and make necessary changes and save them back to the same or different file types. It supports all file operations such as create new, open, or save files.
- Syntax highlighting support:** Highlights the content in the edit control based on the selected language. Essential Edit provides built-in support for programming languages such as C#, Visual Basic, XAML, and XML.
- Custom language support:** Essential Edit WPF extends support for user defined language configurations. Custom language configurations can be created easily and applied to the content in the edit control.
- Undo/Redo support:** Allows to cancel or apply the changes done to the content in the order in which they were performed (LIFO). Undo and Redo can be performed using the built-in context menu or using keyboard shortcuts.
- Line number support:** Displays line numbers for text in the edit control for easy reference. You can enable or disable this feature.
- Expand or collapse support:** Facilitates to outline the programming codes and enables to expand or collapse the text within region or module to enhance the readability.
- Read only mode:** Displays contents of a file or source codes files and prevents user from editing the contents of the file.
- Editing commands:** Essential Edit WPF ships with built-in RoutedUICommands to perform various operations in the edit control externally using command bindings.
- Fully functional context menu:** Edit WPF contains a built-in context menu with options to perform editing operations such as undo, redo, cut, copy, paste, select all, and so on. You can enable or disable the built-in context menu.

### Getting Started with WPF Syntax Editor

This section explains how to create an interactive code editor application like Microsoft Visual Studio Editor using the [EditControl](#).

### Assembly deployment

Refer to the [Control Dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

Refer to this [documentation](#) to find more details about installing nuget packages in a WPF application.

## Creating simple application with EditControl

You can create Windows Forms application with [EditControl](#) as follows:

1. [Creating the project](#)
2. [Adding control via Designer](#)
3. [Adding control via XAML](#)
4. [Adding control via C#](#)
5. [Loading a file into Document](#)
6. [Syntax Highlighting](#)

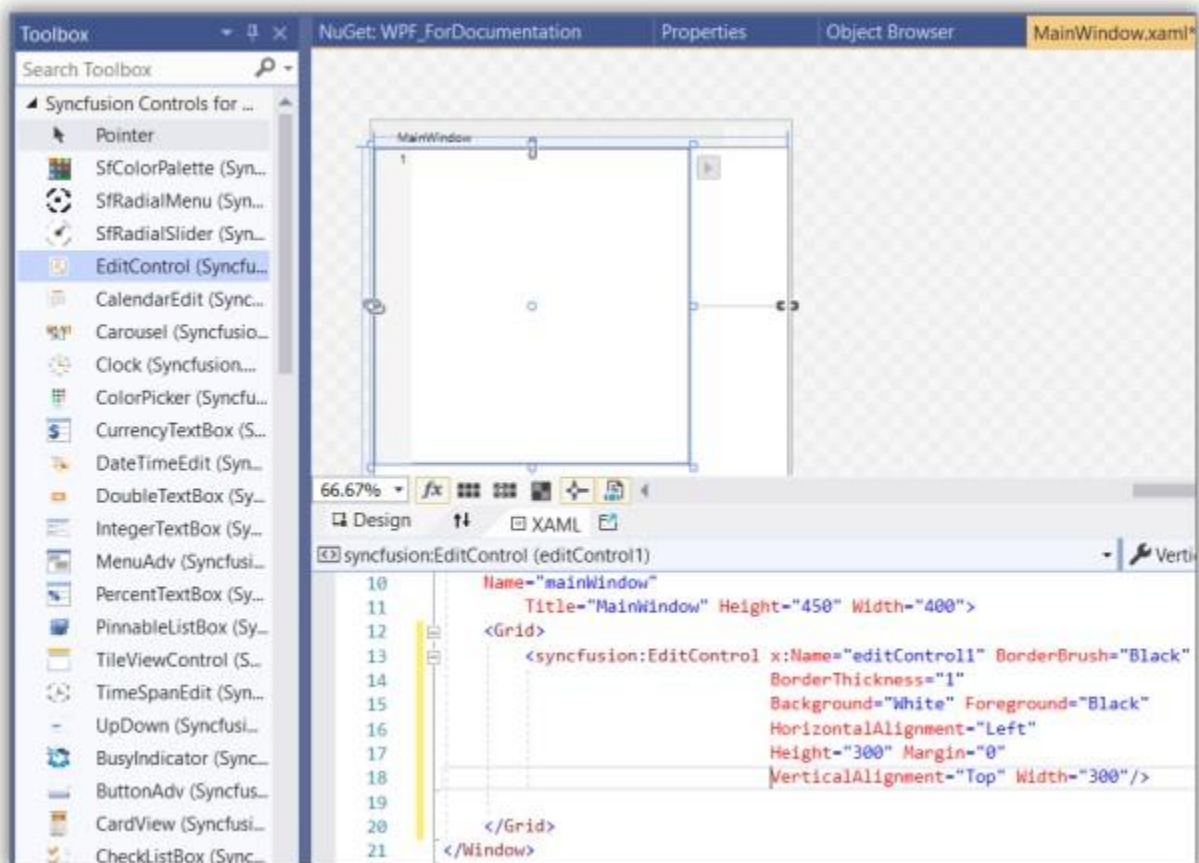
### Creating the project

Create a new WPF project in Visual Studio to display the EditControl.

### Adding EditControl via designer

The [EditControl](#) can be added to the application by dragging it from the toolbox and dropping it in the designer view. The following required assembly references will be added automatically:

- Syncfusion.Edit.WPF
- Syncfusion.GridCommon.WPF
- Syncfusion.Shared.WPF
- Syncfusion.Tools.WPF





### Adding WPF EditControl via XAML

To add the [EditControl](#) manually in XAML, follow these steps:

- 1) Create a new WPF project in Visual Studio. 2) Add the following required assembly references to the project: *Syncfusion.Edit.WPF* *Syncfusion.GridCommon.WPF* *Syncfusion.Shared.WPF* *Syncfusion.Tools.WPF* 3) Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf>, and declare the EditControl in XAML page.

#### XML

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WPF_ForDocumentation"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="WPF_ForDocumentation.MainWindow"
mc:Ignorable="d"
Name="mainWindow"
Title="MainWindow" Height="450" Width="400">
<Grid>
<syncfusion:EditControl x:Name="editControl1" BorderBrush="Black"
BorderThickness="1"
Background="White" Foreground="Black"
HorizontalAlignment="Left"
Height="300" Margin="0"
VerticalAlignment="Top" Width="300"/>
</Grid>
</Window>
```

### Adding WPF EditControl via C#

To add the [EditControl](#) manually in C#, follow these steps:

- 1) Create a new WPF application via Visual Studio. 2) Add the following required assembly references to the project: *Syncfusion.Edit.WPF* *Syncfusion.GridCommon.WPF* *Syncfusion.Shared.WPF* *Syncfusion.Tools.WPF* 3) Include the required namespace.

#### C#

```
using Syncfusion.Windows.Edit;
```

- 4) Create an instance of [EditControl](#), and add it to the window.

#### C#

```
public MainWindow()
{
InitializeComponent();
//Initializing EditControl and setting necessary property values.
EditControl editControl = new EditControl() {Height = 200, Width = 200,
Background = Brushes.White, Foreground = Brushes.Black };
this.Content = editControl;
}
```





### Loading a file into document

This option helps to load a file into the [EditControl](#).

Essential Edit WPF facilitates users to create, open, modify and save text files and programming language files. EditControl provides built-in support for a variety of text based file formats such as txt, cs, VB, SQL, XAML, and XML. It also enables to specify custom file types in the custom language configurations.

### Opening a file

The [DocumentSource](#) property of EditControl is used to specify the file to be opened with [EditControl](#). The following code can be used to set the [DocumentSource](#) property of EditControl is used to specify the file to be opened with EditControl property.

#### XML

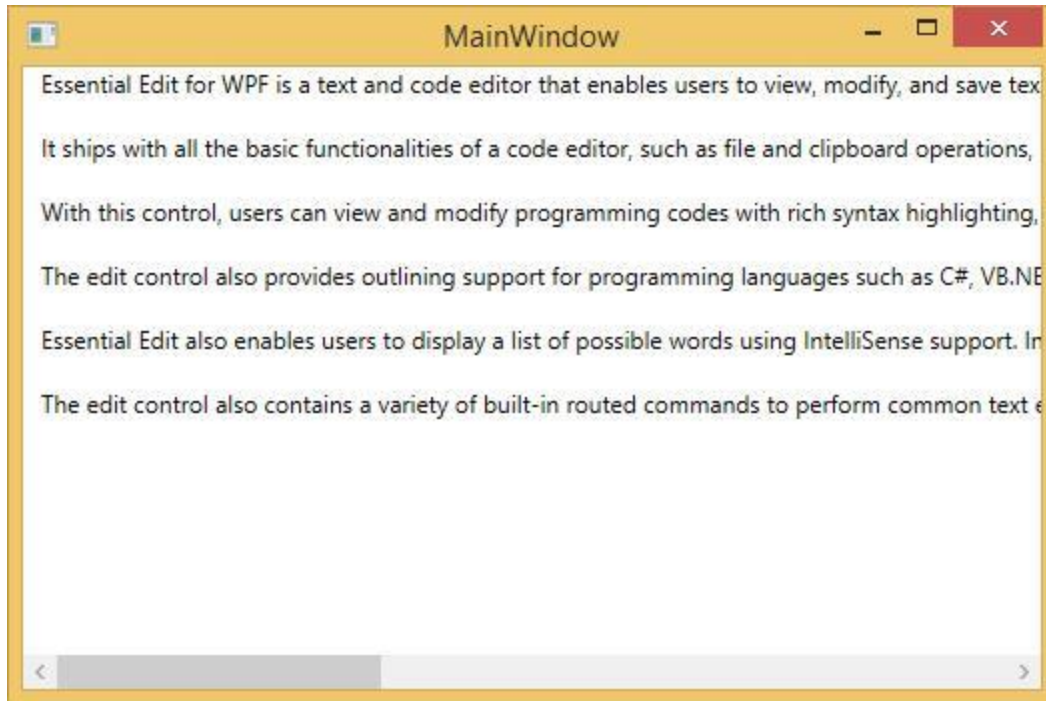
```
<syncfusion:EditControl x:Name="editControl" DocumentSource="C:\Content.txt"
ShowLineNumber="False" EnableOutlining="False"/>
```

#### C#

```
editControl.DocumentSource = @"C:\Content.txt";
```

#### VB.NET

```
editControl.DocumentSource = "C:\Content.txt"
```



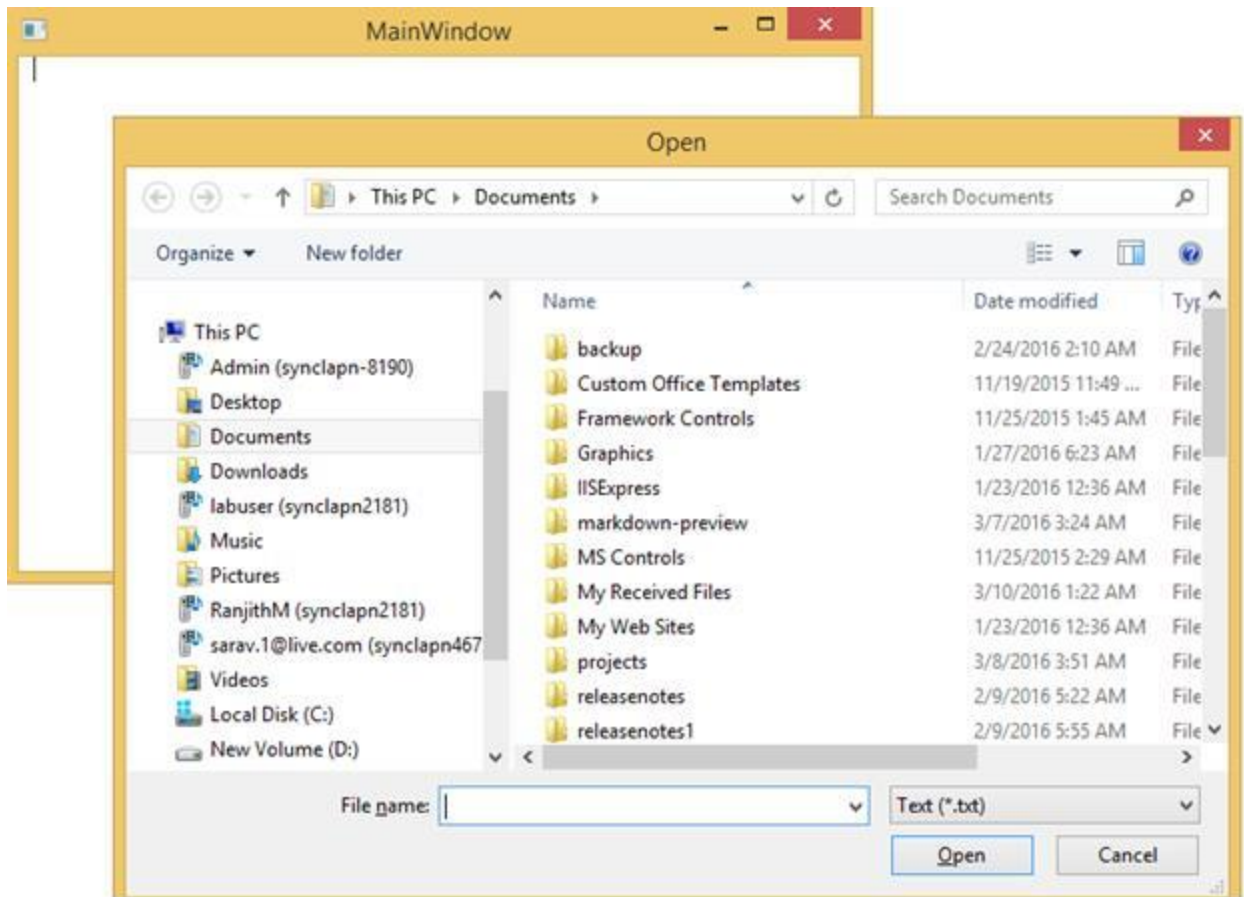
Files can also be opened using the [LoadFile](#) method. LoadFile method displays a OpenFileDialog to enable you to choose the file that needs to be opened in the [EditControl](#).

#### **C#**

```
editControl.LoadFile();
```

#### **VB.NET**

```
editControl.LoadFile()
```



### *Saving the text in a file*

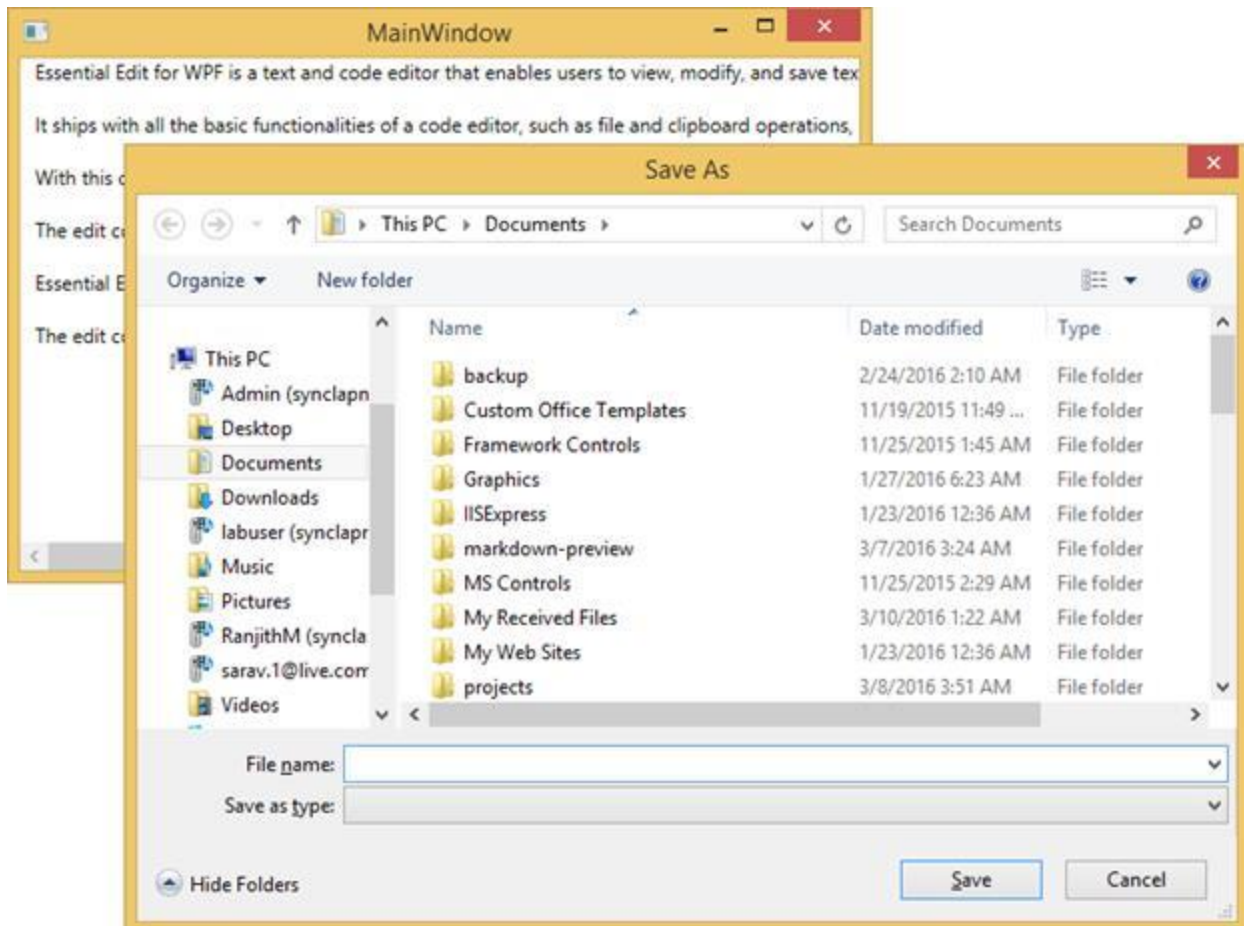
[SaveFile](#) method in the [EditControl](#) class is used to save the text in EditControl to a file. EditControl does support saving all the built-in languages, file types and custom language file type respectively.

### **C#**

```
editControl.SaveFile();
```

### **VB.NET**

```
editControl.SaveFile()
```



### Syntax highlighting

The [EditControl](#) offers mostly used languages like C#, VB, XML, XAML and SQL as built-in languages. It also provides support to configure new custom language.

The [EditControl](#) has built-in syntax highlighting support for the following languages:

- C
- C Sharp
- Custom
- Delphi
- HTML
- Java
- J Script
- PowerShell
- Text
- VBScript
- Visual Basic
- XML
- XAML
- SQL

With the language support, [EditControl](#) enables the users to create, open, modify and save programming codes from different file types. EditControl provides built in Syntax highlighting and outlining support for all supported languages with SQL being exception in outlining support. It also provides built-in IntelliSense support for all procedural languages such as C# and Visual Basic.

The [DocumentLanguage](#) property in the EditControl class enables the users to select the language. DocumentLanguage is a Language enum type property with default value as Text. The following lines of code can be used to change the [DocumentLanguage](#) property.

#### **XML**

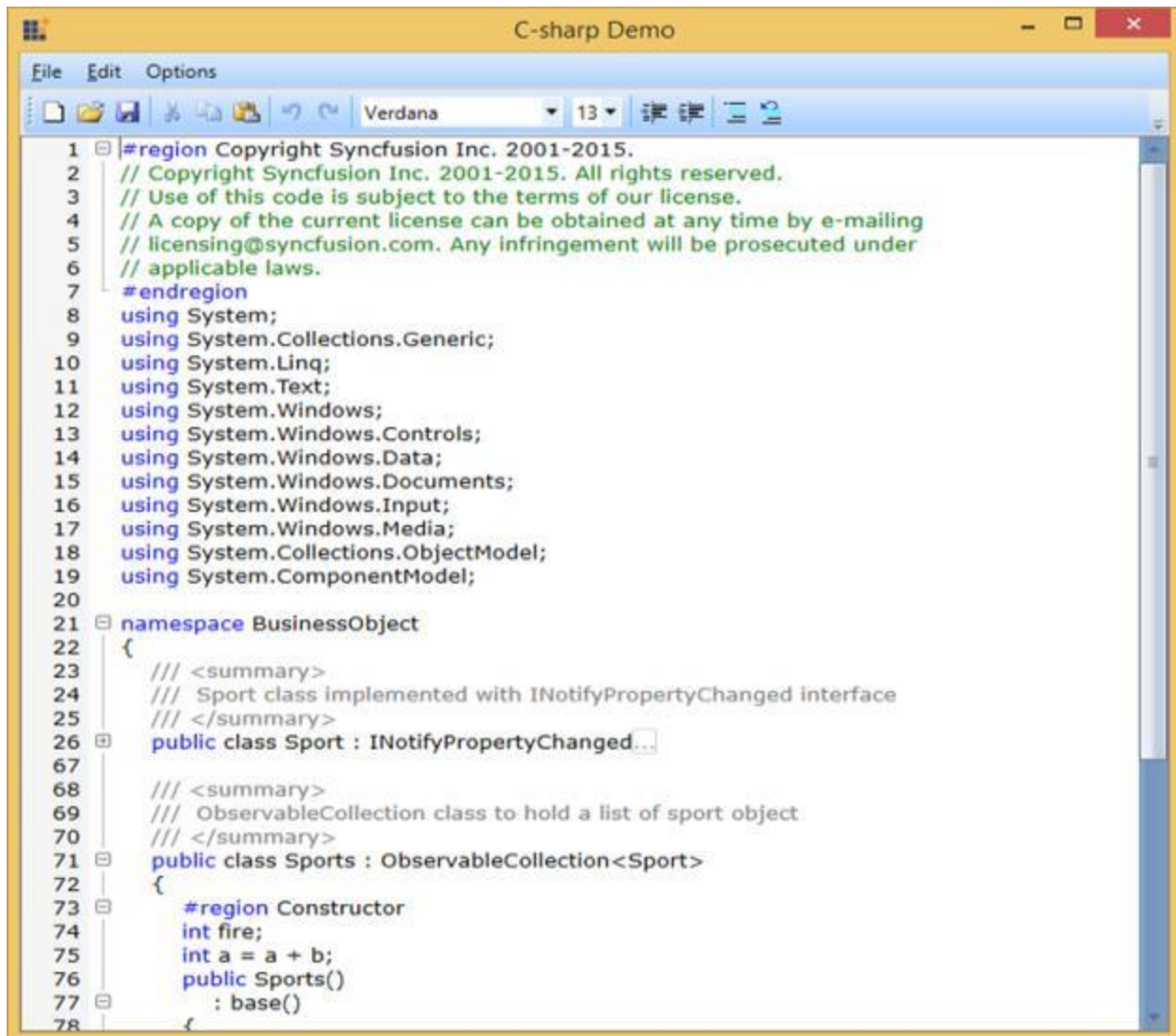
```
<syncfusion:EditControl x:Name="editControl" DocumentLanguage="CSharp"
DocumentSource="C:\Source.cs" FontSize="13"/>
```

#### **C#**

```
editControl.DocumentLanguage = Languages.CSharp;
```

#### **VB.NET**

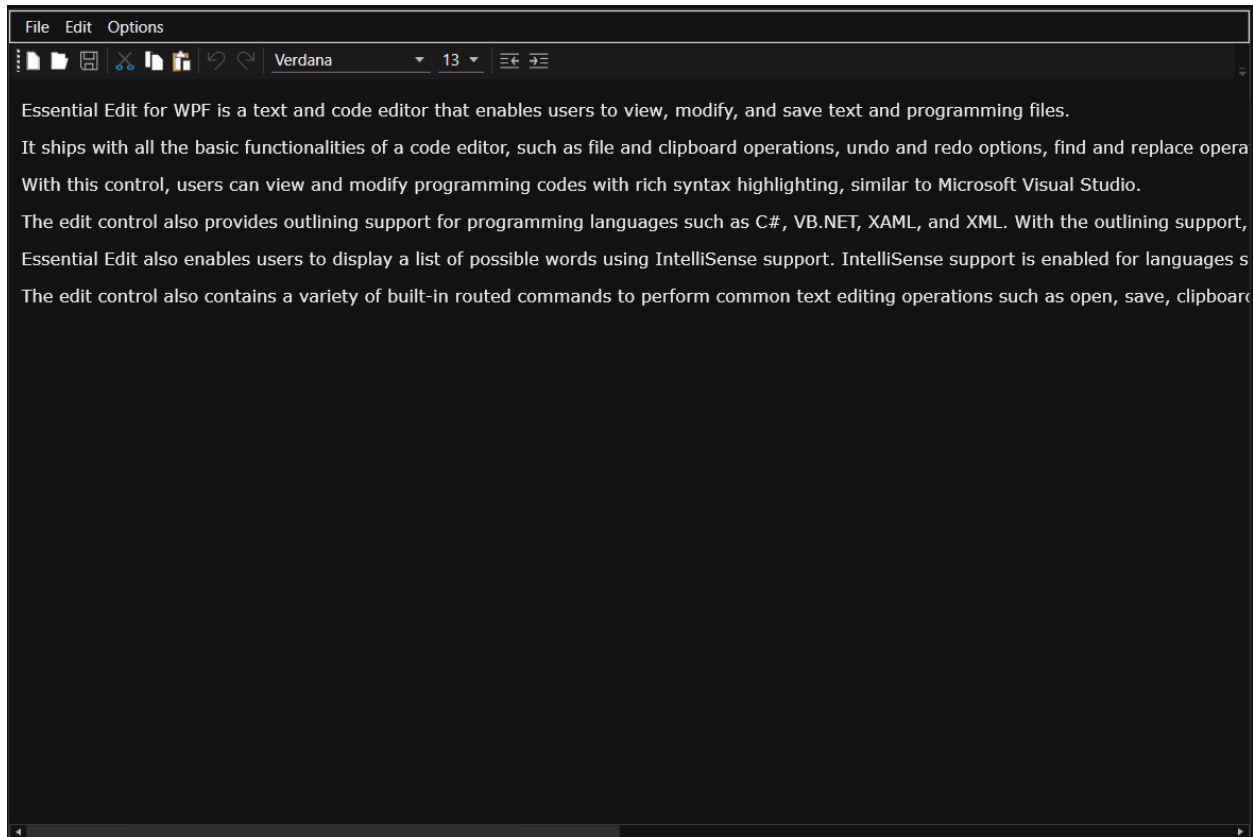
```
editControl.DocumentLanguage = Languages.CSharp
```



### Theme

SyntaxEditor supports various built-in themes. Refer to the below links to apply themes for the SyntaxEditor,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Syntax Editor Members in WPF Syntax Editor

EditControl class contains variety of properties, methods and events to enable the developers utilize this control easily. It is enhanced with maximum possible level of customization and facilities that are provided for the developers to use these controls effectively. This topic discusses about the properties, methods and events available in the EditControl class.

#### Properties

The following table illustrates the properties in EditControl and its usages.

Name of the property	Type	Description
AcceptsTab	Boolean	Gets or sets a boolean property to allow or restrict the usage of TAB key in EditControl.
AssemblyReferences	IEnumerable(Uri)	Gets or sets a collection of type Uri, this property enables you to specify the path of assemblies from EditControl and it should fetch Intellisense items automatically.
CurrentLanguage	LanguageBase	Gets a LanguageBase instance based on the DocumentLanguage property.

CursorIndex	Int	Gets current index of text where the cursor is located in EditControl pane.
CustomLanguage	LanguageBase	Gets or sets an instance of LanguageBase type indicating the custom language.
DocumentLanguage	Languages	Gets or sets Language for the text in EditControl. Syntax highlighting and outlining of text in EditControl are performed based on the language set using this property.
DocumentSource	String	Gets or sets Source for EditControl. The filename specified will be loaded in the EditControl.
EnableIntellisense	Boolean	Gets or sets a value indicating whether Intellisense is enabled in this instance of EditControl.
EnableOutlining	Boolean	Gets or sets a value indicating whether Outlining of text in EditControl should be applied.
FindOptions	FindOptions	Gets or sets an instance of FindOptions indicating the options selected in the FindReplace Window.
FindResultsTabHeight	Double	Gets or sets a value indicating desired height of the find results tab.
FindResultsTabVisibility	TabVisibility	Gets or sets a value indicating the Visibility mode of the find results tab. TabVisibility is enum type containing values such as Auto, Visible and Collapsed.
HorizontalScrollBarVisibility	ScrollBarVisibility	Gets or sets a value indicating Visibility of Horizontal ScrollBar. By default, it is set to Auto, wherein the ScrollBar will be visible when required.
IntellisenseBoxStyle	Style	Gets or sets a Style to be applied to IntelliSense listbox.
IntellisenseCustomItemsSource	IEnumerable	Gets or sets a collection of business object inherited from IntellisenseItem to be displayed in the IntellisenseBox.
IntellisenseItemTemplate	DataTemplate	Gets or sets a DataTemplate to be applied to an IntellisenseBox.
IntellisenseMode	IntellisenseMode	Gets or sets a value indicating the Intellisense mode. It supports two modes viz, Auto and



		<p>Custom.</p> <p>In Auto mode, intellisense items are auto generated based on the language lexems, assembly references included.</p> <p>In Custom mode, users have to set the <code>IntellisenseCustomItemsSource</code> property to specify the items to be displayed in the intellisense.</p>
<code>IntellisensePopupHeight</code>	Double	Gets or sets a value indicating desired height of the <code>IntellisensePopup</code> .
<code>IntellisensePopupWidth</code>	Double	Gets or sets a value indicating desired width of <code>IntellisensePopup</code> .
<code>IsFindResultsTabClosed</code>	Boolean	Gets or sets a value indicating whether the Find Results Tab is closed or not. This property has a value <code>true</code> if the Find results tab in the <code>EditControl</code> is closed; otherwise, <code>false</code> .
<code>IsReadOnly</code>	Boolean	Gets or sets a value indicating whether Read only mode is enabled or not.
<code>IsRedoEnabled</code>	Boolean	Gets or sets a value indicating whether this instance supports Redo operation or not.
<code>IsUndoEnabled</code>	Boolean	Gets or sets a value indicating whether this instance supports Undo operation or not.
<code>LineHeight</code>	Double	Gets and sets the height of each line in <code>EditControl</code> .
<code>LineNumber</code>	Int	Gets the line number where the cursor is currently located.
<code>Lines</code>	<code>LinesCollection</code>	Gets the Collection of Lines in <code>EditControl</code> .
<code>SearchResults</code>	<code>SearchResults</code>	Gets or sets a value indicating the Search results.
<code>SelectedText</code>	String	Gets a value indicating the Selected text of <code>EditControl</code> .
<code>ShowDefaultContextMenu</code>	Boolean	Gets or sets a value indicating whether built-in context menu should be displayed.
<code>ShowFindAndReplace</code>	Boolean	Gets or sets a value indicating whether Find and Replace functionality to enabled or not.

ShowLineNumber	Boolean	Gets or sets a value indicating whether Line Number to be displayed or not. Set this to true to display the Line number. Default value is true.
TabSpaces	Int	Gets or sets a value indicating the number of Spaces to include when the tab key is pressed.
Text	String	Gets or sets a value indicating text in EditControl.
VerticalScrollBarVisibility	ScrollBarVisibility	Gets or sets a value indicating Visibility of Vertical ScrollBar. By default, it is set to Auto, wherein the ScrollBar will be visible when required.

## Methods

The following table lists the methods available in EditControl class and its purpose.

Methods	Type	Description
ExpandLine(int index)	Void	a line, index in the argument refers the index of the line to be expanded (0 based value).
ExpandLineUpTopLevel(int index)	Void	Expands a line and its parent line upto the top most level. Index in the argument refers to the index of the line to be expanded (0 based value).
GetTextRange(int start, int end)	String	Returns the text between Start and End lines. Start and End denotes the index of the Start and End lines (0 based values).
LoadFile()	Boolean	LoadFile method is used to open a file in the EditControl. It shows up a OpenFileDialog in order to select the file to be opened using EditControl and returns a boolean value stating whether file opened is successful.
LoadFile(string filename)	Boolean	This method does not display OpenFileDialog and loads the file specified as the parameter of the method. It returns a boolean value stating the file open is successful.
SaveFile()	Boolean	Save method is used to save the text in the EditControl under a file name with different supported file types.

## Events

The following table lists the events available in EditControl class and its purpose.

Event	Type	Description
DocumentSourceChanged	PropertyChangedCallback	A property changed event gets raised when the DocumentSource property value is changed.
IntellisenseBoxOpening	IntellisenseBoxEventHandler	Gets raised before the Intellisense popup is displayed. This event can be used to cancel the Intellisense popup display or change ItemsSource of the Intellisense ListBox and so on.
IntellisenseBoxClosed	EventHandler	This event will be raised after the Intellisense popup is closed. This event can be used to perform any operation related to Intellisense after the popup is closed.
IntellisenseDrillDown	IntellisenseBoxEventHandler	Drill down in Intellisense occurs when the user types drill down char specified in the CurrentLanguage instance. When the drill down occurs, the Intellisense looks for any sub items are available for the selected Intellisense item and displays the popup if any. This event gets raised before the Intellisense popup is displayed after a drill down char is typed by the user. This event can be used to perform any operations related to Intellisense during drill down.
SelectedTextChanged	PropertyChangedCallback	A PropertyChangedCallback gets raised when the text in the EditControl is selected.
TextChanged	PropertyChangedCallback	A PropertyChangedCallback get raised when the text in the EditControl gets changed.
CaretPositionChanged	EventHandler	<p>This event will be raised when the caret position of the text in the EditControl is changed. The <u>CaretPositionChanged</u> event receives the <u>sender</u> and <u>CaretPositionEventArgs</u> as argument which has the following properties.</p> <ul style="list-style-type: none"> <li>• <u>Line Number</u> : Gets the current line number value of the EditControl.</li> <li>• <u>Cursor Index</u> : Gets the current cursor index value of the EditControl.</li> </ul>

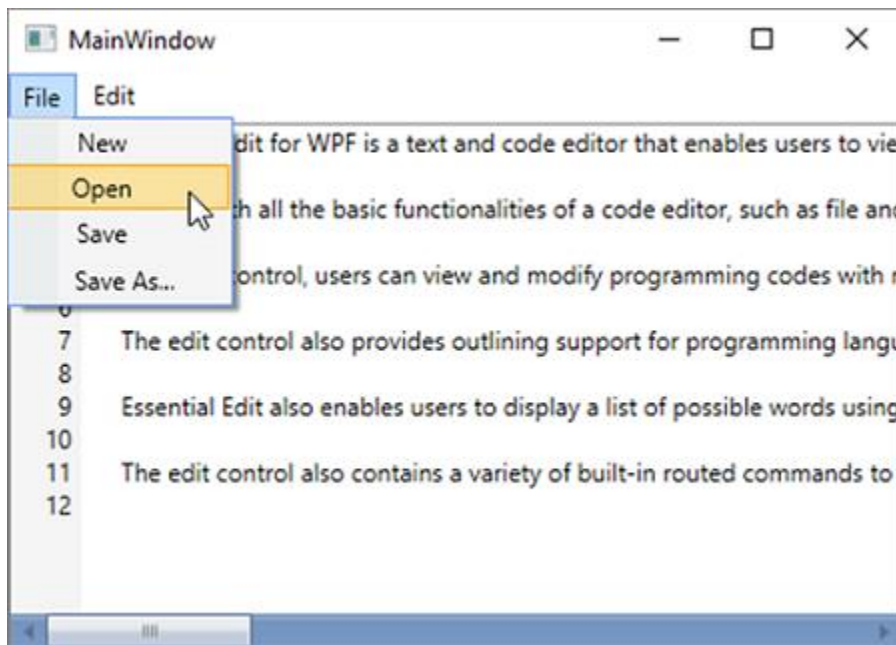
## Edit Commands in WPF Syntax Editor

Essential edit for WPF contains built-in RoutedUICommands for all editing and file operations such as select all, cut, copy, paste, new, open, save, save as and so on. The built-in RoutedUICommands can be bound to the edit control by using the **Command** property of the external controls such as button, menu item, and so on. The following lines of code can be used to bind the RoutedUICommands with external controls.

### XML

```
<Menu BorderBrush="LightGray" BorderThickness="0,0,1,2" Grid.ColumnSpan="1">
  <MenuItem Header="_File" Background="Transparent" Width="{Binding}" >
    <MenuItem Command="{x:Static sfedit:EditCommands.New}"
      CommandTarget="{Binding ElementName=Edit1}"/>
    <MenuItem Command="{x:Static sfedit:EditCommands.Open}"
      CommandTarget="{Binding ElementName=Edit1}"/>
    <MenuItem Command="{x:Static sfedit:EditCommands.Save}"
      CommandTarget="{Binding ElementName=Edit1}"/>
    <MenuItem Command="{x:Static sfedit:EditCommands.SaveAs}"
      CommandTarget="{Binding ElementName=Edit1}"/>
  </MenuItem>
</Menu>
<Border BorderThickness="1" BorderBrush="Gray" Grid.Row="1"
  Grid.ColumnSpan="2">
  <sfedit:EditControl Name="Edit1" EnableOutlining="False" Background="white"
    AllowDrop="True" BorderBrush="Black" BorderThickness="0" Margin="0"
    ShowLineNumber="True"
    Grid.Row="1" >
  </sfedit:EditControl>
</Border>
```

The following image displays **Open** edit command window.



## Text Selection in WPF Syntax Editor

The `SyntaxEditor` control supports the selection of content through mouse, keyboard, and touch interactions.

### Selection using Mouse

User can select text using the mouse by clicking on a position and dragging the selection to the desired line position.

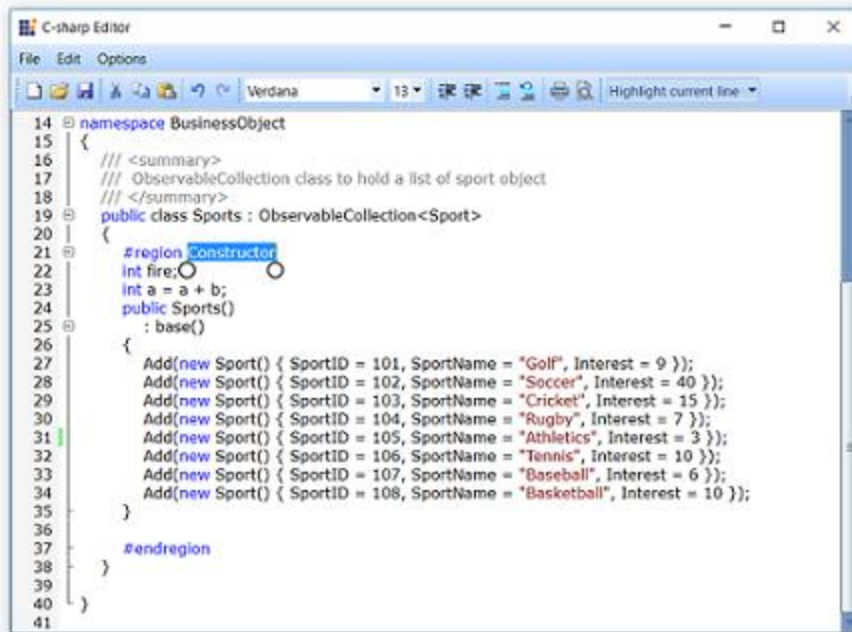
### Selection using Keyboard with shortcuts

The following keyboard shortcuts are supported for content selection.

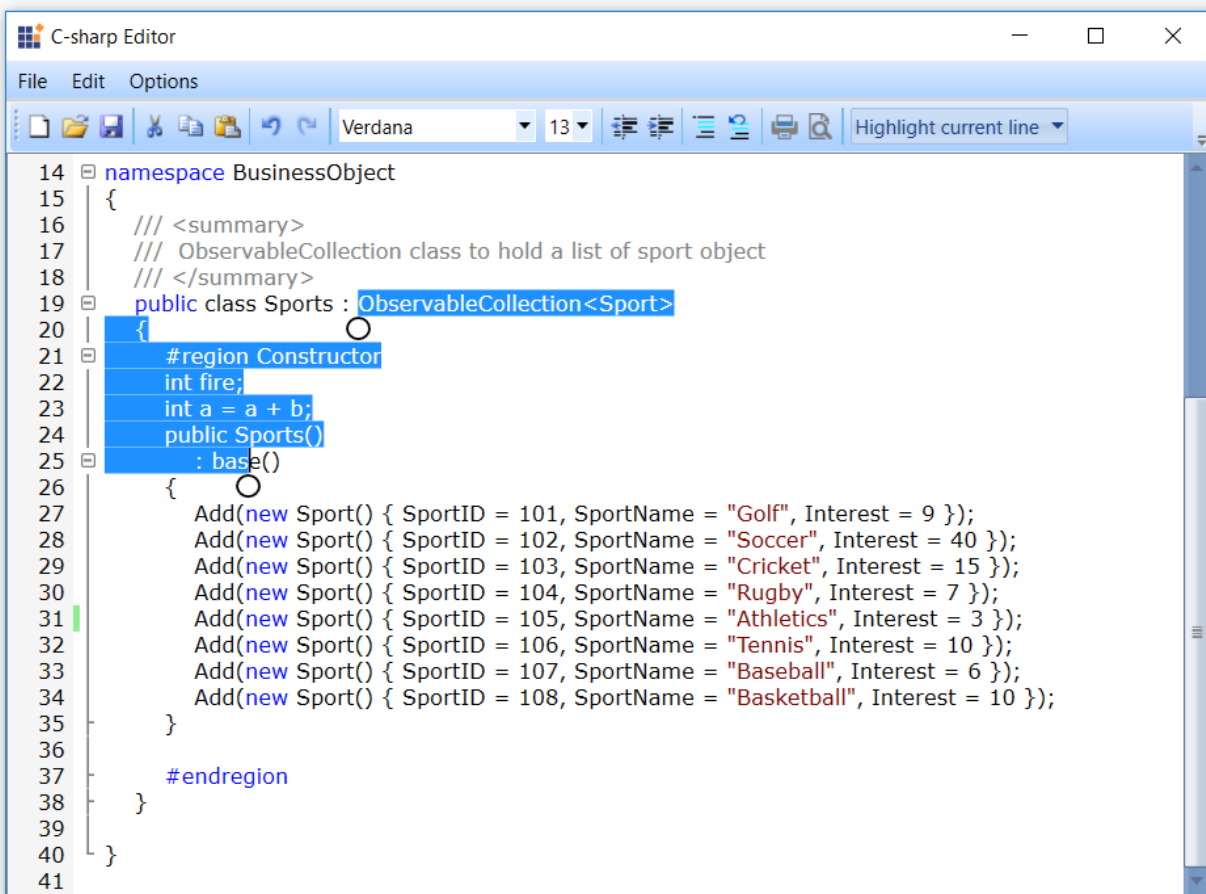
Selection Shortcut Keys	Description
Shift + Right arrow	Extends selection one position forward.
Shift + Left arrow	Extends selection one position backward.
Shift + Down arrow	Extends selection to the same position on the next line.
Shift + Up arrow	Extends selection to the same position on the previous line.
Shift + Home	Extends selection to the start of the current line.
Shift + End	Extends selection to the end of the current line.
Ctrl + Shift + Home	Extends selection to the document start position.
Ctrl + Shift + End	Extends selection to the document end position.
Ctrl + Shift + Right arrow	Extends selection to the current word end position.
Ctrl + Shift + Left arrow	Extends selection to the current word start position.
Ctrl + A	Selects the entire document.

### Selection using Touch

Users can select text by double-tapping the desired word. Selection pointers will be displayed at the start and end positions of the selected content.



Users can extend the selection by dragging the selection pointers. The following screenshot shows an extended selection of content.



When a selection goes beyond the view port of the SyntaxEditor control, a scroll viewer will automatically bring new content into the view and the text selection will extend to that content.

### Text Navigation in WPF Syntax Editor

Syntax Editor offers extensive support to text navigation. Users can perform navigation between characters, words, and line items using the built-in functions or keyboard shortcut keys.

#### Programmatic Navigation

##### *Character based navigation*

The following functions enables text navigation in the Syntax Editor in terms of characters or columns.

1. [MoveToPreviousIndex](#) to move cursor position in backward position.
2. [MoveToNextIndex](#) to move cursor position in forward position in .

#### **C#**

```
//To move cursor to previous index  
this.editControl.MoveToPreviousIndex();
```

#### **VB.NET**

```
//To move cursor to previous index  
Me.editControl.MoveToPreviousIndex()
```

##### *Word based navigation*

The following functions enables text navigation in the Syntax Editor in terms of words.

1. [MoveToNextWord](#) moves cursor to start position of next word.
2. [MoveToPreviousWord](#) moves cursor to start position of previous word.

#### **C#**

```
//To move cursor to next word  
this.editControl.MoveToNextWord();
```

#### **VB.NET**

```
//To move cursor to next word  
Me.editControl.MoveToPreviousWord()
```

##### *Line based navigation*

The following functions enables text navigation in the Syntax Editor in terms of lines.

1. [MoveToNextLine](#) to move cursor to same position on next line.
2. [MoveToPreviousLine](#) to move cursor to same position on previous line.
3. [MoveToLineStart](#) to move cursor to start position of current line.
4. [MoveToLineEnd](#) to move cursor to end position of current line.

#### **C#**

```
//To move cursor to next line
this.editControl.MoveToNextLine();
```

**VB.NET**

```
//To move cursor to next line
Me.editControl.MoveToNextLine()
```

## Keyboard Navigation

The following keyboard shortcuts are supported for cursor navigation in Syntax Editor.

Navigation Shortcut Keys	Descriptions
Right arrow	Moves the cursor to a position forward in the current line item.
Left arrow	Moves the cursor to the end position of the current line item.
Down arrow	Moves the cursor to the same position on the next line item.
Up arrow	Moves the cursor to the same position on the previous line item.
Home	Moves the cursor to the start position of the current line item.
End	Moves the cursor to the end position of the current line item.
Ctrl + Right arrow	Moves the cursor to the start position of the next word.
Ctrl + Left arrow	Moves the cursor to the start position of the previous word.
Ctrl + Home	Moves the cursor to the document start position.
Ctrl + End	Moves the cursor to the end position of document.

## Line Background Customization in WPF Syntax Editor

## Applying line background customization

The `SetLineBackground` function helps to customize the background color of specific lines.

*Method*

`SetLineBackground(lineNumber, fullLine, brush)`: Helps to customize the background of line.

*Arguments*

**lineNumber**: Specifies the line number where the cursor is currently located in EditControl.

**fullLine**: Specifies whether to highlight full line or not.

**brush**: Specifies the `Brush` for background customization.

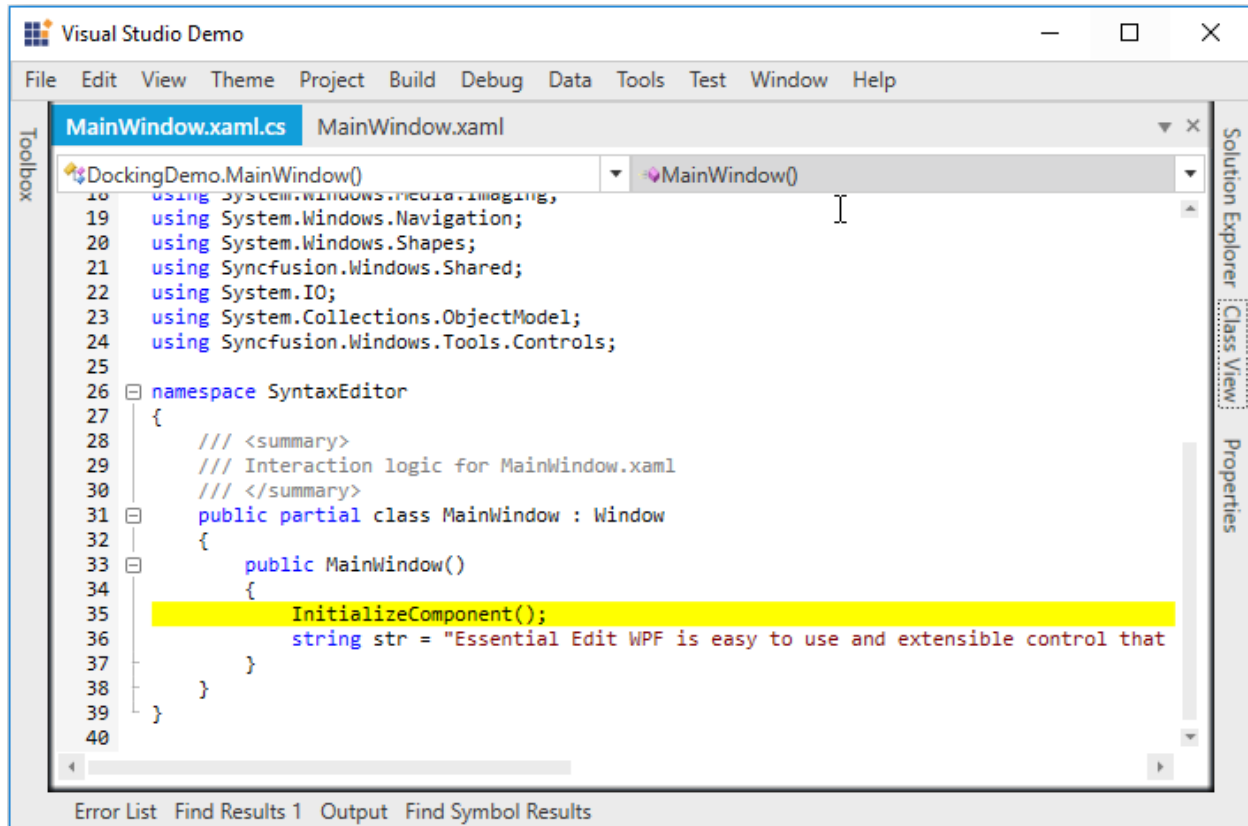
**C#**



```
// Set the background to a specified Line.
this.editControl1.SetLineBackground(this.editControl1.LineNumber, true,
Brushes.Yellow);
```

**VB.NET**

```
' Set the background to a specified Line.
this.editControl1.SetLineBackground(this.editControl1.LineNumber, true,
Brushes.Yellow)
```



## Resetting line background customization

The `ResetLineBackground` function helps to reset the background color of customized lines.

*Method*

`ResetLineBackground(lineNumber)`: Helps to reset the background customization of line.

*Arguments*

**lineNumber**: Specifies the line number where the cursor is currently located in EditControl.

**C#**

```
// Reset the background to a specified Line.
this.editControl1.ResetLineBackground(this.editControl1.LineNumber);
```

**VB.NET**

```
' Reset the background to a specified Line.
```

```
this.editControll1.ResetLineBackground(this.editControll1.LineNumber)
```

### On demand line background customization

The **OnBeforeLineRender** event customizes the background color of the line on demand.

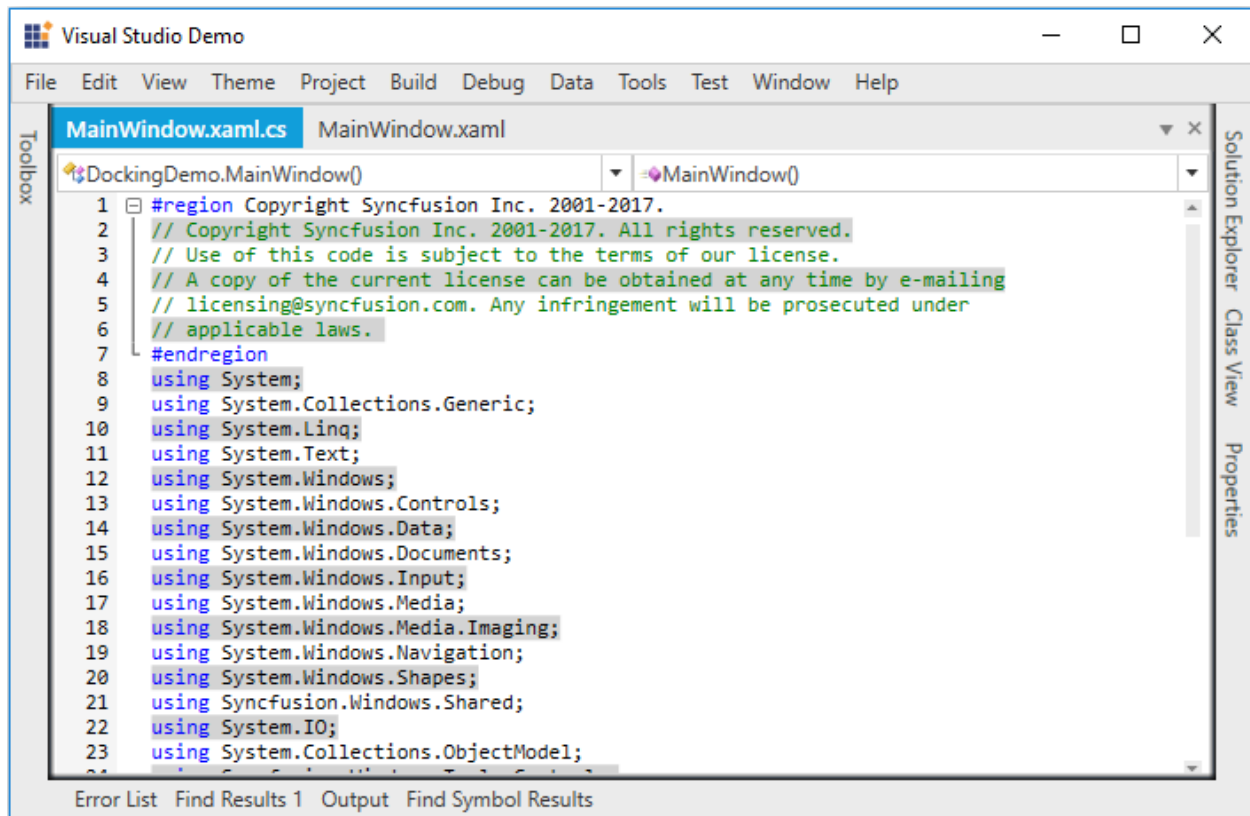
#### C#

```
public MainWindow()
{
    InitializeComponent();
    editControll1.DocumentSource = "../..../Source.cs";
    editControll1.OnBeforeLineRender += new
    Syncfusion.Windows.Edit.OnBeforeLineRenderEventHandler(editControll1_OnBefore
    LineRender);
}
// Invoked when before the Line Render.
private void editControll1_OnBeforeLineRender(object sender,
    Syncfusion.Windows.Edit.OnBeforeLineRenderArgs args)
{
    if (args.LineItem.LineNumber % 2 == 0)
    {
        args.BackgroundColor = Brushes.LightGray;
        args.IsFullLine = false;
    }
}
```

#### VB.NET

```
public MainWindow()
{
    InitializeComponent()
    editControll1.DocumentSource = "../..../Source.cs"
    editControll1.OnBeforeLineRender += new
    Syncfusion.Windows.Edit.OnBeforeLineRenderEventHandler(editControll1_OnBefore
    LineRender)
}
' Invoked when before the Line Render.
private void editControll1_OnBeforeLineRender(object sender,
    Syncfusion.Windows.Edit.OnBeforeLineRenderArgs args)
{
    if (args.LineItem.LineNumber % 2 == 0)
    {
        args.BackgroundColor = Brushes.LightGray
        args.IsFullLine = false;
    }
}
```

**Note:** The on demand line background customization is recommended when the **EditControl** is loaded with huge data.



## Printing in WPF Syntax Editor

EditControl provides support to print the content displayed in the EditControl using **Print** method.

### C#

```
editControl.Print();
```

### VB.NET

```
' Invoke the print method  
editControl.Print
```

## Print Preview

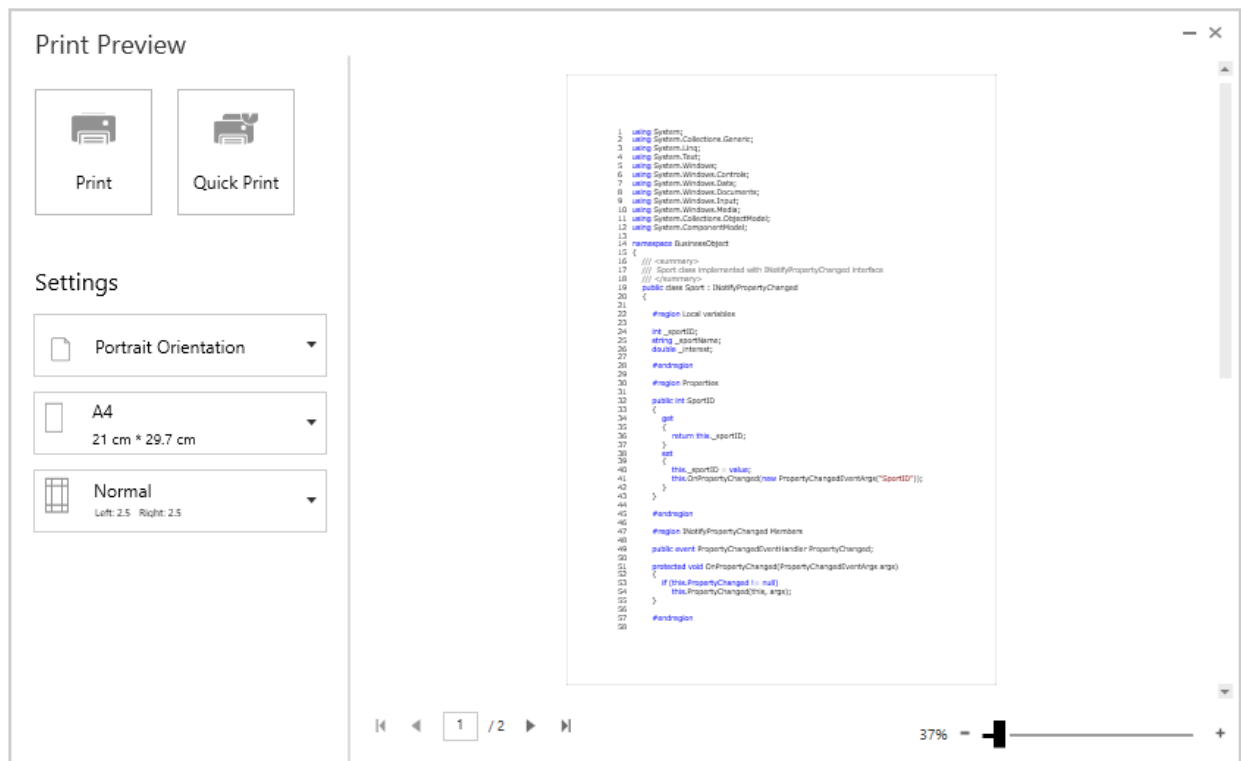
EditControl provides option to display print preview to review and customize the document in desired format before printing. Print preview window can be opened by calling **ShowPrintPreview** method.

### C#

```
editControl.ShowPrintPreview();
```

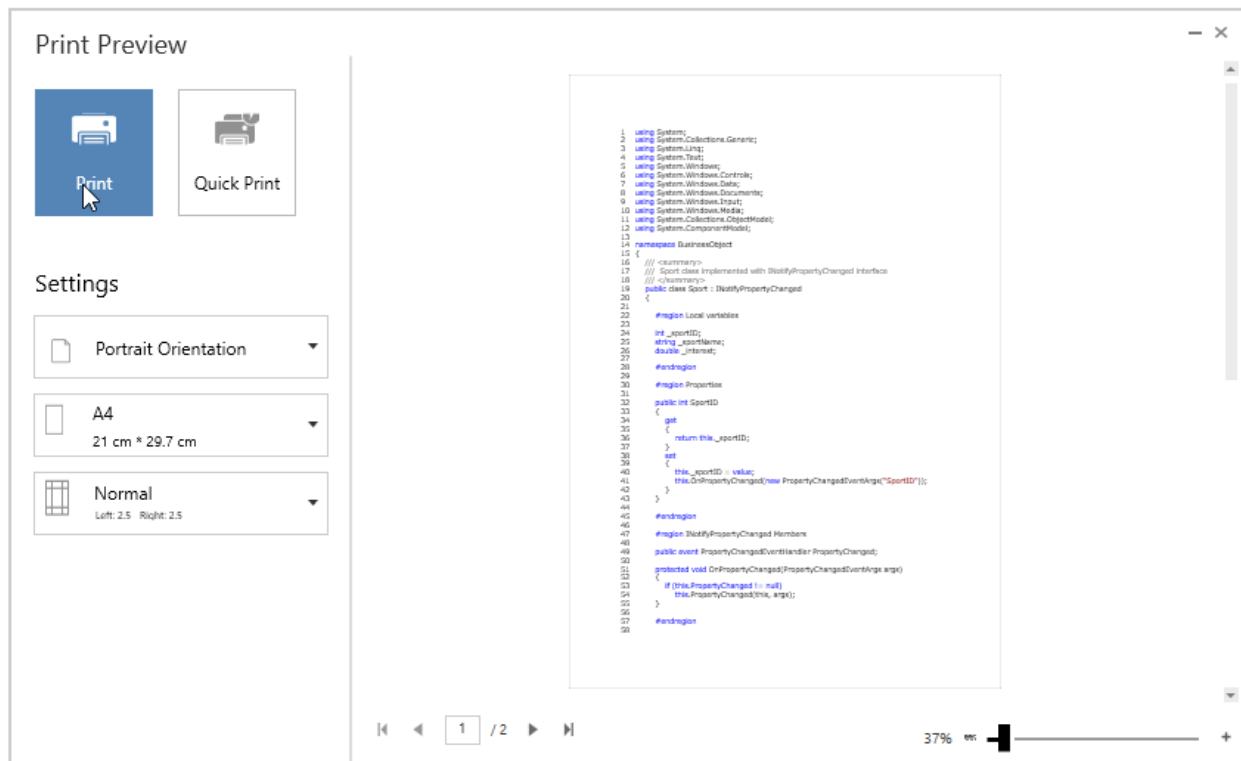
### VB.NET

```
' Invoke the print Preview Window  
editControl.ShowPrintPreview
```

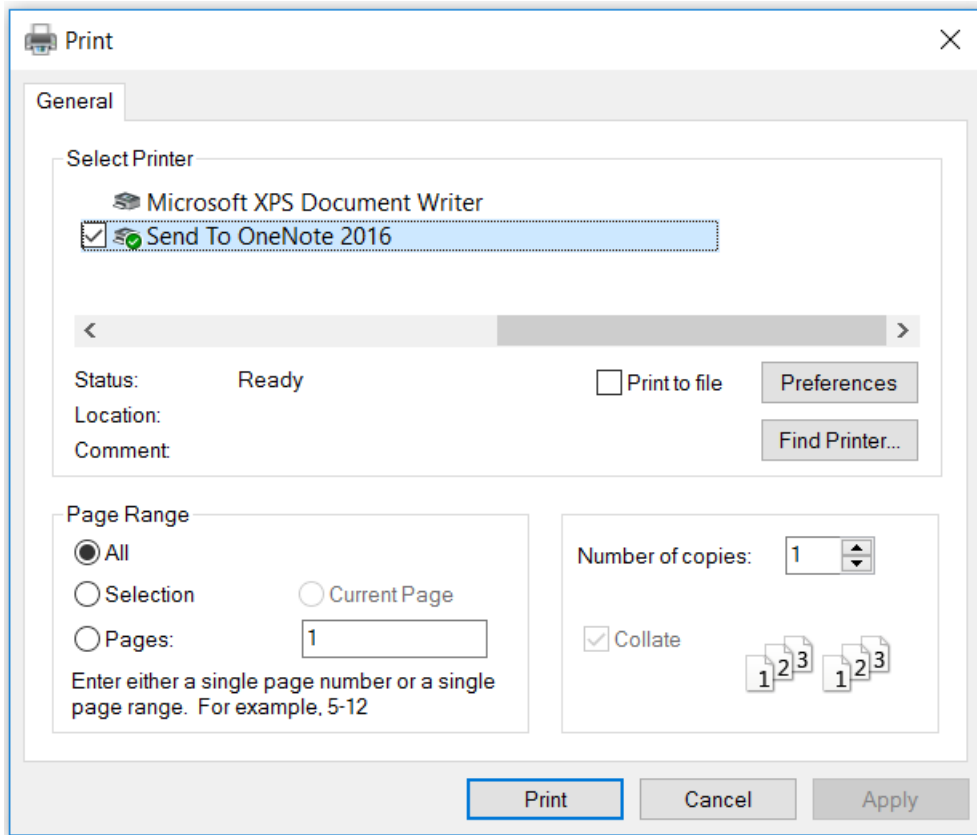


### Print

Print preview window has Print and Quick Print Buttons which needs to be clicked to print the EditControl.



**Step 1:** Clicking the Print button opens the System print dialog where user can select the printer and set the number of copies to be printed.



**Step 2:** Clicking the Quick Print button, directly print the pages using default printer without opening the print dialog.

### Print Settings

EditControl provides various options to customize page settings using `EditControl.PrintSettings` property of type `PrintSettings`.

### Orientation

EditControl provides support to switch between Portrait (more rows but fewer columns) and Landscape (more columns but fewer rows) orientation while printing. Orientation can be changed by setting `PrintSettings.Orientation` Property.

### C#

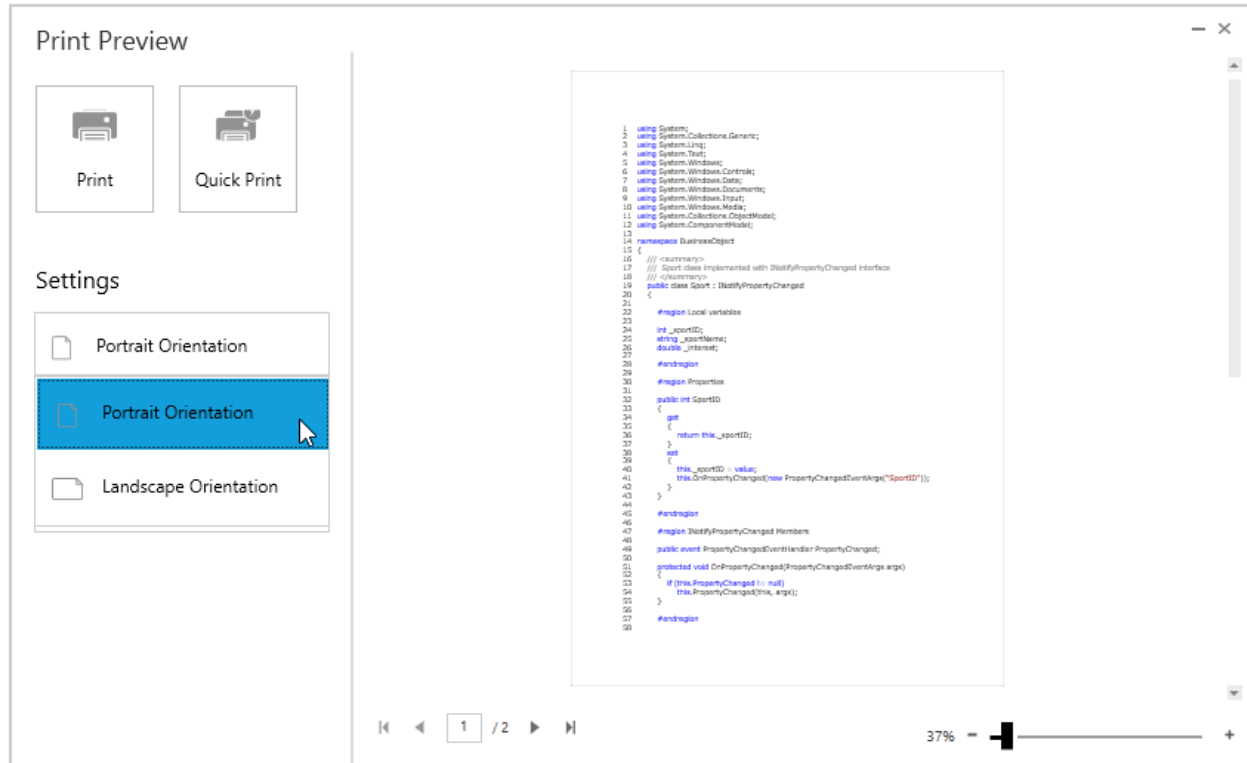
```
editControl.PrintSettings = new PrintSettings();
editControl.PrintSettings.Orientation =
Syncfusion.Windows.Shared.Printing.PrintOrientation.Landscape;
editControl.ShowPrintPreview();
```

### VB.NET

```
editcontrol.PrintSettings = New PrintSettings
editcontrol.PrintSettings.Orientation =
Syncfusion.Windows.Shared.Printing.PrintOrientation.Landscape
```

```
editControl.ShowPrintPreview
```

Print orientation can be changed in print preview at runtime by selecting from orientation drop-down in print preview.



### Page size

EditControl provides support to change the page size. Page size can be changed by setting `PrintSettings.PageWidth` and `PrintSettings.PageHeight` properties.

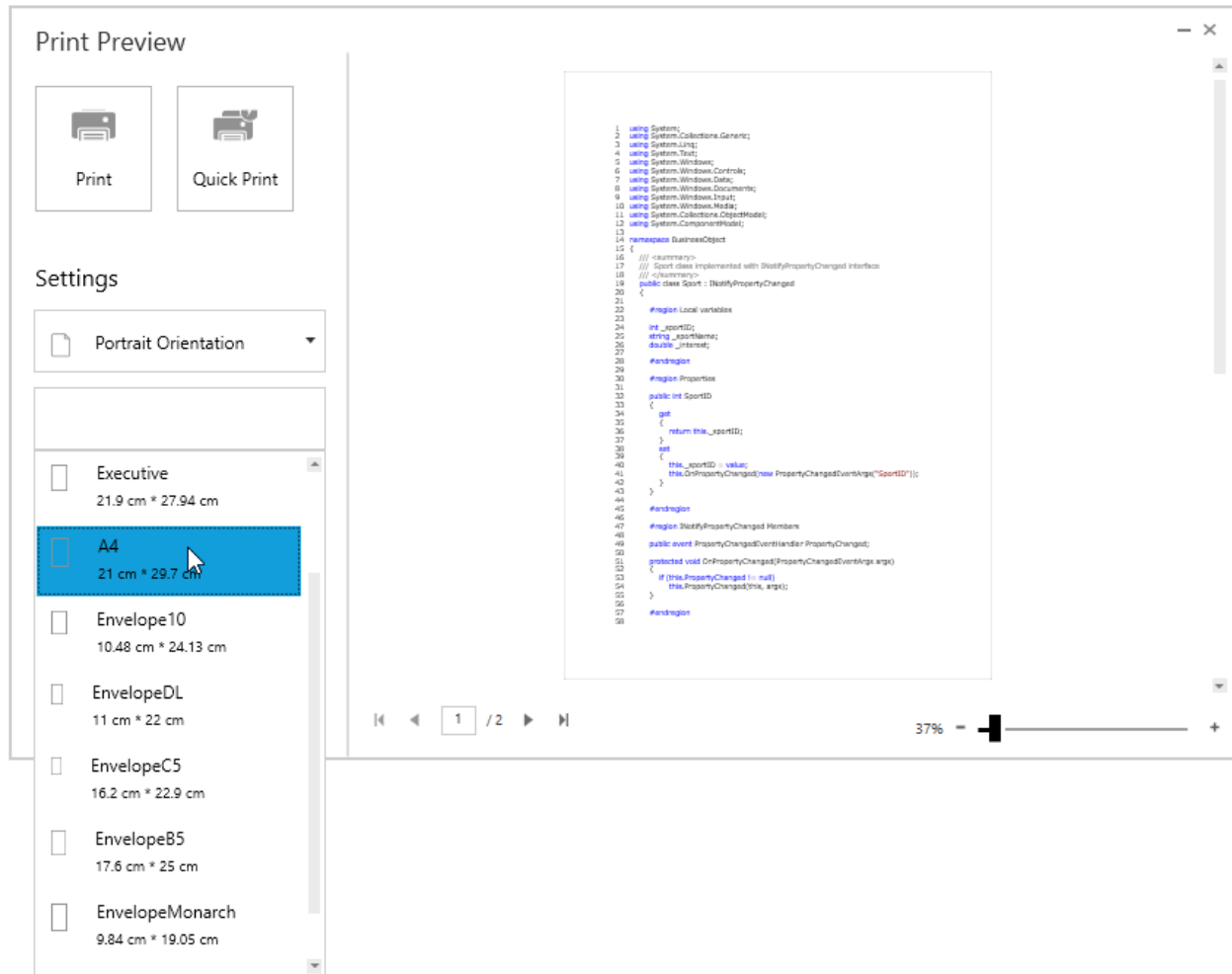
### C#

```
editControl.PrintSettings = new PrintSettings();
editControl.PrintSettings.PageHeight = 800;
editControl.PrintSettings.PageWidth = 800;
editControl.Print();
```

### VB.NET

```
editControl.PrintSettings = New PrintSettings
editControl.PrintSettings.PageHeight = 800
editControl.PrintSettings.PageWidth = 800
editControl.Print
```

Page size can be changed in print preview also by selecting from page-size drop-down which displays pre-defined page sizes. You can also manually enter custom page width and height in the editors below page-size drop-down and press OK to apply the custom width and height for the page.



### Page margin

EditControl provides support to change the page margins to adjust content in printed page. Page margin can be changed by setting `PrintSettings.PageMargin` property.

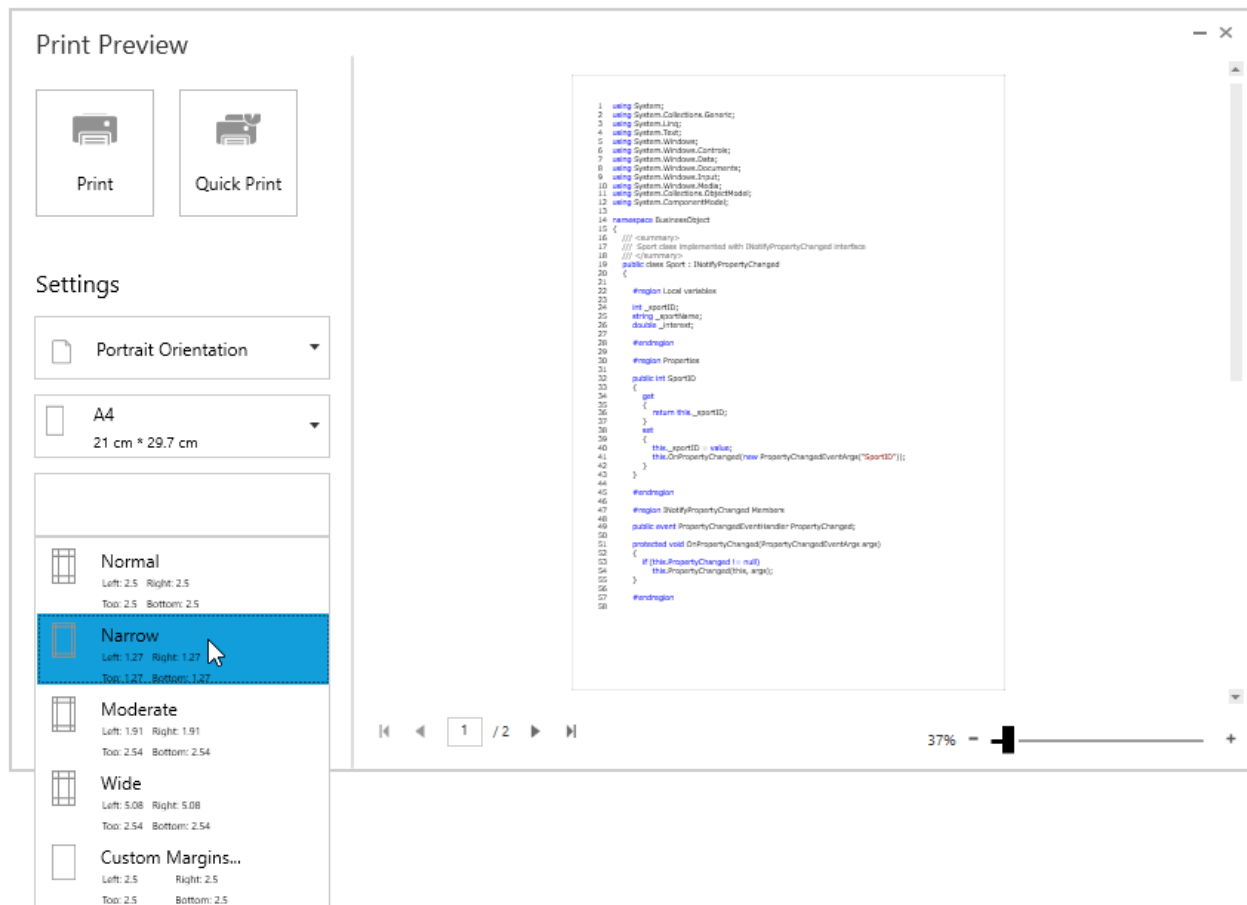
### C#

```
editControl.PrintSettings = new PrintSettings();
editControl.PrintSettings.PageMargin = new Thickness(5);
editControl.Print();
```

### VB.NET

```
editcontrol.PrintSettings = New PrintSettings
editcontrol.PrintSettings.PageMargin = New Thickness(5)
editcontrol.Print
```

Page margin can be changed in print preview also by selecting from pre-defined page margin from margin drop-down. You can manually enter custom margins in the editors below margin drop-down and press OK to apply the custom margin.



### Setting Header and Footer

EditControl provides a way to display additional content at the top (Header) or bottom (Footer) of the page while printing. This can be achieved by setting `PageHeaderHeight`, `PageHeaderTemplate`, `PageFooterHeight`, `PageFooterTemplate` properties in `PrintSettings`.

Steps to add page header while printing,

1. Create DataTemplate in Application.Resources.

### XML

```
<Application.Resources>
<DataTemplate x:Key="pageHeaderTempalte">
<Grid Background="Gray">
<TextBlock Text="Syncfusion"
FontSize="18"
FontWeight="Bold"
Foreground="White"
HorizontalAlignment="Center"/>
</Grid>
</DataTemplate>
</Application.Resources>
```



2. Set the above defined DataTemplate to `PrintSettings.PageHeaderTemplate` and assign value for `PrintSettings.PageHeaderHeight` property also.

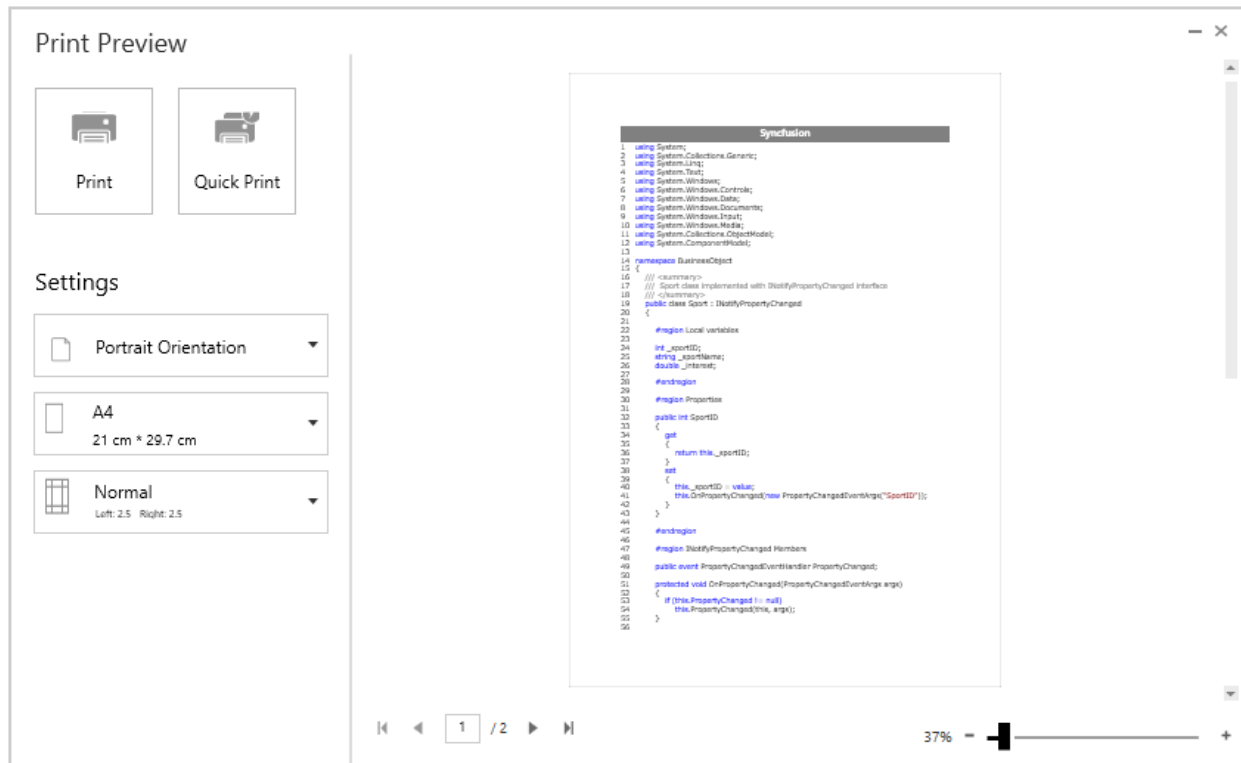
### C#

```
editcontrol.PrintSettings = new PrintSettings();
editcontrol.PrintSettings.PageHeaderHeight = 30;
editcontrol.PrintSettings.PageHeaderTemplate =
Application.Current.Resources["pageHeaderTemplate"] as DataTemplate;
editcontrol.ShowPrintPreview();
```

### VB.NET

```
editcontrol.PrintSettings = New PrintSettings
editcontrol.PrintSettings.PageHeaderHeight = 30
editcontrol.PrintSettings.PageHeaderTemplate =
CType(Application.Current.Resources("pageHeaderTemplate"), DataTemplate)
editcontrol.ShowPrintPreview
```

3. Now run the application and you can see page header in all the pages. In the same way, you can set `PrintSettings.PageFooterTemplate` also.



### Printing Current Date time and Page number

You can print current Date and Time at each page by setting the `PageFooterHeight`.

You can get the page number from `PrintPageControl`.

### XML

```
<Application.Resources>
```

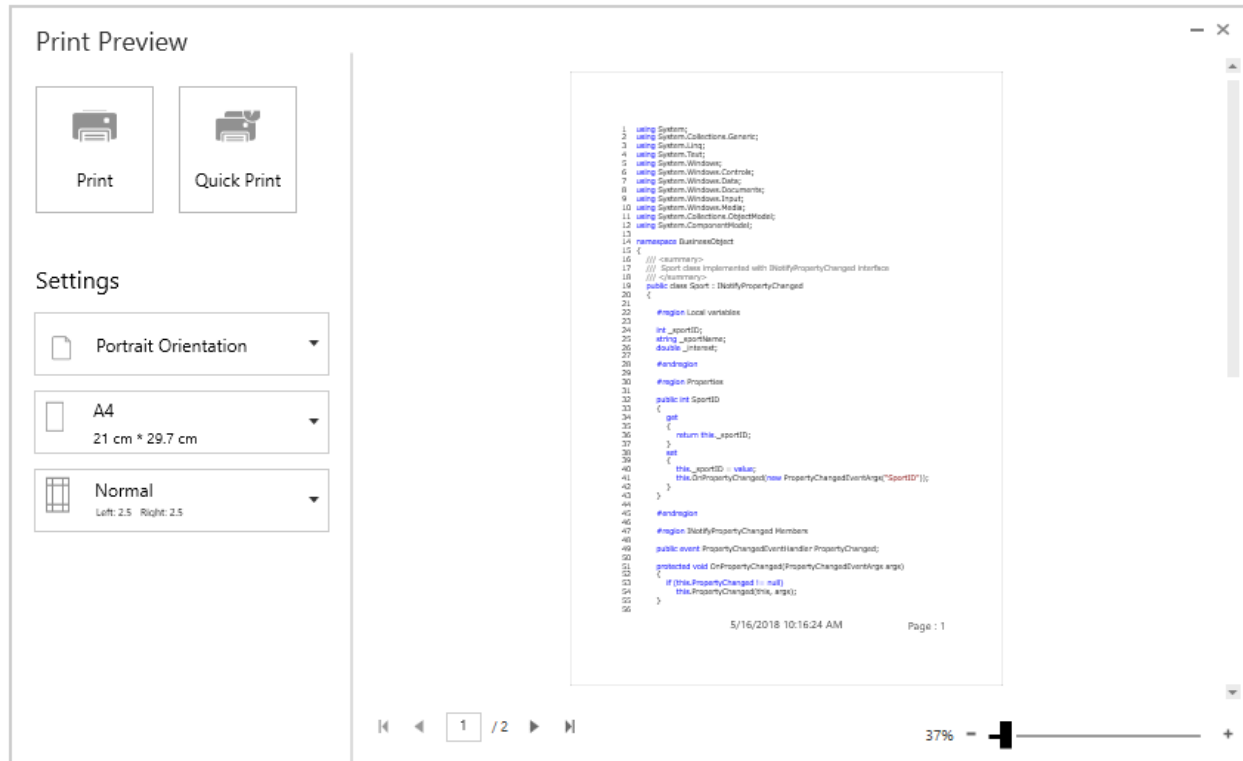
```
<DataTemplate x:Key="pageFooterTempalte">
<Grid>
<TextBlock HorizontalAlignment="Center"
FontSize="20"
Text="{Binding Source={x:Static system:DateTime.Now}}"/>
<TextBlock Margin="0,0,10,0"
HorizontalAlignment="Right"
VerticalAlignment="Center" FontSize="20">
<TextBlock.Text>
<Binding Path="PageIndex"
RelativeSource="{RelativeSource Mode=FindAncestor,
AncestorType={x:Type syncfusion:PrintPageControl}}"
StringFormat="Page : {0}" />
</TextBlock.Text>
</TextBlock>
</Grid>
</DataTemplate>
</Application.Resources>
```

### **C#**

```
editcontrol.PrintSettings = new PrintSettings();
editcontrol.PrintSettings.PageFooterHeight = 30;
editcontrol.PrintSettings.PageFooterTemplate =
Application.Current.Resources["pageFooterTempalte"] as DataTemplate;
editcontrol.ShowPrintPreview();
```

### **VB.NET**

```
editcontrol.PrintSettings = New PrintSettings
editcontrol.PrintSettings.PageHeaderHeight = 30
editcontrol.PrintSettings.PageFooterTemplate =
CType(Application.Current.Resources("pageFooterTemplate"), DataTemplate)
editcontrol.ShowPrintPreview
```



## Basic Editing

### *Editing Text in EditControl WPF*

[EditControl](#) supports displaying or editing string values with any number of lines. It also supports all basic editing operations such as typing, cut, copy, paste, delete, backspace, undo and redo operations.

[EditControl](#) supports performing edit operations through keyboard and mouse. It supports shortcut keys for basic editing operations such as cut, copy, paste, undo and redo operations. EditControl also has a built-in context menu to perform common editing operations such as undo, redo, cut, copy, paste, select all operations using mouse. User can enable/ disable the built-in context menu. For more information refer to [Default Context Menu](#).

[Text](#) property of the EditControl contains the text available in the EditControl at any point of time. The [Text](#) property of EditControl gets updated when you edit the text in the control by typing, editing the text by cut, paste, delete, undo or redo operations or by setting the Text property exclusively at runtime. The TextChanged event of EditControl gets raised when the Text property gets changed. Refer to [EditControl Events](#) section to know more about TextChanged and other events available in EditControl.

Set a simple string as EditControl's Text by using the following line of code.

### XML

```
<syncfusion:EditControl x:Name="editControl" Text="This is Syncfusion's EditControl"/>
```

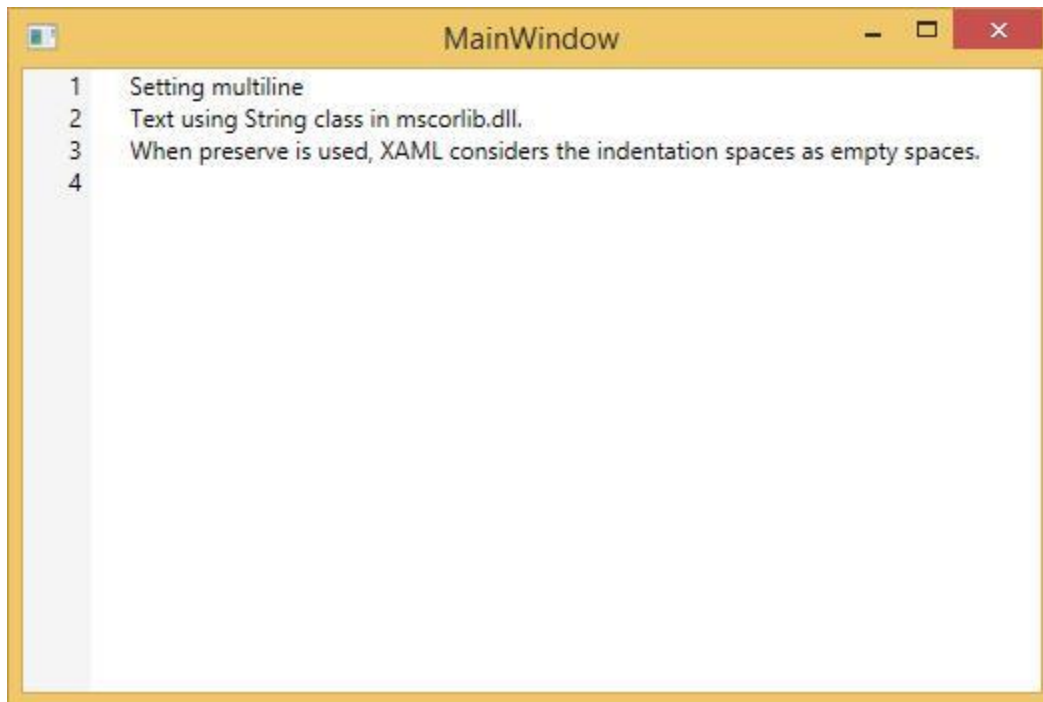
### C#

```
editControl.Text = "Setting Text property from code behind (C#)";
```

To set a multi-line string as Text property through **XAML**, String class in **microsoft.dll** can be used with **xml:space="preserve"**. The following lines of code can be used to set a multiline text from **XAML**.

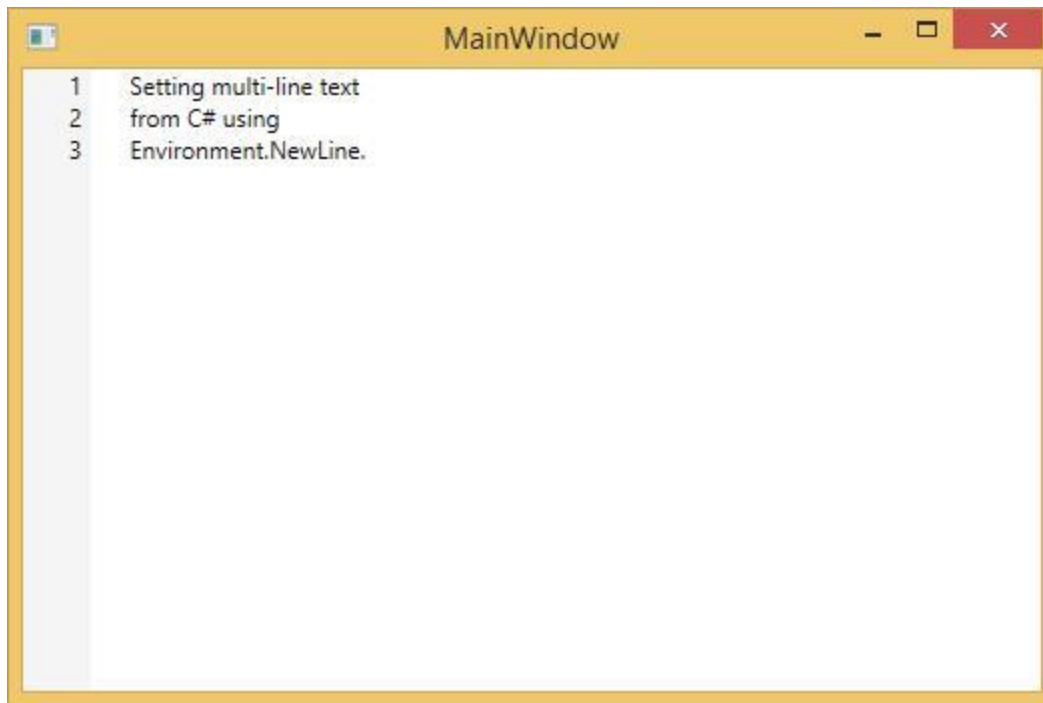
#### **XAML**

```
<syncfusion:EditControl x:Name="editControl">
<library:String xml:space="preserve" xmlns:library="clr-
namespace:System;assembly=microsoft.dll">Setting multiline
Text using String class in microsoft.dll.
When preserve is used, XAML considers the indentation spaces as empty
spaces.
</library:String>
</syncfusion:EditControl>
```



#### **C#**

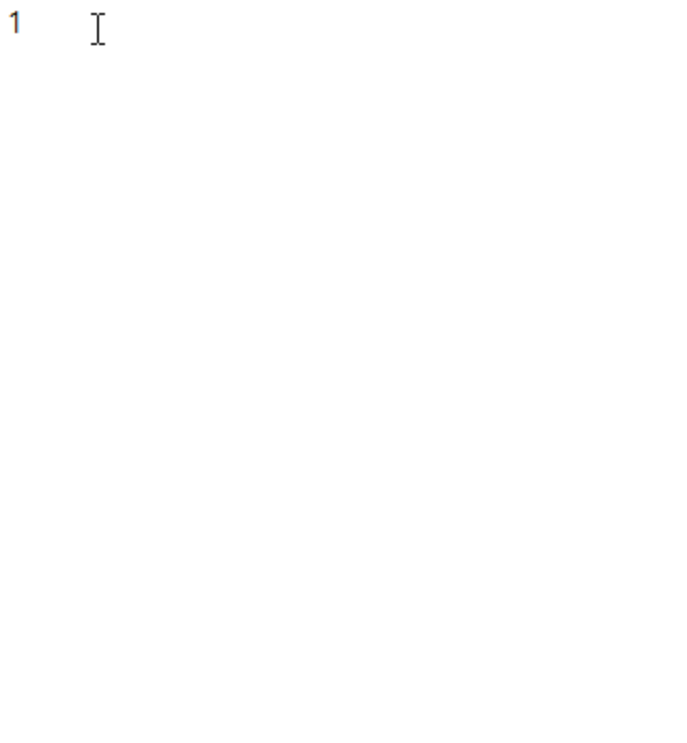
```
editControl.Text = @"Setting multi-line text" + Environment.NewLine + "from
C# using" + Environment.NewLine + "Environment.NewLine.";
```



#### *Indentation in EditControl*

[EditControl](#) allows to auto indent the text when enter key pressed to add new lines. Auto indentation can be enabled by setting [IsAutoIndentationEnabled](#) to 'true'. Auto indentation works based on [IndentingOptions](#) property which has following options,

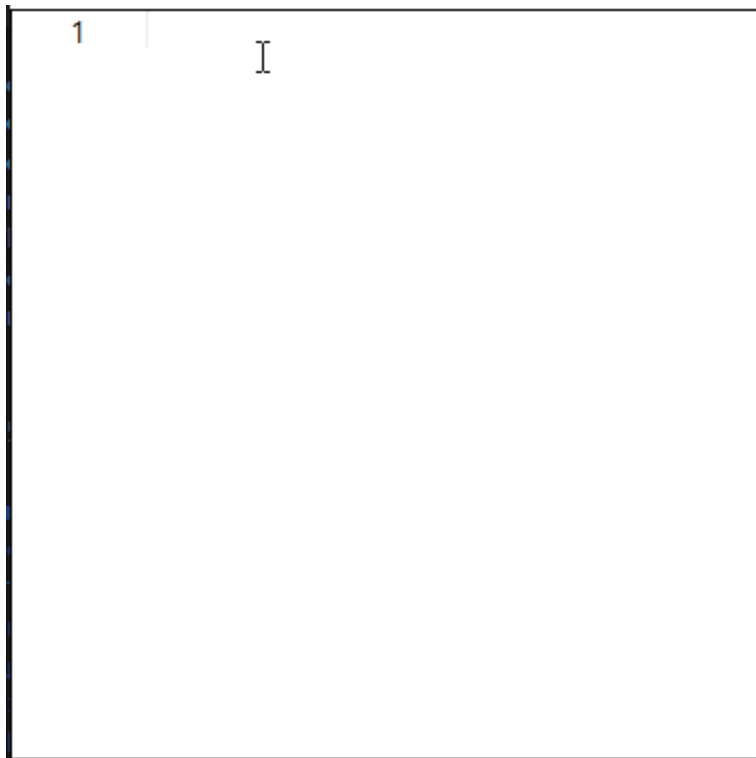
- None - When `ENTER` key is pressed, edit cursor will move to beginning of the next line.



- Block - When `ENTER` key is pressed, edit cursor will move to next line with same indentation of current line.

```
1  if(date != null)
2  {
3      date = Date.Today;
4  }
```

- Smart - When `ENTER` key is pressed, edit cursor will move to next line with one tab space.



#### XML

```
<syncfusion:EditControl Name="Edit1" Background="White" Margin="0"
IsAutoIndentationEnabled="True" IndentingOptions="Smart" Foreground="Black"
/>
```

#### C#

```
EditControl editControl = new EditControl() {IsAutoIndentationEnabled =
true, Height = 200, Width = 200, Background = Brushes.White, Foreground =
Brushes.Black };
editControl.IndentingOptions = IndentingOptions.Smart;
```

### TabSpaces in EditControl

[EditControl](#) supports for changing the number of empty spaces to be added for single Tab key press by setting [TabSpaces](#) property. The default value is 4.

#### XML

```
<syncfusion:EditControl Name="Edit1" Background="White" Margin="0"
IsAutoIndentationEnabled="True" TabSpaces="10" IndentingOptions="Smart"
Foreground="Black" ShowLineNumber="True" />
```

#### C#

```
EditControl editControl = new EditControl() {Height = 200, Width = 200,
Background = Brushes.White, Foreground = Brushes.Black };
editControl.TabSpaces = 10;
```

### File Support in WPF Syntax Editor

Essential Edit WPF facilitates users to create, open, modify and save text files and programming language files. EditControl provides built-in support for a variety of text based file formats such as txt, cs, VB, SQL, XAML, and XML. It also enables to specify custom file types in the custom language configurations.

#### Opening a file

Opening a file in the [EditControl](#) can be done in the following three ways:

#### Using file path

The [DocumentSource](#) property of EditControl is used to specify the file to be opened with EditControl. The following code can be used to set the DocumentSource property.

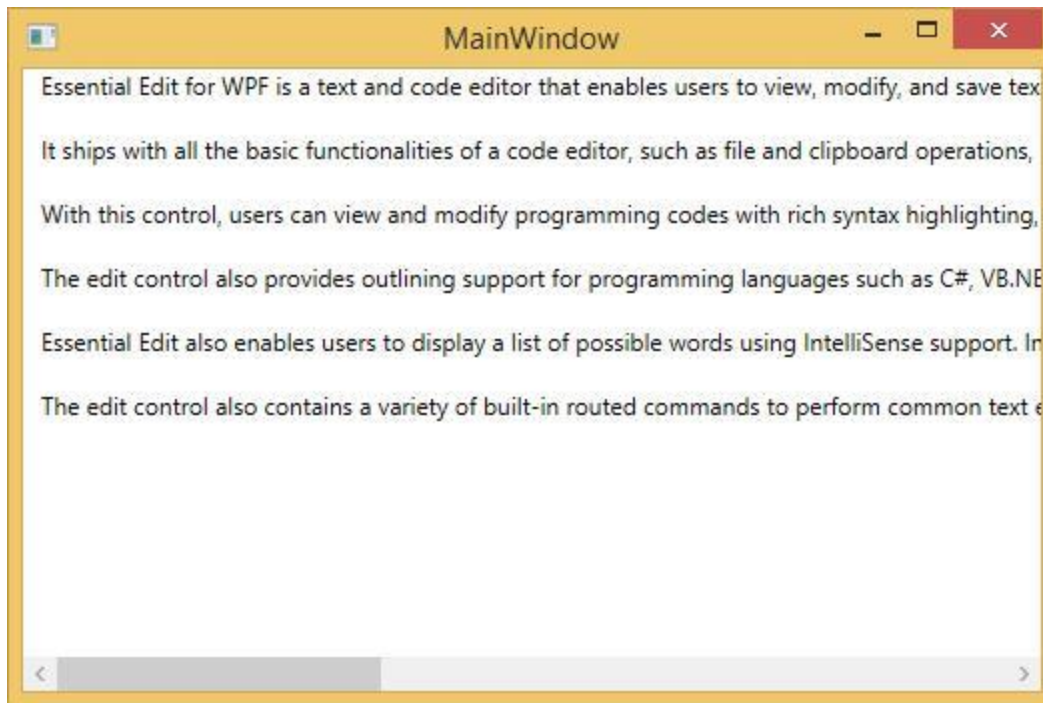
#### XML

```
<sfedit:EditControl x:Name="editControl" DocumentSource="C:\Content.txt"
ShowLineNumber="False" EnableOutlining="False"/>
```

#### C#

```
editControl.DocumentSource = @"C:\Content.txt";
```

The following image displays the contents from file set as DocumentSource window.



#### [Opening through file dialog](#)

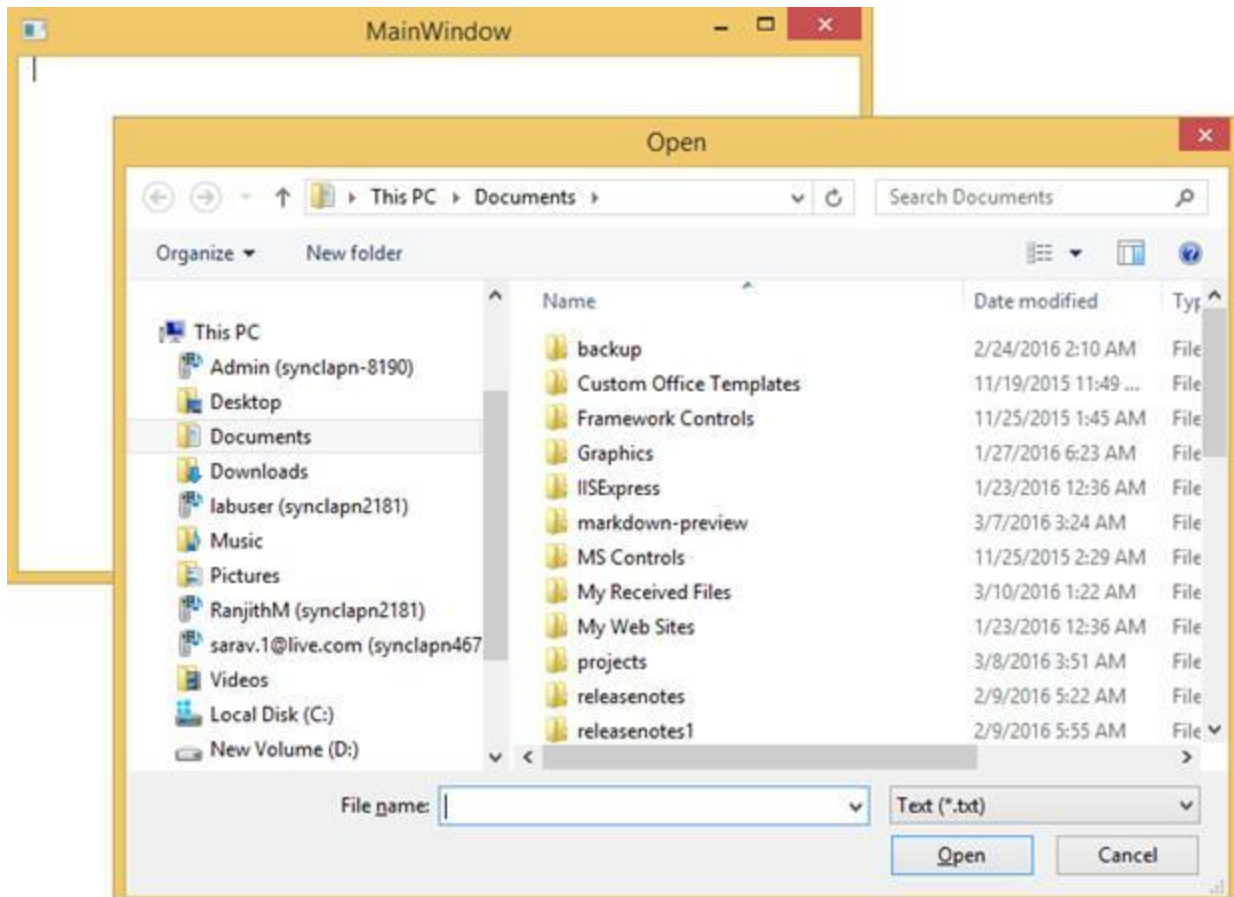
Files can also be opened using the LoadFile method. LoadFile method displays a FileOpenDialog to enable you to choose the file that needs to be opened in the EditControl.

#### **C#**

```
editControl.LoadFile();
```

The following image displays the file open dialog.





### Dropping a file

The `EditControl` allows users to drop a file over it, by setting the `AllowDrop` property to `true`. Users can drop any type of file supported in the `EditControl`. The editor will automatically switch its `DocumentLanguage` based on dropped file's extension. When dropping, if any documents is already in open, it will be closed, and the `DocumentClosing` event will occur. Here, you can control the desired action before closing.

#### C#

```
this.editControl.AllowDrop = true;
```

#### VB.NET

```
Me.editControl.AllowDrop = True
```

### Saving the text in a file

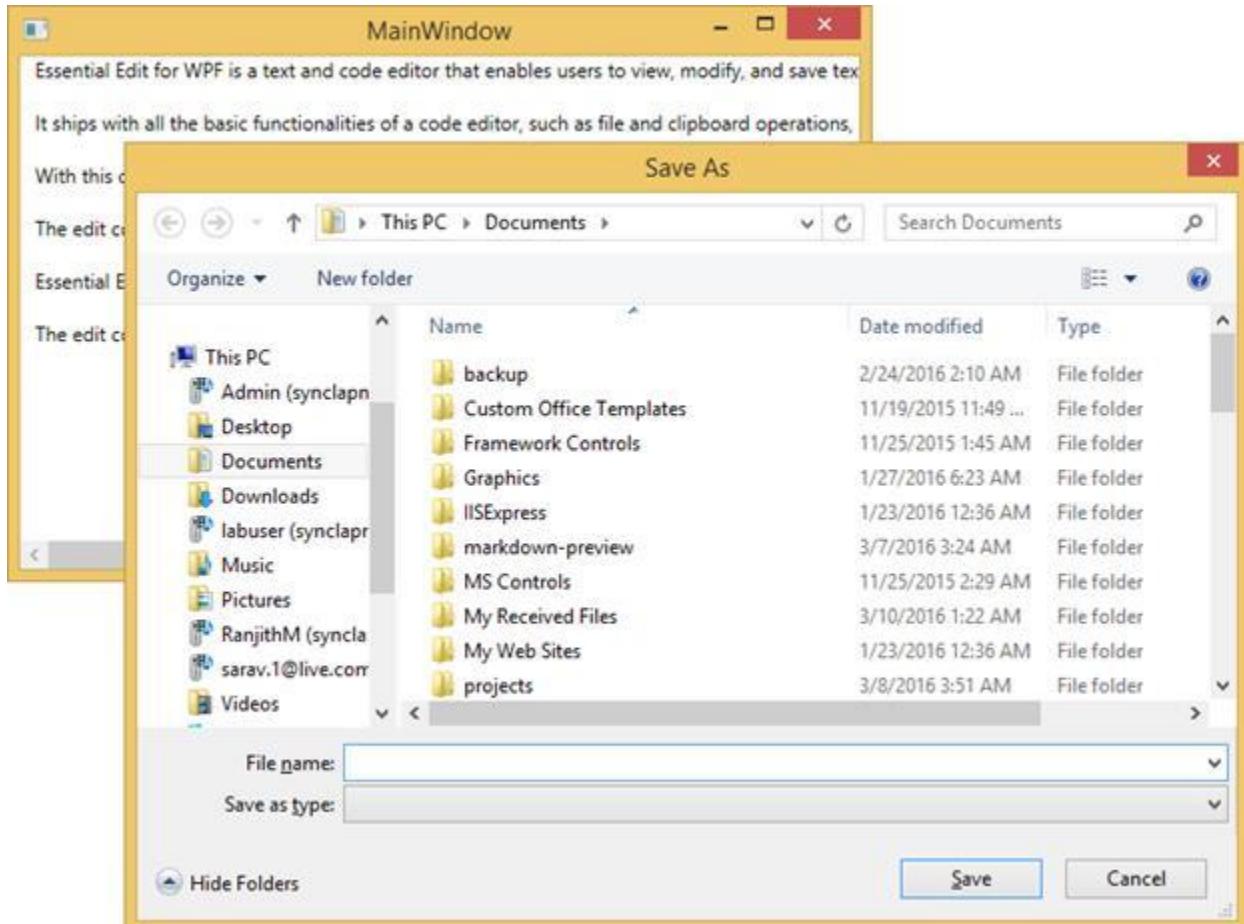
`SaveFile` method in the `EditControl` class is used to save the text in `EditControl` to a file. `EditControl` does support saving all the built-in languages, file types and custom language file type respectively.

Enable save file, by using the following code.

#### C#

```
editControl.SaveFile();
```

The following image displays the save file dialog.



#### *DocumentClosing event*

By default, the existing file will not be saved when loading or dropping a new file. You can control this behavior using the [DocumentClosing](#) event, which occurs when closing a file. You can use the `HasUnsavedChanges` property to identify whether the existing file contains changes, based on which you can choose the action need to be performed.

| [DocumentClosingEventArgs](#) | Description |

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

| `HasUnsavedChanges` | Represents a value that indicates whether the file contains unsaved changes. |

| `Action` | Represents a value to specify the save actions. |

| [SaveAction](#) | Description |

|---|---|

| `Save` | Saves the changes before closing document. |

| `Discard` | Ignores the changes and closes the document. |

| `Cancel` | Cancels the current action that is being performed. |

| Prompt | Opens a dialog for allowing users to choose one of the above actions. |

### **C#**

```
public MainWindow()
{
    InitializeComponent();
    editControl.DocumentClosing += EditControl_DocumentClosing;
}
private void EditControl_DocumentClosing(object sender,
    DocumentClosingEventArgs args)
{
    if (HasUnsavedChanges)
    {
        args.Action = SaveAction.Save;
    }
}
```

### **VB.NET**

```
editControl.DocumentClosing += AddressOf EditControl_DocumentClosing
Private Sub EditControl_DocumentClosing(ByVal sender As Object, ByVal args
As DocumentClosingEventArgs)
If HasUnsavedChanges Then
args.Action = SaveAction.Save
End If args.Action = SaveAction.Save
End Sub
```

### *Default Context Menu*

#### ShowDefaultContextMenu

EditControl has a built-in context menu which enables users to easily perform common text editing operations. The default context menu can be enabled/disabled using **ShowDefaultContextMenu** property. By default, this property is set to **true**.

Set the ShowDefaultContextMenu property of EditControl by using the following code.

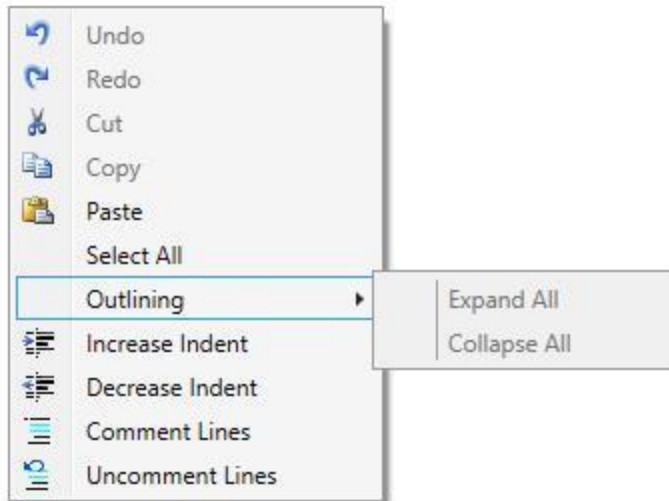
### **XML**

```
<sfedit:EditControl x:Name="editControl" ShowDefaultContextMenu="True"/>
```

### **C#**

```
editControl.ShowDefaultContextMenu = true;
```

The following image displays the Outlining Menu Expanded window.



### Functionalities supported by EditControl's ContextMenu

EditControl's built-in context menu supports the following functionalities.

Command	Usage
Undo	Revert the previous action performed in the EditControl
Redo	Performs the action again that was reverted using Undo command
Cut	Cut the selected text
Copy	Copy the selected text
Paste	Paste the text in the clipboard in the current cursor location.
Select All	Selects all the text in the EditControl
Outlining -> Expand All	Expands all the collapsed blocks in the text. This functionality is supported only when the EnableOutlining is set to <code>true</code> and the language supports outlining.
Outlining -> Collapse All	Hides all the blocks in the EditControl's text. This functionality is supported only when the EnableOutlining is set to <code>true</code> and the language supports outlining.
Increase Indent	Appends a series of empty characters (tab) in front of the first valid character in the line. This command can be performed on an individual line or selected lines.

Decrease Indent	Removes a series of empty characters (tab) if any, in front of the first valid character in the line. This command can be performed on an individual line or selected lines
Comment Lines	It detects the supported comment Lexem from the Language configuration of EditControl™s current language and appends it to individual or multiple selected lines.
Uncomment Line	Removes the comment from the individual or selected lines.

### Line Numbers

Edit WPF enables users to display line numbers for the content in the EditControl. Line numbers can be displayed or hidden by using the `ShowLineNumber` property of the EditControl class. The following lines of code can be used to display or hide line numbers in EditControl.

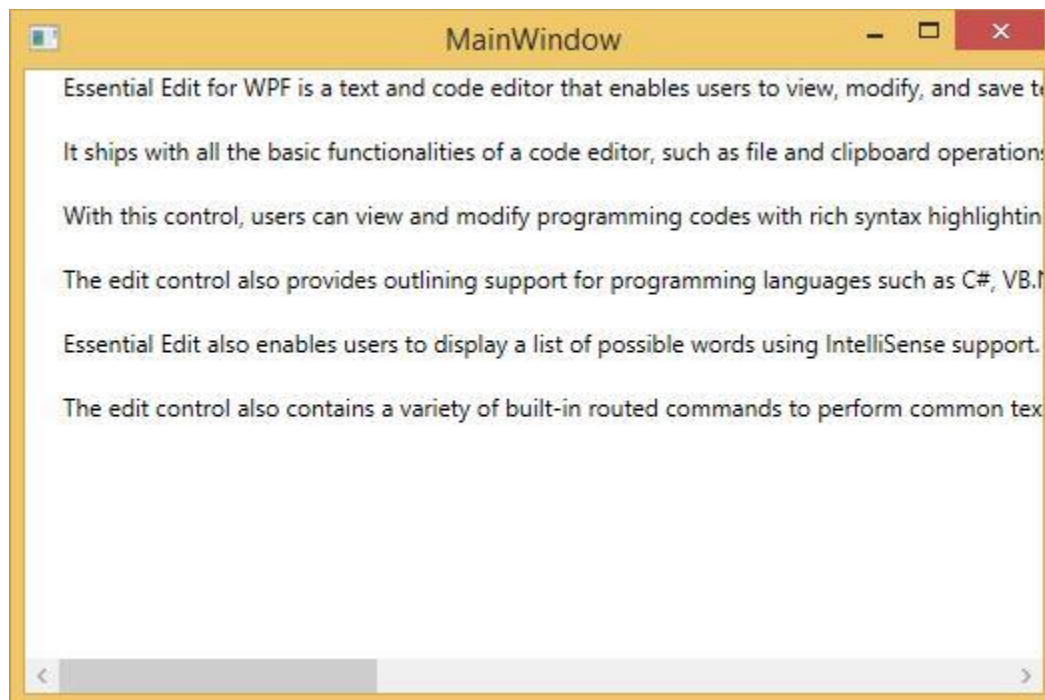
#### XML

```
<sfedit:EditControl Name="editControl" ShowLineNumber="False">
</sfedit:EditControl>
```

#### C#

```
editControl.ShowLineNumber = false;
```

The following image displays hidden line numbers.



### Events

#### CaretPositionChanged

[CaretPositionChanged](#) event occurs when the caret position of the text in the `EditControl` is changed.

This event receives two arguments namely `sender` that handles `EditControl` and [CaretPositionEventArgs](#) as objects.

The [CaretPositionEventArgs](#) object contains the following properties:

- [LineNumber](#) - Gets the current line number value of the `EditControl`.
- [CursorIndex](#) - Gets the current cursor index value of the `EditControl`.

### Status Bar in WPF Syntax Editor

The status bar option is used to view primary information such as the loaded document's path, encoding type, line number and column number based on the cursor position. By default, [Visibility](#) property for status bar is set to be **Collapsed**.

### XML

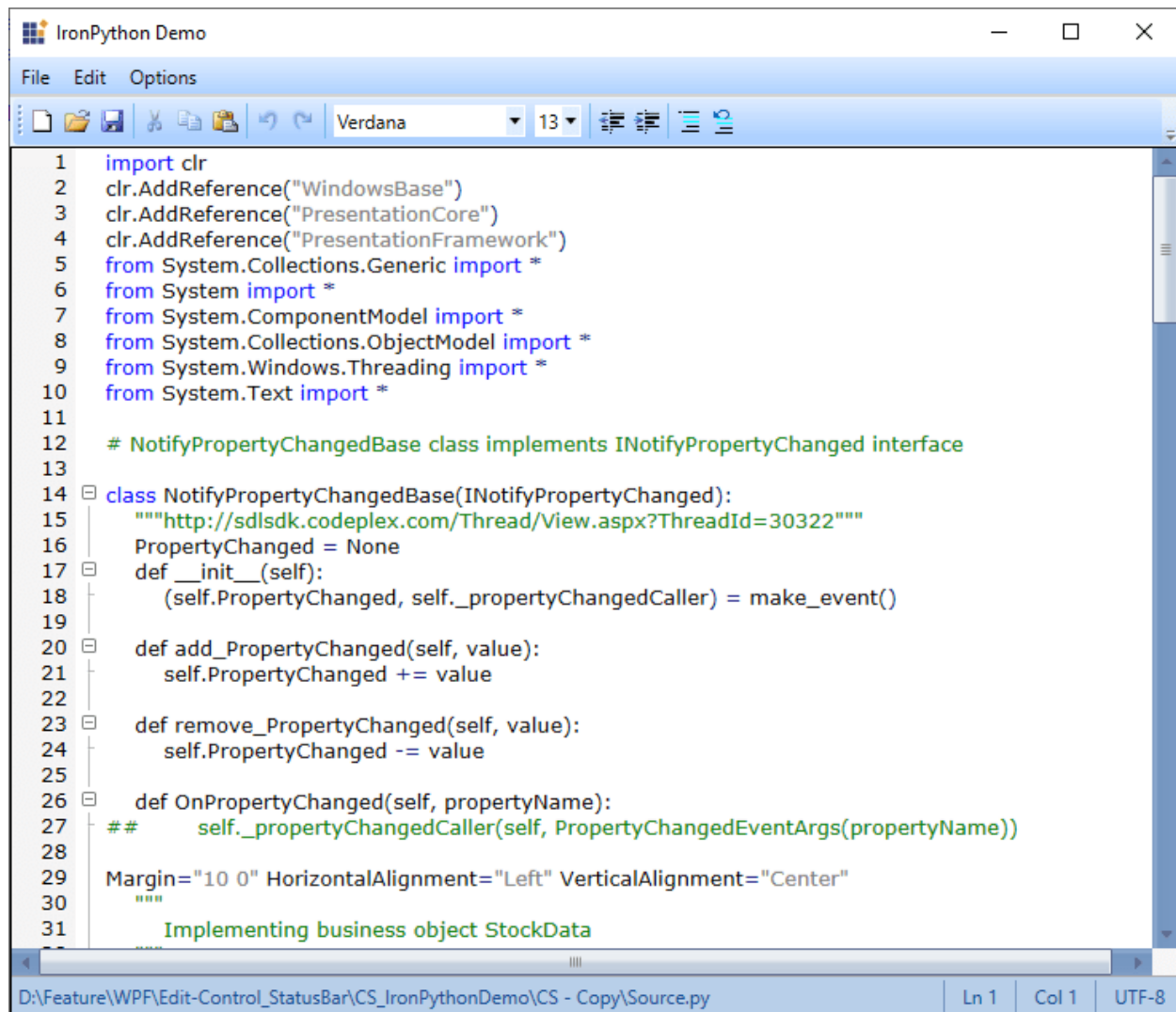
```
<!--Adding EditControl to application and setting its StatusBarVisiblity to Visible-->  
<syncfusion:EditControl Background="White" Name="EditControl1">  
<syncfusion:EditControl.StatusBarSettings >  
<syncfusion:StatusBarSettings Visibility="Visible"/>  
</syncfusion:EditControl.StatusBarSettings>  
</syncfusion:EditControl>
```

### C#

```
public partial class MainWindow : Window  
{  
    public MainWindow()  
    {  
        InitializeComponent();  
        this.EditControl1.DocumentSource = "../..../Source.cs";  
        this.EditControl1.StatusBarSettings.Visibility = Visibility.Visible;  
    }  
}
```

### VB.NET

```
public partial class MainWindow : Window  
{  
    public MainWindow()  
    {  
        InitializeComponent();  
        Me.EditControl1.DocumentSource = "../..../Source.cs";  
        Me.EditControl1.StatusBarSettings.Visibility = Visibility.Visible;  
    }  
}
```



The status bar shows the file path, encoding type, line number and column number. The visibility of each item in the status bar can be customized by using the below properties.

- [ShowFilePath](#): It shows the exact application directory and the file path which was loaded in Edit Control. File Path property can be enabled/disabled by changing it's visibility property.
- [ShowLineNumber](#): It shows the current line number where the cursor is placed on Edit Control. By changing it's visibility user can enable/disable the line number property.
- [ShowColumnNumber](#): It shows the current column number where the cursor is placed on Edit Control. By changing it's visibility user can enable/disable the Column number property.
- [ShowEncoding](#): It shows the current encoding type of the loaded file in Edit Control. Encoding property can be enable/disable by changing it's visibility property.

## XML

```

<!--Adding EditControl to application and setting its StatusBarVisibility to Visible-->
<syncfusion:EditControl Background="White" Name="EditControl1">
<syncfusion:EditControl.StatusBarSettings >

```

```
<syncfusion:StatusBarSettings Visibility="Visible" ShowFilePath="Visible"
ShowLineNumber="Visible" ShowColumnNumber="Visible"
ShowEncoding="Collapsed"/>
</syncfusion:EditControl.StatusBarSettings>
</syncfusion:EditControl>
```

## C#

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        this.EditControl1.DocumentSource = "../..../Source.cs";
        this.EditControl1.StatusBarSettings.Visibility = Visibility.Visible;
        this.EditControl1.StatusBarSettings.ShowFilePath = Visibility.Visible;
        this.EditControl1.StatusBarSettings.ShowLineNumber = Visibility.Visible;
        this.EditControl1.StatusBarSettings.ShowColumnNumber = Visibility.Visible;
        this.EditControl1.StatusBarSettings.ShowEncoding = Visibility.Collapsed;
    }
}
```

## VB.NET

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        Me.EditControl1.DocumentSource = "../..../Source.cs";
        Me.EditControl1.StatusBarSettings.Visibility = Visibility.Visible;
        Me.EditControl1.StatusBarSettings.ShowFilePath = Visibility.Visible;
        Me.EditControl1.StatusBarSettings.ShowLineNumber = Visibility.Visible;
        Me.EditControl1.StatusBarSettings.ShowColumnNumber = Visibility.Visible;
        Me.EditControl1.StatusBarSettings.ShowEncoding = Visibility.Collapsed;
    }
}
```

The following screenshot shows the Edit Control after changing the [ShowEncoding](#) visibility property to **Collapsed**.



```

1  import clr
2  clr.AddReference("WindowsBase")
3  clr.AddReference("PresentationCore")
4  clr.AddReference("PresentationFramework")
5  from System.Collections.Generic import *
6  from System import *
7  from System.ComponentModel import *
8  from System.Collections.ObjectModel import *
9  from System.Windows.Threading import *
10 from System.Text import *
11
12 # NotifyPropertyChangedBase class implements INotifyPropertyChanged interface
13
14 class NotifyPropertyChangedBase(INotifyPropertyChanged):
15     """http://sdlsdk.codeplex.com/Thread/View.aspx?ThreadId=30322"""
16     PropertyChanged = None
17     def __init__(self):
18         (self.PropertyChanged, self._propertyChangedCaller) = make_event()
19
20     def add_PropertyChanged(self, value):
21         self.PropertyChanged += value
22
23     def remove_PropertyChanged(self, value):
24         self.PropertyChanged -= value
25
26     def OnPropertyChanged(self, propertyName):
27         ## self._propertyChangedCaller(self, PropertyChangedEventArgs(propertyName))
28
29     Margin="10 0" HorizontalAlignment="Left" VerticalAlignment="Center"
30     """
31     Implementing business object StockData
32     """
33     def Symbol(self, value):
34         self.Symbol = value
35         self.OnPropertyChanged("Symbol")

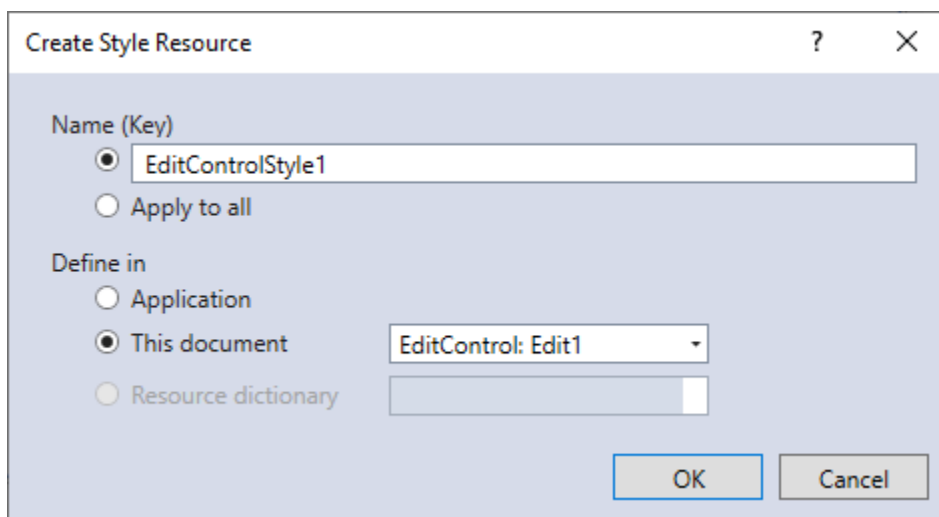
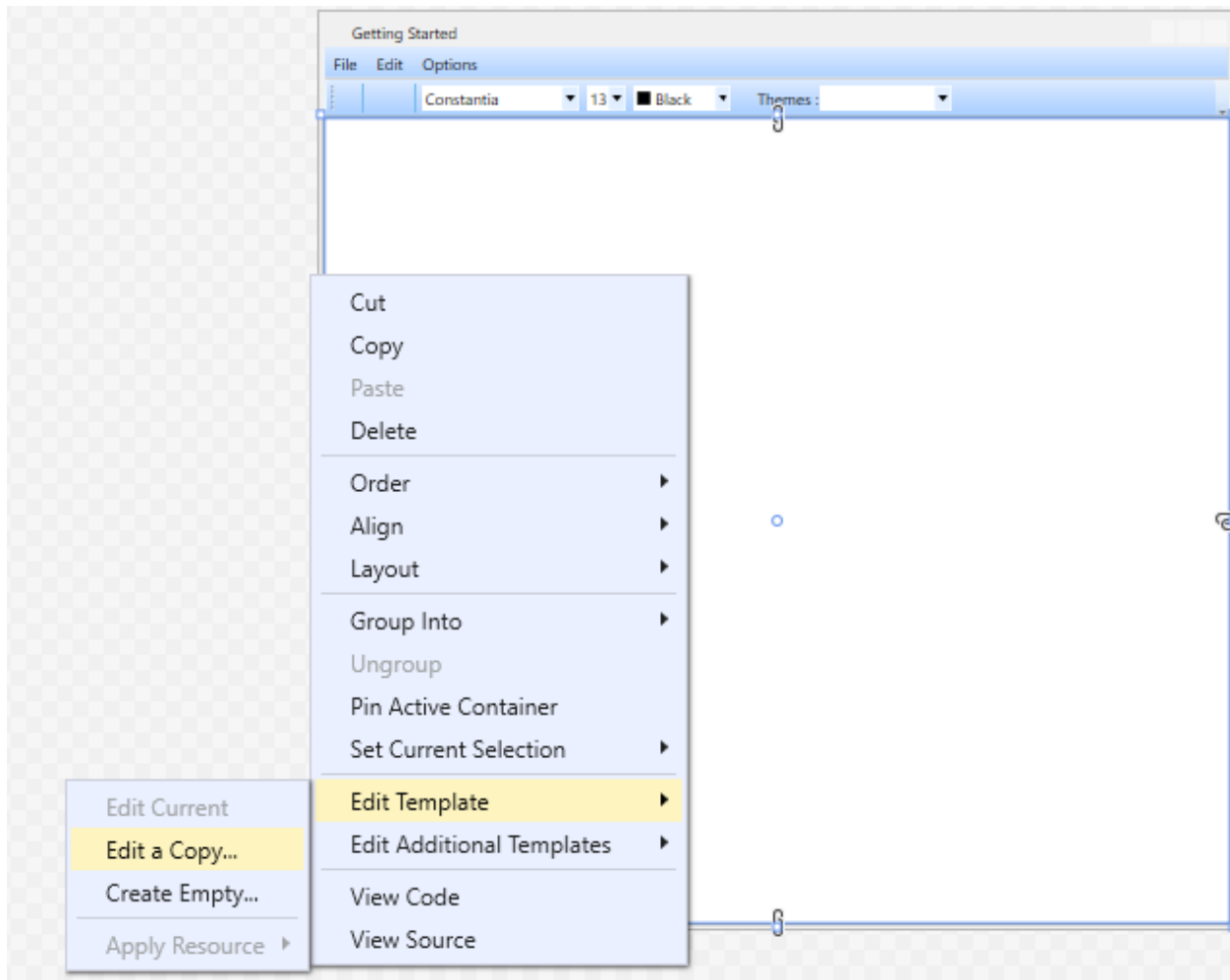
```

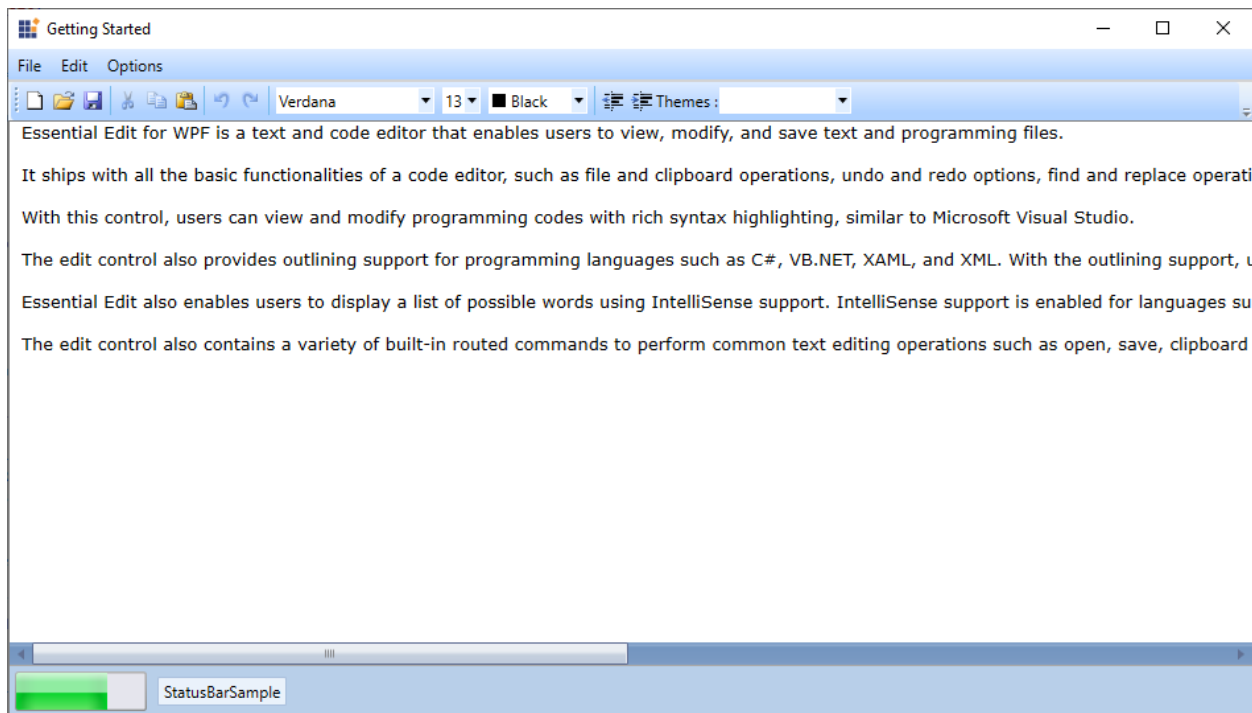
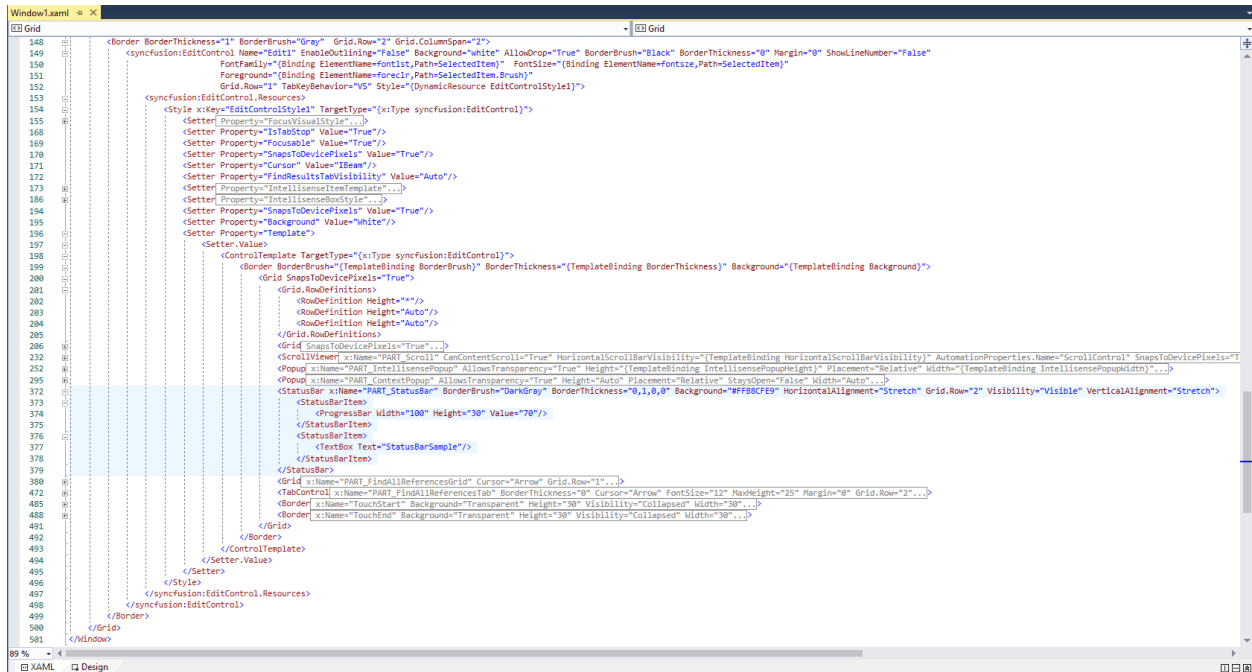
D:\Feature\WPF\Edit-Control\_StatusBar\CS\_IronPythonDemo\CS - Copy\Source.py Ln 1 Col 1

### Customization of status bar

The status bar can be customized by opening context menu on right clicking the Edit Control in designer page. In that select **Edit Template** and then click **Edit a Copy....** A new dialog will be shown in that choose the options enabled in combo box **EditControl: Edit1** which adds the Edit Control template to the current Edit Control class.

Then user can able to add custom controls such as Progress Bar, TextBox, etc... as needed to be displayed in that status bar by modifying the default template in it. The following screenshots illustrate how to do the same thing step-by-step.





### Styling the status bar

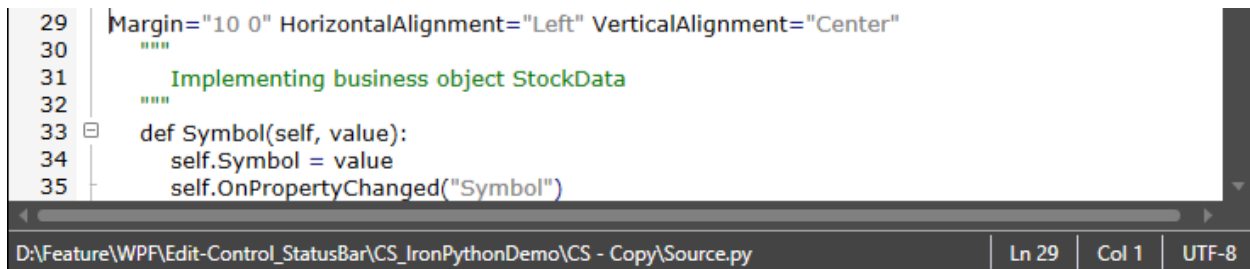
The status bar supports following built-in styles,

- Blend
- Default
- Metro
- Office2007Black
- Office2007Blue

- Office2007Silver
- VS2010

The following illustrations shows how the status bar is applied with different built-in styles.

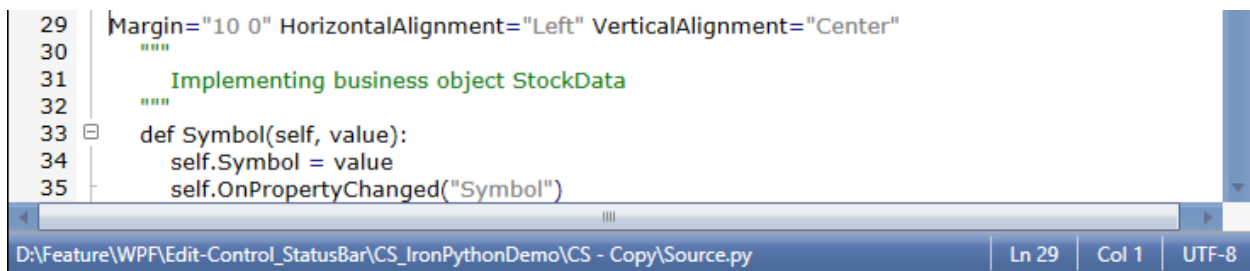
### Blend



```
29 | Margin="10 0" HorizontalAlignment="Left" VerticalAlignment="Center"
30 | """
31 |     Implementing business object StockData
32 |     """
33 | def Symbol(self, value):
34 |     self.Symbol = value
35 |     self.OnPropertyChanged("Symbol")
```

D:\Feature\WPF\Edit-Control\_StatusBar\CS\_IronPythonDemo\CS - Copy\Source.py | Ln 29 | Col 1 | UTF-8

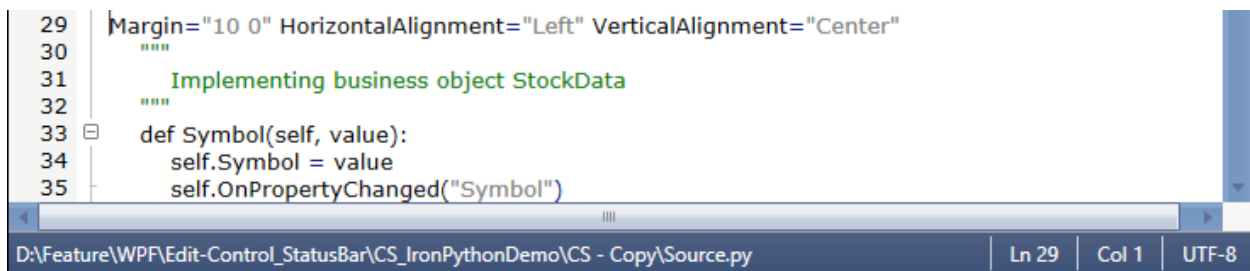
### Default



```
29 | Margin="10 0" HorizontalAlignment="Left" VerticalAlignment="Center"
30 | """
31 |     Implementing business object StockData
32 |     """
33 | def Symbol(self, value):
34 |     self.Symbol = value
35 |     self.OnPropertyChanged("Symbol")
```

D:\Feature\WPF\Edit-Control\_StatusBar\CS\_IronPythonDemo\CS - Copy\Source.py | Ln 29 | Col 1 | UTF-8

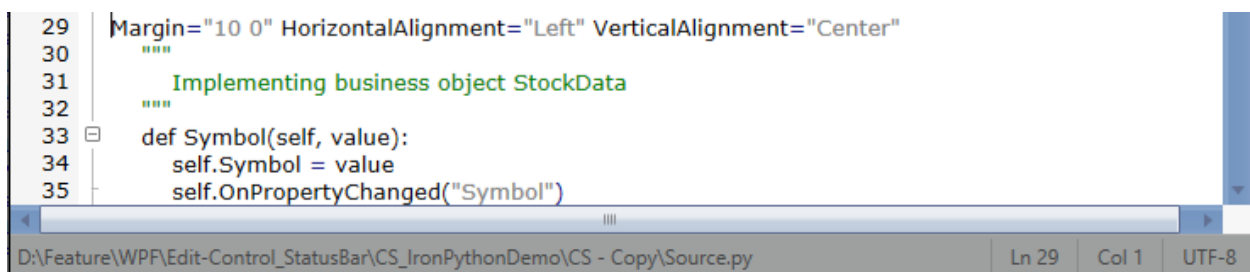
### Metro



```
29 | Margin="10 0" HorizontalAlignment="Left" VerticalAlignment="Center"
30 | """
31 |     Implementing business object StockData
32 |     """
33 | def Symbol(self, value):
34 |     self.Symbol = value
35 |     self.OnPropertyChanged("Symbol")
```

D:\Feature\WPF\Edit-Control\_StatusBar\CS\_IronPythonDemo\CS - Copy\Source.py | Ln 29 | Col 1 | UTF-8

### Office2007Black



```
29 | Margin="10 0" HorizontalAlignment="Left" VerticalAlignment="Center"
30 | """
31 |     Implementing business object StockData
32 |     """
33 | def Symbol(self, value):
34 |     self.Symbol = value
35 |     self.OnPropertyChanged("Symbol")
```

D:\Feature\WPF\Edit-Control\_StatusBar\CS\_IronPythonDemo\CS - Copy\Source.py | Ln 29 | Col 1 | UTF-8

### Office2007Blue

```

29 | Margin="10 0" HorizontalAlignment="Left" VerticalAlignment="Center"
30 | """
31 |     Implementing business object StockData
32 |     """
33 | def Symbol(self, value):
34 |     self.Symbol = value
35 |     self.OnPropertyChanged("Symbol")

```

D:\Feature\WPF\Edit-Control\_StatusBar\CS\_IronPythonDemo\CS - Copy\Source.py | Ln 29 | Col 1 | UTF-8

**Office2007Silver**

```

29 | Margin="10 0" HorizontalAlignment="Left" VerticalAlignment="Center"
30 | """
31 |     Implementing business object StockData
32 |     """
33 | def Symbol(self, value):
34 |     self.Symbol = value
35 |     self.OnPropertyChanged("Symbol")

```

D:\Feature\WPF\Edit-Control\_StatusBar\CS\_IronPythonDemo\CS - Copy\Source.py | Ln 29 | Col 1 | UTF-8

**VS2010**

```

29 | Margin="10 0" HorizontalAlignment="Left" VerticalAlignment="Center"
30 | """
31 |     Implementing business object StockData
32 |     """
33 | def Symbol(self, value):
34 |     self.Symbol = value
35 |     self.OnPropertyChanged("Symbol")

```

D:\Feature\WPF\Edit-Control\_StatusBar\CS\_IronPythonDemo\CS - Copy\Source.py | Ln 29 | Col 1 | UTF-8

*Read Only Mode*

Edit WPF can also be used as a static control in order to view only the contents of the file. ReadOnly mode is enabled/disabled by using the `IsReadOnly` property of the EditControl class. The following code is used to set the ReadOnly mode for EditControl.

**XML**

```
<sfedit:EditControl Name="editControl" IsReadOnly="True">
</sfedit:EditControl>
```

**C#**

```
editControl.IsReadOnly = true;
```

*Single Line Mode*

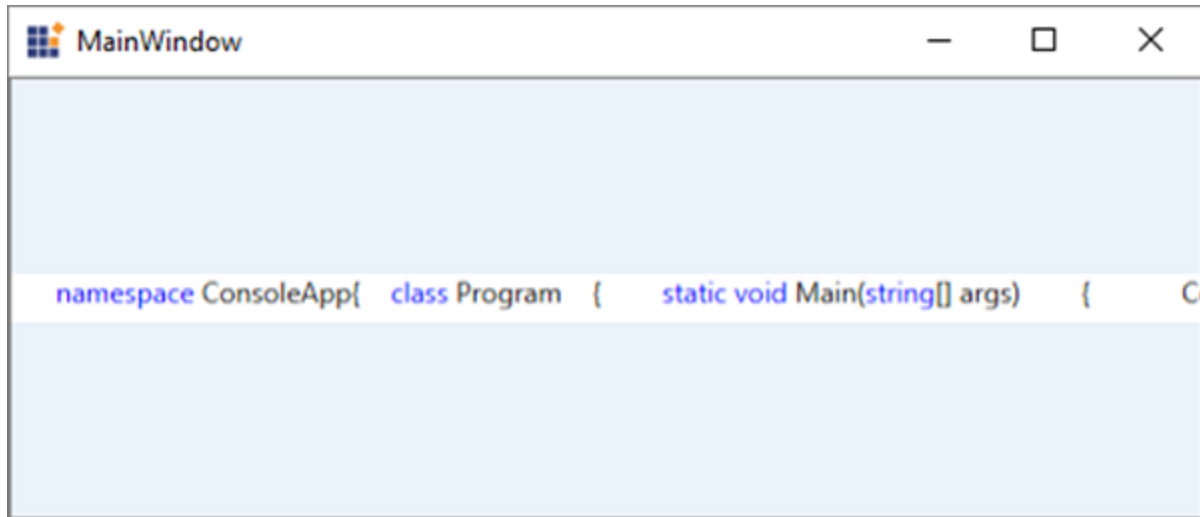
Single Line Mode interprets all lines of code as a single line. It appears as a regular textbox with syntax highlighting, editing, clipboard operation, etc. by setting the `IsMultiLine` property to `false`. The default value is `true`.

**XML**

```
<sfedit:EditControl Name="editControl" IsMultiLine = "False">
</sfedit:EditControl>
```

**C#**

```
editControl.IsMultiLine = false;
```

*Find and Replace*

Essential Edit WPF is now enhanced with Find and Replace feature, which enables users to search a text and replace it with an alternate text. Various options available in text search are:

- Match case
- Match whole word
- Search hidden Text
- Substring
- Prefix

To enable/disable Find and Replace feature, use `ShowFindAndReplace` property in `EditControl`.

The following code can be used to set the `ShowFindAndReplace` property of `EditControl` class.

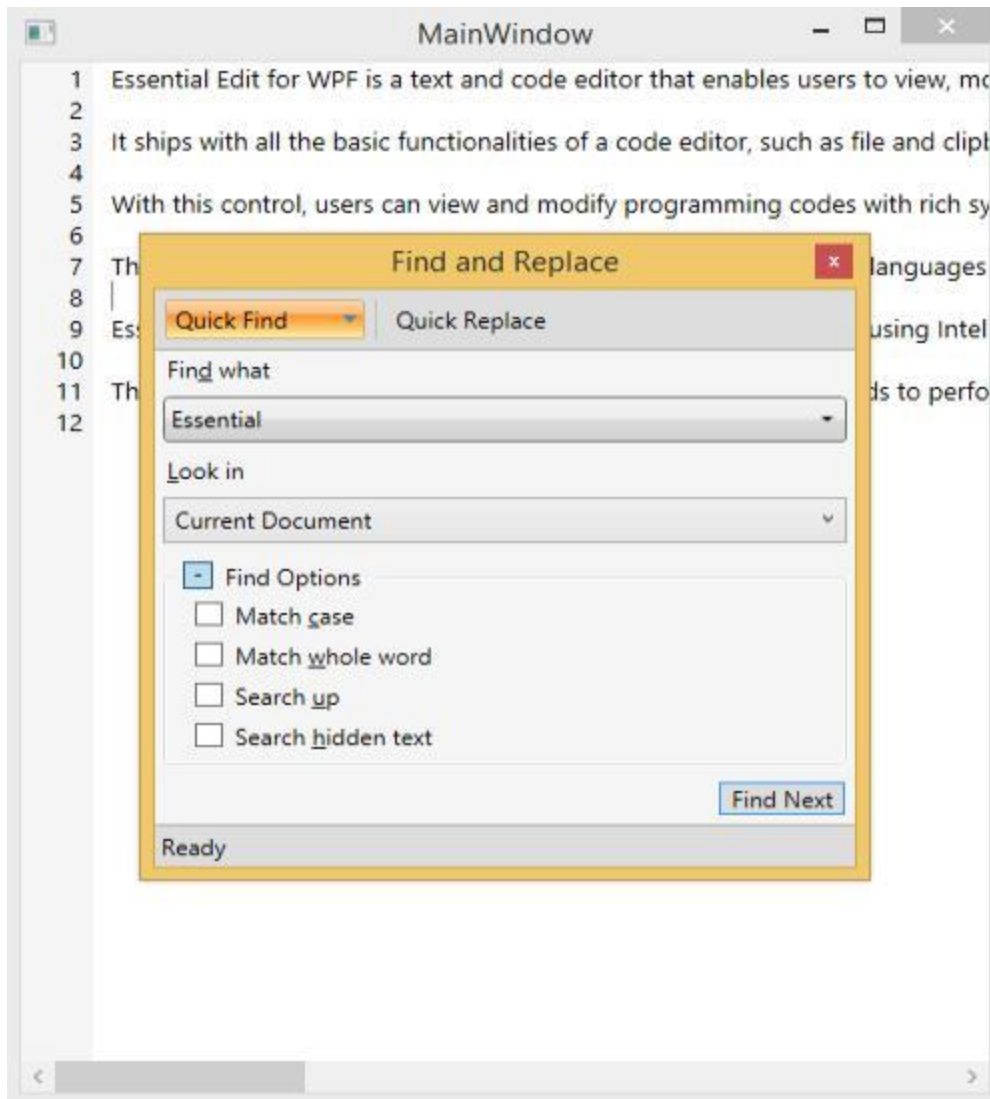
**XML**

```
<sfedit:EditControl x:Name="editControl" DocumentSource="C:\Content.txt"
FontSize="13" EnableOutlining="False" ShowFindAndReplace="True"/>
```

**C#**

```
editControl.ShowFindAndReplace = true;
```

The following image displays the Find and Replace Window.



The Find and Replace dialog provides the basic search functionality as mentioned below.

- Quick Find
- Quick Replace
- Find Symbol

**Quick Find:** Quick Find tab enables to search a text in an open document or a selection for a string.

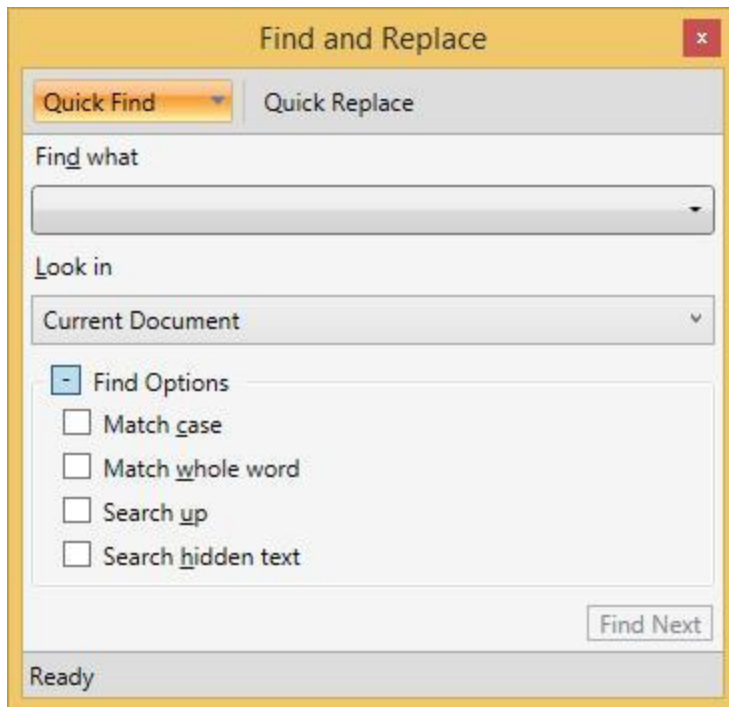
**Find Symbol:** Find Symbol tab enables to search for words and the position of the word in the document.

**Quick Replace:** Quick Replace tab enables to find a text and replace it with an alternative text.

#### Quick find

Quick Find tab enables users to search all the occurrences of a text in an open document or a selection. Select Quick Find in Find and Replace dialog or use Ctrl+F to enable Quick Find.

The following image displays Quick Find Tab.

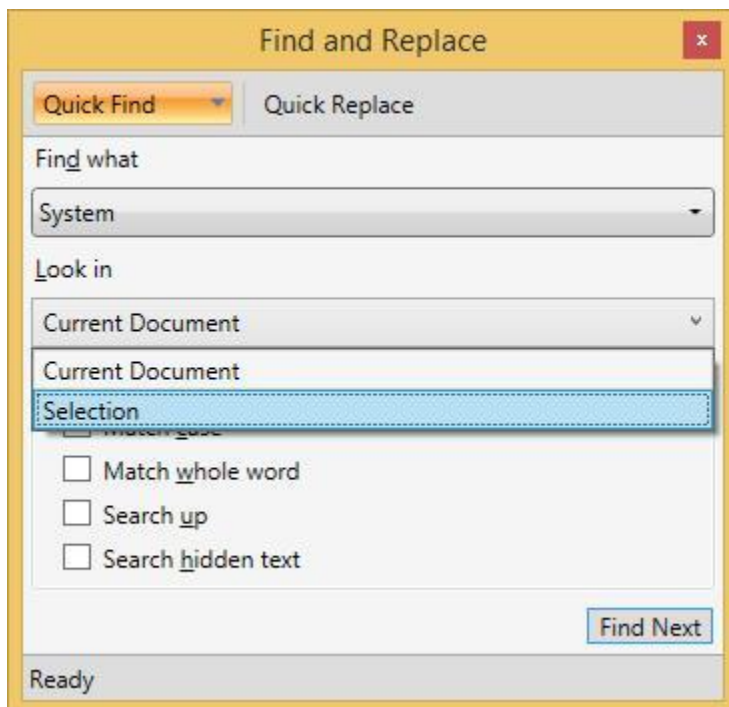


**Find what:** Enter the search text in Find what field to search the text in the document.

**Look in:** You can choose the search area (the whole document or a selection) in this field.

**Note:** Select an area in the document before opening Find and Replace dialog, Selection is automatically selected in this field. This can also be selected in the drop-down box.

The following image displays look in field in Find and Replace Window.



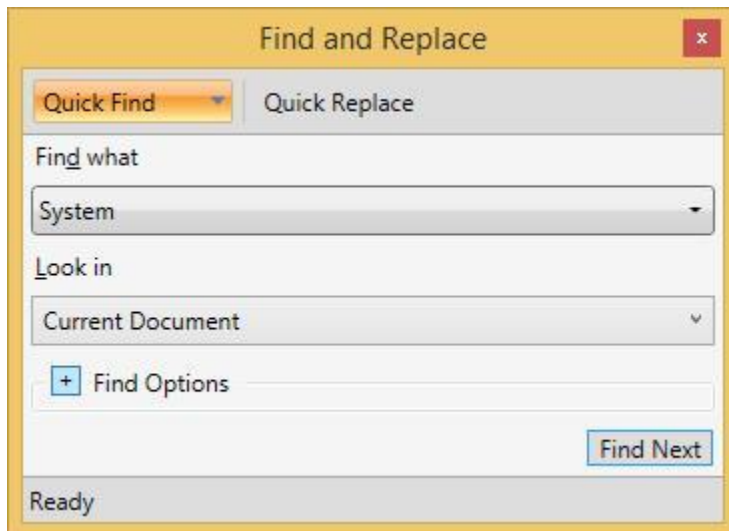


**Find options:** Find options are placed under collapsible GroupBox control to have a compact view of the Find and Replace window.

Find and Replace window facilitates you to search the text based on the following options.

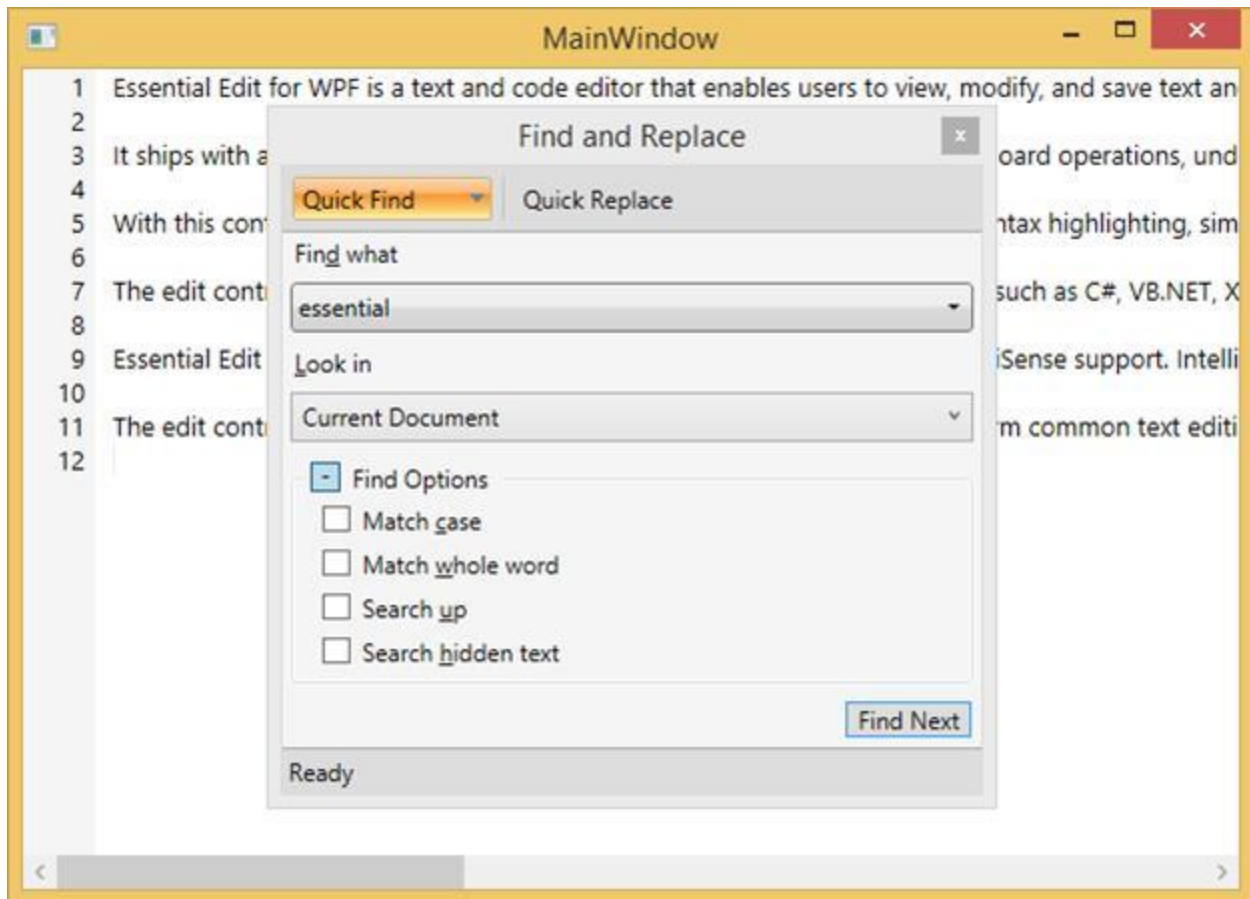
- **Match case:** Performs a case-sensitive search.
- **Match whole word:** Searches for the text specified with in a word boundary.
- **Search up:** Searches in lines above the current cursor location.
- **Search hidden text:** Text from the collapsed region will also be included in search area.

The following image displays Find Options Collapsed window.



Click the Find Next or Enter to search the text in document. The Find Next Button will be enabled only when the search text is entered in Find what field.

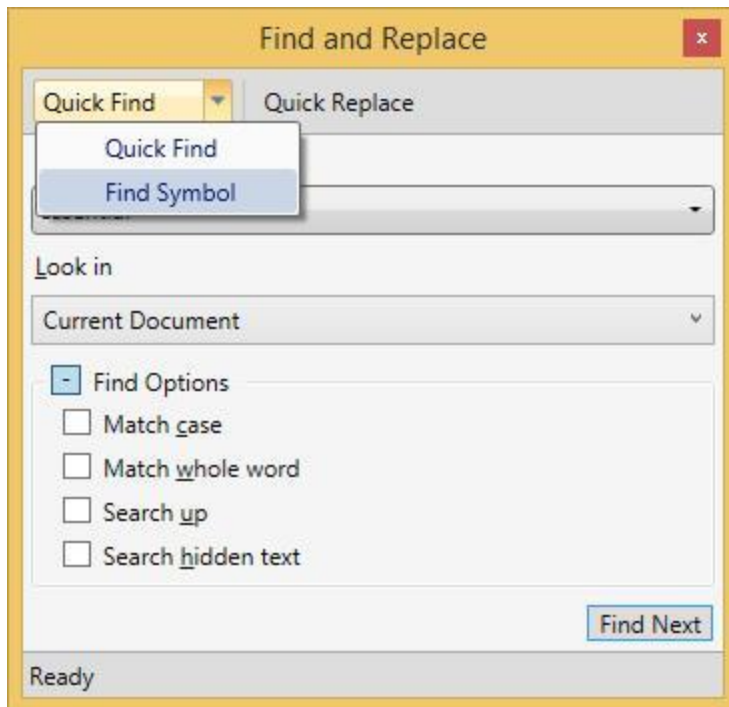
The following image displays EditControl Sample Window.



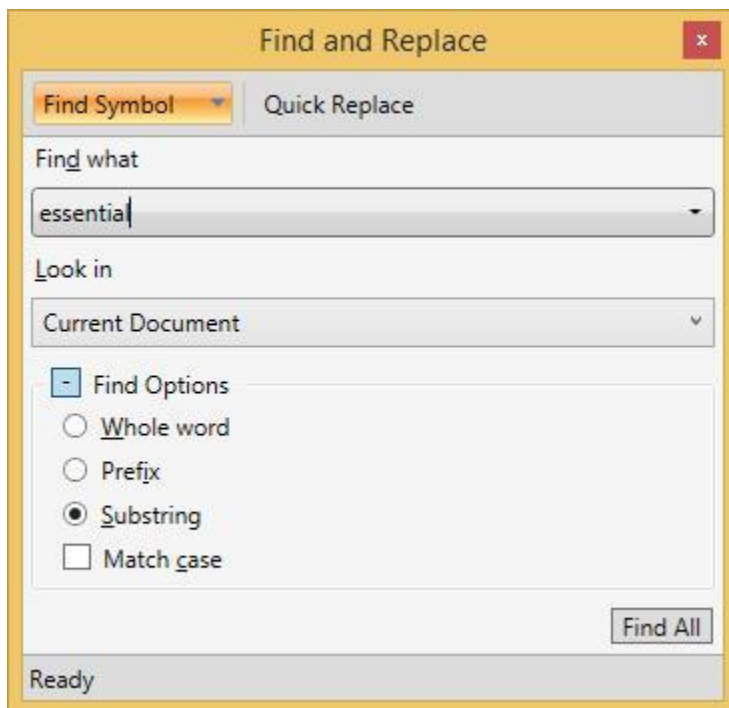
### Find symbol

Find symbol tab in Find and Replace window facilitates the users to find all the occurrences of the specified text in the entire Edit Control's text. Find Symbol tab can be enabled by selecting the drop-down option in the Quick Find tab.

The following image displays Find and Replace Window.



The following image displays Find Symbol Tab in Find and Replace Window.



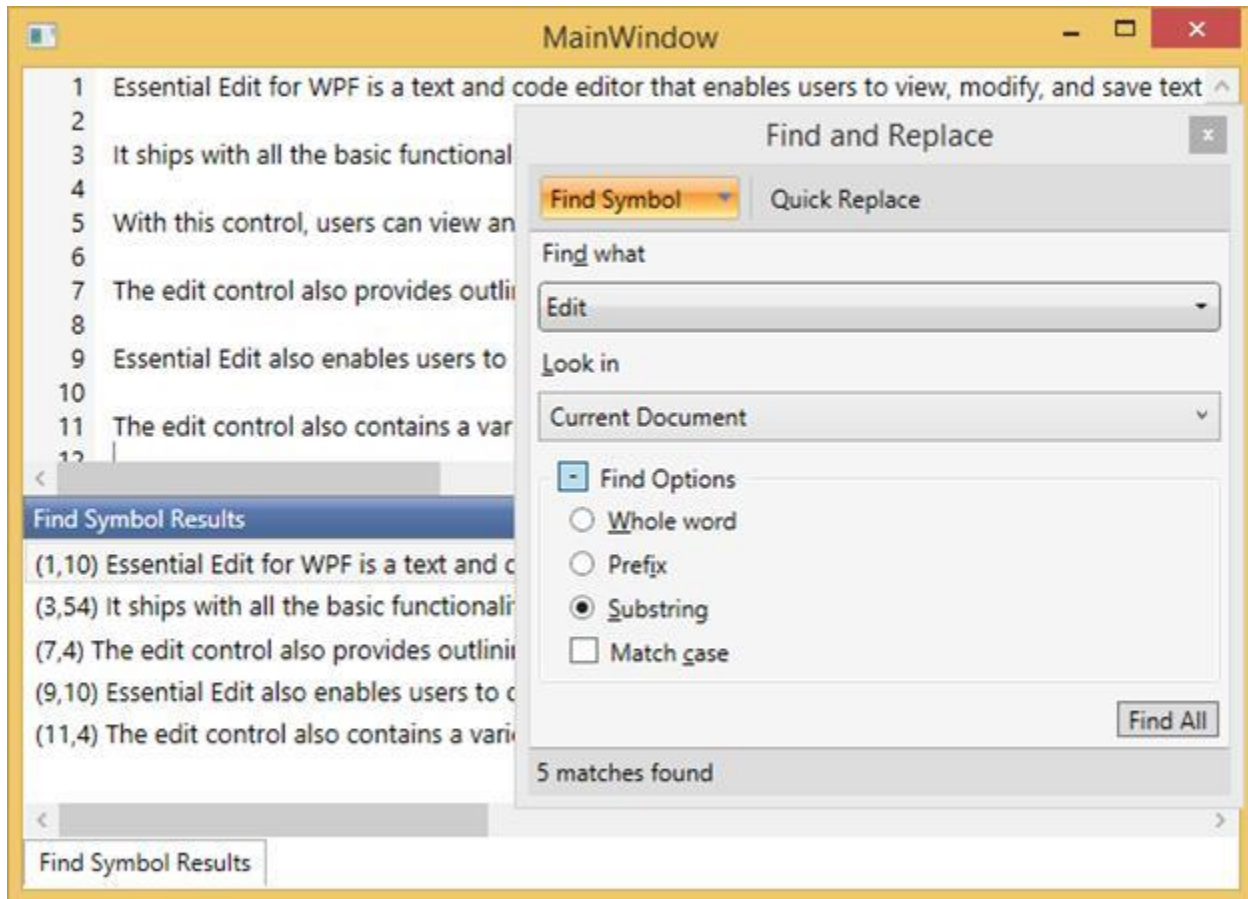
Find Symbol tab supports the following search options to refine the search results.

- **Whole word:** Searches the text when the whole word matches.
- **Prefix:** Locates the line, when the line starts with the search text.
- **Substring:** Locates the line, when the line contains the search text.
- **Match case:** Performs a case sensitive search.

**Find Symbol Results (Shift+F12):** The Whole word, Prefix, Substring options are radio buttons and Sub string will be selected in default.

Find symbol results tab lists all the occurrences of the search text with additional details of line number and position of the text in the line. Double clicking on any item listed in the Find symbol will navigate the cursor to the result selected.

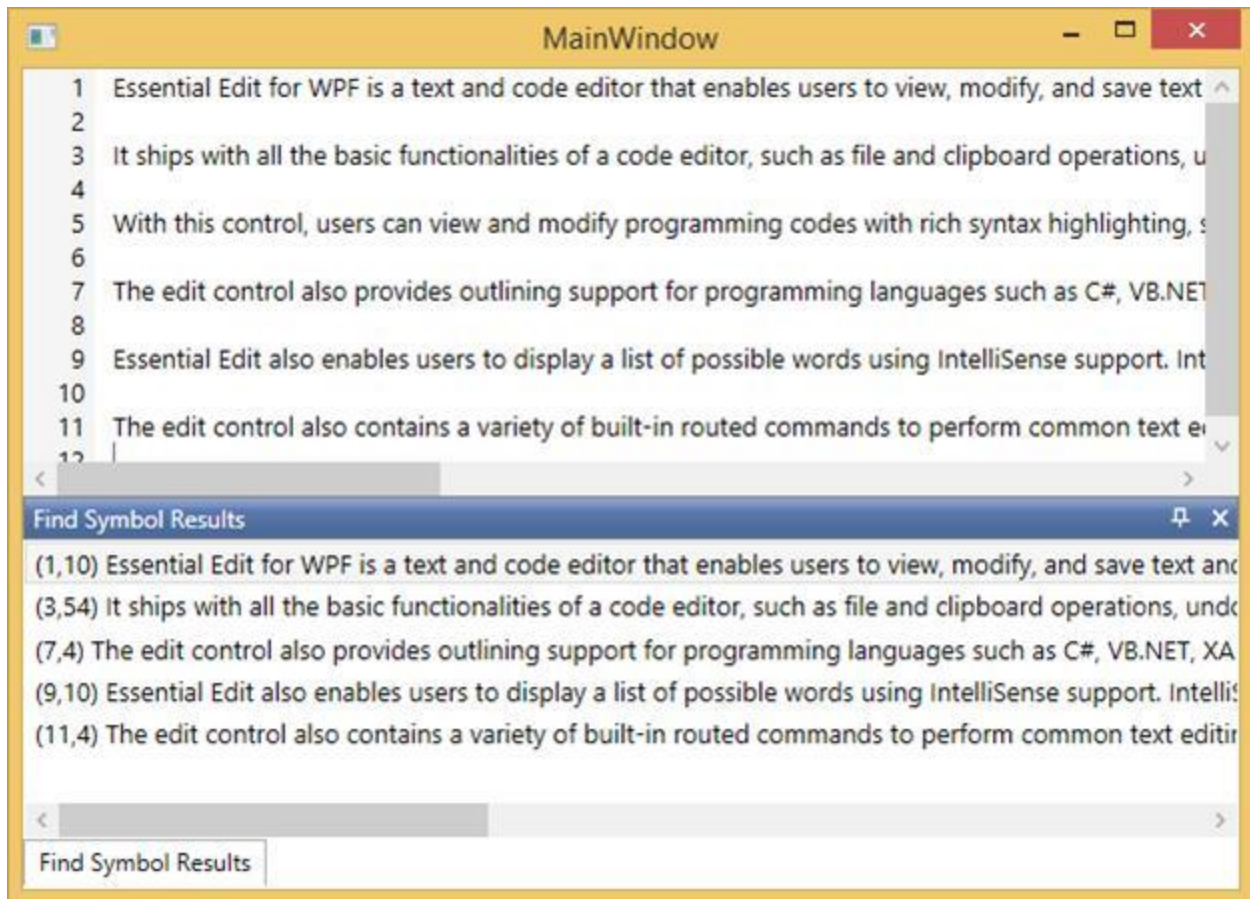
The following image displays EditControl Sample Window.



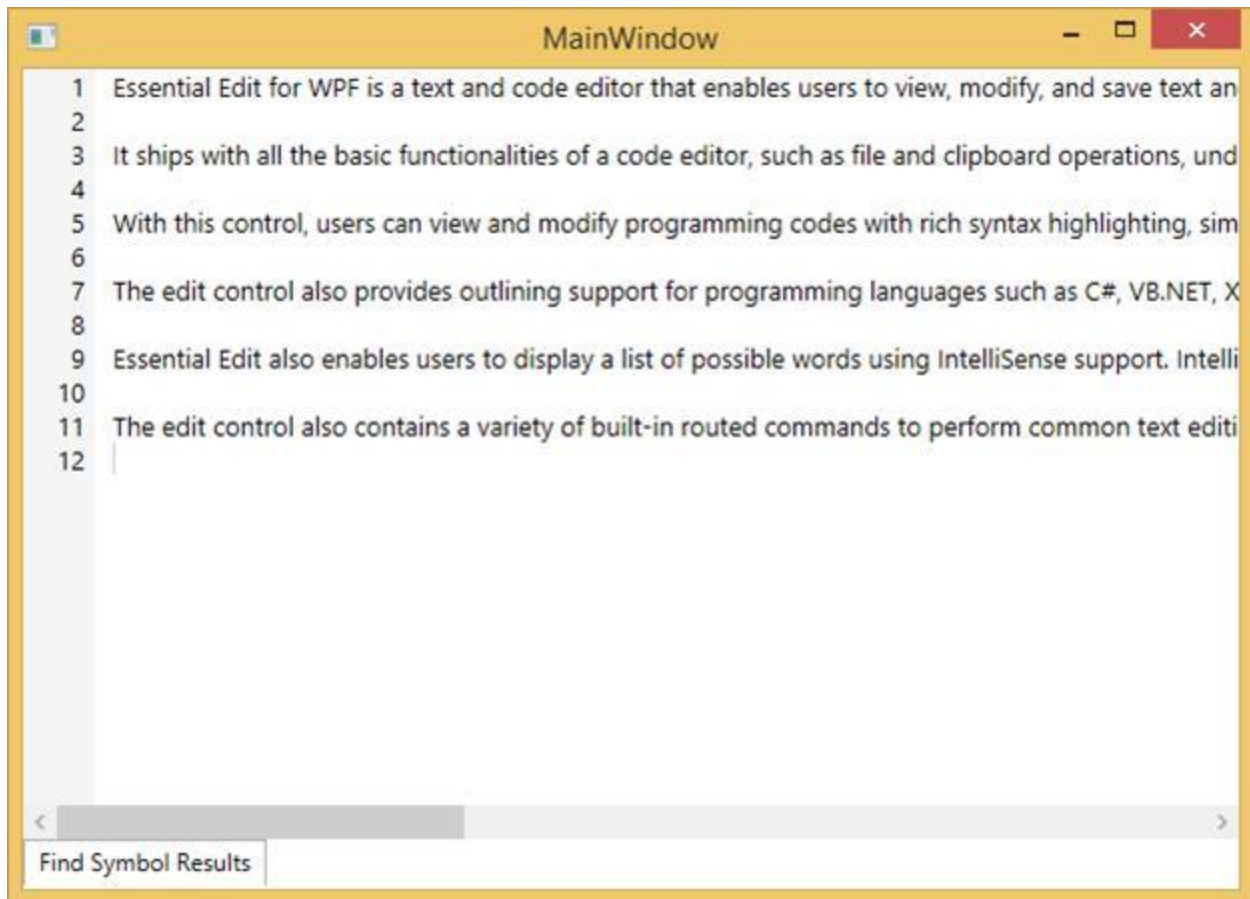
Click the Find All or Enter to search the text in document.

**Note:** Find Symbol results tab is a dockable so that it can be hidden, pinned or closed as necessary.

The following image displays Find Symbol Results.



The following image displays Find Symbol Results tab in Hidden State Window.

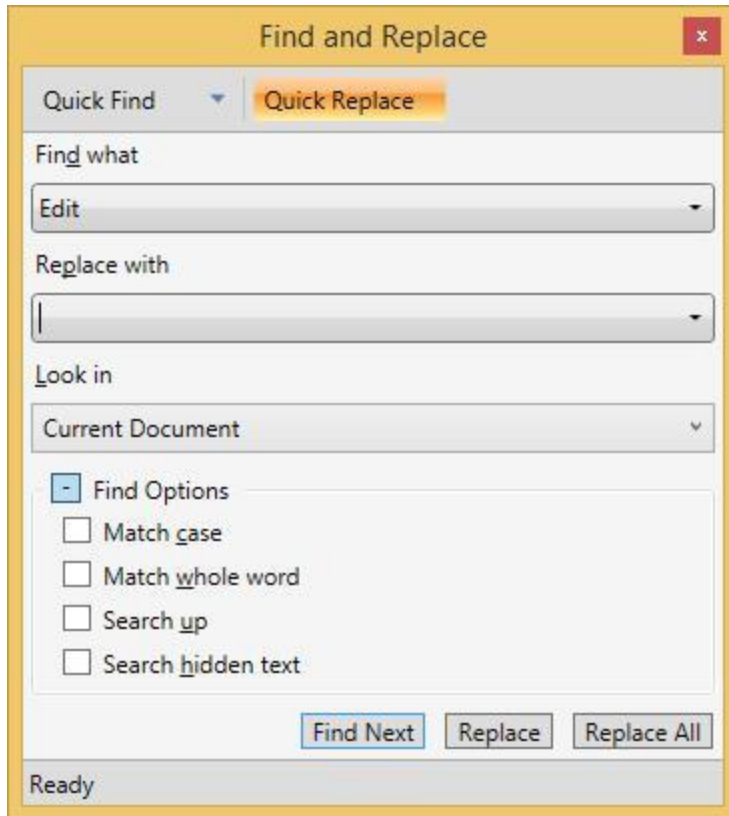


### Quick replace

Quick replace tab in Find and replace window enables the users to search a text and replace it with an alternate text. Quick replace tab can be enabled by clicking on the Quick Replace button at the top of the Find and Replace window or by using Ctrl+H key from EditControl.

The following image shows Quick Replace Tab in Find and Replace Window.





1. Quick Replace tab is similar to that of Quick Find except for **Replace With** field.
2. Replace with: Enter the alternative text to be replaced in this field.
3. Quick replace supports two functionalities as follows:
  - Replace: Replaces the immediate occurrence of text specified in **Find what** with text specified in Replace with field.
  - Replaces: Replaces all the occurrences of the text specified in **Find what** with the text specified in the Replace with field.

### GoToLine support

EditControl supports GoToLine functionality helps to reach out the line by programmatically or at run time using KeyBoard ShortKey as inspired from the Microsoft Visual Studio Editor. The GoToLine method is used to position the mouse pointer on any specified line. It not only positions the pointer on the appropriate line, but it also scrolls the concerned line as per the user requirement. Some of the common usage of this method associated with the purpose such as:

Edit control method	Description
GoToLine(Int LineNumber)	This function helps to scroll to a particular line, based on user input/requirement
ShowGoToLine()	This function helps to show the GoToLine Window programmatically and current cursor line index will be displayed in it.

ShowGoToLine(int LineNumber)	This function helps to show the GoToLine Window programmatically with the given line number included.
------------------------------	---

### GoToLine method

#### C#

```
//Places the Cursor at the given line number  
editcontrol.GoToLine(1);
```

#### VB.NET

```
'Places the Cursor at the given line number  
editcontrol.GoToLine(100)
```

### ShowGoToLine

The GoToLine Window is invoked using the ShowGoToLine method. The KeyBoard Shortcut key is Ctrl+G.

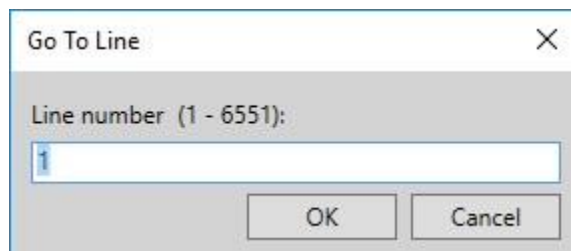
#### C#

```
// Method used to invoke the GoToLine Dialog Box  
editcontrol.ShowGoToLine();
```

#### VB.NET

```
'Method used to invoke the GoToLine Dialog Box  
editcontrol.ShowGoToLine()
```

The following output shows the GoToLine Dialog Window.



### ShowGoToLine with LineNumber

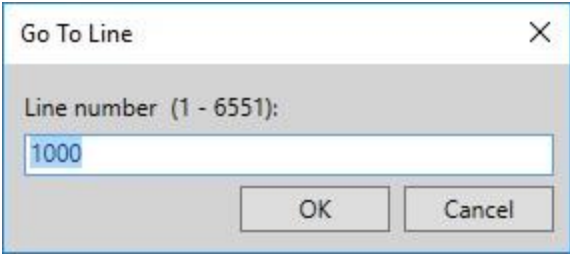
#### C#

```
// Method used to invoke the GoToLine Window with user given line number  
editcontrol.ShowGoToLine(1000);
```

#### VB.NET

```
'Method used to invoke the GoToLine Window with user given line number  
editcontrol.ShowGoToLine(1000)
```





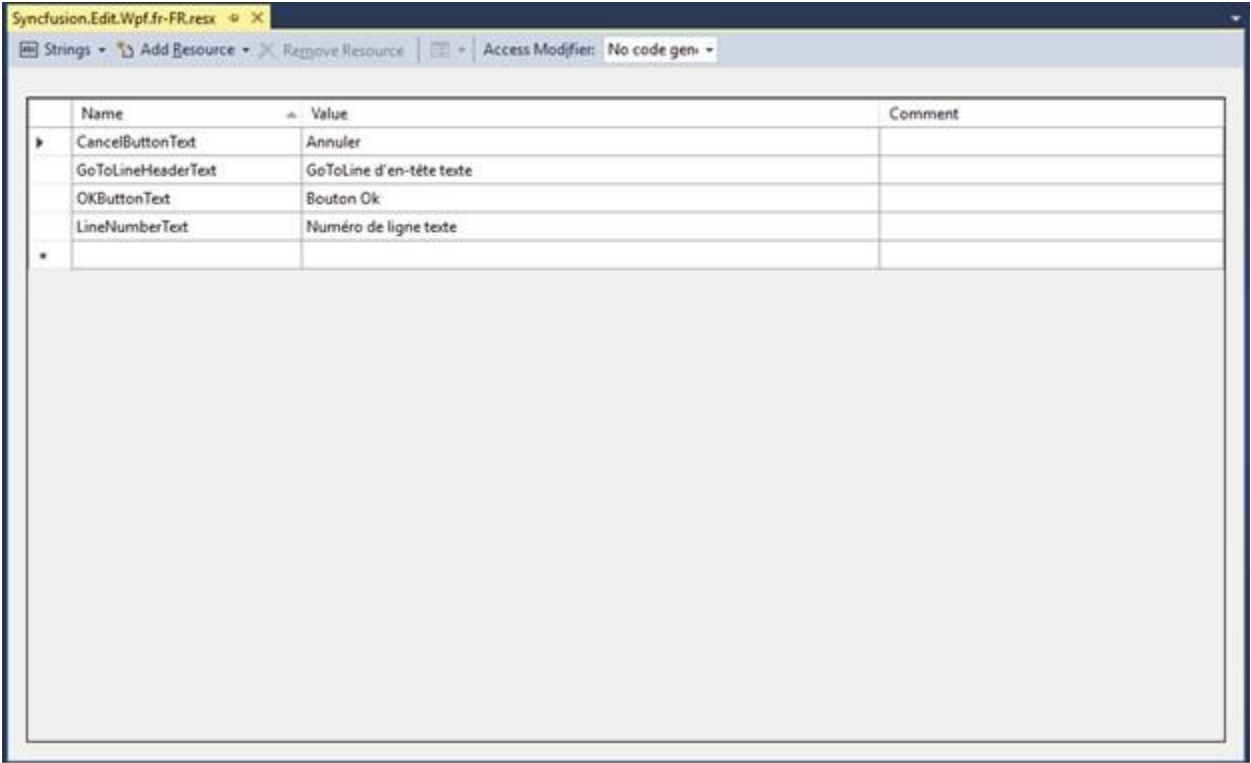
Localization support

EditControl also provides the Localization support for the GoToLine functionality. This allows to configure the content of the GoToLine window in the multi-lingual according to the cultures.

As explained in the [Localization](#) section, you can localize the content of the EditControl using the same steps.

The key text associated with the GoToLine Dialog Window can be localized as follows:

Localization text	Description
OkButtonText	Helps to localize the OK Button Text.
GoToLineHeaderText	Helps to localize the GoToLine Dialog Window Header Text.
LineNumberText	Helps to Localize the Line Number Text
CancelButtonText	Helps to localize the Cancel Button Text.



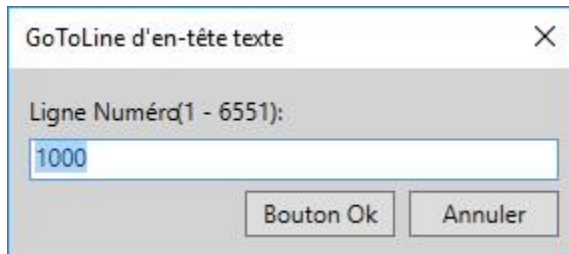
C#

```
public MainWindow()
{
    //used to assign the culture
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("fr-FR");
    InitializeComponent();
}
```

## VB.NET

```
Public Sub New()
    used to assign the culture
    System.Threading.Thread.CurrentThread.CurrentUICulture = New
    System.Globalization.CultureInfo("fr-FR")
    InitializeComponent()
End Sub
```

The following output shows the GoToLine Window localized in French culture.



### Selected Text Drag and Drop

You can drag and drop the selected text in a paragraph by setting the [AllowDragDrop](#) property to true. By default, Dragging the selected text will perform a move operation, while dragging using the Ctrl key will perform a copy operation.

## XML

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:Syntaxeditor_sample"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="Syntaxeditor_sample.MainWindow"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
    <Grid>
        <syncfusion:EditControl Name="editControl" BorderThickness="1"
AllowDragDrop="True" AllowDrop="True" EnableOutlining="False"
DocumentLanguage="Custom" Text="Setting multi-line text from C# using
Environment.NewLine." />
    </Grid>
</Window>
```

## C#

```
editControl.AllowDragDrop = true;  
editControl.AllowDrop = true;
```

## Language Support

### Supported Languages in WPF Syntax Editor

Edit for WPF provides built-in support for a procedural and markup languages such as C#, Visual Basic, XAML and XML. It also supports SQL language and facilitates the users to provide custom language configurations.

With the language support, EditControl enables the users to create, open, modify and save programming codes from different file types. EditControl provides built in Syntax highlighting and outlining support for all supported languages with SQL being exception in outlining support. It also provides built-in IntelliSense support for all procedural languages such as C# and Visual Basic.

The `DocumentLanguage` property in the EditControl class enables the users to select the language. DocumentLanguage is a Language enum type property with default value as Text. The following lines of code can be used to change the DocumentLanguage property.

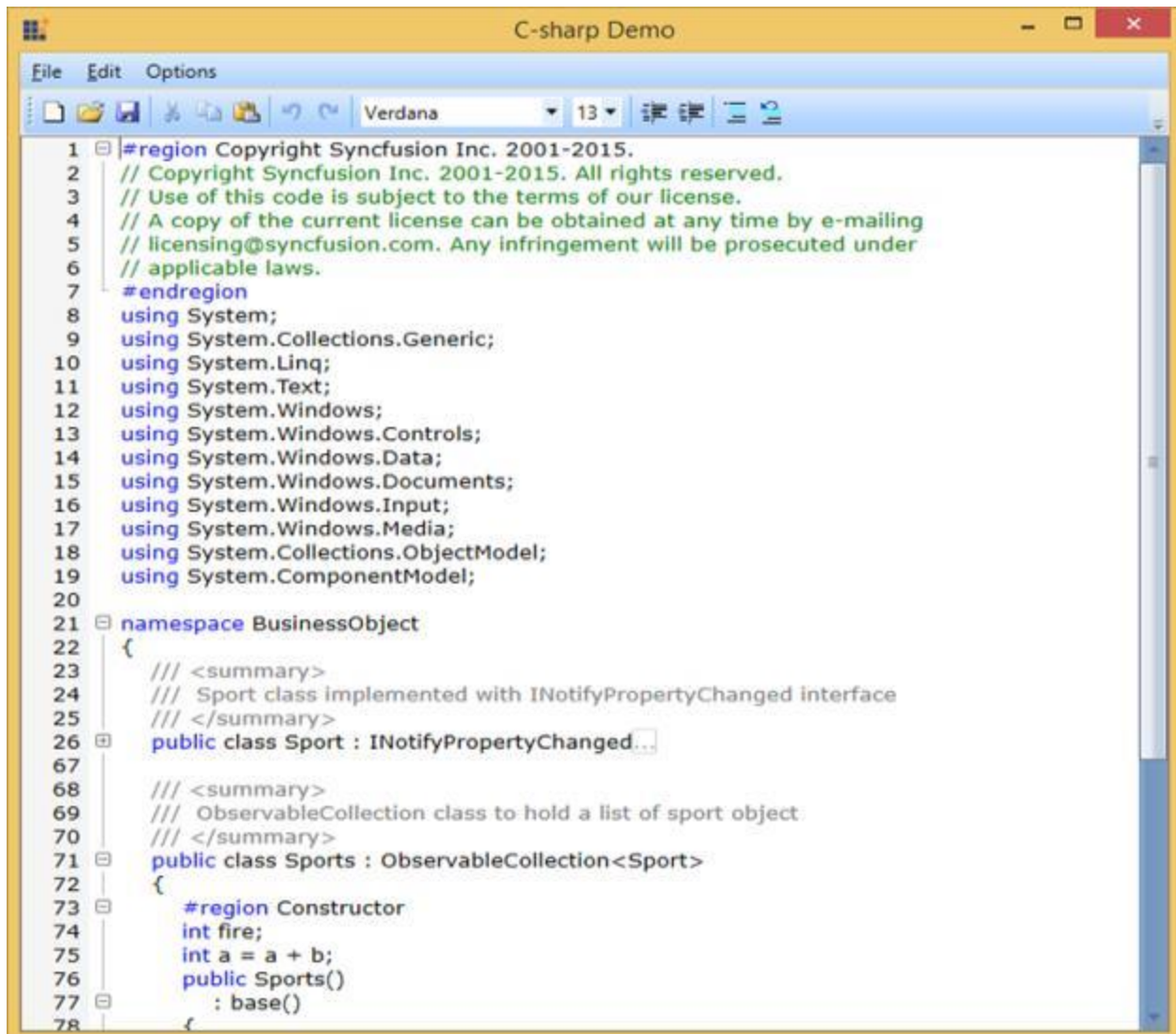
### XML

```
<sfedit:EditControl x:Name="editControl" DocumentLanguage="CSharp"  
DocumentSource="C:\Source.cs" FontSize="13"/>
```

### C#

```
editControl.DocumentLanguage = Languages.CSharp;
```

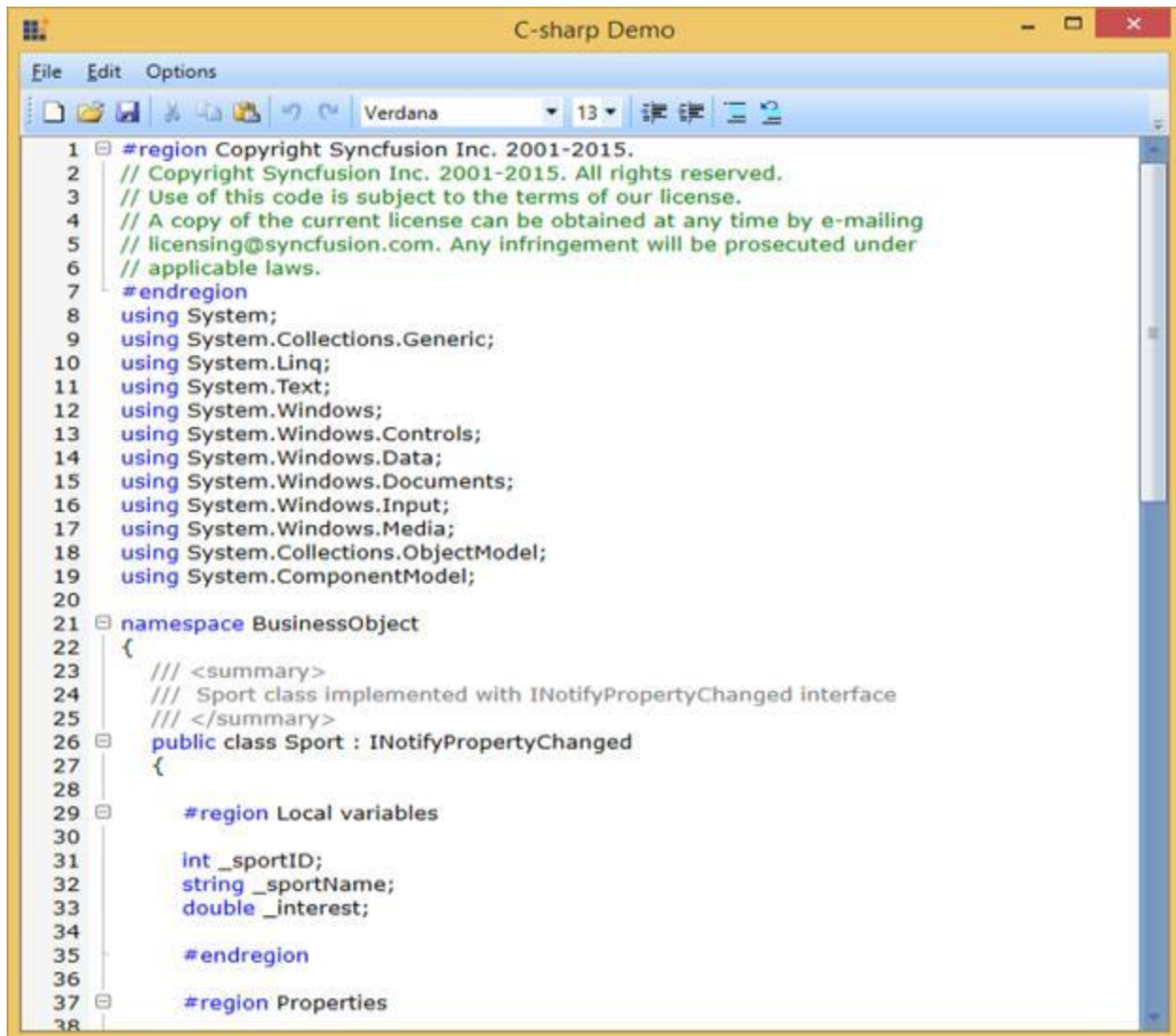
The following image displays the EditControl displaying contents.



### Syntax Highlighting in WPF Syntax Editor

EditControl is enhanced with the Syntax highlighting support. This applies different colorings to keywords, literals, comments and so on. EditControl provides syntax highlighting support for all supported languages and also based on the custom language configurations provided by the users.

The following image displays the EditControl with DocumentLanguage.



### Expand Collapse Support in WPF Syntax Editor

The EditControl provides built-in support for outlining. With this support, users can expand or collapse a block of text. EditControl provides expand-collapse support in C#, Visual Basic, XAML and Xml languages. It also provides expand-collapse support for custom languages based on the base class used for custom language (built-in expand and collapse support will be available if the custom language is implemented inheriting from `ProceduralLanguageBase` or `MarkupLanguageBase` class). Users can also implement their custom expand-collapse logic using `ApplyExpandCollapse` override method available in the custom language class. Refer to [Creating a custom language](#) topic for more information on expand-collapse implementations for custom languages.

The EditControl automatically identify the collapsible blocks using the language configurations of the current `DocumentLanguage`. EditControl displays + or – button in the expand collapse area of the EditControl to indicate that the lines under the block can be collapsed. The lines can be collapsed by pressing on the “–” button and can be expanded using the “+” button.

### Enabling expand-collapse button

Expand-collapse feature can be enabled/disabled using `EnableOutlining` property of EditControl class. The following code can be used to set the `EnableOutlining` property.

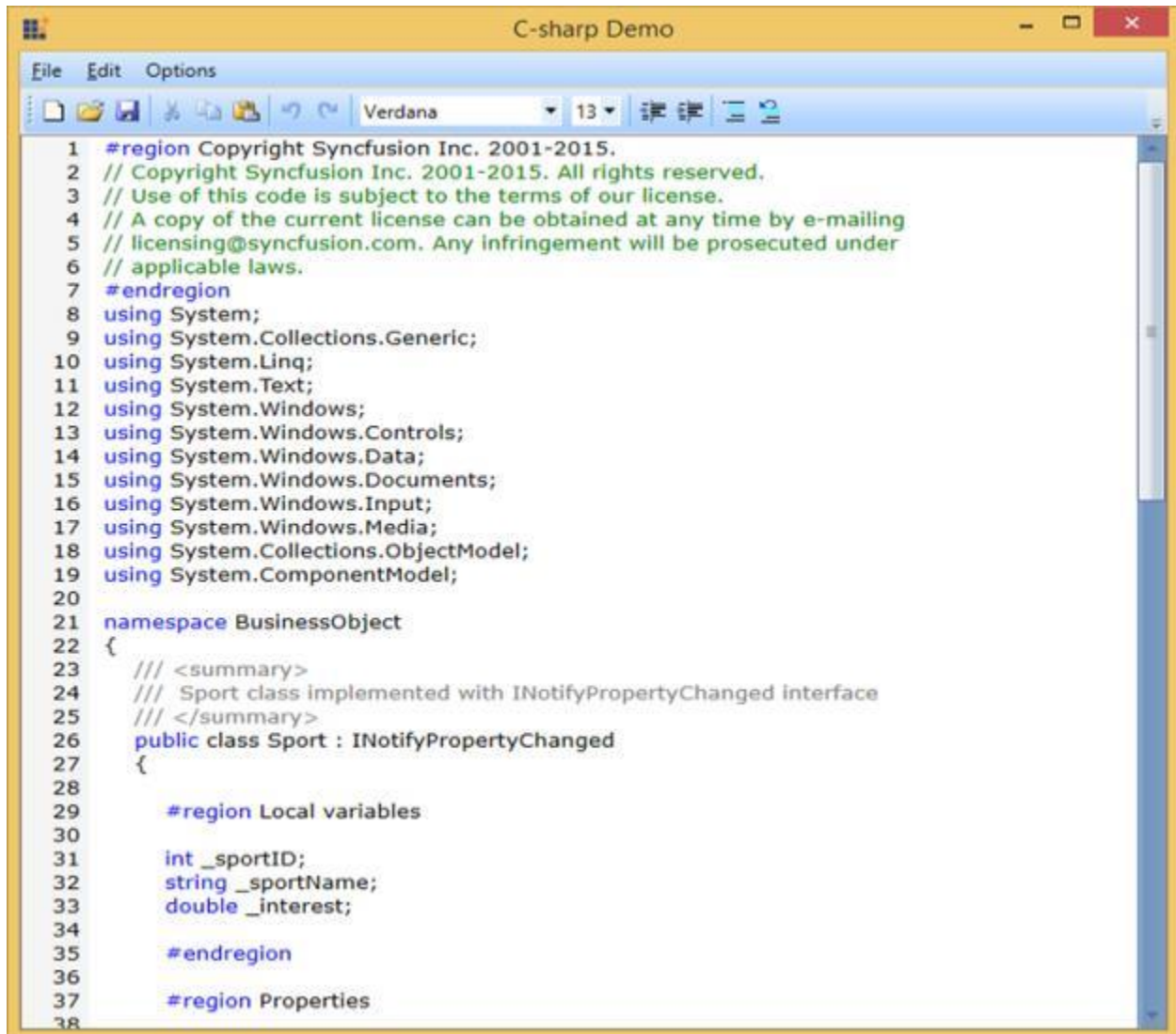
## XML

```
<sfedit:EditControl x:Name="editControl" DocumentLanguage="CSharp"
DocumentSource="C:\Source.cs" FontSize="13" EnableOutlining="False"/>
```

## C#

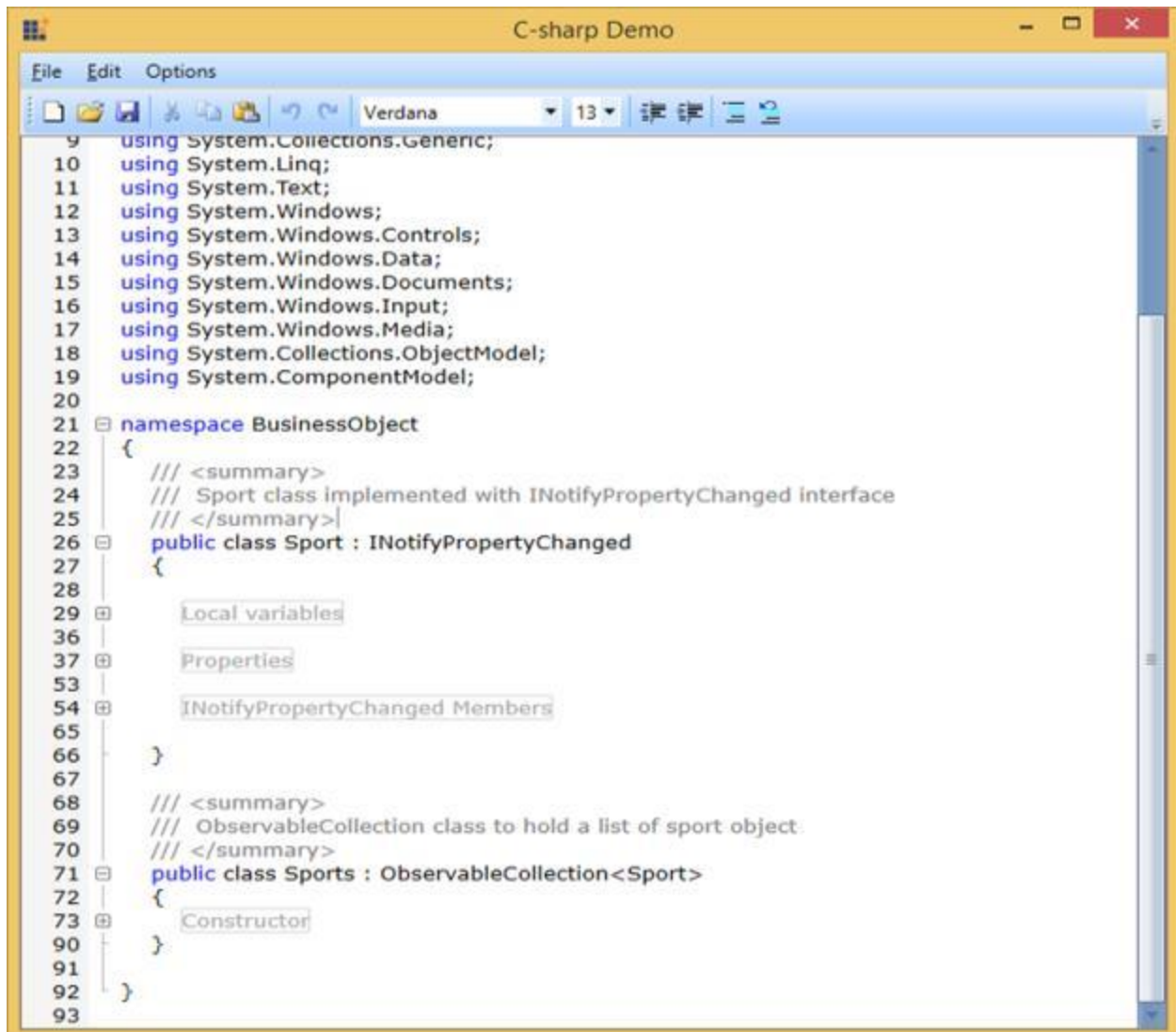
```
editControl.EnableOutlining = false;
```

The following image displays the EnableOutlining Is Set to False Window.



The following image displays EnableOutlining Set to True Window.

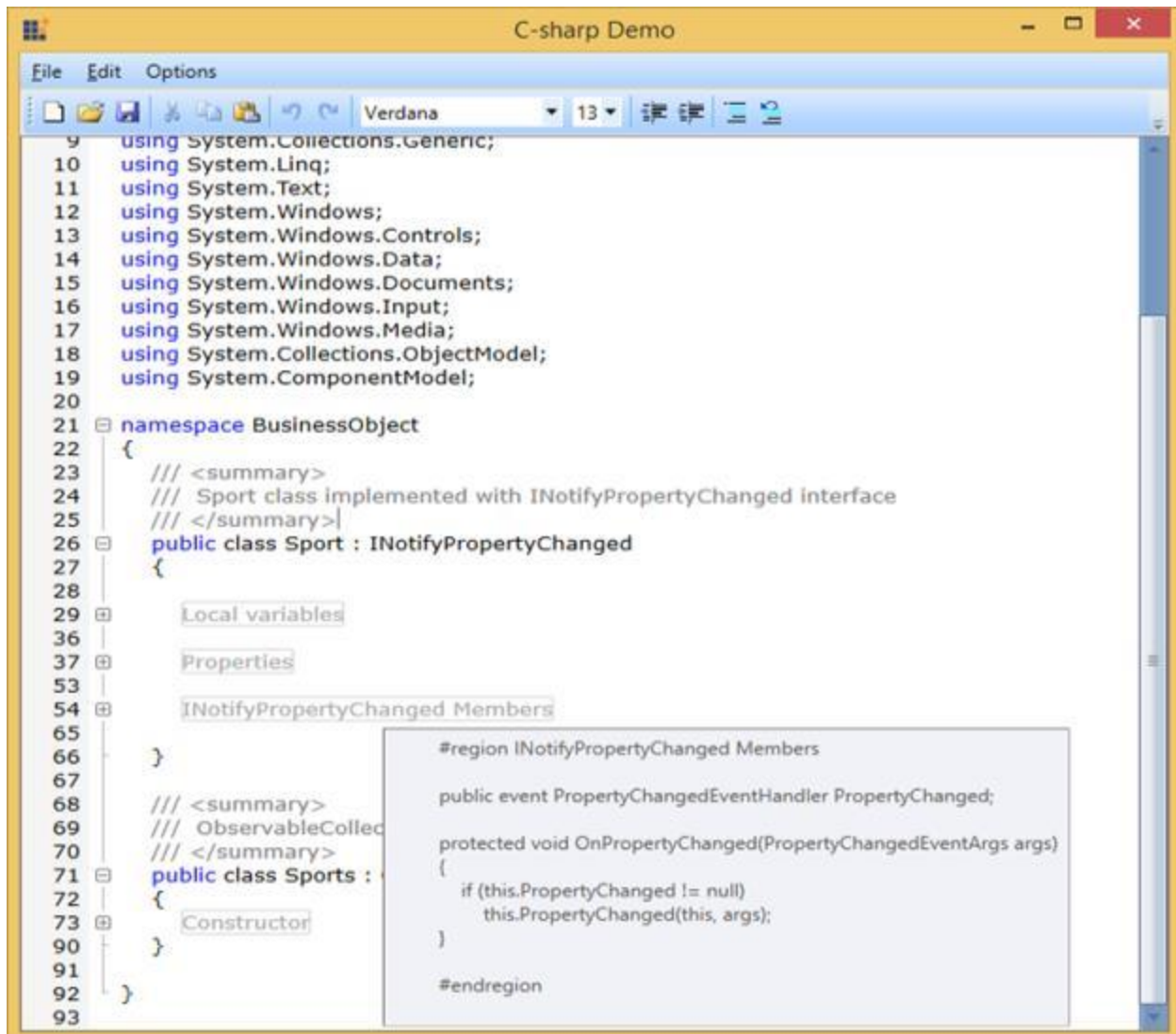




#### *ToolTip and selection support*

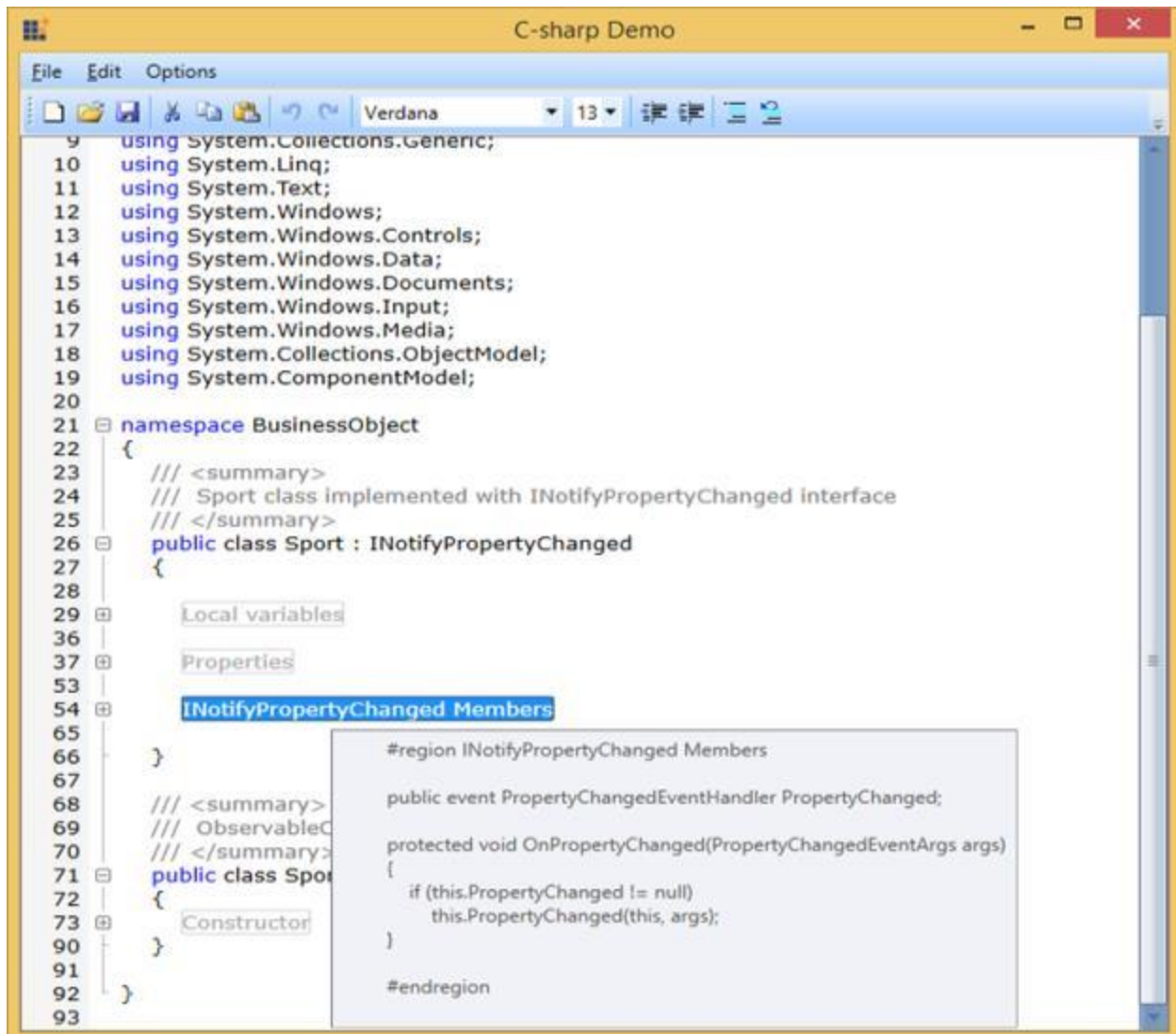
The EditControl is enhanced with ToolTip displaying the collapsed lines of text when mouse is hovered on a collapsed region or ellipses. It also enables the users to select the entire collapsed area by clicking on collapsed region or ellipses.

The following screenshot displays the Tooltip.

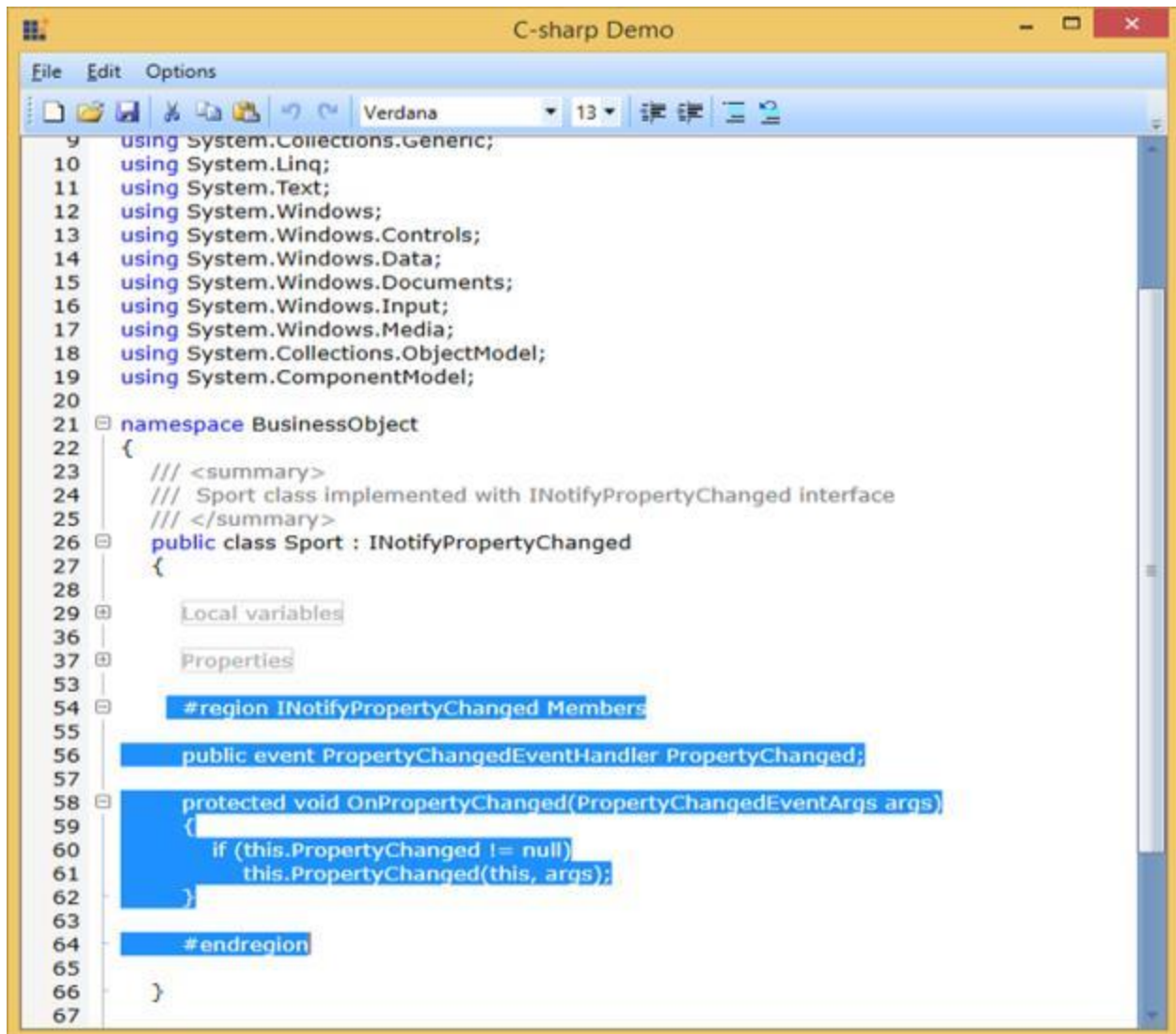


The following screenshot displays the Collapsed Region Window.





The following screenshot displays the Text Selection Window.



### LanguageBase Members

LanguageBase class in Syncfusion.Windows.Edit namespace plays a vital role in the implementation of language support in EditControl. In order to include a support for a new or custom language, a class inherited from LanguageBase or its sub classes need to be implemented. LanguageBase contains a number of properties and methods to enable the custom language developers configure their languages easily. This topic discusses about the properties and methods available in the LanguageBase class.

### Properties

The following table lists the properties available in LanguageBase class and its usage.

Property name	Type	Description
ApplyColoring	Boolean	Gets or sets a value indicating if the language supports syntax highlighting.

BlockEnd	String	Gets or sets a value indicating the end text that denotes end of a block in code.
BlockStart	String	Gets or sets a value indicating the start text that denotes Start of a block in code.
CaseSensitive	Boolean	Gets or sets a value indicating whether the Language has Case Sensitive or not.
CommitsIntellisenseItemOnSpaceBar	Boolean	Gets or sets a value indicating whether the selected Intellisense items to be appended when space bar is pressed.
EllipsisText	String	Gets or sets a value indicating the text to be displayed when a block is collapsed. By default it is set to "...".
FileExtension	String	Gets or sets File Extension supported by the language.
Formats	IEnumerable	Gets or sets a collection of type of IFormat indicating the language configurations. It has been modified to IEnumerable type to allow the users to binding a custom collection as language formats.
IntellisenseCommitCharacters	String	Gets or sets a value indicating when the selected Intellisense items to be appended. The string given will be converted to individual characters internally and verified to append the selected item to text.
IntellisenseDrillDownChar	Char	Gets or sets a value indicating a char on which the sub-items of the intellisense items to displayed.
IsSplitTextToWords	Boolean	Gets or sets a value indicating if the text in lines has to be spitted into tokens.
Lexem	IEnumerable	Gets or sets a collection of type of ILexem indicating the language configurations. It has been modified to IEnumerable type to allow the users to bind a custom collection as language lexems.
Name	String	Gets or sets Name of the Language.

ParentControl	EditControl	Gets a value indicating the Parent EditControl's reference.
SplitLinesRegex	String	Gets or sets a value indicating the Regex to be applied for splitting the text in lines.
SplitWordsRegex	String	Gets or sets a value indicating the Regex to be applied for splitting the lines into individual tokens.
SupportsIntellisense	Boolean	Gets or sets a value indicating whether the language supports IntelliSense or not.
SupportsOutlining	Boolean	Gets or sets a value indicating whether the language supports Outlining.
TextForeground	Brush	Gets or sets Foreground brush to be applied when no Lexems are applicable for the text.

## Methods

The following table lists the methods available in LanguageBase class and its purpose.

Method	Return Type	Description
ApplyColor(string text, int line);	IFormat	Helper method to apply coloring to the text based on the Lexems in the Language configurations. It is a protected method and can be overridden in the sub classes. The method is used to manipulate the Format to be applied to a particular token in a line.
ApplyExpandCollapse(ApplyExpandCollapseArgs args)	Void	Helper method to perform Expand Collapse for line items. When custom expand collapse logics is implemented, the methods can be overridden. This method runs in a background thread to overcome performance hits. Usage of any properties and methods outside the scope of the thread may result in an exception.
ApplyExpandItems()	Void	Helper method to Apply Expansions for the content in the EditControl. This method

		can be called if the Expand Collapse has to be refreshed for entire text in EditControl.
HideIntellisensePopup()	Void	Helper method to Hide Intellisense Pop-up.
InitializeExpandCollapse()	Void	This method is called before the ApplyExpandCollapse and can be used to perform any Initialization operations.
PositionIntellisensePopup(int line, int index)	Void	Helper method to adjust the Position of the Intellisense box. Line value in parameter represents the line number (starts of 0) and index in the parameter represents the cursor index
RefreshExpandItems(int line)	Void	Helper method to Refresh lines expansions from a specified line number. Line value in parameter refers to the index (starts from 0)
ShowIntellisenseBox(EditIntellisenseArgs args)	Void	Helper method to Show the Intellisense pop-up.
SplitTextToLines()	Void	A helper method to Split the text in the EditControl into individual lines.

#### *ProceduralLanguageBase Class*

ProceduralLanguageBase class contains syntax highlighting, outlining, and Auto mode IntelliSense implementations for procedural languages. C# and Visual Basic language support have been implemented using ProceduralLanguageBase class.

ProceduralLanguageBase class can be directly used as a base class for implementing a custom language. Refer to [Custom Language Support](#) topic for more information on how to configure the custom language in EditControl.

#### *MarkupLanguageBase Class*

MarkupLanguageBase class contains syntax highlighting and outlining implementations for markup languages. XAML and XML language support have been implemented using MarkupLanguageBase class.

MarkupLanguageBase class can be directly used as a base class for implementing a custom markup language. Refer to [Custom Language Support](#) topic for more information on how to configure the custom language in EditControl.

### Custom Language Support in WPF Syntax Editor

The EditControl provides built-in syntax highlighting and outlining support for common procedural languages such as C# and Visual Basic, markup languages like XAML and XML, and SQL. It also allows the developer to create Custom Language configurations to apply syntax highlighting and outlining.

The EditControl's Custom Language support helps the developers to create code editors for custom languages by specifying their language configurations class and applying it as a Custom Language in EditControl.

The Edit WPF provides the following classes to enable the users create their Custom Languages by inheriting from any of the following base classes.

- LanguageBase
- ProceduralLanguageBase
- MarkupLanguageBase

Each of these classes has basic level of implementations to let the users define minimum set of configurations to get started with the Custom Languages easily.

#### Important features

- Allows the developer to choose the base class considering the complexity and type of language that developer is planning to create.
- Provides much more flexibility to developers than before in writing their custom logics in syntax coloring and outlining.

Supports DataBinding of Lexem and formats by implementing their custom business classes with ILexem and IFormat Interfaces.

### Customization of features

#### Language base classes

As described in earlier topics, the Edit WPF provides three base classes for the developers to choose, in order to create their Custom Languages. Following are the level of implementations available in each of the base classes.

**LanguageBase:** Contains implementation that are common for all the languages such as rendering lines, backspace and delete key operations, comment and uncomment commands and provides methods to override language specific functionalities such as syntax coloring, outlining, and IntelliSense.

**ProceduralLanguageBase:** Contains implementations of syntax highlighting, outlining and IntelliSense for procedural languages such C# and Visual Basic.

**MarkupLanguageBase:** Contains implementation of Regex based syntax highlighting and outlining features specific to markup languages such as XAML and XML.

### Creating a custom language

This section of the documentation discusses on creating a Custom Language configuration with EditControl. A language specification document for which custom language is being created will help the users in finding the details of languages such as keywords, literals, comments etc. In this section, a custom language for IronPython is created as an example.

Create a new class inheriting from `ProceduralLanguageBase` class and set basic properties of the language, by using the following code.

### C#

```
public class PythonLanguage : ProceduralLanguageBase
{
    public PythonLanguage(EditControl control)
    : base(control)
    {
        this.Name = "Python";
        this.FileExtension = ".py";
        this.ApplyColoring = true;
        this.SupportsIntellisense = false;
        this.SupportsOutlining = true;
        this.TextForeground = Brushes.Black;
    }
}
```

Create Lexem and Formats collection for Custom Language, to enable data binding with Lexem and Format properties; we have modified the `Lexem` and `Formats` property type to `IEnumerable` from `LexemCollection` and `FormatCollection` respectively.

**Note:** In previous versions, Lexem and Formats were added directly to Lexems and Formats properties of `EditLanguage` class. Now since it is `IEnumerable`, a custom collection need to be created and apply it to Lexem and Format properties of custom language class, here it is `PythonLanguage` class.

### Formats for IronPython

Create a collection of `IFormat` implemented classes to apply in Formats property of custom language, by using the following code.

**Note:** `EditControl` uses this collection to fetch the color to be applied to the tokens.

### XML

```
<syncfusion:FormatsCollection x:Key="pythonLanguageFormats">
  <syncfusion:EditFormats Foreground="Green" FormatName="CommentFormat"/>
  <syncfusion:EditFormats Foreground="Black"
    FormatName="MultilineCommentFormat"/>
  <syncfusion:EditFormats Foreground="Blue" FormatName="KeywordFormat"/>
  <syncfusion:EditFormats Foreground="Navy" FormatName="OperatorFormat"/>
  <syncfusion:EditFormats Foreground="Gray" FormatName="LiteralsFormat"/>
</syncfusion:FormatsCollection>
```

### Lexem for IronPython

Create a collection of `ILexem` implemented class, by using the following code.

**Note:** This collection will be applied to Lexem property of the custom language. `EditControl` uses the Lexem property to retrieve all keywords, comments, literals, preprocessors etc.

### XML

```
<syncfusion:LexemCollection x:Key="pythonLanguageLexems">
  <syncfusion:Lexem StartText="class \w+[\s:\w,() ]+" IsRegex="True"
    IsMultiline="True" ContainsEndText="True" LexemType="CodeSnippet"
```

```

EndText="\r\n" ScopeLevel="Class" ShowAlternateIntellisenseText="True"
IntellisenseDisplayText="class"/>
<syncfusion:Lexem StartText="def \w+[\s:\w,() ]+" IsRegex="True"
IsMultiline="True" ContainsEndText="True" LexemType="CodeSnippet"
EndText="\r\n" ScopeLevel="Member" ShowAlternateIntellisenseText="True"
IntellisenseDisplayText="def"/>
<syncfusion:Lexem StartText="#" EndText="\r\n" IsMultiline="False"
ContainsEndText="True" LexemType="Comment" FormatName="CommentFormat"/>
<syncfusion:Lexem StartText="&#34;&#34;&#34;" EndText="&#34;&#34;&#34;"
IsMultiline="True" ContainsEndText="True" LexemType="Comment"
FormatName="CommentFormat" />
<syncfusion:Lexem StartText="and" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="as" ContainsEndText="False" IsMultiline="False"
LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="assert" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="break" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="class" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="continue" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="def" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="del" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="elif" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="else" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="except" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="exec" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="finally" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="for" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="from" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="global" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="if" ContainsEndText="False" IsMultiline="False"
LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="import" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="in" ContainsEndText="False" IsMultiline="False"
LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="is" ContainsEndText="False" IsMultiline="False"
LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="lambda" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="not" ContainsEndText="False"
IsMultiline="False" LexemType="Keyword" FormatName="KeywordFormat"/>
<syncfusion:Lexem StartText="or" ContainsEndText="False" IsMultiline="False"
LexemType="Keyword" FormatName="KeywordFormat"/>

```



[illegible]

```

<syncfusion:Lexem StartText="&#38;" ContainsEndText="False"
IsMultiline="False" LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="~" ContainsEndText="False" IsMultiline="False"
LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="=" ContainsEndText="False" IsMultiline="False"
LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="&#60;" ContainsEndText="False"
IsMultiline="False" LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="&#62;" ContainsEndText="False"
IsMultiline="False" LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="==" ContainsEndText="False" IsMultiline="False"
LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="!=" ContainsEndText="False" IsMultiline="False"
LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="&#60;=" ContainsEndText="False"
IsMultiline="False" LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="&#62;=" ContainsEndText="False"
IsMultiline="False" LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="+=" ContainsEndText="False" IsMultiline="False"
LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="--" ContainsEndText="False" IsMultiline="False"
LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="*=" ContainsEndText="False" IsMultiline="False"
LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="%=" ContainsEndText="False" IsMultiline="False"
LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="/=" ContainsEndText="False" IsMultiline="False"
LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="&#38;=" ContainsEndText="False"
IsMultiline="False" LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="^=" ContainsEndText="False" IsMultiline="False"
LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="|=" ContainsEndText="False" IsMultiline="False"
LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="&#60;&#60;" ContainsEndText="False"
IsMultiline="False" LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="&#60;&#60;=" ContainsEndText="False"
IsMultiline="False" LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="=&#62;" ContainsEndText="False"
IsMultiline="False" LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="&#60;&#62;" ContainsEndText="False"
IsMultiline="False" LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="&#62;&#62;=" ContainsEndText="False"
IsMultiline="False" LexemType="Operator" FormatName="OperatorFormat"/>
<syncfusion:Lexem StartText="**;" ContainsEndText="False"
IsMultiline="False" LexemType="Operator" FormatName="OperatorFormat"/>
</syncfusion:LexemCollection>

```

Initialize Lexem and Formats properties of PythonLanguage by using the following code.

**C#**

```

customLanguage.Lexem = this.Resources["pythonLanguageLexems"] as
LexemCollection;
customLanguage.Formats = this.Resources["pythonLanguageFormats"] as
FormatsCollection;

```

Syntax highlighting will work with these changes itself as the implementations are same for procedural languages. Moreover, if there are any tweaks needs to be done, ApplyColoring method can be overridden to apply syntax colorings.

Outlining in IronPython differs when compared to C# and Visual Basic as IronPython does not have specific keyword or symbol to mark the close of a block. So the regular expand collapse logic in ProceduralLanguageBase needs to be overridden to write custom outlining logic for IronPython.

To provide interactivity and overcome UI blockages, ApplyExpandCollapse method runs in the Background thread hence any attempt to access resources outside this thread will throw a thread access exception. Therefore helper classes that would serve the necessary information as argument for this ApplyExpandCollapse method is created.

#### LineItemExpandInformation

Name of the property	Description
ContainsLines	A boolean property to determine whether the line contains child items. Default value is set to false. This property has to be set based on the language's block configuration. Set this property to <code>true</code> to display a button in the outlining area of EditControl otherwise a line is displayed.
ContainsPreprocessor	A boolean property to determine if the LineItem contains a preprocessor text. This property is used to display the collapsed lines similar to Visual Studio. For instance, region name within rounded rectangle.
EndLine	Specifies the line number of block's end.
IsExpanded	Specifies the state of the LineItem. This property displays '-' symbol for <code>true</code> value and '+' symbol for <code>false</code> value.
LineStartBlock	A property to hold Block information. This property is used as a reference to previous lines block information.
ParentLineNumber	Specifies the LineNumber of its Parent.
PreprocessorText	Specifies the text to be displayed when the preprocessor is collapsed.
StartLine	Specifies the Start Line of the block.
ToggleExpansion	<p>A boolean property to expand all the child lines of a line item. This property is used to unhide all the collapsed lines when a collapsed line loses the block status.</p> <p>For instance, the lines under a namespace need to be expanded when the namespace is commented.</p>

Property values of **LinItemExpandInformation** need to be updated in the **ApplyExpandCollapse** method which will be reflected in the **LinItem** once the Background thread is completed.

### ApplyExpandCollapseArgs

ApplyExpandCollapseArgs class contains following properties to enable the developers get necessary information about the language and content of the EditControl.

Name of the property	Description	Type of the property
Assemblies	Contains the assemblies added as references in EditControl, if any operations related to IntelliSense are to be performed while outlining like creating scope definitions etc.	List (Uri)
ExpandInformation	A property of type LinItemExpandInformation containing all necessary properties related to current LinItem.	LinItemExpandInformation
Formats	Returns the Formats property of custom language.	IEnumerable (IFormat)
LanguageBlocks	Provides a collection of blocks definitions from the Lexem included.	List (BlockListener)
Lexems	Returns the Lexem property of custom language.	IEnumerable (ILexem)
Source	Returns a collection of LinItemExpandInformation for all lines in EditControl.	List (LinItemExpandInformation)

Here we proceed to override the ApplyExpandCollapse method of LanguageBase class to apply custom outlining. In this method, use LanguageBlocks property from argument as reference for Block definitions. These block definitions are generated using the Lexem property.

### C#

```

/// <summary>
/// Override method of ApplyExpandCollapse to perform language specific
/// expand
/// options. This method runs in a background thread to provide better
/// interactivity.
/// </summary>
/// <param name="args">represents the instance of ApplyExpandCollapseArgs
/// contains
/// all necessary information related to each line</param>
protected override void ApplyExpandCollapse(ApplyExpandCollapseArgs args)

```

```

{
bool createListener = false;
if (args.LanguageBlocks == null)
{
return;
}
/// LanguageBlocks of args is a collection block definitions added in Lexem
property
/// we use it as identify if the current line contains any block start
var selblocks = from blk in args.LanguageBlocks
where (!blk.IsRegex &&
args.ExpandInformation.Text.Trim().StartsWith(blk.BlockStart) ||
(blk.IsRegex && Regex.Match(args.ExpandInformation.Text,
blk.BlockStart).Success))
select blk;
if (selblocks.Count() > 0)
{
var block = selblocks.ElementAt(0);
///Checks if the Line starts with a comment or comes under a comment block.
if (CheckCommentBlock(args.ExpandInformation))
{
createListener = true;
}
if (createListener)
{
///currentListener property is used to hold the current block information.
Whenever a block is encountered
///the currentListener value is moved in a stack and a new currentListener
is created with the necessary values.
if (currentListener != null)
{
///IronPython does not have a specific Block end and a condition is checked
whether a class is contained in the current block.
if (currentListener.BlockStart.StartsWith("class") &&
block.BlockStart.StartsWith("class"))
{
var parentLineItem = args.Source[currentListener.ParentLineNumber - 1];
parentLineItem.EndLine = args.Source.IndexOf(args.ExpandInformation) - 1;
currentListener.EndLineNumber = parentLineItem.EndLine;
lastBlockEndLine = parentLineItem.EndLine;
}
else
{
args.ExpandInformation.ParentLineNumber = currentListener.ParentLineNumber;
args.ExpandInformation.StartLine =
args.Source.IndexOf(args.ExpandInformation) + 2;
if (blocksStack == null)
{
blocksStack = new Stack<BlockListener>();
}
blocksStack.Push(currentListener);
}
}
currentListener = new BlockListener()
{
BlockStart = block.BlockStart,
BlockEnd = block.BlockEnd,

```

```
IsPreprocessor = block.IsPreprocessor,
ParentLineNumber = args.Source.IndexOf(args.ExpandInformation) + 1,
IsRegex = block.IsRegex,
CheckParentType = block.CheckParentType,
ParentLexemType = block.ParentLexemType,
LexemType = block.LexemType,
ScopeLevel = block.ScopeLevel
};
args.ExpandInformation.ContainsPreprocessor = block.IsPreprocessor;
args.ExpandInformation.PreprocessorText = block.IsPreprocessor ?
args.ExpandInformation.Text.Trim().Substring(block.BlockStart.Length).Trim()
: string.Empty;
args.ExpandInformation.ContainsLines = true;
int tempInd = args.Source.IndexOf(args.ExpandInformation);
args.ExpandInformation.StartLine = block.IsPreprocessor ? tempInd + 1 :
tempInd + 2;
}
else if (currentListener != null)
{
args.ExpandInformation.ParentLineNumber = currentListener.ParentLineNumber;
args.ExpandInformation.IsExpanded = true;
args.ExpandInformation.ContainsLines = false;
}
else
{
args.ExpandInformation.ParentLineNumber = -1;
args.ExpandInformation.IsExpanded = true;
args.ExpandInformation.ContainsLines = false;
}
}
else if (currentListener != null)
{
args.ExpandInformation.ContainsLines = false;
args.ExpandInformation.ParentLineNumber = currentListener.ParentLineNumber;
///Checking if the text is empty to close the block. If there is a block end
available we can check the condition and close the block.
if (args.ExpandInformation.Text.Trim() == string.Empty)
{
var parentLineItem = args.Source[currentListener.ParentLineNumber - 1];
parentLineItem.EndLine = args.Source.IndexOf(args.ExpandInformation);
currentListener.EndLineNumber = parentLineItem.EndLine;
lastBlockEndLine = parentLineItem.EndLine;
currentListener = null;
if (blocksStack.Count > 0)
{
currentListener = blocksStack.Pop();
}
}
else if (currentListener.BlockEnd == null)
{
currentListener = null;
if (blocksStack.Count > 0)
{
currentListener = blocksStack.Pop();
}
}
}
```

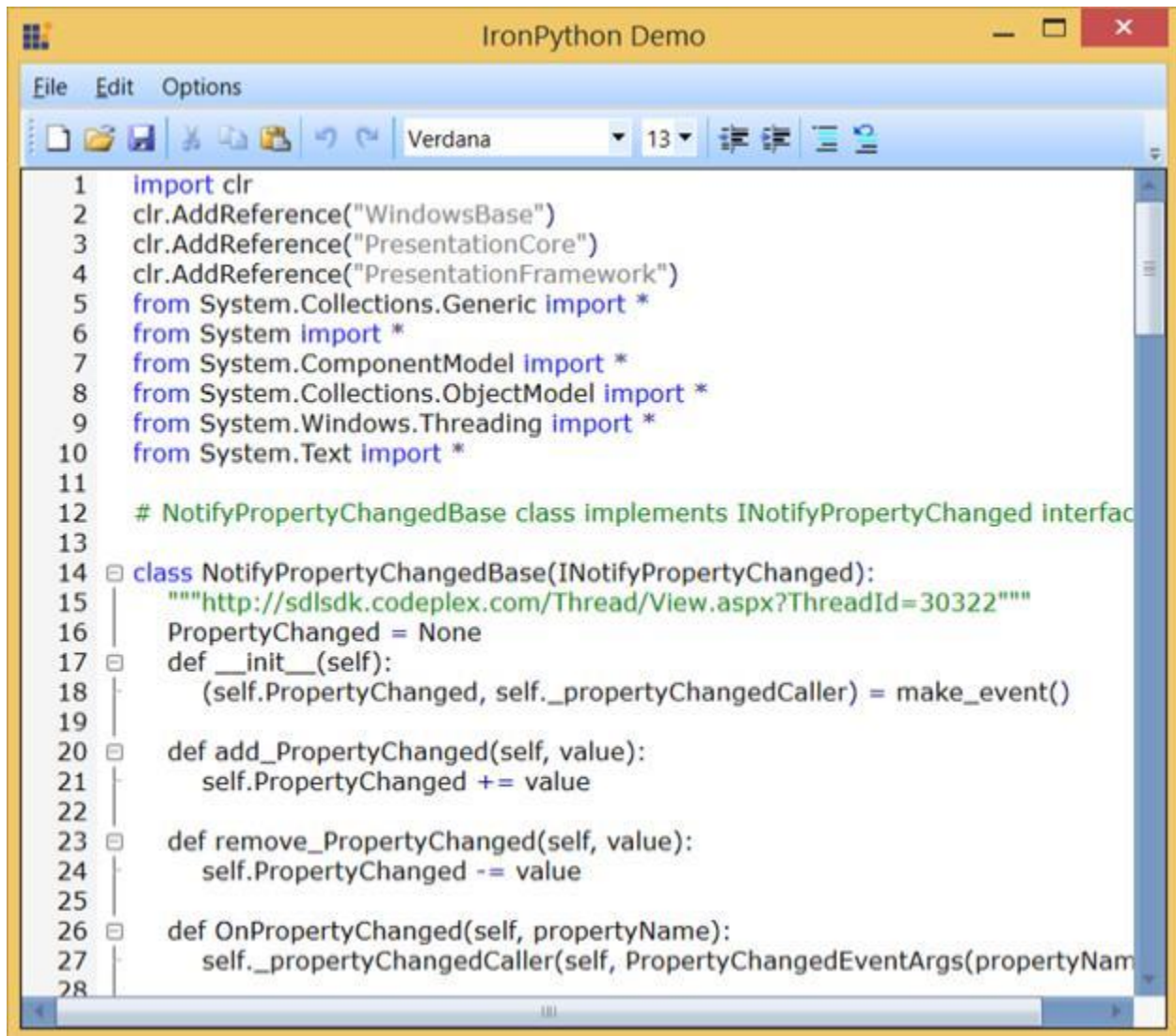
```

else
{
args.ExpandInformation.ContainsLines = false;
if (!args.ExpandInformation.IsExpanded)
{
args.ExpandInformation.IsExpanded = true;
args.ExpandInformation.ToggleExpansion = true;
}
args.ExpandInformation.IsExpanded = true;
args.ExpandInformation.ContainsLines = false;
}
///Closing block for last line since there will not be any empty lines
if (args.Source.IndexOf(args.ExpandInformation) == args.Source.Count - 1)
{
if (currentListener != null)
{
var parentLineItem = args.Source[currentListener.ParentLineNumber - 1];
parentLineItem.EndLine = lastBlockEndLine;
currentListener.EndLineNumber = parentLineItem.EndLine;
}
}
}
/// <summary>
/// Override method of ApplyExpandCollapse to perform any initialization
/// before Expand
///collapse is applied.
/// </summary>
protected override void InitializeApplyExpandCollapse()
{
currentListener = null;
}
/// <summary>
/// Helper method to check if the line starts with single line or multiline
/// comment
/// </summary>
/// <param name="item">represents the LineItemExpandInformation
/// instance.</param>
/// <returns>
/// a boolean value indicating if the line starts with comment type of
/// lexem.
/// </returns>
private bool CheckCommentBlock(LineItemExpandInformation item)
{
if (item.LineStartBlock != null && item.LineStartBlock.LexemType ==
EditTokenType.Comment)
{
return false;
}
if (commentsCollection != null)
{
LineItemExpandInformation tempItem = item;
RegexOptions options = this.CaseSensitive ? RegexOptions.None :
RegexOptions.IgnoreCase;
var blocks = commentsCollection.Where(lexem => (!lexem.IsMultiline &&
tempItem.Text.Trim().StartsWith(lexem.StartText)) || (lexem.IsMultiline &&
tempItem.Text.Trim().StartsWith(lexem.StartText) &&

```

```
(tempItem.Text.IndexOf(lexem.EndText) == -1 ||  
tempItem.Text.IndexOf(lexem.EndText) == tempItem.Text.Length -  
lexem.EndText.Length));  
if (blocks.Count() > 0)  
{  
    return false;  
}  
}  
return true;  
}
```

The following screenshot displays IronPython Demo Window.



**Note:** Refer to IronPython Demo in the Essential Studio WPF SampleBrowser to view its functionality or debug the application.

#### IntelliSense Support in WPF Syntax Editor

The Essential Edit for WPF provides Visual Studio like IntelliSense support. With IntelliSense support users can quickly choose the possible words while typing text in the control.



IntelliSense support in Edit WPF facilitates you to select possible words while typing text in the EditControl.

When you type the text in the EditControl, it displays a list of possible words in a popup. You can navigate, using Up and Down arrow keys or mouse and Scrollbar, to appropriate items. Select an item from the list to append to the text in the EditControl.

### Important features

- **IntelliSense** in Edit WPF works in two modes namely **Auto** or **Custom**.
- **Auto mode**: This automatically generates the list of items to be displayed from the pre-built assemblies specified in `AssemblyReferences` property. This mode of operation is currently supported for C# and Visual Basic language respectively.
- **Custom mode**: This enables the users to provide the list of items to be displayed in the IntelliSense.
- Exclusive properties in EditControl enable the users to customize the look and feel of the IntelliSense pop-up and its items.

This provides the facility to modify the characters on the selected item to be appended, to the text similar to that of Visual Studio IntelliSense settings. It also provides options to enable or disable appending text when space bar is pressed.

#### Customization of IntelliSense modes

IntelliSense in Edit WPF works in two modes: Auto and Custom. IntelliSense modes can be switched by using `IntelliSenseMode` property in EditControl class. It is an enum of the type of `IntelliSenseMode`. By default, `IntelliSenseMode` property is set to Auto.

#### Auto mode

In Auto mode, EditControl generates the IntelliSense list box items similar to Visual Studio based on the current language configurations (Lexem). IntelliSense also displays Types, Properties, Events and Methods from pre-built assemblies specified using `AssemblyReferences` property of EditControl class.

---

**Note:** Auto IntelliSense mode is currently supported for C# and Visual Basic languages and will be extended to other markup languages supported by EditControl in forthcoming releases.

---

#### Adding EditControl to the application

Add EditControl to the application and set its `IntelliSenseMode` to Auto by using the following code.

#### XML

```
<!--Adding EditControl to application and setting its IntelliSenseMode to Auto-->  
<syncfusion:EditControl Background="White" DocumentLanguage="CSharp"  
Name="EditControll1" IntellisenseMode="Auto"/>
```

#### C#

```
public partial class MainWindow : Window  
{  
    ObservableCollection<Uri> uriList;  
    public MainWindow()  
    {  
        InitializeComponent();  
    }  
}
```

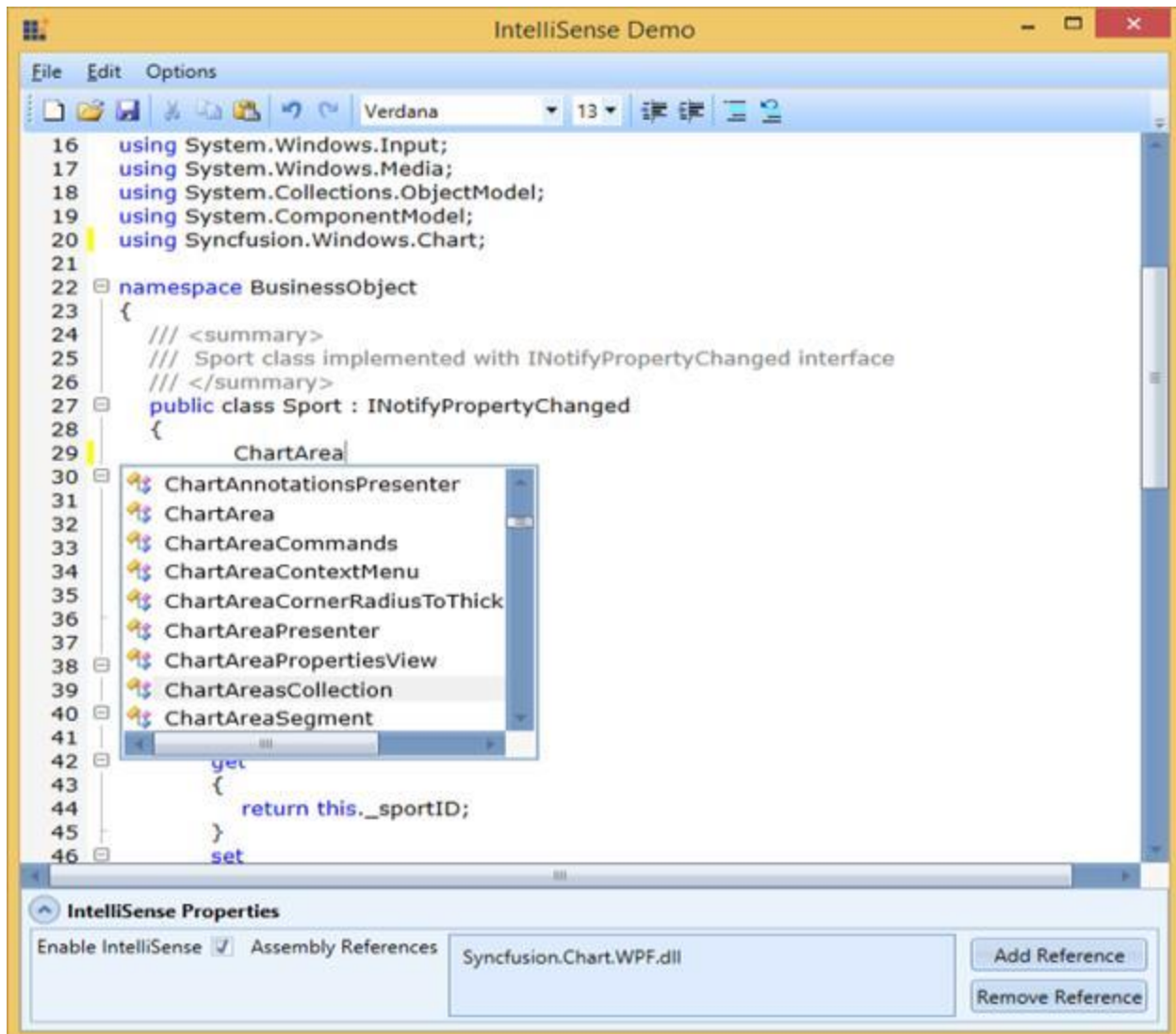
```
uriList = new ObservableCollection<Uri>();
EditControll.DocumentSource = "../.. /Source.cs";
EditControll.AssemblyReferences = uriList;
}
private void Button_Click(object sender, RoutedEventArgs e)
{
    OpenFileDialog fileDialog = new OpenFileDialog();
    fileDialog.Filter = "Assembly Files (*.dll) | *.dll";
    fileDialog.ShowDialog();
    if (fileDialog.FileName.Trim() != string.Empty)
    {
        uriList.Add(new Uri(fileDialog.FileName));
    }
}
```

---

**Note:** Having an INotifyCollectionChanged implemented collection as AssemblyReferences will update the IntelliSense items automatically when an assembly reference is added at runtime.

---

The following image displays the Intellisense Demo Window.



### Custom mode

IntelliSense support in Edit WPF enables you to bind collections of your business object as an ItemsSource of IntelliSenseListBox. It also provides the flexibility to change the ItemTemplate of the IntelliSenseListBox to suite your business object or requirements.

This ensures that your business object is implemented from IIntelliSenseItem interface to your business object compatible with the IntelliSenseListBox. The EditControl have exclusive properties implemented to enable the users bind custom collections and apply custom ItemTemplates.

- **IntelliSenseMode:** Set `IntelliSenseMode` property to custom to apply a custom ItemsSource to IntelliSense.
- **IntelliSenseCustomItemsSource:** IEnumerable type of property to bind custom ItemsSource to IntelliSense ListBox.
- **IntelliSenseItemTemplate:** DataTemplate type of property to apply custom ItemTemplate to IntelliSense ListBox.

### Creating DataTemplate in the ResourceDictionary

Create DataTemplate in the ResourceDictionary to apply it as `IntellisenseItemTemplate` property of `EditControl` by using the following code.

#### XML

```
<DataTemplate x:Key="CustomIntelliSenseItemTemplate">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Image Source="{Binding Icon}" MaxHeight="16" MaxWidth="16" Margin="3"/>
<TextBlock Text="{Binding Text}" Grid.Column="1" Margin="3"/>
</Grid>
</DataTemplate>
```

Apply `IntellisenseMode` and `IntellisenseItemTemplate` properties by using the following code.

#### XML

```
<!--Adding EditControl to application and setting its IntelliSenseMode to Custom-->
<syncfusion:EditControl Background="White" Name="EditControl1"
IntellisenseMode="Custom"
IntellisenseItemTemplate="{StaticResource
CustomIntelliSenseItemTemplate}"/>
```

### Creating a custom business object

Create a Custom Business Object implemented using `IIntellisenseItem` interface by using the following code.

#### C#

```
/// <summary>
/// Business object implemented from IIntelliSenseItem interface in
/// Syncfusion.Windows.Edit namespace
/// </summary>
public class CustomIntelliSenseItem : IIntellisenseItem
{
    /// <summary>
    /// Gets or sets a value indicating Icon to be displayed in the
    /// IntelliSenseListBox
    /// </summary>
    public ImageSource Icon
    {
        get;
        set;
    }
    /// <summary>
    /// Gets or sets a value indicating Text to be displayed in the
    /// IntelliSenseListBox
    /// </summary>
    public string Text
    {
        get;
```

```

set;
}
/// <summary>
/// Gets or sets a collection of sub-items to be displayed
/// </summary>
public IEnumerable<IIntellisenseItem> NestedItems
{
    get;
    set;
}
}

```

### Creating a custom collection of the business object

Now, create a Custom Collection of the Business Object and set as Custom ItemsSource using `IntellisenseCustomItemsSource` property by using the following code.

### C#

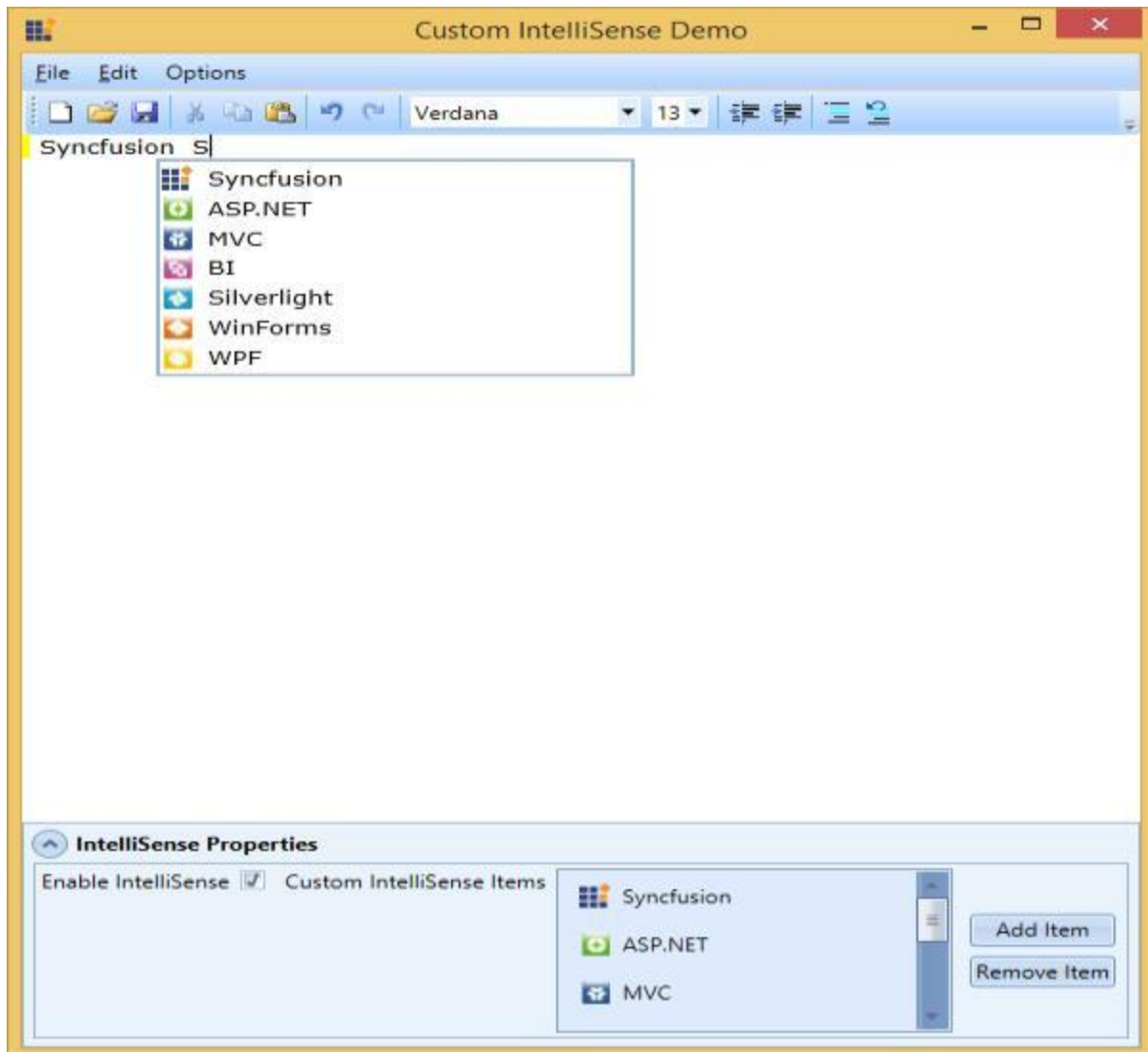
```

// Initializing custom business object collection
ObservableCollection<CustomIntelliSenseItem> customItems = new
ObservableCollection<CustomIntelliSenseItem>();
customItems.Add(new CustomIntelliSenseItem()
{
    Text = "Syncfusion",
    Icon = new BitmapImage(new
Uri("/CustomIntelliSenseDemo;component/Resources/syncfusion.png",
UriKind.Relative))
});
customItems.Add(new CustomIntelliSenseItem()
{
    Text = "Silverlight",
    Icon = new BitmapImage(new
Uri("/CustomIntelliSenseDemo;component/Resources/silverlight.png",
UriKind.Relative))
});
customItems.Add(new CustomIntelliSenseItem()
{
    Text = "WPF",
    Icon = new BitmapImage(new
Uri("/CustomIntelliSenseDemo;component/Resources/wpf.png",
UriKind.Relative))
});
// Applying custom business object collection as
IntelliSenseCustomItemsSource
EditControl1.IntellisenseCustomItemsSource = customItems;

```

When the code runs, the following output displays.

The following image displays the Custom Intellisense Demo Window.



### Customizing IntelliSense list box style

The EditControl enables the users to customize the look and feel of the IntelliSense listbox by applying custom style to the IntelliSense listbox. The `IntelliSenseBoxStyle` property of EditControl class can be used to apply custom style for IntelliSense listbox.

Customize the IntelliSense List Box Style, by using the following code.

### XML

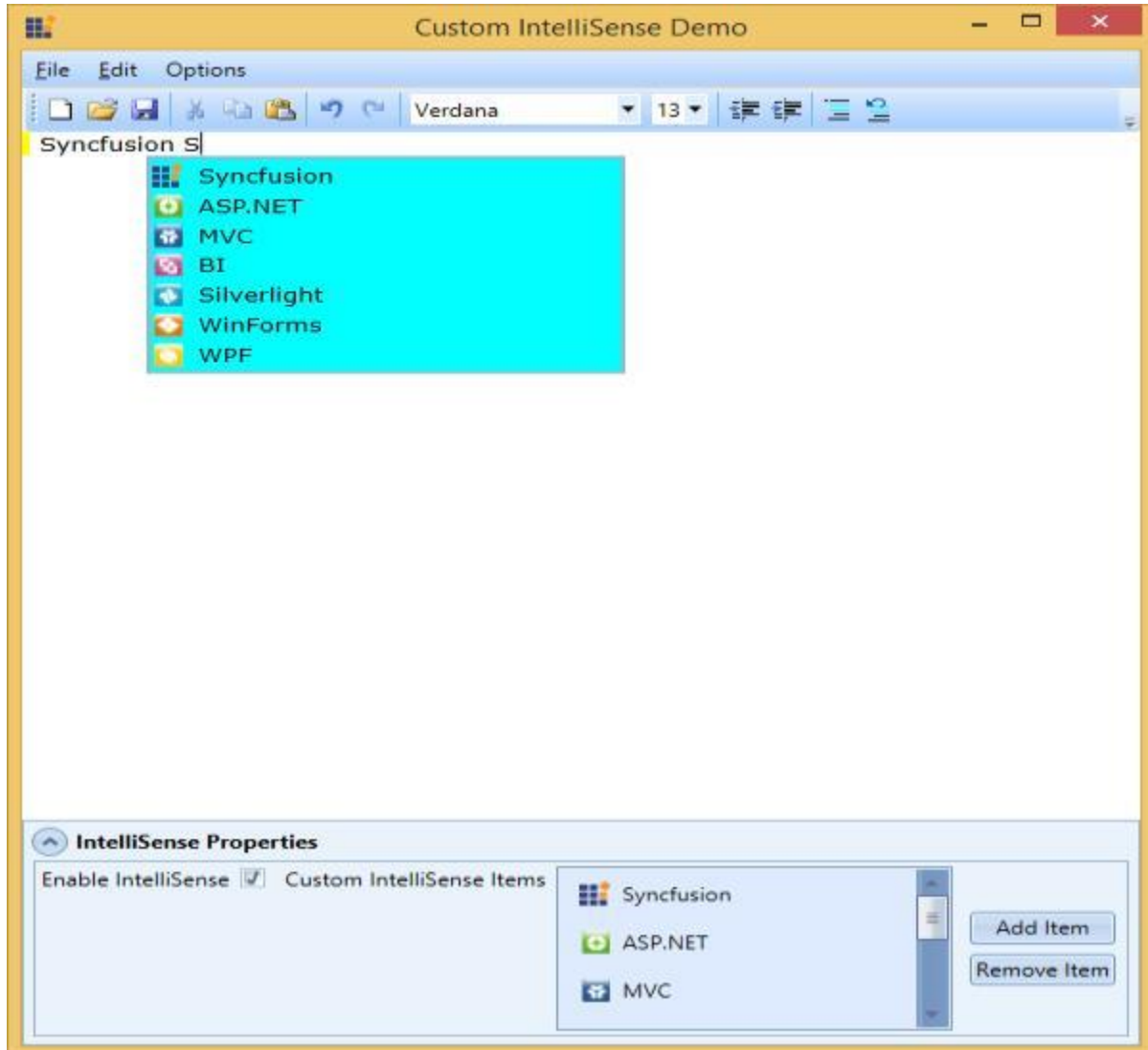
```
<Style x:Key="ListBoxItemStyle" TargetType="ListBoxItem">
  <Setter Property="FontSize" Value="11"/>
  <Setter Property="FontFamily" Value="Verdana"/>
  <Setter Property="FocusVisualStyle" Value="{x:Null}" />
  <Setter Property="syncfusion:SkinStorage.VisualStyle" Value="Default"/>
</Style>
<Style x:Key="ListBoxStyle" TargetType="ListBox">
  <Setter Property="ItemContainerStyle" Value="{StaticResource
ListBoxItemStyle}"/>
  <Setter Property="syncfusion:SkinStorage.VisualStyle" Value="Default"/>
  <Setter Property="Background" Value="Aqua"/>
</Style>
```

```

</Style>
<syncfusion:EditControl Name="EditControll1"
IntellisenseBoxStyle="{StaticResource ListBoxStyle}"/>

```

The following figure displays the window that appears after applying IntelliSenseBox Style.



#### Applying multi-level IntelliSense items in custom mode

As mentioned in earlier topics, EditControl supports applying custom collection of business objects as IntelliSense, when the business objects are implemented using `IIntelliSenseItem` interface. This `IIntelliSenseItem` interface has a `NestedItems` property which can be used to display sub items in IntelliSense. The EditControl has a property under `CurrentLanguage`, a `DrillDownChar` property of the type `char` to specify on which character press, the sub-items are to be displayed in IntelliSense. The default value of `DrillDownChar` is "." (Periods) and when "." (Periods) key is pressed, the EditControl will automatically get the collection from the `NestedItems` property and displays it in the EditControl.

Create a Custom IntelliSense for tables and fields for SQL language by using the following code.

#### XML

```

<!--Resources-->
<DataTemplate x:Key="CustomIntelliSenseItemTemplate">
<TextBlock Text="{Binding Text}" Margin="3"/>
</DataTemplate>
<!--Adding EditControl to application and setting its IntelliSenseMode to
Auto-->
<syncfusion:EditControl Background="White" Name="EditControl1"
DocumentLanguage="SQL"  IntellisenseMode="Custom"
IntellisenseItemTemplate="{StaticResource CustomIntelliSenseItemTemplate}"/>

```

**C#**

```

/// <summary>
/// Business object implemented from IIntelliSenseItem interface in
/// Syncfusion.Windows.Edit namespace
/// </summary>
public class CustomIntelliSenseItem : IIntellisenseItem
{
    /// <summary>
    /// Gets or sets a value indicating Icon to be displayed in the
    /// IntelliSenseListBox
    /// </summary>
    public ImageSource Icon
    {
        get;
        set;
    }
    /// <summary>
    /// Gets or sets a value indicating Text to be displayed in the
    /// IntelliSenseListBox
    /// </summary>
    public string Text
    {
        get;
        set;
    }
    /// <summary>
    /// Gets or sets a collection of sub-items to be displayed
    /// </summary>
    public IEnumerable<IIntellisenseItem> NestedItems
    {
        get;
        set;
    }
}

```

**C#**

```

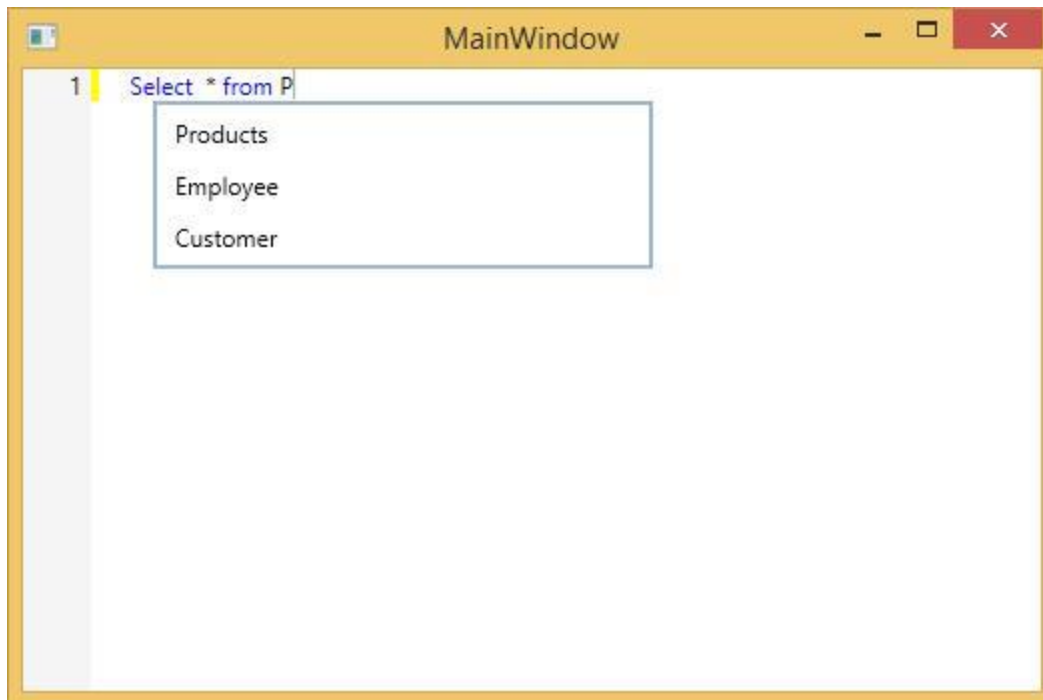
ObservableCollection<CustomIntelliSenseItem> customItems = new
ObservableCollection<CustomIntelliSenseItem>();
//Intializing sub-items for products
ObservableCollection<CustomIntelliSenseItem> productsSubItem = new
ObservableCollection<CustomIntelliSenseItem>();
productsSubItem.Add(new CustomIntelliSenseItem() { Text = "ID" });
productsSubItem.Add(new CustomIntelliSenseItem() { Text = "Name" });
productsSubItem.Add(new CustomIntelliSenseItem() { Text = "Manufacturer" });

```



```
productsSubItem.Add(new CustomIntelliSenseItem() { Text = "Price" });
productsSubItem.Add(new CustomIntelliSenseItem() { Text = "OrderQuantity"
});
productsSubItem.Add(new CustomIntelliSenseItem() { Text = "Units" });
//Intializing sub-items for employee
ObservableCollection<CustomIntelliSenseItem> employeeSubItem = new
ObservableCollection<CustomIntelliSenseItem>();
employeeSubItem.Add(new CustomIntelliSenseItem() { Text = "ID" });
employeeSubItem.Add(new CustomIntelliSenseItem() { Text = "Name" });
employeeSubItem.Add(new CustomIntelliSenseItem() { Text = "DOB" });
employeeSubItem.Add(new CustomIntelliSenseItem() { Text = "City" });
employeeSubItem.Add(new CustomIntelliSenseItem() { Text = "ContactNumber"
});
//Intializing sub-items for customer
ObservableCollection<CustomIntelliSenseItem> customerSubItem = new
ObservableCollection<CustomIntelliSenseItem>();
customerSubItem.Add(new CustomIntelliSenseItem() { Text = "ID" });
customerSubItem.Add(new CustomIntelliSenseItem() { Text = "Name" });
customerSubItem.Add(new CustomIntelliSenseItem() { Text = "City" });
customerSubItem.Add(new CustomIntelliSenseItem() { Text = "State" });
customerSubItem.Add(new CustomIntelliSenseItem() { Text = "Country" });
customerSubItem.Add(new CustomIntelliSenseItem() { Text = "ContactNumber"
});
//adding items to main collection
customItems.Add(new CustomIntelliSenseItem()
{
Text = "Products",
NestedItems =
productsSubItem
});
customItems.Add(new CustomIntelliSenseItem()
{
Text = "Employee",
NestedItems =
employeeSubItem
});
customItems.Add(new CustomIntelliSenseItem()
{
Text = "Customer",
NestedItems =
customerSubItem
});
// Applying custom business object collection as
IntelliSenseCustomItemsSource
EditControll1.IntellisenseCustomItemsSource = customItems;
```

The following screenshot illustrates IntelliSense displaying first-level of items from custom collection.



The following screenshot illustrates IntelliSense displaying sub-items from selected item.



Following classes in EditControl will be helpful to perform various operations related to IntelliSense.

Name of Property	Description	Type of Property	Value It Accepts
EnableIntelliSense	Gets or sets a value	Boolean	True/false

	indicating if IntelliSense Support has to be enabled or disabled.		
IntelliSenseMode	Gets or sets a value representing IntelliSense mode of operations.	IntelliSenseMode	IntellisenseMode.Auto/IntellisenseMode.Custom
IntelliSenseCustomItemsSource	To specify the custom collection of business objects to be displayed in custom IntelliSense.	IEnumerable	Any collection of items implemented using IIntellisenseItem
IntelliSenseBoxStyle	IntelliSense.	Style	Style object
IntelliSenseItemTemplate	IntelliSense.	DataTemplate	DataTemplate object

Following classes in LanguageBase will be helpful to perform various operations related to IntelliSense specific support and custom languages. These properties can be modified in custom language classes inherited from LanguageBase or ProceduralLanguageBase or MarkupLanguageBase class. It can also be modified using **CurrentLanguage** property of EditControl class.

Name of the property	Description	Type of the property
DrillDownChar	The character specified is used to drill down to nested items. When the user presses the corresponding key, IntelliSense drills down and displays the sub-items in the IntelliSense.	Char

IntelliSenseCommitCharacters	Specifies the characters when used. The IntelliSense should append the selected IntelliSense item to EditControl's text.	String
CommitsIntelliSenseItemOnSpaceBar	Specifies if the selected IntelliSense item to be appended to the EditControl's text when a space bar is pressed.	Boolean
SupportsIntelliSense	To allow or restrict IntelliSense auto mode in custom languages.	Boolean

## Events

Event	Usage	Handler Type	Handler
IntelliSenseBoxOpening	This event gets triggered before the IntelliSense pop-up is displayed.	IntelliSenseBoxEventHandler	Set IsCancel property in the EventArgs to cancel displaying of IntelliSense popup.
IntelliSenseDrillDown	This event gets triggered when a DrillDownChar specified is encountered.	IntelliSenseBoxEventHandler	Set IsCancel property in the EventArgs to cancel displaying of IntelliSense popup.

IntelliSenseBoxEventArgs contains following arguments.

Argument	Usage
Assemblies	Contains the list of assemblies included as AssemblyReferences.
Cancel	A boolean argument can be used to cancel the event.
CursorIndex	Contains current Cursor Index where the event is triggered.
ItemsSource	Contains the ItemsSource to be applied to the IntelliSense, it can also be changed if there are any Custom Filtrations to be done in the event.
LineIndex	Contains current Line Index where the event is triggered (0 based values).

## RoutedUICommands in WPF Syntax Editor

### EditCommands members

The EditControl ships with a variety of built-in RoutedUICommands to perform various editing operations in it using commands. The EditCommands class in the Edit WPF library contains lot of necessary commands. The following are the list of RoutedUICommands available in the EditControl.

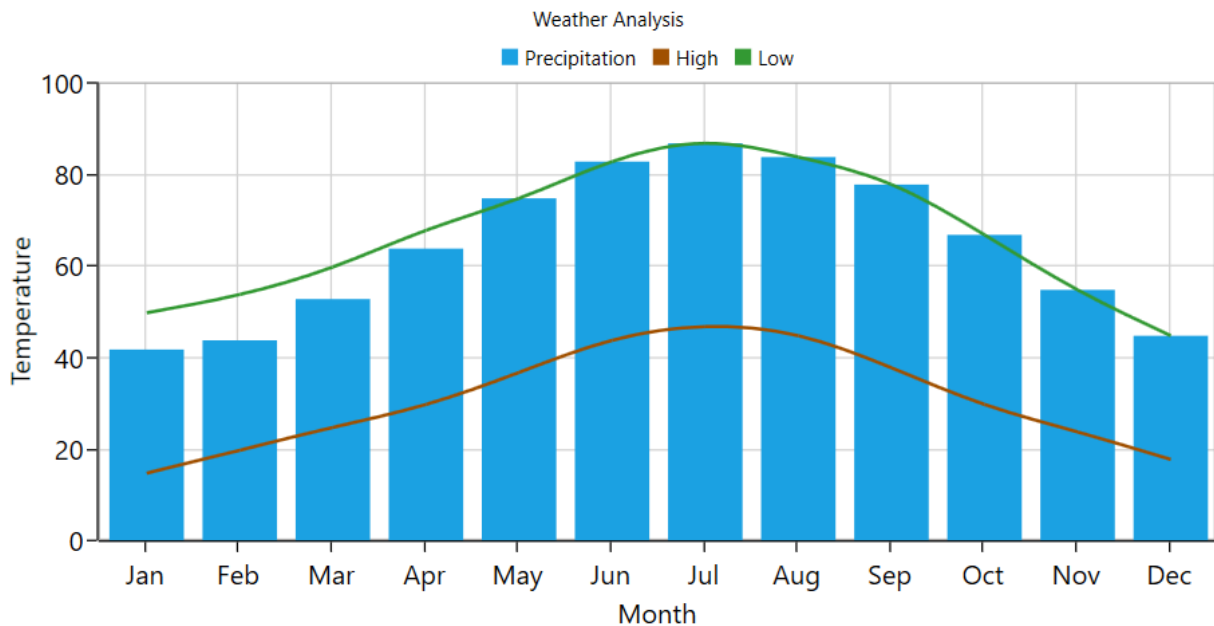
RoutedUICommand	Description
Backspace	Performs backspace key operations.
Collapse All	Collapses all the blocks in the EditControl text.
CommentSelection	Selects comments or individual lines based on the selected language.
Copy	Performs copying of text.
Cut	Cuts the text.
DecreaseIndent	Decreases the indent of individual or selected lines.
Delete	Performs delete key operation.
ExpandAll	Expands all the blocks in the EditControl text.
Find	Opens the find and replace window with the selected quick find tab.
FindAllReferences	Finds all the occurrences of the specified text.
IncreaseIndent	Increases the indent of individual or selected lines.
New	Resets all the text in the EditControl.
Open	Opens an existing file and displays the OpenFileDialog to select the file.
Paste	Paste the copied text to the clipboard.
Redo	Performs redo operations.
Replace	Replaces the text specified in the Find What field with the field in Find and Replace window.
Save	Saves the text in the EditControl to a file. It displays the SaveFileDialog to enter the name and location to save the file.

Search	Finds the text specified in the last search, without opening the Find and Replace window.
SearchInSelected	Finds the text specified in the last search within the selected text, without opening the Find and Replace window.
SelectAll	Selects all the text in EditControl.
ShowIntellisense	Displays the provided IntellisensePopup if the IntelliSense items are not empty.
UncommentSelection	Removes the comment Lexem from individual or selected lines.
Undo	Performs undo operations.

## SfChart

### WPF Charts (SfChart) Overview

SfChart provides a perfect way to visualize data with a high level of user interactivity that focus on development, productivity and simplicity of use. SfChart also provides a wide variety of charting features that can be used to visualize large quantities of data, flexibility of binding data and user customization.



### Key features

- SfChart supports 38 different types of [series](#), ranging from simple [bar series](#) to complex [financial charts](#). Each type of chart represents a unique style of representing data with more user friendly and greater UI visualization.
- Capable of rendering large amount of data within the few milliseconds (ms).
- Allows you to map data from the specified path, by achieving [data binding](#) concept.

- Interactive zooming[<https://help.syncfusion.com/wpf/charts/interactive-features/zoompan>] can be done with touch mode enabled that allows you to explore portions of large charts in more detail, with excellent performance.
- When you need more information about particular segment in a chart, a little mouse over on the series provides much more information by including [tooltip](#), [crosshair](#) and [track ball](#) behavior.
- Supports 10 different types of [technical indicators](#) that determine financial, stock or economic trends by analyzing a set of recorded data.
- Supports multiple axes that can be stacked and spanned for multiple panes.
- SfChart provides support for rendering multiple series at same time, with options to compare and visualize two different chart series simultaneously.
- User friendly and provides various options for you to customize chart features like [axis](#), [labels](#), [legends](#), [series](#) etc and visualize them accordingly.

---

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

---

### Getting Started with WPF Charts (SfChart)

This section explains you the steps required to populate the Chart with data, header, add data labels, legend and tooltips to the Chart. This section covers only the minimal features that you need to learn to get started with the Chart.

To get start quickly with [WPF Chart](#), you can check this video:

```
<style>#WPFChartVideoTutorial{width : 90% !important; height:400px !important }</style>
```

```
<iframe id='WPFChartVideoTutorial' src='https://www.youtube.com/embed/5b8nEevQPC8'></iframe>
```

### Adding chart reference

Refer to this [article](#) to learn how to add Syncfusion controls to Visual Studio projects in various ways. You can also refer to [this](#) link to learn about the assemblies required for adding Chart to your project.

### Initialize chart

Import the [WPF Chart](#) (SfChart) namespace as follows in your respective Window.

#### XML

```
xmlns:syncfusion="clr-namespace:Syncfusion.UI.Xaml.Charts;assembly=Syncfusion.SfChart.WPF"
```

#### C#

```
using Syncfusion.UI.Xaml.Charts;
```

#### VB.NET

```
Imports Syncfusion.UI.Xaml.Charts
```

Then initialize an empty chart with two axes as shown below,

#### XML

```
<syncfusion:SfChart>
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:CategoryAxis />
</syncfusion:SfChart.PrimaryAxis>
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis/>
</syncfusion:SfChart.SecondaryAxis>
</syncfusion:SfChart>
```

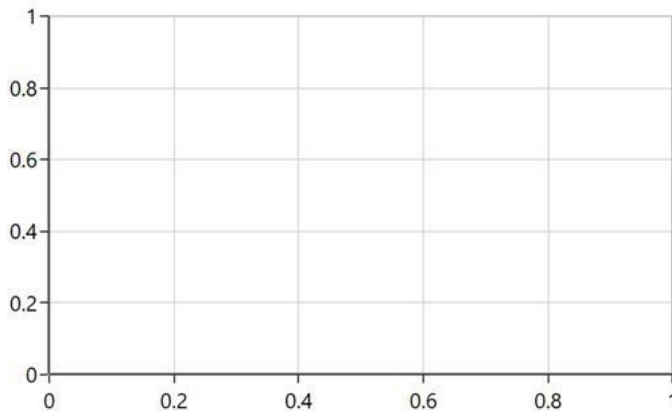
## C#

```
SfChart chart = new SfChart();
CategoryAxis primaryAxis = new CategoryAxis();
chart.PrimaryAxis = primaryAxis;
NumericalAxis secondaryAxis = new NumericalAxis();
chart.SecondaryAxis = secondaryAxis;
```

## VB.NET

```
Dim chart As New SfChart()
Dim primaryAxis As New CategoryAxis ()
chart.PrimaryAxis = primaryAxis
Dim secondaryAxis As New NumericalAxis ()
chart.SecondaryAxis = secondaryAxis
```

Run the project and check if you get following output to make sure you have configured your project properly to add [SfChart](#).



**Note:** [SfChart](#) supports default axes, so that these axes ([PrimaryAxis](#) and [SecondaryAxis](#)) will get generated automatically based upon the data bind to the chart, if you didn't specify the axes explicitly.

### Initialize view model

Now, let us define a simple data model that represents a data point in [WPF Chart](#) (SfChart).

## C#

```
public class Person
{
    public string Name { get; set; }
    public double Height { get; set; }
}
```



**VB.NET**

```
Public Class Person
Public Property Name As String
Public Property Height As Double
End Class
```

Next, create a view model class and initialize a list of **Person** objects as follows.

**C#**

```
public class ViewModel
{
public List<Person> Data { get; set; }
public ViewModel()
{
Data = new List<Person>()
{
new Person { Name = "David", Height = 180 },
new Person { Name = "Michael", Height = 170 },
new Person { Name = "Steve", Height = 160 },
new Person { Name = "Joel", Height = 182 }
};
}
}
```

**VB.NET**

```
Public Class ViewModel
Public Property Data As List(Of Person)
Public Sub New()
Data = New List(Of Person)() From
{
New Person With {.Name = "David", .Height = 180},
New Person With {.Name = "Michael", .Height = 170},
New Person With {.Name = "Steve", .Height = 160},
New Person With {.Name = "Joel", .Height = 182}
}
End Sub
End Class
```

Set the **ViewModel** instance as the **DataContext** of your window; this is done to bind properties of **ViewModel** to [SfChart](#).

**Note:** Add namespace of **ViewModel** class to your XAML window if you prefer to set **DataContext** in XAML.

**XML**

```
<Window x:Class=" ChartDemo.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
```

```
xmlns:local="clr-namespace:ChartDemo "
xmlns:syncfusion ="clr-
namespace:Syncfusion.UI.Xaml.Charts;assembly=Syncfusion.SfChart.WPF"
mc:Ignorable="d"
Title="MainWindow" Height="350" Width="525">
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
</Window>
```

**C#**

```
this.DataContext = new ViewModel();
```

**VB.NET**

```
Me.DataContext = New ViewModel()
```

## Populate chart with data

As we are going to visualize the comparison of heights in the data model, add [ColumnSeries](#) to [SfChart.Series](#) property, and then bind the [Data](#) property of the above [ViewModel](#) to the [ColumnSeries.ItemsSource](#) property as follows.

**Note:** You need to set [XBindingPath](#) and [YBindingPath](#) properties, so that [SfChart](#) would fetch values from the respective properties in the data model to plot the series.

**XML**

```
<syncfusion:SfChart>
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:CategoryAxis Header="Name" />
</syncfusion:SfChart.PrimaryAxis>
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis Header="Height(in cm)" />
</syncfusion:SfChart.SecondaryAxis>
<syncfusion:ColumnSeries ItemsSource="{Binding Data}" XBindingPath="Name"
YBindingPath="Height" >
</syncfusion:ColumnSeries>
</syncfusion:SfChart>
```

**C#**

```
SfChart chart = new SfChart();
//Adding horizontal axis to the chart
CategoryAxis primaryAxis = new CategoryAxis();
primaryAxis.Header = "Name";
chart.PrimaryAxis = primaryAxis;
//Adding vertical axis to the chart
NumericalAxis secondaryAxis = new NumericalAxis();
secondaryAxis.Header = "Height(in cm)";
chart.SecondaryAxis = secondaryAxis;
//Initialize the two series for SfChart
ColumnSeries series = new ColumnSeries();
series.ItemsSource = (new ViewModel()).Data;
```

```
series.XBindingPath = "Name";  
series.YBindingPath = "Height";  
//Adding Series to the Chart Series Collection  
chart.Series.Add(series);
```

### VB.NET

```
Dim chart As New SfChart()  
'Adding horizontal axis to the chart  
Dim primaryAxis As New CategoryAxis()  
primaryAxis.Header = "Name"  
chart.PrimaryAxis = primaryAxis  
'Adding vertical axis to the chart  
Dim secondaryAxis As New NumericalAxis()  
secondaryAxis.Header = "Height(in cm)"  
chart.SecondaryAxis = secondaryAxis  
'Initialize the two series for SfChart  
Dim series As New ColumnSeries()  
series.ItemsSource = New ViewModel().Demands  
series.XBindingPath = "Name"  
series.YBindingPath = "Height"  
'Adding Series to the Chart Series Collection  
chart.Series.Add(series)
```

**Note:** Syncfusion Chart also supports rendering combination of multiple series. Refer to [this](#) for details.

### Add title

The header of the chart acts as the title to provide quick information to the user about the data being plotted in the chart. You can set title using the [Header](#) property of chart as follows.

### XML

```
<Grid>  
<syncfusion:SfChart Header="Chart">  
</syncfusion:SfChart>  
</Grid>
```

### C#

```
chart.Header = "Chart";
```

### VB.NET

```
chart.Header = "Chart"
```

Refer to [this](#) link to learn more about the options available in [SfChart](#) to customize chart header.

### Enable data labels

You can add data labels to improve the readability of the chart and it can be enabled using [AdornmentInfo](#) property of [ChartSeries](#). By default, there is no label displayed, you have to set [ShowLabel](#) property of [ChartAdornmentInfo](#) to True.

### XML

```
<syncfusion:SfChart>
...
<syncfusion:ColumnSeries >
<syncfusion:ColumnSeries.AdornmentsInfo>
<syncfusion:ChartAdornmentInfo ShowLabel="True"/>
</syncfusion:ColumnSeries.AdornmentsInfo>
</syncfusion:ColumnSeries>
...
</syncfusion:SfChart>
```

### C#

```
series.AdornmentsInfo = new ChartAdornmentInfo () { ShowLabel = true };
```

### VB.NET

```
series.AdornmentsInfo = New ChartAdornmentInfo() With {.ShowLabel = True}
```

Refer to [this](#) link to learn more about the options available in [SfChart](#) to customize chart adornments.

### Enable legend

You can enable legend using the [SfChart.Legend](#) property as follows.

### XML

```
<syncfusion:SfChart>
...
<syncfusion:SfChart.Legend>
<syncfusion:ChartLegend/>
</syncfusion:SfChart.Legend>
...
</syncfusion:SfChart>
```

### C#

```
chart.Legend = new ChartLegend ();
```

### VB.NET

```
chart.Legend = New ChartLegend ()
```

Additionally, you need to set label for each series using the [Label](#) property of ChartSeries, which will be displayed in corresponding legend.

### XML

```
<syncfusion:SfChart>
...
<syncfusion:ColumnSeries Label="Heights" ItemsSource="{Binding Data}"
XBindingPath="Name" YBindingPath="Height" />
...
</syncfusion:SfChart>
```

## C#

```
ColumnSeries series = new ColumnSeries ();  
series.ItemsSource = (new ViewModel()).Data;  
series.XBindingPath = "Name";  
series.YBindingPath = "Height";  
series.Label = "Heights";
```

## VB.NET

```
Dim series As New ColumnSeries ()  
series.ItemsSource = New ViewModel().Data  
series.XBindingPath = "Name"  
series.YBindingPath = "Height"  
series.Label = "Heights"
```

Refer to this [link](#) to learn more about the options available in [SfChart](#) to customize legend.

## Enable tooltip

Tooltips are used to show information about the segment, when you click the segment. You can enable tooltip by setting series [ShowTooltip](#) property to true.

## XML

```
<syncfusion:SfChart>  
...  
<syncfusion:ColumnSeries ShowTooltip="True" ItemsSource="{Binding Data}"  
XBindingPath="Name" YBindingPath="Height"/>  
...  
</syncfusion:SfChart>
```

## C#

```
ColumnSeries series = new ColumnSeries();  
series.ItemsSource = (new ViewModel()).Data;  
series.XBindingPath = "Name";  
series.YBindingPath = "Height";  
series.ShowTooltip = true;
```

## VB.NET

```
Dim series As New ColumnSeries ()  
series.ItemsSource = New ViewModel().Data  
series.XBindingPath = "Name"  
series.YBindingPath = "Height"  
series.ShowTooltip = True
```

Refer to [this](#) link to learn more about the options available in [SfChart](#) to customize tooltip.

The following code example gives you the complete code of above configurations.

## XML

```
<Window x:Class="Sample_WPF.MainWindow"  
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:Sample_WPF"
xmlns:syncfusion="clr-
namespace:Syncfusion.UI.Xaml.Charts;assembly=Syncfusion.SfChart.WPF"
mc:Ignorable="d"
Title="MainWindow" Height="350" Width="525">
<!--Setting DataContext for SfChart-->
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<syncfusion:SfChart Header="Chart" Height="300" Width="500">
<!--Initialize the horizontal axis for SfChart-->
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:CategoryAxis Header="Name" FontSize="14"/>
</syncfusion:SfChart.PrimaryAxis>
<!--Initialize the vertical axis for SfChart-->
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis Header="Height(in cm)" FontSize="14"/>
</syncfusion:SfChart.SecondaryAxis>
<!--Adding Legend to the SfChart-->
<syncfusion:SfChart.Legend>
<syncfusion:ChartLegend/>
</syncfusion:SfChart.Legend>
<!--Initialize the series for SfChart-->
<syncfusion:ColumnSeries Label="Heights" ItemsSource="{Binding Data}"
XBindingPath="Name" YBindingPath="Height" ShowTooltip="True" >
<syncfusion:ColumnSeries.AdornmentsInfo>
<syncfusion:ChartAdornmentInfo ShowLabel="True" >
</syncfusion:ChartAdornmentInfo>
</syncfusion:ColumnSeries.AdornmentsInfo>
</syncfusion:ColumnSeries>
</syncfusion:SfChart>
</Grid>
</Window>

```

## C#

```

using Syncfusion.UI.Xaml.Charts;
namespace ChartDemo
{
public sealed partial class MainWindow : Window
{
public MainWindow()
{
InitializeComponent();
SfChart chart = new SfChart() { Header = "Chart", Height = 300, Width = 500
};
//Adding horizontal axis to the chart
CategoryAxis primaryAxis = new CategoryAxis();
primaryAxis.Header = "Name";
primaryAxis.FontSize = 14;
chart.PrimaryAxis = primaryAxis;
//Adding vertical axis to the chart

```

```

NumericalAxis secondaryAxis = new NumericalAxis();
secondaryAxis.Header = "Height(in cm)";
secondaryAxis.FontSize = 14;
chart.SecondaryAxis = secondaryAxis;
//Adding Legends for the chart
ChartLegend legend = new ChartLegend();
chart.Legend = legend;
//Initializing column series
ColumnSeries series = new ColumnSeries();
series.ItemsSource = (new ViewModel()).Data;
series.XBindingPath = "Name";
series.YBindingPath = "Height";
series.ShowTooltip = true;
series.Label = "Heights";
//Setting adornment to the chart series
series.AdornmentsInfo = new ChartAdornmentInfo() { ShowLabel = true };
//Adding Series to the Chart Series Collection
chart.Series.Add(series);
this.Content = chart;
}
}
}

```

## VB.NET

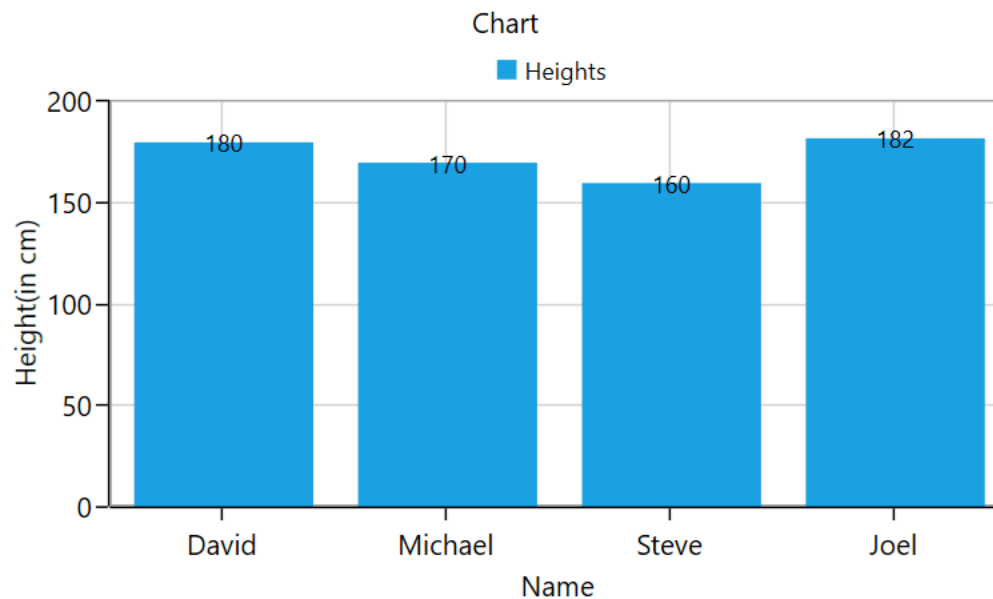
```

Imports Syncfusion.UI.Xaml.Charts
Partial Public Class MainWindow
Inherits Window
Public Sub New()
InitializeComponent()
Dim chart As New SfChart()
chart.Header = "Chart"
chart.Height = 300
chart.Width = 500
'Adding horizontal axis to the chart
Dim primaryAxis As New CategoryAxis()
primaryAxis.Header = "Name"
primaryAxis.FontSize = 14
chart.PrimaryAxis = primaryAxis
'Adding vertical axis to the chart
Dim secondaryAxis As New NumericalAxis()
secondaryAxis.Header = "Height(in cm)"
secondaryAxis.FontSize = 14
chart.SecondaryAxis = secondaryAxis
'Adding Legends for the chart
Dim legend As New ChartLegend()
chart.Legend = legend
'Initializing column series
Dim series As New ColumnSeries()
series.ItemsSource = New ViewModel().Data
series.XBindingPath = "Name"
series.YBindingPath = "Height"
series.Label = "Heights"
series.ShowTooltip = True
'Setting adornment to the chart series
series.AdornmentsInfo = New ChartAdornmentInfo() With {.ShowLabel = True}

```

```
'Adding Series to the Chart Series Collection
chart.Series.Add(series)
Me.Content = chart
End Sub
End Class
```

The following chart is created as a result of the previous codes.



You can find the complete getting started sample from this [link](#).

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configure with built-in support for creating stunning visual effects.

See also

[How to create chart in VB .NET WPF](#)

[How to create a chart control in WPF application using XAML](#)

[How to create chart in C# WPF](#)

[How to create Chart in F# WPF](#)

[How to add Context Menu for WPF Chart](#)

[How to add WPF SfChart inside the SfDataGrid](#)

[How to add footer content to WPF Chart](#)

[How to access data from Meta Trader to chart using TradePlatform.Net in WPF](#)

[How to synchronize the selection between the chart and data grid](#)

[How to generate dynamic number of series based on common items source](#)

[How to display the chart area alone in Chart](#)



[How to redraw the chart while dragging the series out of the range](#)

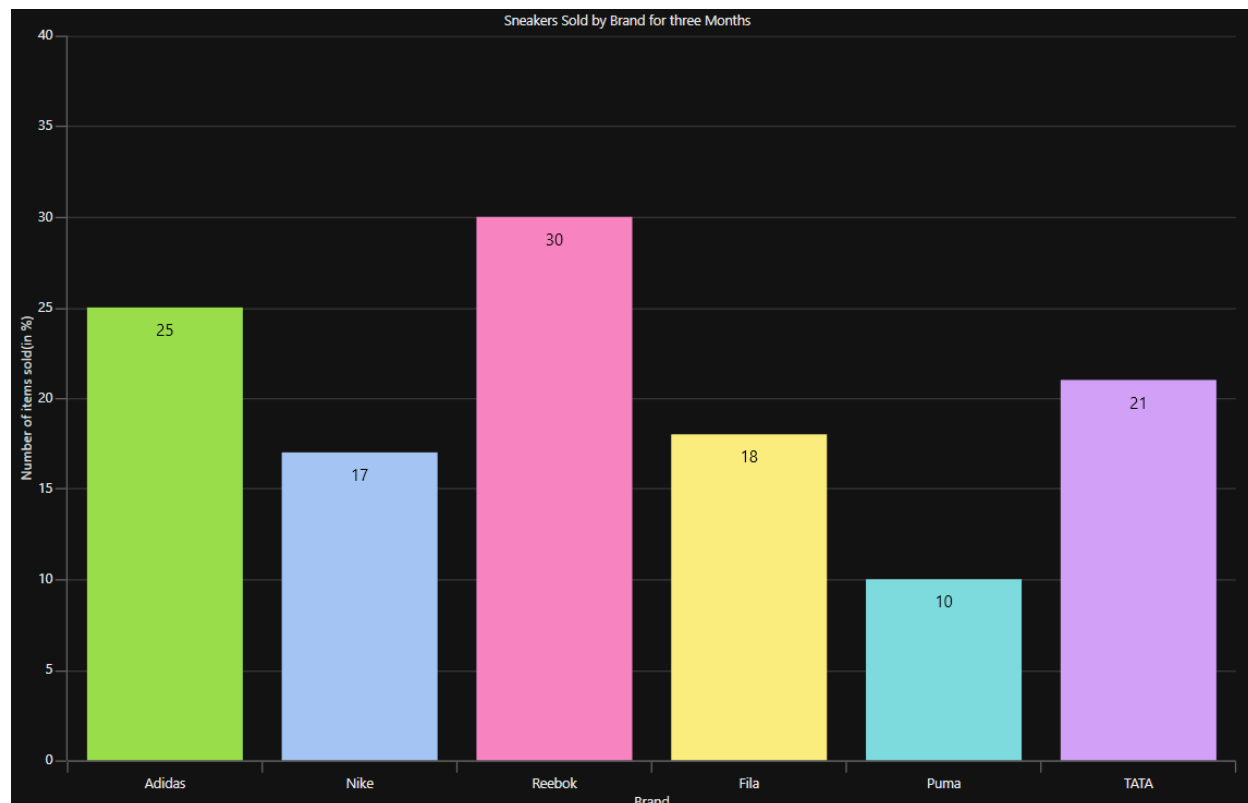
[How to create a real time Chart using MVVM in WPF](#)

[How to add watermark to chart](#)

Theme

SfChart supports various built-in themes. Refer to the below links to apply themes for the SfChart,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Area in WPF Charts (SfChart)

Chart area represents the entire chart and all its elements. It's a virtual rectangular area that includes all the chart elements like axis, legends, series, etc.

The following are the major properties of chart(SfChart):

- [PrimaryAxis](#) – Gets or sets the horizontal x axis for the chart.
- [SecondaryAxis](#) – Gets or sets the vertical y axis for the chart.
- [Legend](#) – Gets or sets the legend for the chart.
- [Series](#) – Gets or sets the list of series in the chart.
- [TechnicalIndicators](#) – Gets or sets the various financial indicators for the chart.
- [Behaviors](#) – Used to add one more interactive features to the chart.

### Customization

SfChart provides the properties like [AreaBorderBrush](#), [AreaBorderThickness](#), [AreaBackground](#) and [Background](#) for customizing the plot area.

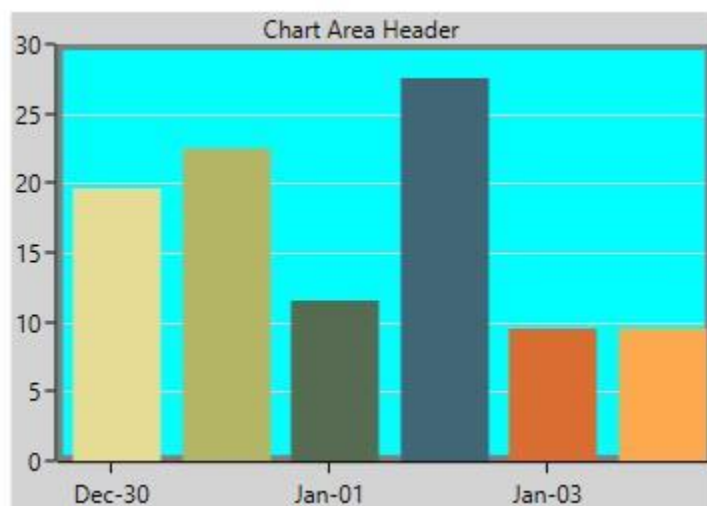
The following code examples illustrates the usage of these properties:

#### XML

```
<chart:SfChart Height="250" Width="350"
Header="Chart Area Header"
AreaBackground="Cyan"
Background="LightGray"
AreaBorderBrush="Gray"
AreaBorderThickness="3" >
```

#### C#

```
SfChart chart = new SfChart();
chart.Header = "Chart Area Header";
chart.AreaBackground = new SolidColorBrush(Colors.Cyan);
chart.Background = new SolidColorBrush(Colors.LightGray);
chart.AreaBorderBrush = new SolidColorBrush(Colors.Gray);
chart.AreaBorderThickness = new Thickness(3);
```



### Multiple Area

You can split plot area into multiple rows and columns using [ChartRowDefinition](#) and [ChartColumnDefinition](#) like Grid panel's row and column definition.

The following code example demonstrates, how you can create multiple panes in the chart area:

#### XML

```
<chart:SfChart >
<!--Adding row definition to the chart-->
<chart:SfChart.RowDefinitions>
<chart:ChartRowDefinition/>
<chart:ChartRowDefinition/>
</chart:SfChart.RowDefinitions>
```

```

<!--Adding column definition to the chart-->
<chart:SfChart.ColumnDefinitions>
<chart:ChartColumnDefinition/>
<chart:ChartColumnDefinition/>
</chart:SfChart.ColumnDefinitions>
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis chart:ChartBase.ColumnSpan="2"/>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis PlotOffset="13" chart:ChartBase.ColumnSpan="2" />
</chart:SfChart.SecondaryAxis>
<chart:ColumnSeries Palette="LightCandy"
ItemsSource="{Binding SneakersDetail}"
XBindingPath="Brand"
YBindingPath="ItemsCount1" />
<chart:ColumnSeries Palette="Metro"
ItemsSource="{Binding SneakersDetail}"
XBindingPath="Brand"
YBindingPath="ItemsCount" >
<chart:ColumnSeries.YAxis>
<chart:NumericalAxis PlotOffset="10"
chart:SfChart.Row="1" >
</chart:NumericalAxis>
</chart:ColumnSeries.YAxis>
</chart:ColumnSeries>
</chart:SfChart>

```

## C#

```

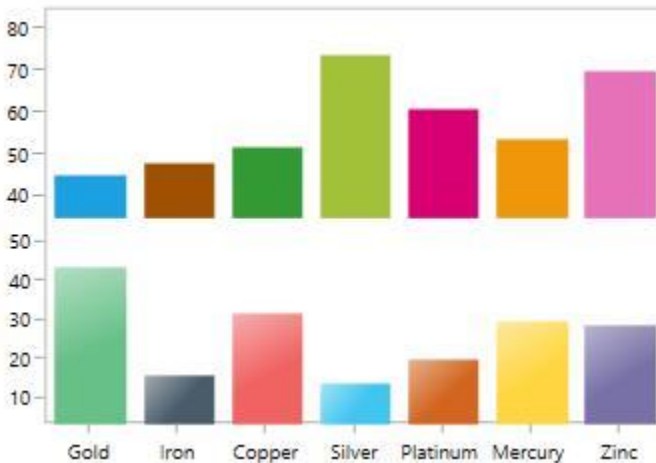
SfChart chart = new SfChart();
chart.RowDefinitions.Add(new ChartRowDefinition());
chart.RowDefinitions.Add(new ChartRowDefinition());
chart.ColumnDefinitions.Add(new ChartColumnDefinition());
chart.ColumnDefinitions.Add(new ChartColumnDefinition());
CategoryAxis xAxis = new CategoryAxis();
ChartBase.SetColumnSpan(xAxis, 2);
chart.PrimaryAxis = xAxis;
NumericalAxis yAxis = new NumericalAxis();
yAxis.PlotOffset = 13;
ChartBase.SetColumnSpan(yAxis, 2);
chart.SecondaryAxis = yAxis;
ColumnSeries columnSeries1 = new ColumnSeries()
{
    ItemsSource = new ViewModel().SneakersDetail,
    XBindingPath = "Brand",
    YBindingPath = "ItemsCount",
    Palette = ChartColorPalette.LightCandy,
};
NumericalAxis axis = new NumericalAxis();
axis.PlotOffset = 10;
SfChart.SetRow(axis, 1);
ColumnSeries columnSeries2 = new ColumnSeries()
{
    ItemsSource = new ViewModel().SneakersDetail,
    XBindingPath = "Brand",
    YBindingPath = "ItemsCount1",
};

```

```

Palette = ChartColorPalette.Metro,
YAxis = axis
};
chart.Series.Add(columnSeries1);
chart.Series.Add(columnSeries2);

```



### Column Span and Row Span

These can be used to specify the number of column or rows up to which the axis can extend. Same like Grid's RowSpan or ColumnSpan property, it is also an attached property.

You can set the row span in chart like the following code example.

### XML

```

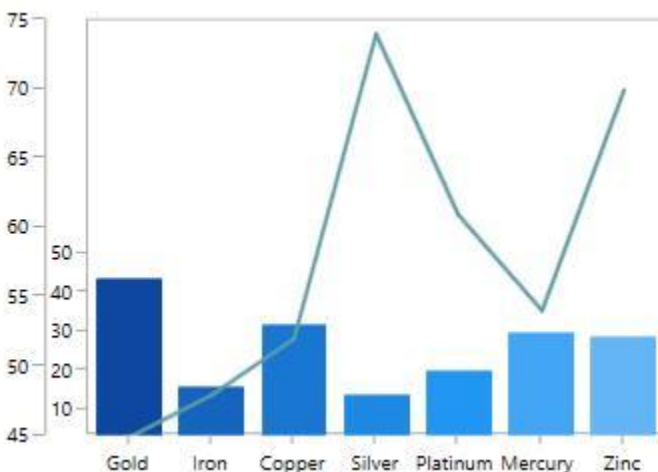
<chart:SfChart>
  <!--Adding row definition to the chart-->
  <chart:SfChart.RowDefinitions>
    <chart:ChartRowDefinition/>
    <chart:ChartRowDefinition/>
  </chart:SfChart.RowDefinitions>
  <chart:SfChart.PrimaryAxis>
    <chart:CategoryAxis chart:ChartBase.RowSpan="2"
      chart:SfChart.Row="0"
      TickLinesPosition="Outside">
    </chart:CategoryAxis>
  </chart:SfChart.PrimaryAxis>
  <chart:SfChart.SecondaryAxis>
    <chart:NumericalAxis />
  </chart:SfChart.SecondaryAxis>
  <chart:ColumnSeries XBindingPath="Brand"
    ItemsSource="{Binding SneakersDetail}"
    YBindingPath="ItemsCount1"
  />
  <chart:LineSeries Interior="CadetBlue" XBindingPath="Brand"
    ItemsSource="{Binding SneakersDetail}"
    YBindingPath="ItemsCount">
    <chart:LineSeries.YAxis>
      <chart:NumericalAxis chart:ChartBase.RowSpan="2">
    </chart:NumericalAxis>
    </chart:LineSeries.YAxis>
  </chart:LineSeries>

```

```
</chart:LineSeries>
</chart:SfChart>
```

**C#**

```
SfChart chart = new SfChart();
chart.RowDefinitions.Add(new ChartRowDefinition());
chart.RowDefinitions.Add(new ChartRowDefinition());
chart.ColumnDefinitions.Add(new ChartColumnDefinition());
chart.ColumnDefinitions.Add(new ChartColumnDefinition());
CategoryAxis xAxis = new CategoryAxis();
xAxis.TickLinesPosition = AxisElementPosition.Outside;
ChartBase.SetColumnSpan(xAxis, 2);
SfChart.SetRow(xAxis, 0);
chart.PrimaryAxis = xAxis;
NumericalAxis yAxis = new NumericalAxis();
ChartBase.SetColumnSpan(yAxis, 2);
chart.SecondaryAxis = yAxis;
ColumnSeries columnSeries1 = new ColumnSeries()
{
    ItemsSource = new ViewModel().SneakersDetail,
    XBindingPath = "Brand",
    YBindingPath = "ItemsCount1",
};
NumericalAxis axis = new NumericalAxis();
axis.PlotOffset = 10;
ChartBase.SetColumnSpan(axis, 2);
LineSeries lineSeries = new LineSeries()
{
    ItemsSource = new ViewModel().SneakersDetail,
    XBindingPath = "Brand",
    YBindingPath = "ItemsCount",
    Interior = new SolidColorBrush(Colors.CadetBlue),
    YAxis = axis
};
chart.Series.Add(columnSeries1);
chart.Series.Add(lineSeries);
```



Clone or copy the chart

More like serialization, you can use [Clone](#) method for SfChart control state persistence. This method creates a copy of the chart instance.

#### C#

```
var chartCopy = chart.Clone() as SfChart;  
grid.Children.Add(chartCopy as SfChart);  
//Here, 'grid' is an empty container in the application to hold the chart.
```

**Tips:** You can use this method for copy and paste like requirement, by cloning chart upon copy and reload while pasting.

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

#### See also

[PointToValue](#)

[ValueToPoint](#)

[How to add space between multiple columns in a Chart segment](#)

[How to allot space between the segments](#)

[How to display the chart area alone in Chart](#)

[How to redraw the chart while dragging the series out of the range](#)

[How to set the Z Index to the series](#)

[How to get coordinates of x and y in MouseDown event](#)

[How to set ItemWidthPercent before drawing chart](#)

[How to view corner segments without cutting in edge of WPF Chart](#)

Header in WPF Charts (SfChart)

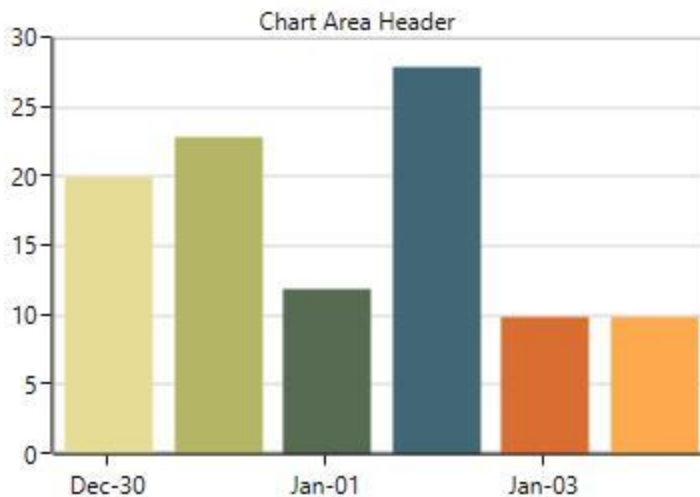
[Header](#) property is used to define the title for the chart. This allows you to add any object (.Net object) as content for chart title.

#### XML

```
<syncfusion:SfChart Header="Chart Area Header" />
```

#### C#

```
SfChart chart = new SfChart();  
chart.Header = "Usage of Metals";
```



Header can be positioned left or right side of the chart using [HorizontalHeaderAlignment](#) property.

Also you can add more customization for the header as below:

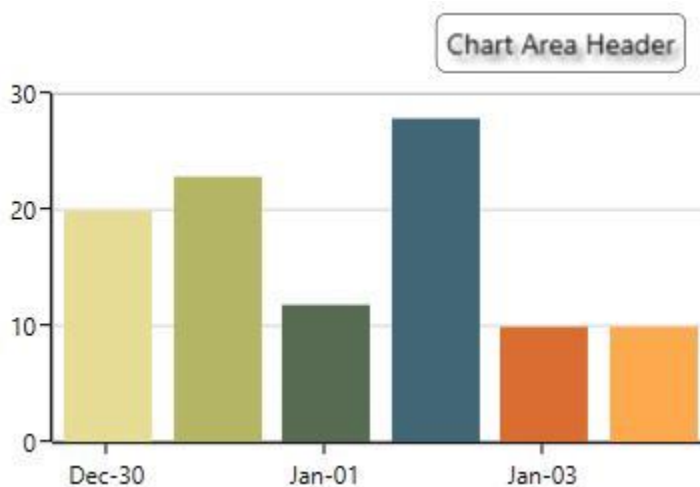
#### XML

```
<chart:SfChart.Header>
  <Border BorderThickness="0.5" BorderBrush="Black" Margin="10"
    CornerRadius="5">
    <TextBlock FontSize="14" Text="Chart Area Header" Margin="5">
      <TextBlock.Effect>
        <DropShadowEffect Color="Black"
          Opacity="0.5" />
      </TextBlock.Effect>
    </TextBlock>
  </Border>
</chart:SfChart.Header>
```

#### C#

```
SfChart chart = new SfChart();
Border border = new Border()
{
    BorderThickness = new Thickness(0.5),
    BorderBrush = new SolidColorBrush(Colors.Black),
    Margin = new Thickness(10),
    CornerRadius = new CornerRadius(5)
};
TextBlock textBlock = new TextBlock()
{
    Text = "Chart Area Header",
    Margin = new Thickness(5),
    FontSize = 14
};
textBlock.Effect = new DropShadowEffect()
{
    Color = Colors.Black,
    Opacity = 0.5
};
```

```
border.Child = textBlock;
chart.Header = border;
```



**Note:** Here, HorizontalHeaderAlignment is set as 'Right'.

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configure with built-in support for creating stunning visual effects.

### Axis in WPF Charts (SfChart)

[ChartAxis](#) is used to locate a data point inside the chart area. Charts typically have two axes that are used to measure and categorize data: a vertical (Y) axis, and a horizontal (X) axis.

Vertical(Y) axis always uses numerical scale. Horizontal(X) axis supports the following types of scale:

- Category
- Numeric
- Date time
- Logarithmic Axis

The following are the API's in ChartAxis

- [ArrangeRect](#) – Represents the bounds of chart axis size.
- [VisibleRange](#) – Represents the axis start and end visible values as follows.

<a href="#">Start</a>	Gets the start value of the visible range.
<a href="#">End</a>	Gets the end value of the visible range.
<a href="#">Delta</a>	Gets the delta value of the visible range.
<a href="#">Empty</a>	Gets the empty value of the visible range.
<a href="#">IsEmpty</a>	Gets a value indicating whether the visible range is empty or not.



<a href="#">Median</a>	Gets the median value of the visible range.
------------------------	---

- [VisibleLabels](#) – Represents the axis label collection which are visible in axis.

The following topics explain in detail about the axis and its parts

### Header

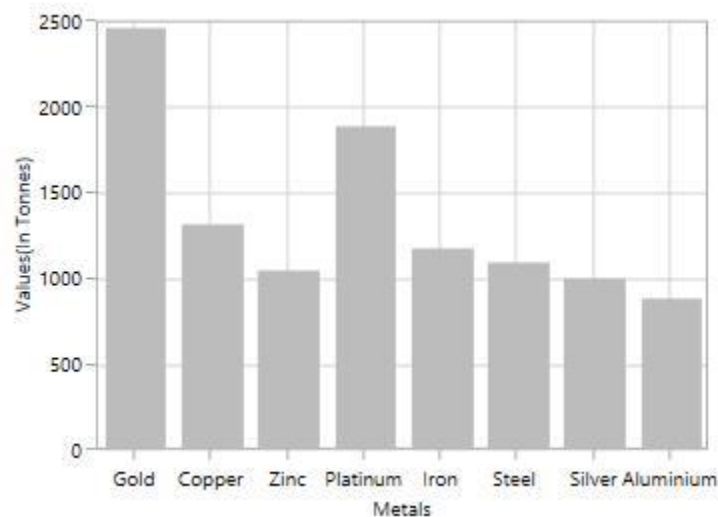
In [ChartAxis](#) you can define any object as header using [Header](#) property. The following code example demonstrates the defining header in primary and secondary axis.

### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:CategoryAxis Header="Metals"/>
</syncfusion:SfChart.PrimaryAxis>
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis Header="Values(In Tonnes)"/>
</syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis() { Header = "Medals" };
chart.SecondaryAxis = new NumericalAxis() { Header = "Values(In Tonnes)" };
```



### Header Customization

Default appearance of the header can be customized using [HeaderTemplate](#) property. The following code snippet demonstrates the header customization.

### XML

```
<syncfusion:SfChart x:Name="chart">
<syncfusion:SfChart.Resources>
<DataTemplate x:Key="headerTemplate1">
```

```

<Border BorderBrush="Black" CornerRadius="5" BorderThickness="1">
  <TextBlock Text="Demands" FontSize="12"
    FontStyle="Italic"
    FontWeight="Bold" Margin="3"/>
</Border>
</DataTemplate>
<DataTemplate x:Key="headerTemplate2">
  <Border BorderBrush="Black" CornerRadius="5" BorderThickness="1">
    <TextBlock FontSize="12" Margin="3"
      FontStyle="Italic" FontWeight="Bold"/>
  </Border>
</DataTemplate>
</syncfusion:SfChart.Resources>
<syncfusion:SfChart.PrimaryAxis>
  <syncfusion:CategoryAxis HeaderTemplate="{StaticResource headerTemplate1}"/>
</syncfusion:SfChart.PrimaryAxis>
<syncfusion:SfChart.SecondaryAxis>
  <syncfusion:NumericalAxis Header="Values(In Tonnes)"
    HeaderTemplate="{StaticResource headerTemplate2}"/>
</syncfusion:SfChart.SecondaryAxis>
</syncfusion:SfChart>

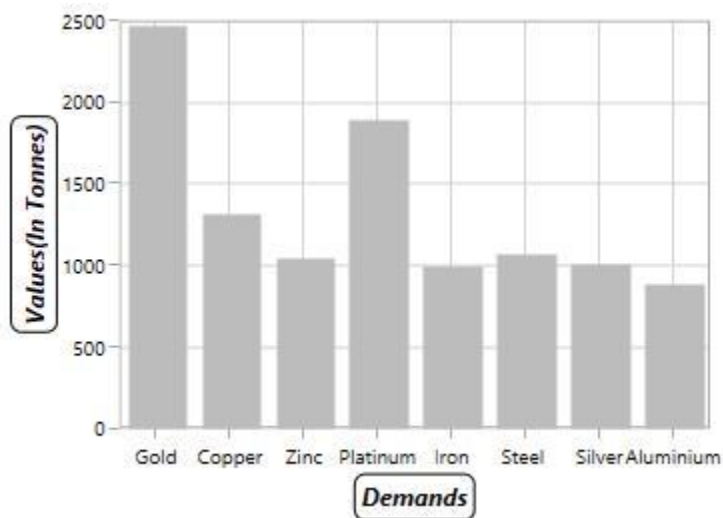
```

## C#

```

chart.PrimaryAxis = new CategoryAxis()
{
    HeaderTemplate = chart.Resources["headerTemplate1"] as DataTemplate
};
chart.SecondaryAxis = new NumericalAxis()
{
    HeaderTemplate = chart.Resources["headerTemplate2"] as DataTemplate
};

```



## HeaderStyle

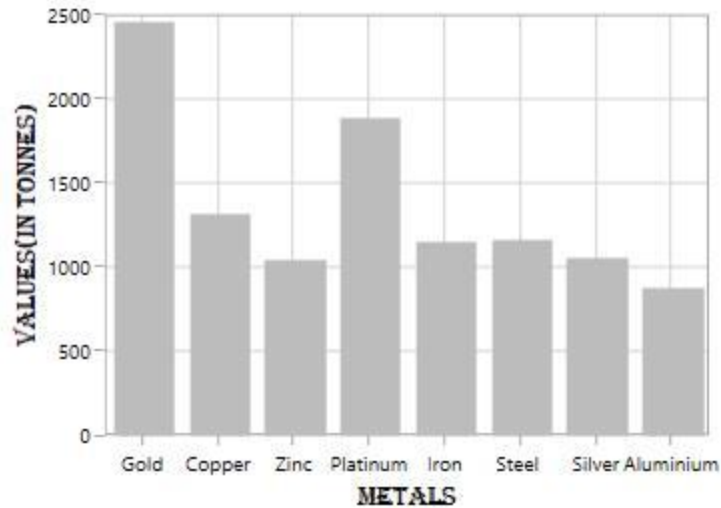
[HeaderStyle](#) property is used to provide style for the axis header. The following code example explains header style customization.

## XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:CategoryAxis Header="Metals" >
<syncfusion:CategoryAxis.HeaderStyle>
<syncfusion:LabelStyle FontFamily="Algerian" FontSize="13"
Foreground="Black">
</syncfusion:LabelStyle>
</syncfusion:CategoryAxis.HeaderStyle>
</syncfusion:CategoryAxis>
</syncfusion:SfChart.PrimaryAxis>
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis Header="Values(In Tonnes)"/>
<syncfusion:NumericalAxis.HeaderStyle>
<syncfusion:LabelStyle FontFamily="Algerian" FontSize="13"
Foreground="Black">
</syncfusion:LabelStyle>
</syncfusion:NumericalAxis.HeaderStyle>
</syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>
```

## C#

```
LabelStyle style = new LabelStyle()
{
    FontFamily = new FontFamily("Algerian"),
    FontSize = 13,
    Foreground = new SolidColorBrush(Colors.Black)
};
chart.PrimaryAxis = new CategoryAxis()
{
    Header = "Medals",
    LabelStyle = style
};
chart.SecondaryAxis = new NumericalAxis()
{
    Header = "Values(In Tonnes)",
    LabelStyle = style
};
```



### Axis Labels

Labels will be generated by the range and the values binded to [XBindingPath](#) or [YBindingPath](#) properties.

### Positioning the Labels

The [LabelsPosition](#) property is used to position the axis label either inside or outside the chart plotting area. By default, LabelsPosition is [Outside](#).

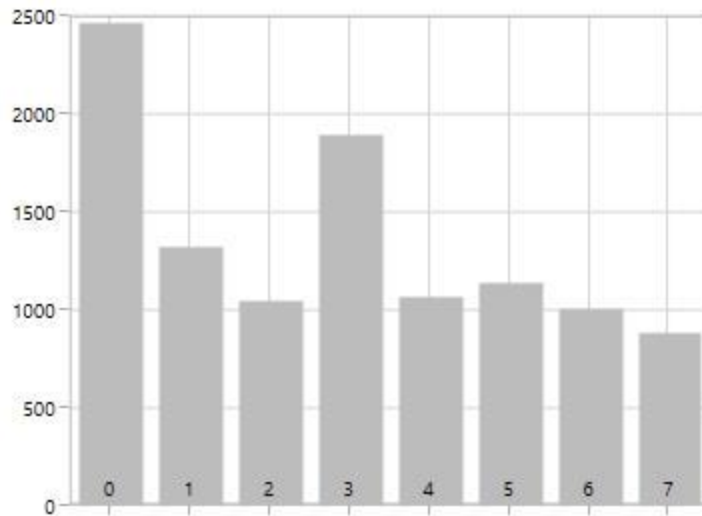
#### Inside

#### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:NumericalAxis LabelsPosition="Inside">
</syncfusion:NumericalAxis>
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new NumericalAxis()
{
    LabelsPosition = AxisElementPosition.Inside
};
```



### LabelRotationAngle

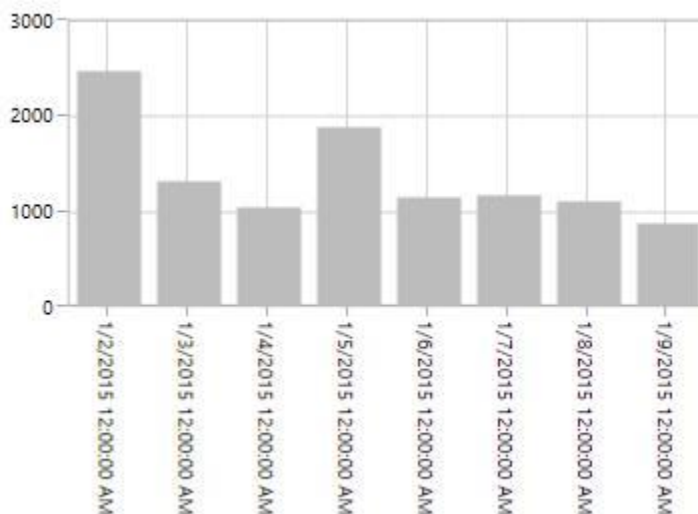
[LabelRotationAngle](#) property allows you to define the angle for the label content. The following code example illustrates the [LabelRotationAngle](#).

#### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:DateTimeAxis LabelRotationAngle="90" >
</syncfusion:DateTimeAxis>
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new DateTimeAxis()
{
    LabelRotationAngle = 90
};
```



## Custom Labels

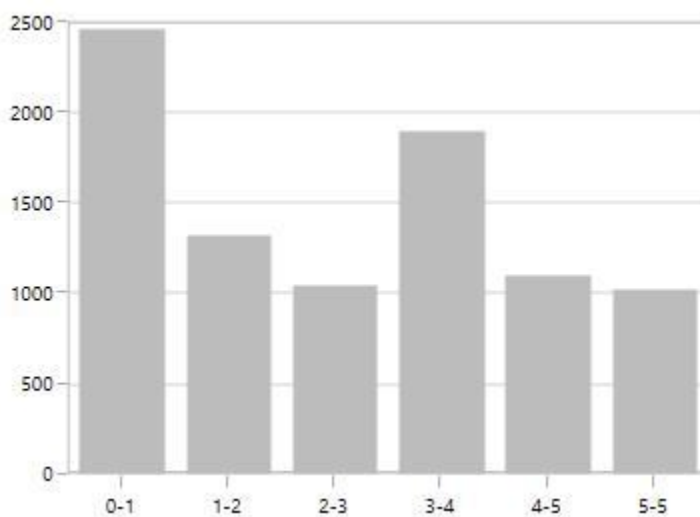
SfChart allows user to define the labels for the axis. For defining the axis label you have to set the [LabelContent](#) and [Position](#) property. You can define the labels using [CustomLabels](#) property as in the below code snippet.

### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:CategoryAxis >
<syncfusion:CategoryAxis.CustomLabels>
<syncfusion:ChartAxisLabel Position="0" LabelContent="0-1"/>
<syncfusion:ChartAxisLabel Position="1" LabelContent="1-2"/>
<syncfusion:ChartAxisLabel Position="2" LabelContent="2-3"/>
<syncfusion:ChartAxisLabel Position="3" LabelContent="3-4"/>
<syncfusion:ChartAxisLabel Position="4" LabelContent="4-5"/>
<syncfusion:ChartAxisLabel Position="5" LabelContent="5-5"/>
</syncfusion:CategoryAxis.CustomLabels>
</syncfusion:CategoryAxis>
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
CategoryAxis axis = new CategoryAxis();
axis.CustomLabels.Add(new ChartAxisLabel() { Position = 0, LabelContent =
"0-1" });
axis.CustomLabels.Add(new ChartAxisLabel() { Position = 1, LabelContent =
"1-2" });
axis.CustomLabels.Add(new ChartAxisLabel() { Position = 2, LabelContent =
"2-3" });
axis.CustomLabels.Add(new ChartAxisLabel() { Position = 3, LabelContent =
"3-4" });
axis.CustomLabels.Add(new ChartAxisLabel() { Position = 4, LabelContent =
"4-5" });
axis.CustomLabels.Add(new ChartAxisLabel() { Position = 5, LabelContent =
"5-5" });
chart.PrimaryAxis = axis;
```



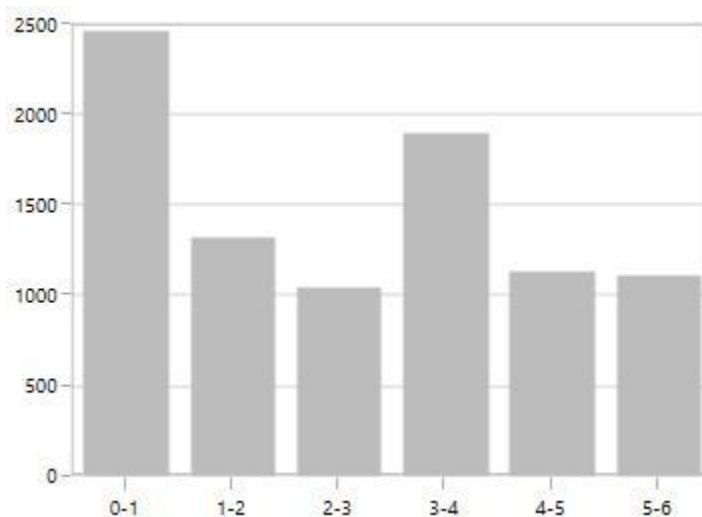
You can also directly bind the collection of labels to the [LabelsSource](#) property for defining custom labels. The following code example demonstrates the defining the label collection in code behind and binding the property in XAML page.

### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:CategoryAxis LabelsSource="{Binding Labels}"
ContentPath="Content" PositionPath="Position">
</syncfusion:CategoryAxis>
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    ContentPath = "Content",
    PositionPath = "Position",
    LabelsSource = Labels
};
public List<LabelItem> Labels { get; set; }
Labels = new List<LabelItem>
{
    new LabelItem() { Position=0, Content = "0-1"},
    new LabelItem() { Position=1, Content = "1-2"},
    new LabelItem() { Position=2, Content = "2-3"},
    new LabelItem() { Position=3, Content = "3-4"},
    new LabelItem() { Position=4, Content = "4-5"},
    new LabelItem() { Position=5, Content = "5-6"},
    new LabelItem() { Position=6, Content = "6-7"},
    new LabelItem() { Position=7, Content = "7-8"},
};
public class LabelItem
{
    public string Content { get; set; }
    public int Position { get; set; }
}
```



## Label Formatting

Axis labels can be formatted by predefined formatting types based on the axis types.

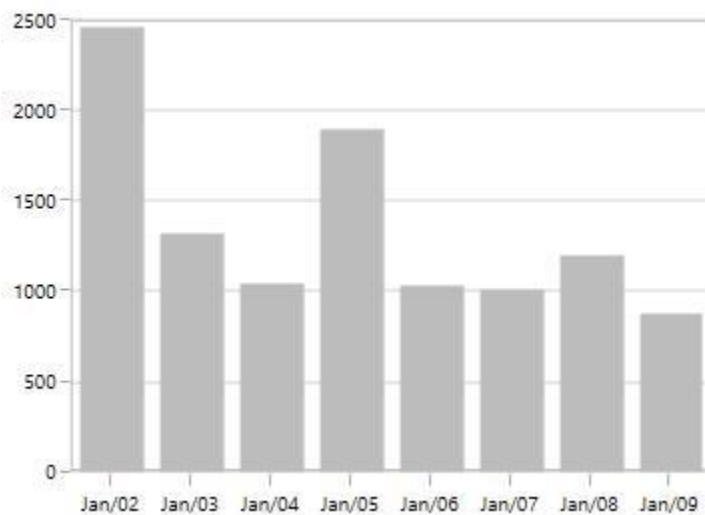
### DateTimeAxis

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis LabelFormat="MMM/dd" FontSize="12" >  
</syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    LabelFormat = "MM/dd",  
    FontSize = 12  
};
```



### TimeSpanAxis

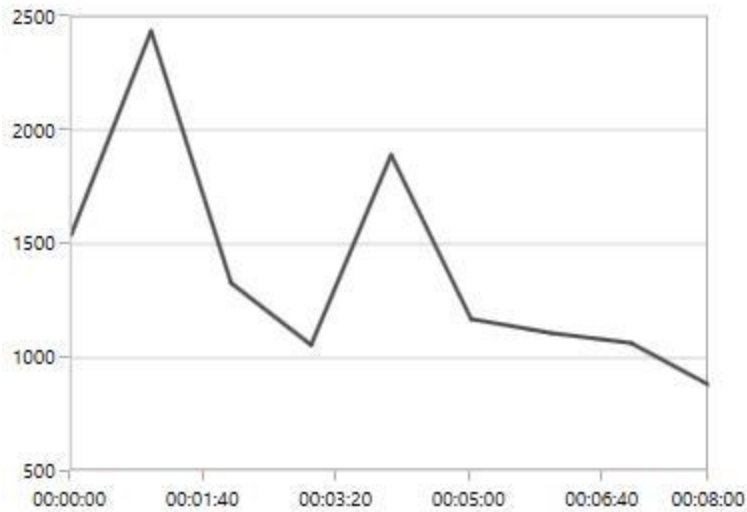
#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:TimeSpanAxis LabelFormat="g" ></syncfusion:TimeSpanAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new TimeSpanAxis()  
{  
    LabelFormat = "g",  
};
```





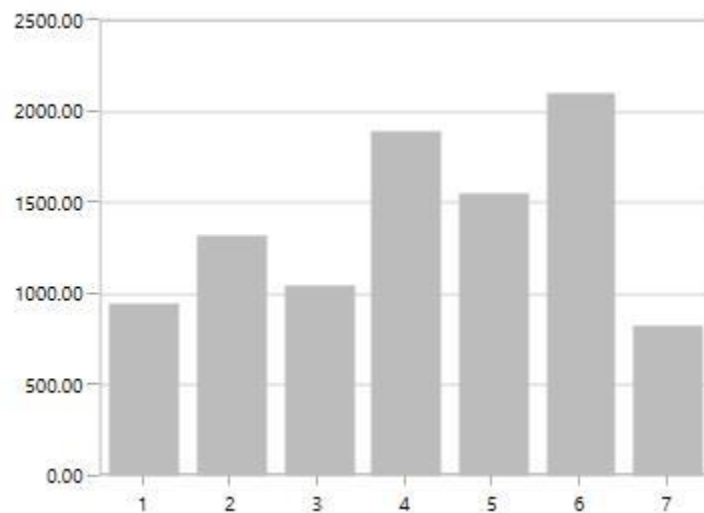
## NumericalAxis

### XML

```
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis LabelFormat="0.00" />
</syncfusion:SfChart.SecondaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()
{
    LabelFormat = "0.00",
};
```



## Adding Units to Labels

To display the measuring units, it can be added as a prefix or suffix to the axis labels. This can be achieved using the [PrefixLabelTemplate](#) and [PostfixLabelTemplate](#) properties.

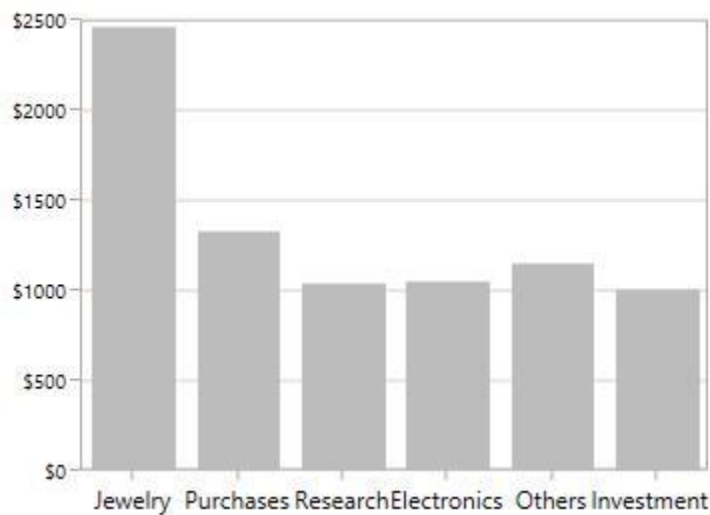
## PrefixLabelTemplate

### XML

```
<syncfusion:SfChart x:Name="chart">
  <syncfusion:SfChart.Resources>
    <DataTemplate x:Key="prefixLabelTemplate">
      <TextBlock FontSize="10" VerticalAlignment="Center"
        Text="$" />
    </DataTemplate>
  </syncfusion:SfChart.Resources>
  <syncfusion:SfChart.SecondaryAxis>
    <syncfusion:NumericalAxis FontSize="10"
      PrefixLabelTemplate="{StaticResource prefixLabelTemplate}" />
  </syncfusion:SfChart.SecondaryAxis>
</syncfusion:SfChart>
```

### C#

```
chart.SecondaryAxis = new NumericalAxis()
{
    PrefixLabelTemplate = chart.Resources["prefixLabelTemplate"] as DataTemplate
};
```



## PostfixLabelTemplate

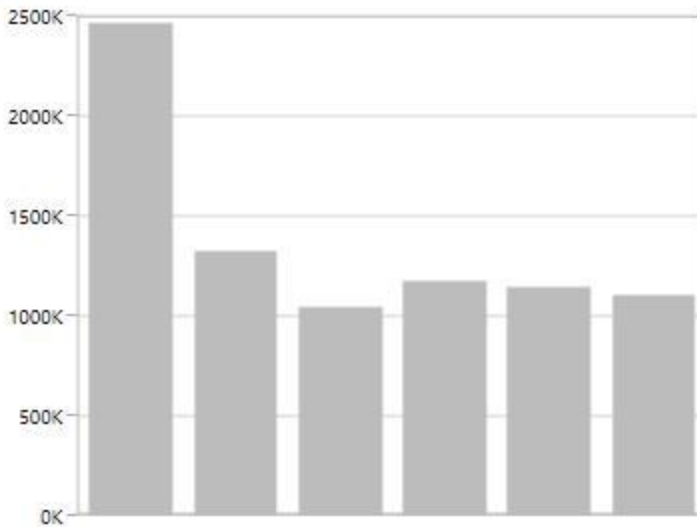
### XML

```
<syncfusion:SfChart x:Name="chart">
  <syncfusion:SfChart.Resources>
    <DataTemplate x:Key="postfixLabelTemplate">
      <TextBlock FontSize="10" VerticalAlignment="Center" Text="K" />
    </DataTemplate>
  </syncfusion:SfChart.Resources>
  <syncfusion:SfChart.SecondaryAxis>
    <syncfusion:NumericalAxis FontSize="10"
      PostfixLabelTemplate="{StaticResource postfixLabelTemplate}" />
  </syncfusion:SfChart.SecondaryAxis>
```

```
</syncfusion:SfChart>
```

## C#

```
chart.SecondaryAxis = new NumericalAxis()
{
    PostfixLabelTemplate = chart.Resources["postfixLabelTemplate"] as
    DataTemplate
};
```



## LabelTemplate

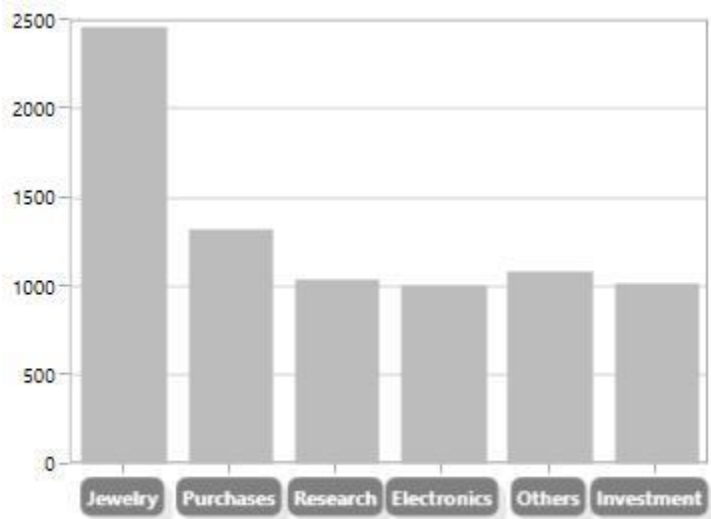
[LabelTemplate](#) property allows you to define the appearance for the axis labels .the following code example illustrates the [LabelTemplate](#) property.

## XML

```
<syncfusion:SfChart x:Name="chart">
  <syncfusion:SfChart.Resources>
    <DataTemplate x:Key="labelTemplate">
      <Border Background="Gray" CornerRadius="5" >
        <TextBlock Text="{Binding LabelContent}" Foreground="White"
          FontStyle="Normal" FontSize="10"
          FontWeight="Bold" Margin="3"/>
      <Border.Effect>
        <DropShadowEffect ShadowDepth="6" Direction="315"
          Color="LightGray" Opacity="0.25"
          BlurRadius="0.8" />
      </Border.Effect>
    </Border>
  </DataTemplate>
</syncfusion:SfChart.Resources>
<syncfusion:SfChart.PrimaryAxis>
  <syncfusion:CategoryAxis LabelTemplate="{StaticResource labelTemplate}"/>
</syncfusion:SfChart.PrimaryAxis>
</syncfusion:SfChart>
```

**C#**

```
chart.PrimaryAxis = new CategoryAxis()  
{  
    LabelTemplate = chart.Resources["labelTemplate"] as DataTemplate  
};
```

**LabelExtent**

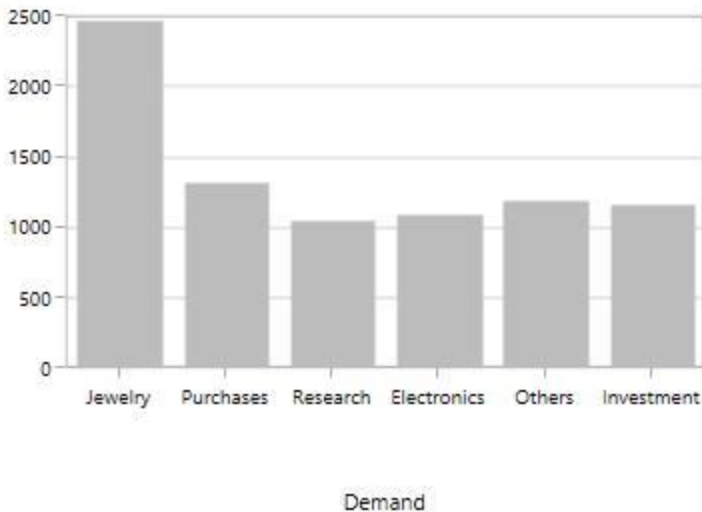
This property allows us to set the distance between the axis header and the axis using [LabelExtent](#) property. The following code snippet defines the [LabelExtent](#) property.

**XML**

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:CategoryAxis Header="Demand" LabelExtent="50" >  
</syncfusion:CategoryAxis>
```

**C#**

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    Header = "Demand",  
    LabelExtent = 50  
};
```



### Smart Axis Labels

When there are more number of axis labels, they may overlap with each other. SfChart provides support to handle the overlapping labels using the [LabelsIntersectAction](#) property. By default the [LabelsIntersectAction](#) value is [Hide](#).

The following are the options for intersecting action.

- None
- Hide
- MultipleRows
- Rotate

### None

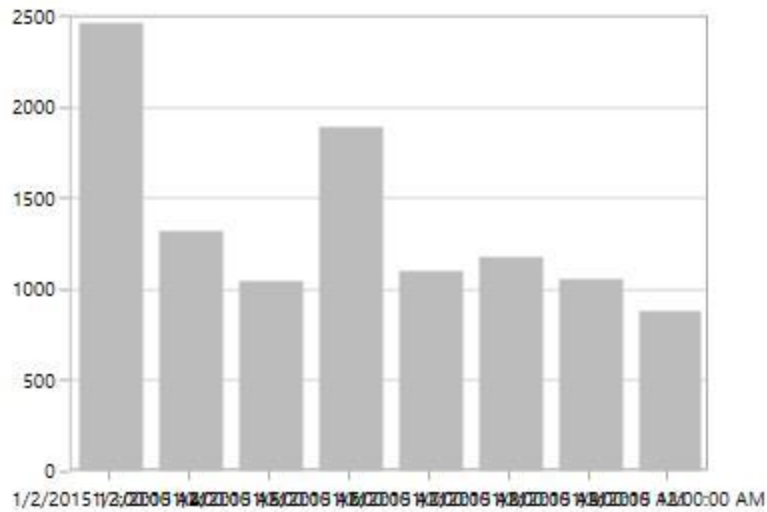
None option is used to display all the label even if it intersects. The following code snippet demonstrates the LabelsIntersectAction as None option.

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:CategoryAxis LabelsIntersectAction="None"/>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new CategoryAxis()  
{  
    LabelsIntersectAction = AxisLabelsIntersectAction.None  
};
```



### Hide

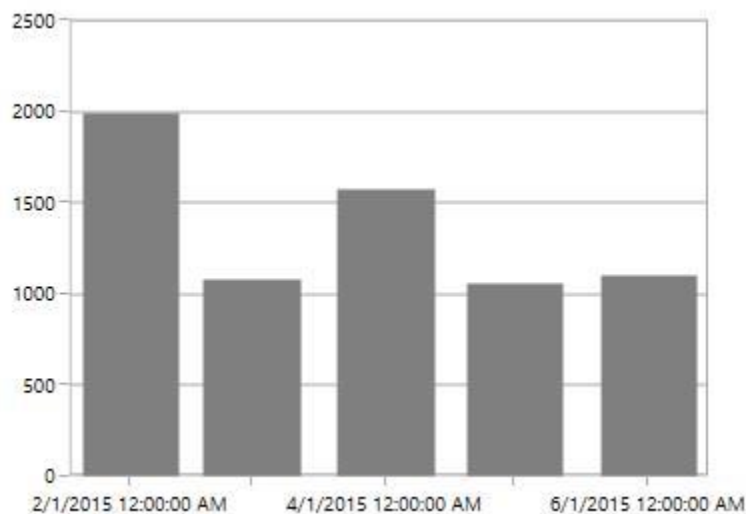
Hide option is used to hide the labels if it intersects .You can define the hide as shown in the below code snippet.

### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:DateTimeAxis LabelsIntersectAction="Hide"/>
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new DateTimeAxis()
{
    LabelsIntersectAction = AxisLabelsIntersectAction.Hide
};
```



### MultipleRows

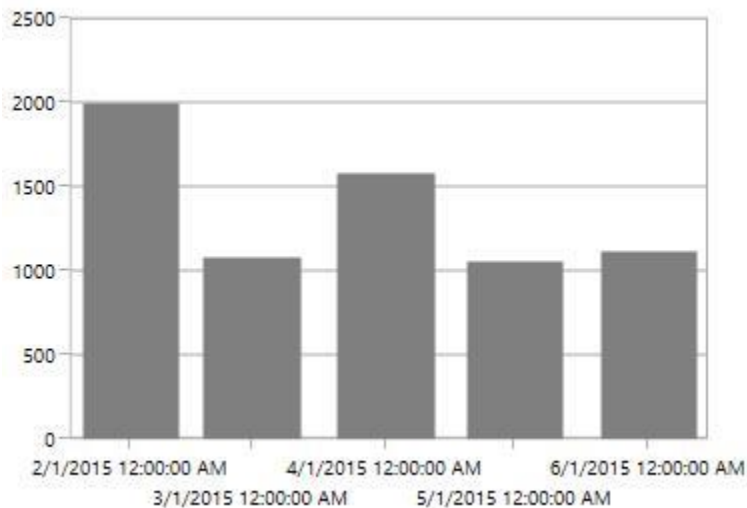
This option is used to move the labels to next row if it intersects. The following code demonstrates the `MultipleRows` option in [LabelsIntersectAction](#).

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis LabelsIntersectAction="MultipleRows">  
</syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    LabelsIntersectAction = AxisLabelsIntersectAction.MultipleRows  
};
```



#### Auto

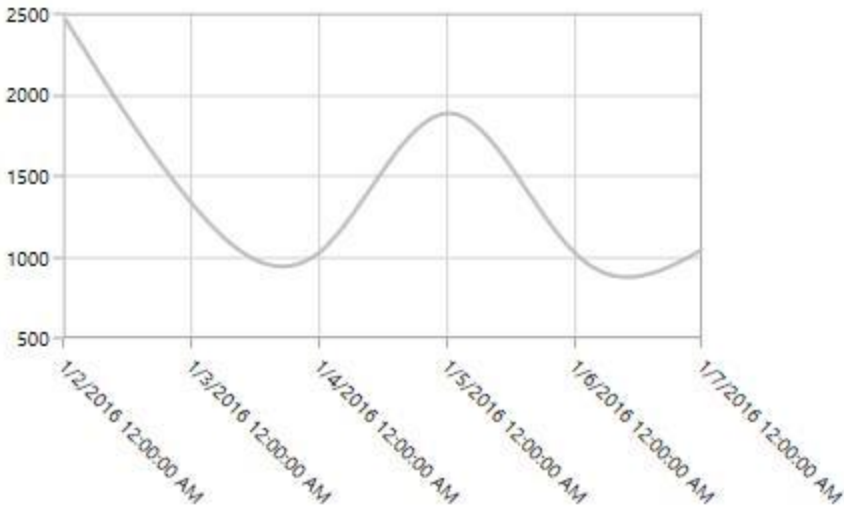
This option in [LabelsIntersectAction](#) property is used to rotate the labels if it intersects. The following code snippet and image demonstrates the rotate option in [LabelsIntersectAction](#) property.

#### XML

```
<syncfusion:CategoryAxis LabelsIntersectAction="Auto">  
</syncfusion:CategoryAxis>
```

#### C#

```
chart.PrimaryAxis = new CategoryAxis()  
{  
    LabelsIntersectAction = AxisLabelsIntersectAction.Auto  
};
```



### EdgeLabelsDrawingMode

SfChart provides support to customize the position of the edge labels in axis using the [EdgeLabelsDrawingMode](#) property. [EdgeLabelsDrawingMode](#) property default value is [Center](#).

The following are the customizing options in [EdgeLabelsDrawingMode](#).

- Center- Positions the label with tick line as center.
- Fit- Position the gridline inside based on the edge label size.
- Hide- Hides the edge labels.
- Shift- Shifts the edge labels to the left or right so that it comes inside the chart area.

### Center

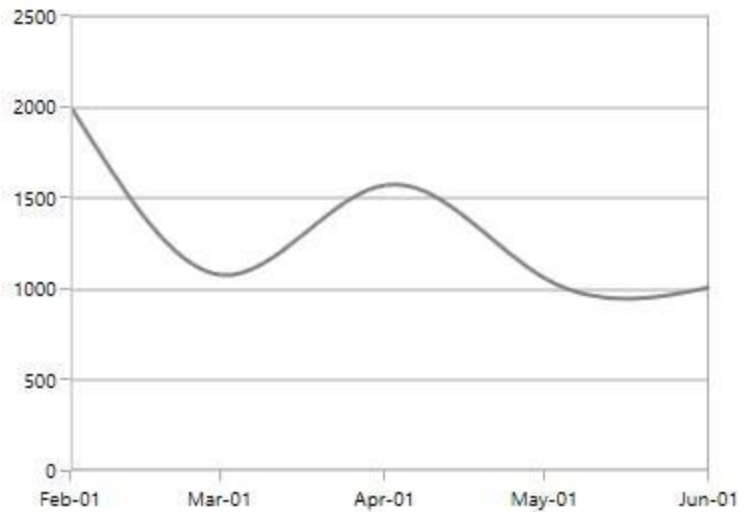
#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis EdgeLabelsDrawingMode="Center" >  
</syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    EdgeLabelsDrawingMode = EdgeLabelsDrawingMode.Center  
};
```





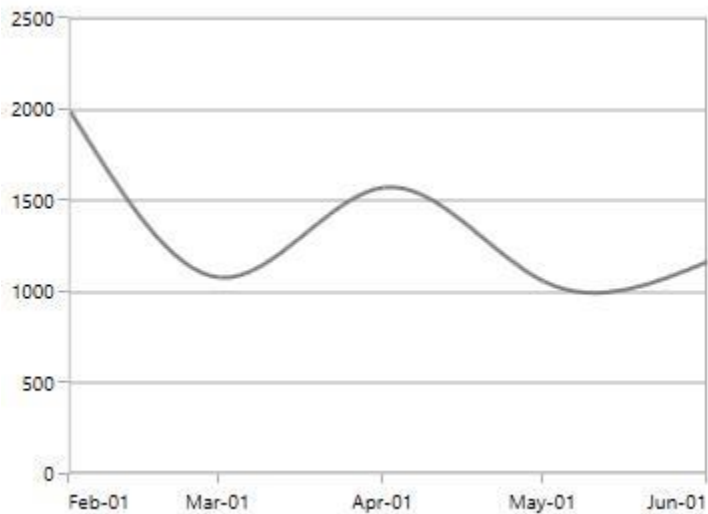
### Shift

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis EdgeLabelsDrawingMode="Shift"  
></syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    EdgeLabelsDrawingMode = EdgeLabelsDrawingMode.Shift  
};
```



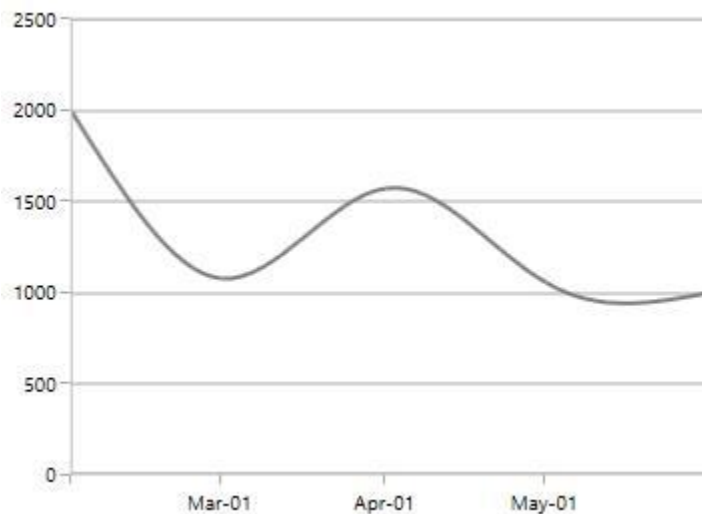
### Hide

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis EdgeLabelsDrawingMode="Hide">  
</syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    EdgeLabelsDrawingMode = EdgeLabelsDrawingMode.Hide  
};
```



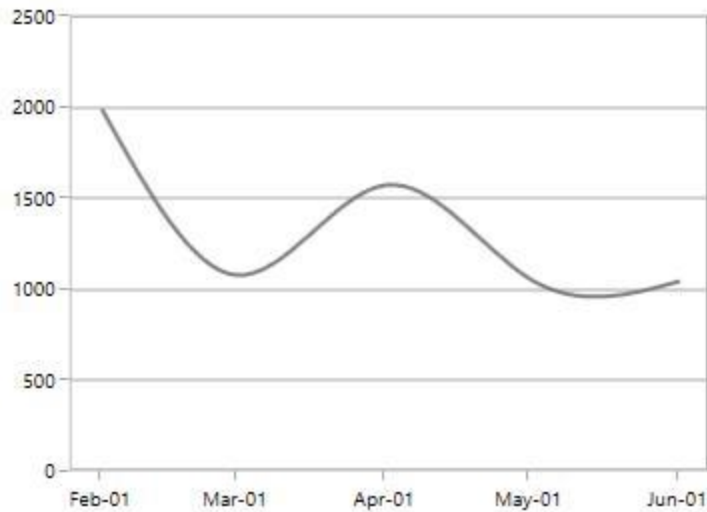
### Fit

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis EdgeLabelsDrawingMode="Fit">  
</syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

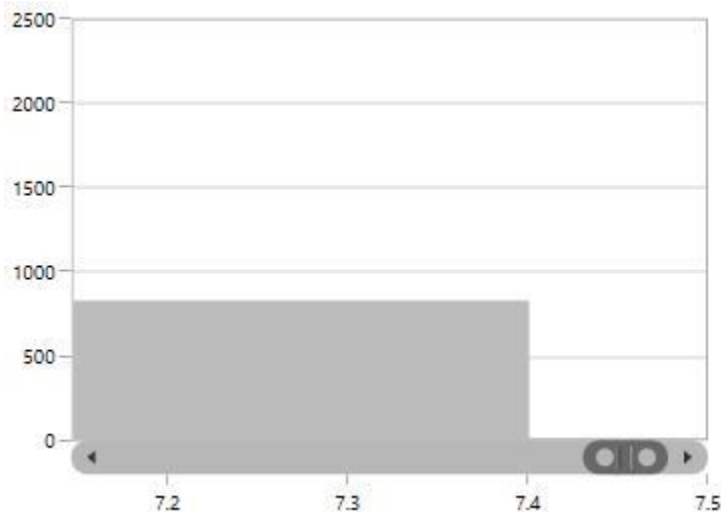
### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    EdgeLabelsDrawingMode = EdgeLabelsDrawingMode.Fit  
};
```



### EdgeLabelsVisibilityMode

The visibility of the extreme labels of the axis can be controlled using [EdgeLabelsVisibilityMode](#) property. By default the [Default](#) option in [EdgeLabelsVisibilityMode](#) is set, which displays the edge label based on auto interval calculations. The following image depicts the default option in [EdgeLabelsVisibilityMode](#) while zooming.



### Always Visible

AlwaysVisible option in [EdgeLabelsVisibilityMode](#) is used to view the edge labels even while performing zooming.

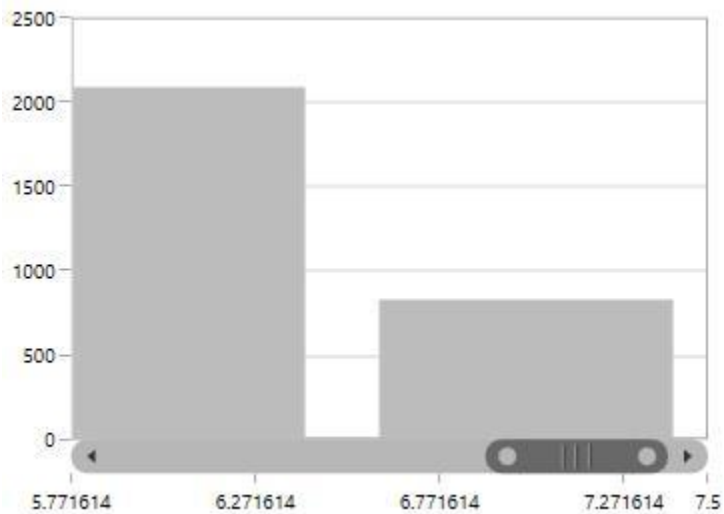
The following code example and image demonstrates the AlwaysVisible option while zooming.

### XML

```
<syncfusion:SfChart.PrimaryAxis>
  <syncfusion:NumericalAxis EdgeLabelsVisibilityMode="AlwaysVisible"
    EnableScrollBar="True">
  </syncfusion:NumericalAxis>
</syncfusion:SfChart.PrimaryAxis>
```

**C#**

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    EdgeLabelsVisibilityMode = EdgeLabelsVisibilityMode.AlwaysVisible  
};
```

**Visible**

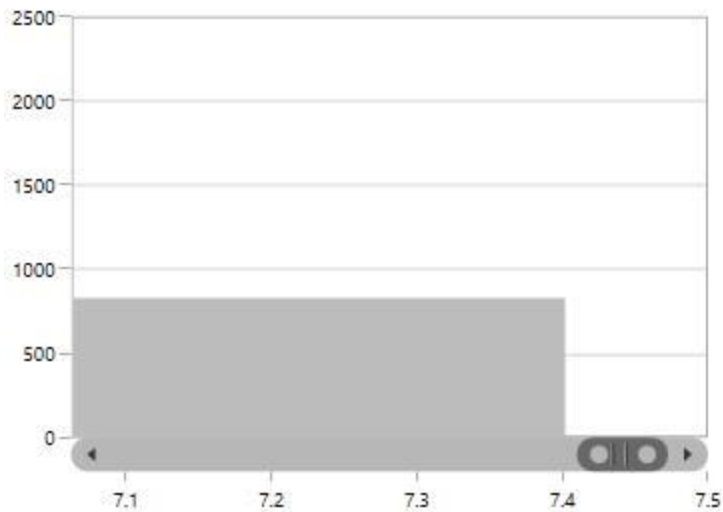
Visible option is used to display the edge labels (first and last label) irrespective of the auto interval calculation until zooming (i.e., in normal state)

**XML**

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis EdgeLabelsVisibilityMode="Visible"  
    EnableScrollBar="True" >  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

**C#**

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    EdgeLabelsVisibilityMode = EdgeLabelsVisibilityMode.Visible  
};
```



### Axis Label Border

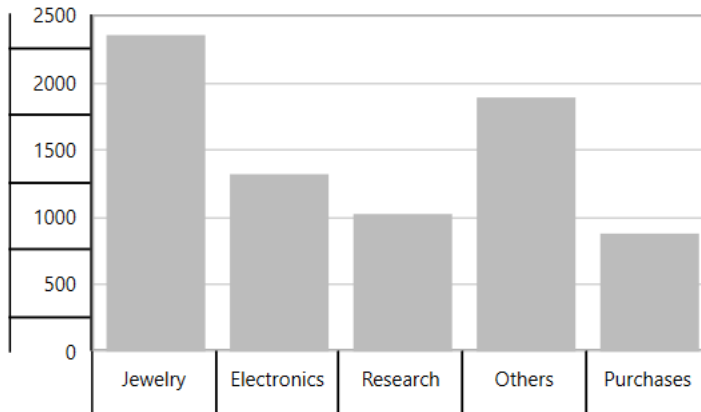
[ChartAxis](#) provides support to place the border around its label. To place the border around axis, enable [ShowLabelBorder](#) property of axis; it can be set as shown in the following code example.

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:CategoryAxis ShowLabelBorder="True"/>  
</syncfusion:SfChart.PrimaryAxis>  
<syncfusion:SfChart.SecondaryAxis>  
</syncfusion:NumericalAxis ShowLabelBorder="True" />  
</syncfusion:SfChart.SecondaryAxis>
```

#### C#

```
chart.PrimaryAxis = new CategoryAxis()  
{  
    ShowLabelBorder = true,  
};  
chart.SecondaryAxis = new NumericalAxis()  
{  
    ShowLabelBorder = true  
};
```



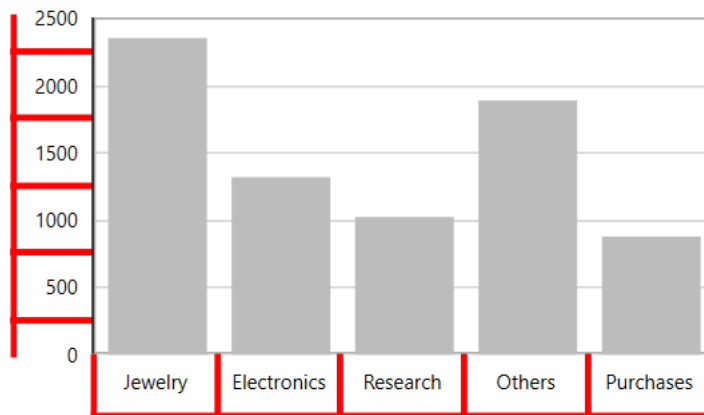
The border color and width can be customized with [LabelBorderBrush](#) and [LabelBorderWidth](#) properties of chart axis; it can be set as shown in the following code example.

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:CategoryAxis LabelBorderWidth="3" ShowLabelBorder="True"  
LabelBorderBrush="Red"/>  
</syncfusion:SfChart.PrimaryAxis>  
<syncfusion:SfChart.SecondaryAxis>  
</syncfusion:NumericalAxis ShowLabelBorder="True" LabelBorderWidth="3"  
LabelBorderBrush="Red"/>  
</syncfusion:SfChart.SecondaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis()  
{  
    ShowLabelBorder = true,  
    LabelBorderWidth = 3,  
    LabelBorderBrush = new SolidColorBrush(Colors.Red)  
};  
chart.SecondaryAxis = new NumericalAxis()  
{  
    ShowLabelBorder = true,  
    LabelBorderWidth = 3,  
    LabelBorderBrush = new SolidColorBrush(Colors.Red),  
};
```



### Grid lines

By default, gridlines are automatically added to the [ChartAxis](#) in its defined intervals. SfChart supports customization of gridline. The visibility of the gridlines can be controlled using the [ShowGridLines](#) property.

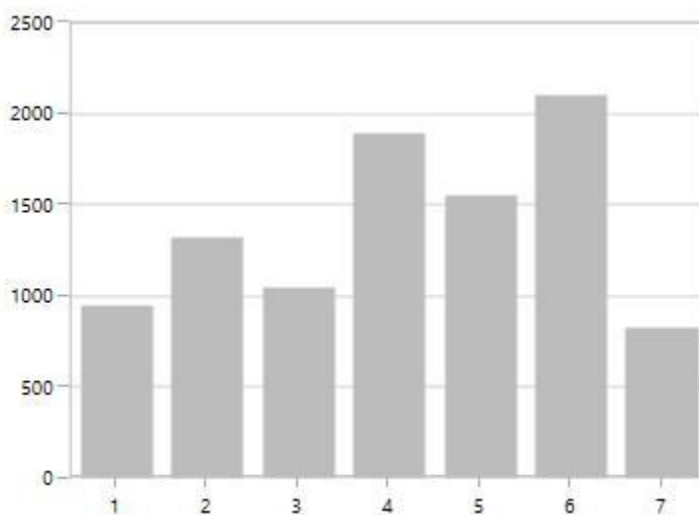
The following code example illustrates the [ShowGridLines](#) property as false in the primary axis.

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis ShowGridLines="False" >  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    ShowGridLines = false  
};
```



Style can also be applied to Major and Minor Gridlines using [MajorGridLineStyle](#) and [MinorGridLineStyle](#) properties.

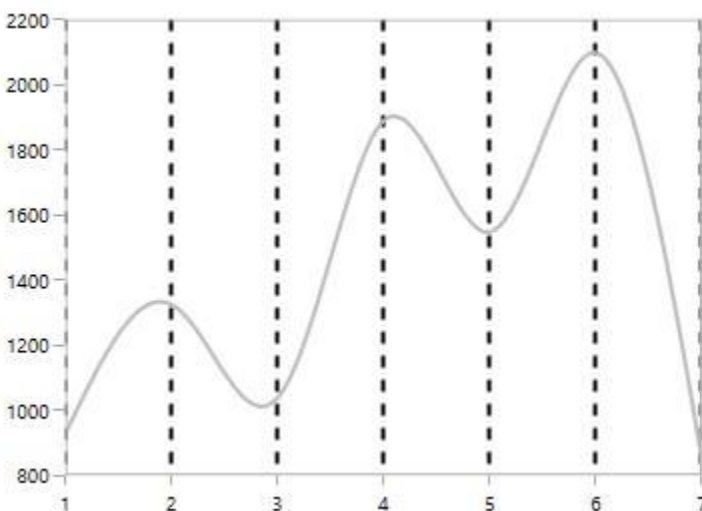
### MajorGridLineStyle

#### XML

```
<syncfusion:SfChart x:Name="chart">
  <syncfusion:SfChart.Resources>
    <Style TargetType="Line" x:Key="lineStyle">
      <Setter Property="StrokeThickness" Value="2"/>
      <Setter Property="Stroke" Value="Black"/>
      <Setter Property="StrokeDashArray" Value="3,3"/>
    </Style>
  </syncfusion:SfChart.Resources>
  <syncfusion:SfChart.PrimaryAxis>
    <syncfusion:NumericalAxis MajorGridLineStyle="{StaticResource lineStyle}"/>
  </syncfusion:SfChart.PrimaryAxis>
</syncfusion:SfChart>
```

#### C#

```
chart.PrimaryAxis = new NumericalAxis()
{
    MajorGridLineStyle = chart.Resources["lineStyle"] as Style
};
```



### MinorGridLineStyle

Minor gridlines will be added automatically when the small tick lines is defined inside the chart area.

#### XML

```
<syncfusion:SfChart x:Name="chart">
  <syncfusion:SfChart.Resources>
    <Style TargetType="Line" x:Key="lineStyle">
      <Setter Property="StrokeThickness" Value="1"/>
      <Setter Property="Stroke" Value="DarkGray"/>
      <Setter Property="StrokeDashArray" Value="3,3"/>
    </Style>
  </syncfusion:SfChart.Resources>
  <syncfusion:SfChart.PrimaryAxis>
    <syncfusion:NumericalAxis MinorGridLineStyle="{StaticResource lineStyle}"/>
  </syncfusion:SfChart.PrimaryAxis>
</syncfusion:SfChart>
```



```

</Style>
</syncfusion:SfChart.Resources>
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis SmallTicksPerInterval="3"
MinorGridLineStyle="{StaticResource lineStyle}"/>
</syncfusion:SfChart.SecondaryAxis>
</syncfusion:SfChart>

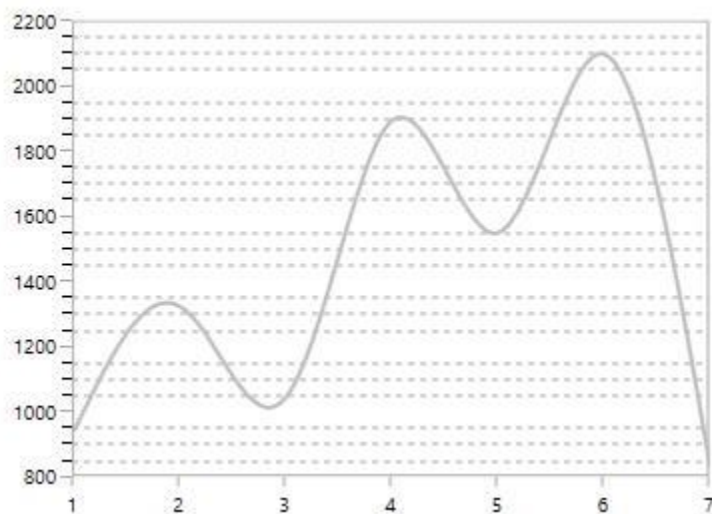
```

**C#**

```

chart.PrimaryAxis = new NumericalAxis()
{
    SmallTicksPerInterval = 3,
    MinorGridLineStyle = chart.Resources["lineStyle"] as Style
};

```

**Tick lines**

Tick line are the small lines which is drawn on the axis line representing the axis labels .Tick lines will be drawn outside of the axis by default.

**TickLineSize**

Tick lines thickness can be customized using [TickLineSize](#) property as shown in the below code snippet.

**XML**

```

<syncfusion:SfChart.PrimaryAxis>
<syncfusion:NumericalAxis TickLineSize="10" ></syncfusion:NumericalAxis>
</syncfusion:SfChart.PrimaryAxis>

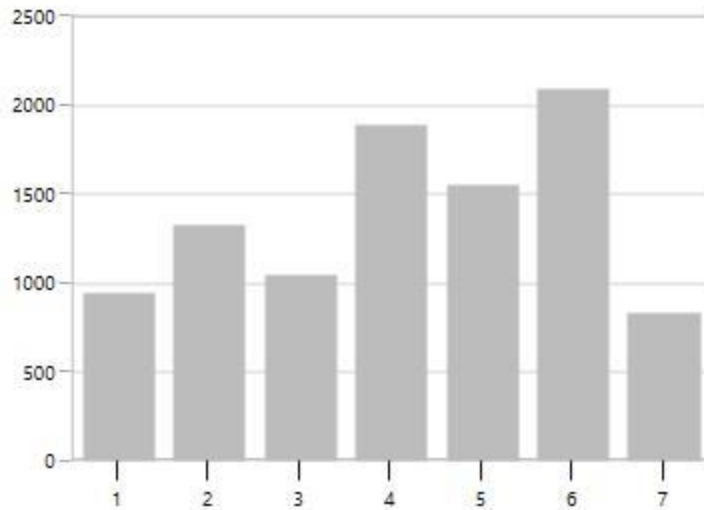
```

**C#**

```

chart.PrimaryAxis = new NumericalAxis()
{
    TickLineSize = 10
};

```



### Positioning the Major Tick Lines

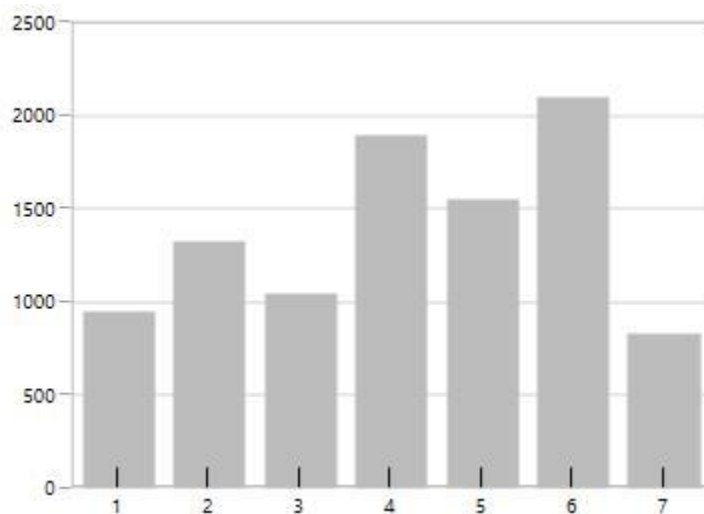
Tick lines can be positioned inside or outside of the chart area using [TickLinesPosition](#) property. By default the tick lines will positioned outside of the chart area. The following code example demonstrates the positioning tick lines inside chart area.

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis TickLinesPosition="Inside">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    TickLinesPosition = AxisElementPosition.Inside  
};
```



## Customization

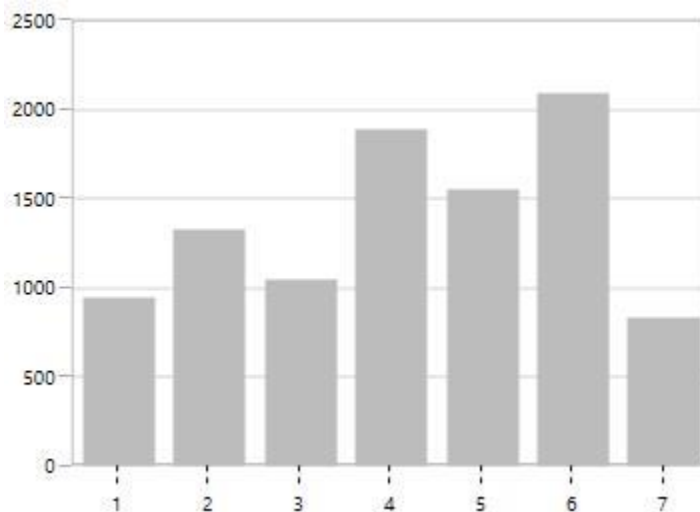
Style can be applied to major tick lines using [MajorTickLineStyle](#) property. The following code snippet demonstrates the styling of major tick lines.

### XML

```
<syncfusion:SfChart x:Name="chart">
  <syncfusion:SfChart.Resources>
    <Style TargetType="Line" x:Name="lineStyle">
      <Setter Property="StrokeThickness" Value="1"/>
      <Setter Property="Stroke" Value="Black"/>
      <Setter Property="StrokeDashArray" Value="3,3"/>
    </Style>
  </syncfusion:SfChart.Resources>
  <syncfusion:SfChart.SecondaryAxis>
    <syncfusion:NumericalAxis TickLineSize="10"
    MajorTickLineStyle="{StaticResource lineStyle}"/>
  </syncfusion:SfChart.SecondaryAxis>
</syncfusion:SfChart>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()
{
    TickLineSize = 10,
    MajorTickLineStyle = chart.Resources["lineStyle"] as Style
};
```



## MinorTickLines

Minor tick lines can be added by defining [SmallTicksPerInterval](#) property. This property will add the tick lines to every interval.

The following code example demonstrates the small ticks set for every interval.

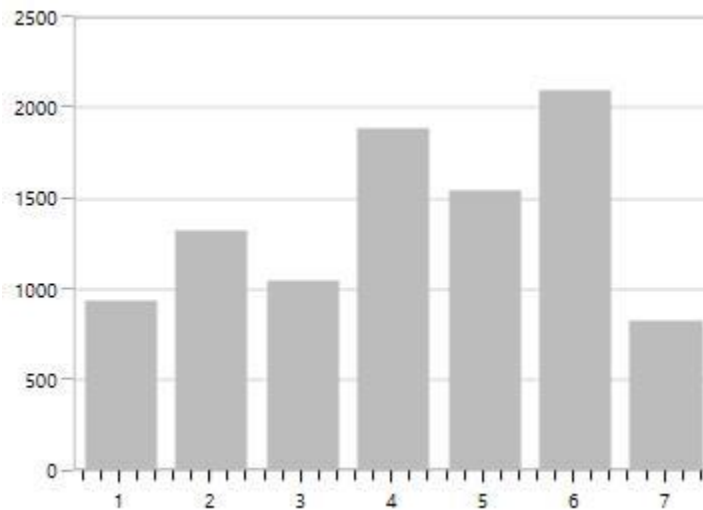
### XML

```
<syncfusion:SfChart.PrimaryAxis>
```

```
<syncfusion:NumericalAxis Interval="1" SmallTicksPerInterval="4" >  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    Interval = 1,  
    SmallTicksPerInterval = 4  
};
```



### Positioning the minor tick lines

Minor tick lines can be positioned inside or outside using [SmallTickLinesPosition](#) property. By default the minor tick lines will be positioned outside.

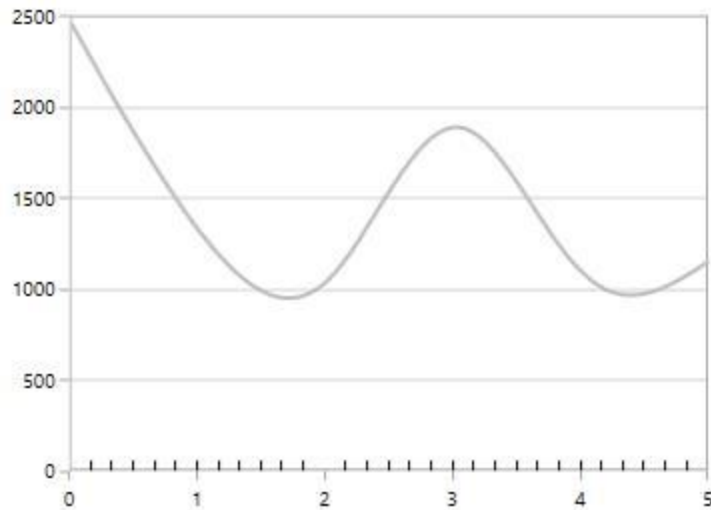
The following code example demonstrates the positioning of minor tick lines inside the chart area.

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis SmallTicksPerInterval="2"  
    SmallTickLinesPosition="Inside">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    SmallTicksPerInterval = 2,  
    SmallTickLinesPosition = AxisElementPosition.Inside  
};
```



### Customization of Minor Ticklines

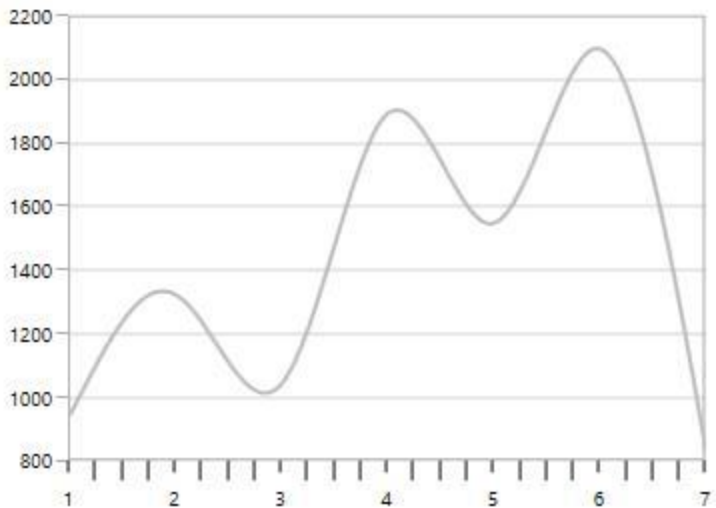
The thickness of the minor tick lines can be customized using [SmallTickLineSize](#) property as shown in the below code snippet.

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis Interval="1" SmallTicksPerInterval="3"  
SmallTickLineSize="10">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    Interval = 1,  
    SmallTicksPerInterval = 3,  
    SmallTickLineSize = 10  
};
```



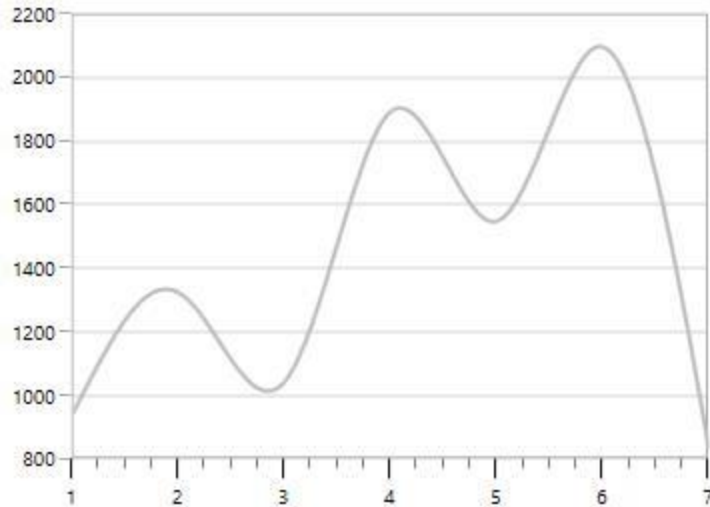
Styling customization of minor tick lines can be defined using [MinorTickLineStyle](#) property. The following code example and image demonstrates the style for minor tick lines.

#### XML

```
<syncfusion:SfChart x:Name="chart">
  <syncfusion:SfChart.Resources>
    <Style TargetType="Line" x:Name="lineStyle" >
      <Setter Property="StrokeThickness" Value="0.5"/>
      <Setter Property="Stroke" Value="Black"/>
    </Style>
  </syncfusion:SfChart.Resources>
  <syncfusion:SfChart.SecondaryAxis>
    <syncfusion:NumericalAxis FontSize="12" Interval="1"
      SmallTicksPerInterval="3"
      TickLineSize="10" SmallTickLineSize="5"
      MinorTickLineStyle="{StaticResource lineStyle}"/>
  </syncfusion:SfChart.SecondaryAxis>
</syncfusion:SfChart>
```

#### C#

```
chart.PrimaryAxis = new NumericalAxis()
{
    Interval = 1,
    SmallTicksPerInterval = 3,
    TickLineSize = 10,
    SmallTickLineSize = 5
    MinorTickLineStyle = chart.Resources["lineStyle"] as Style
};
```



**Note:** For category axis, small tick lines is not applicable since it is rendered based on index positions.

#### Customize individual axis elements

The [RangeStyles](#) can be used to customize the gridlines, ticks and axis labels for a specific region of ChartAxis. The following properties are used to customize the specific range in an axis:

- [Start](#) - Sets the start range of an axis.
- [End](#) - Sets the end range of an axis.
- [MajorGridLineStyle](#) - Customizes the major grid lines of an axis.
- [MinorGridLineStyle](#) - Customizes the minor grid lines of an axis.
- [MajorTickStyle](#) - Customizes the major tick lines of an axis.
- [MinorTickStyle](#) - Customizes the minor tick lines of an axis.
- [LabelStyle](#) - Customizes the axis labels for a specific range.

#### XML

```
<syncfusion:SfChart.Resources>
<Style TargetType="Line" x:Key="RangeLineStyle">
<Setter Property="StrokeThickness" Value="2"/>
<Setter Property="Stroke" Value="RoyalBlue"/>
</Style>
<Style TargetType="Line" x:Key="lineStyle">
<Setter Property="StrokeThickness" Value="2"/>
<Setter Property="Stroke" Value="Green"/>
</Style>
</syncfusion:SfChart.Resources>
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis MajorGridLineStyle= "{StaticResource lineStyle}"
MajorTickLineStyle="{StaticResource lineStyle}">
<syncfusion:NumericalAxis.LabelStyle>
<syncfusion:LabelStyle Foreground="Green"/>
</syncfusion:NumericalAxis.LabelStyle>
<syncfusion:NumericalAxis.RangeStyles>
<syncfusion:ChartAxisRangeStyleCollection>
<syncfusion:ChartAxisRangeStyle Start="18" End="22" MajorGridLineStyle=
"{StaticResource RangeLineStyle}" MajorTickLineStyle = "{StaticResource
```

```

RangeLineStyle}">
<syncfusion:ChartAxisRangeStyle.LabelStyle>
<syncfusion:LabelStyle Foreground="RoyalBlue" />
</syncfusion:ChartAxisRangeStyle.LabelStyle>
</syncfusion:ChartAxisRangeStyle>
</syncfusion:ChartAxisRangeStyleCollection>
</syncfusion:NumericalAxis.RangeStyles>
</syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>

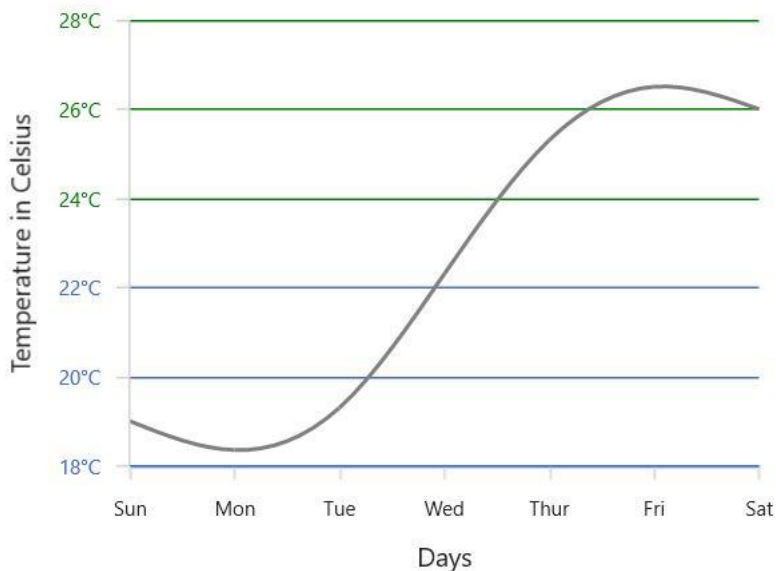
```

## C#

```

NumericalAxis secondaryAxis = new NumericalAxis();
secondaryAxis.MajorGridLineStyle = chart.Resources["lineStyle"] as Style;
secondaryAxis.MajorTickLineStyle = chart.Resources["lineStyle"] as Style;
secondaryAxis.LabelStyle.Foreground = new SolidColorBrush(Colors.Green);
ChartAxisRangeStyleCollection axisRangeStyles = new
ChartAxisRangeStyleCollection();
ChartAxisRangeStyle rangeStyle = new ChartAxisRangeStyle() { Start = 18, End
= 22 };
rangeStyle.MajorGridLineStyle = chart.Resources["RangeLineStyle"] as Style;
rangeStyle.LabelStyle = new LabelStyle();
rangeStyle.LabelStyle.Foreground = new SolidColorBrush(Colors.RoyalBlue);
rangeStyle.MajorTickLineStyle = chart.Resources["RangeLineStyle"] as Style;
axisRangeStyles.Add(rangeStyle);
secondaryAxis.RangeStyles = axisRangeStyles;
chart.SecondaryAxis = secondaryAxis;

```



## AxisLine

SfChart provides support to customize the style of the axis line by defining the [AxisLineStyle](#) property as shown in the below code snippet.

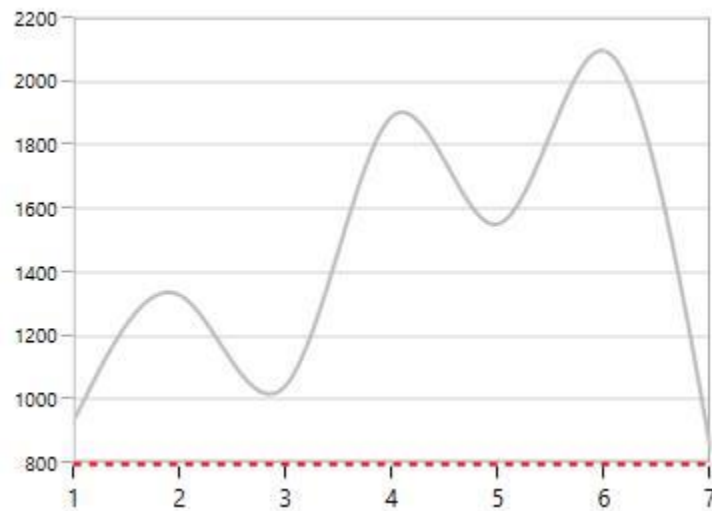


**XML**

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:NumericalAxis Interval="1" >
<syncfusion:NumericalAxis.AxisLineStyle>
<Style TargetType="Line" >
<Setter Property="StrokeThickness" Value="2"/>
<Setter Property="Stroke" Value="Red"/>
<Setter Property="StrokeDashArray" Value="2,2"/>
</Style>
</syncfusion:NumericalAxis.AxisLineStyle>
</syncfusion:NumericalAxis>
</syncfusion:SfChart.PrimaryAxis>
```

**C#**

```
chart.PrimaryAxis = new NumericalAxis()
{
    Interval = 1,
    AxisLineStyle = chart.Resources["lineStyle"] as Style
};
```

**Applying Padding to the Axis line**

The padding to the axis line is defined using [AxisLineOffset](#) property. The following code example demonstrates the setting [AxisLineOffset](#) for x axis.

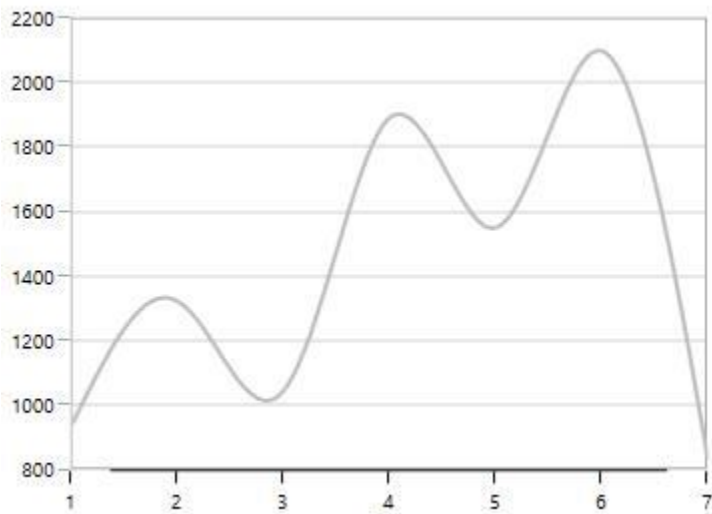
**XML**

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:NumericalAxis AxisLineOffset="20" >
</syncfusion:NumericalAxis>
</syncfusion:SfChart.PrimaryAxis>
```

**C#**

```
chart.PrimaryAxis = new NumericalAxis()
{
```

```
AxisLineOffset = 20  
};
```



### Origin Customization

SfChart allows you to customize the origin. By default the axis will be rendered having (0,0) as origin in x and y axes.

#### ShowAxisNextToOrigin

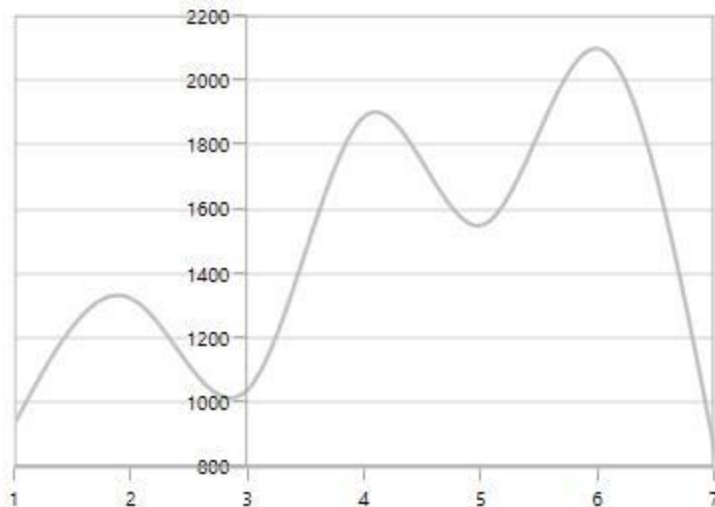
[ShowAxisNextToOrigin](#) property is used to move the axis line to the origin value in [Origin](#) property based on the x or y axes. The following code example demonstrates shifting the axis in the origin value in numerical axis.

#### XML

```
<syncfusion:SfChart.SecondaryAxis>  
<syncfusion:NumericalAxis Origin="3" ShowAxisNextToOrigin="True">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.SecondaryAxis>
```

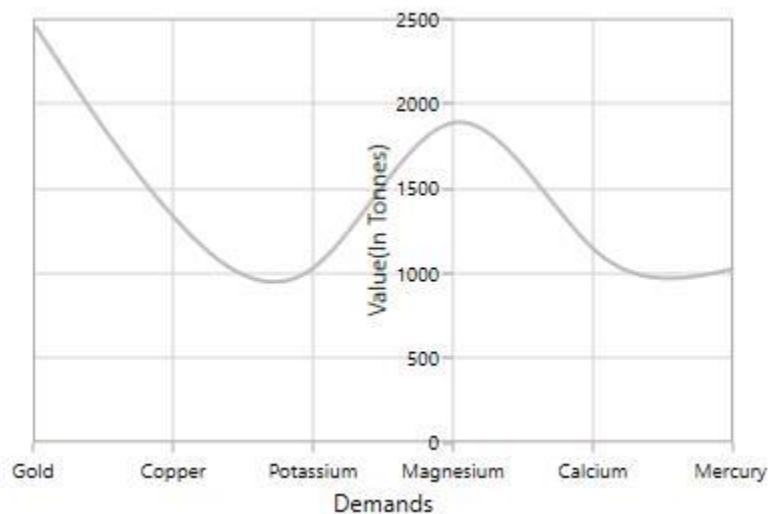
#### C#

```
chart.SecondaryAxis = new NumericalAxis()  
{  
    Origin = 3,  
    ShowAxisNextToOrigin = true  
};
```



### Positioning the Header

The following image demonstrates the default positioning of header when the axis is moved inside based on the origin value.



If you want to position the header outside of the chart area then you can set the [Far](#) option in [HeaderPosition](#) property.

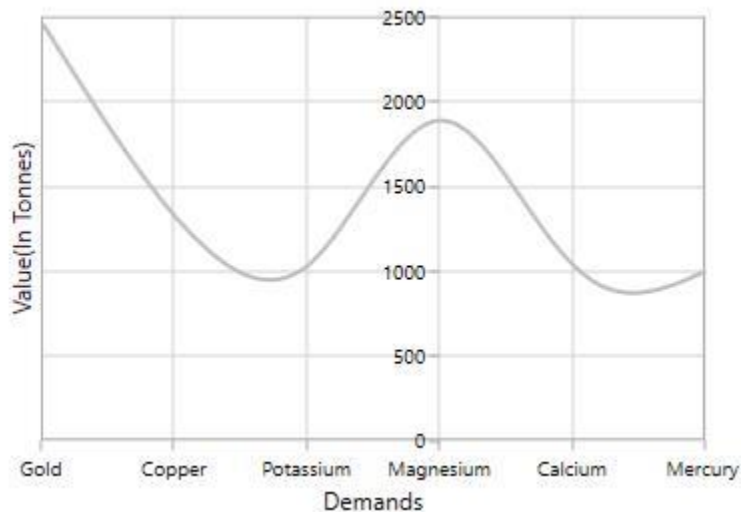
The following code example demonstrates the positioning of the header outside even when the axis is moved inside.

### XML

```
<syncfusion:SfChart.SecondaryAxis>
  <syncfusion:NumericalAxis HeaderPosition="Far"
    Origin="3" ShowAxisNextToOrigin="True" Header="Value(In Tonnes)" >
  </syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>
```

### C#

```
chart.SecondaryAxis = new NumericalAxis()
{
    Origin = 3,
    ShowAxisNextToOrigin = true,
    Header = "Value(In Tonnes)",
    HeaderPosition = AxisHeaderPosition.Far
};
```



### Adding Origin line

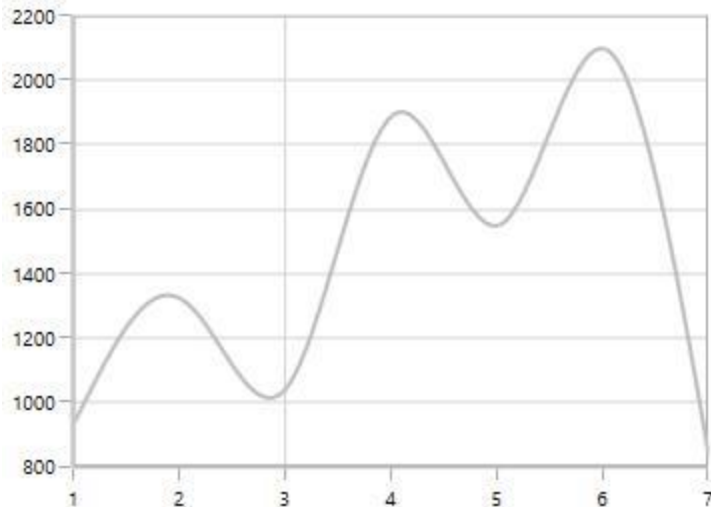
The origin line can be added to chart area by setting the [ShowOrigin](#) property to true. The following code example demonstrates the displaying origin line at (3,0) position value as set in [Origin](#) property.

#### XML

```
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis Origin="3" ShowOrigin="True">
</syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>
```

#### C#

```
chart.SecondaryAxis = new NumericalAxis()
{
    Origin = 3,
    ShowOrigin = true
};
```



### Customizing the OriginLine

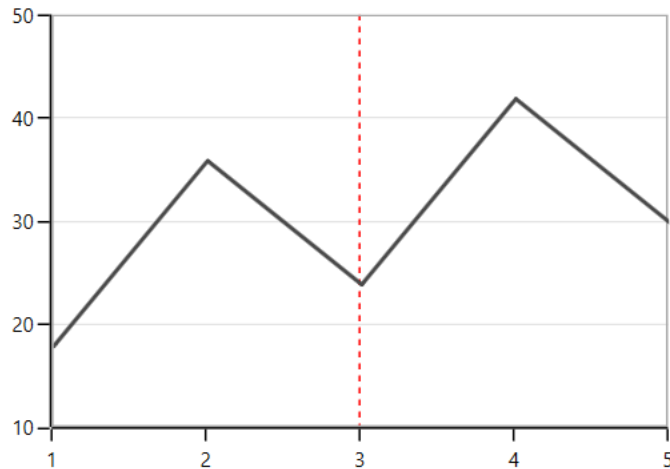
The origin line of axis can be customized by using the [OriginLineStyle](#) property of the ChartAxis. The following code demonstrates how to apply style for origin line.

#### XML

```
<syncfusion:SfChart.SecondaryAxis>
  <syncfusion:NumericalAxis Origin="3" ShowOrigin="True">
    <syncfusion:NumericalAxis.OriginLineStyle>
      <Style TargetType="Line">
        <Setter Property="Stroke" Value="Red"/>
        <Setter Property="StrokeDashArray" Value="3"/>
      </Style>
    </syncfusion:NumericalAxis.OriginLineStyle>
  </syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>
```

#### C#

```
NumericalAxis numericalAxis = new NumericalAxis()
{
    Origin = 3,
    ShowOrigin = true
};
Style style = new Style(typeof(Line));
style.Setters.Add(new Setter(Line.StrokeProperty, new
SolidColorBrush(Colors.Red)));
style.Setters.Add(new Setter(Line.StrokeDashArrayProperty, new
DoubleCollection() { 3 }));
numericalAxis.OriginLineStyle = style;
chart.SecondaryAxis = numericalAxis;
```



### Types of Axis

[ChartAxis](#) supports the following types.

- [NumericalAxis](#)
- [CategoryAxis](#)
- [DateTimeAxis](#)
- [DateTimeCategoryAxis](#)
- [TimeSpanAxis](#)
- [LogarithmicAxis](#)

You can choose any type of [ChartAxis](#), like [DateTimeAxis](#), [NumericalAxis](#), [CategoryAxis](#), [LogarithmicAxis](#) or [TimeSpanAxis](#) depending on the value type. [DateTimeCategoryAxis](#) is a special type, used to plot date and time values for the given data points.

### *NumericalAxis*

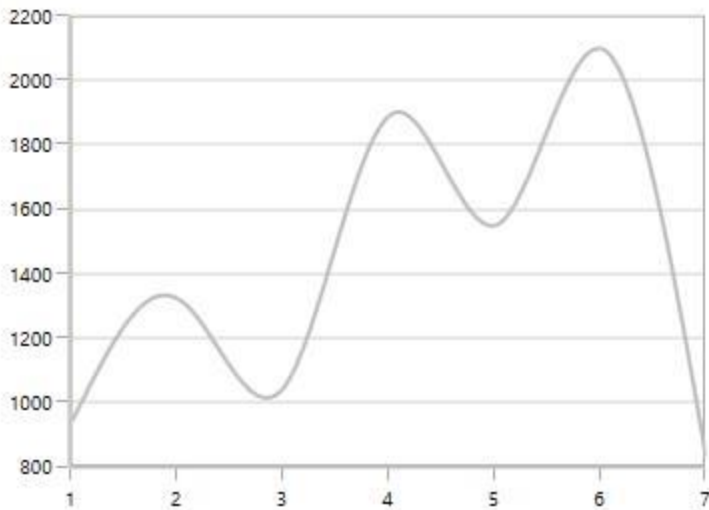
[NumericalAxis](#) is used to plot numerical values to the chart. [NumericalAxis](#) can be defined for both [PrimaryAxis](#) and [SecondaryAxis](#). The following code snippet shows how to define the [NumericalAxis](#).

### **XML**

```
<syncfusion:SfChart.PrimaryAxis>  
  <syncfusion:NumericalAxis  >  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>  
<syncfusion:SfChart.SecondaryAxis>  
  <syncfusion:NumericalAxis  >  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.SecondaryAxis>
```

### **C#**

```
chart.PrimaryAxis = new NumericalAxis();  
chart.SecondaryAxis = new NumericalAxis();
```



### Customizing the NumericalAxis Range

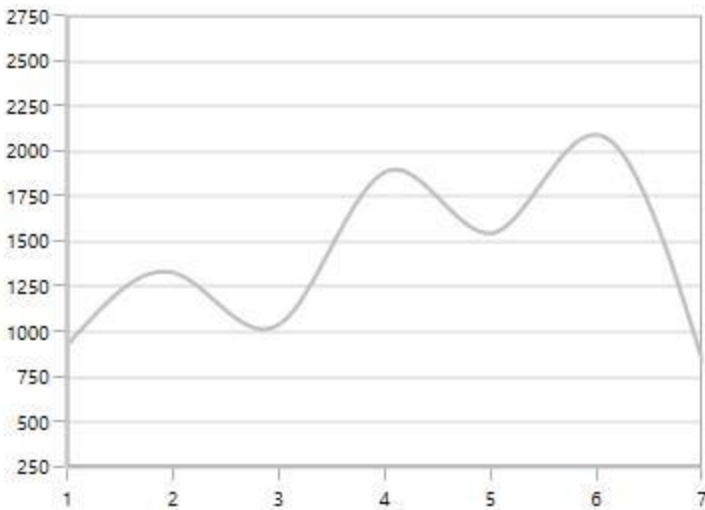
[Maximum](#) property used for setting the maximum value for the axis range and [Minimum](#) property is used for setting the minimum value for the axis range.

#### XML

```
<syncfusion:SfChart.SecondaryAxis>  
<syncfusion:NumericalAxis Maximum="2750" Minimum="250" Interval="250">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.SecondaryAxis>
```

#### C#

```
chart.SecondaryAxis = new NumericalAxis()  
{  
    Maximum = 2750,  
    Minimum = 250,  
    Interval = 250  
};
```



---

**Note:** If minimum or maximum value is set, the other value is calculated by default internally.

---

### StartRangeFromZero

[NumericalAxis](#) will calculate the range based on the data points binded to the axis. To start the range from zero have to define the [StartRangeFromZero](#) property to True. The following code example demonstrates the NumericalAxis range starting from zero.

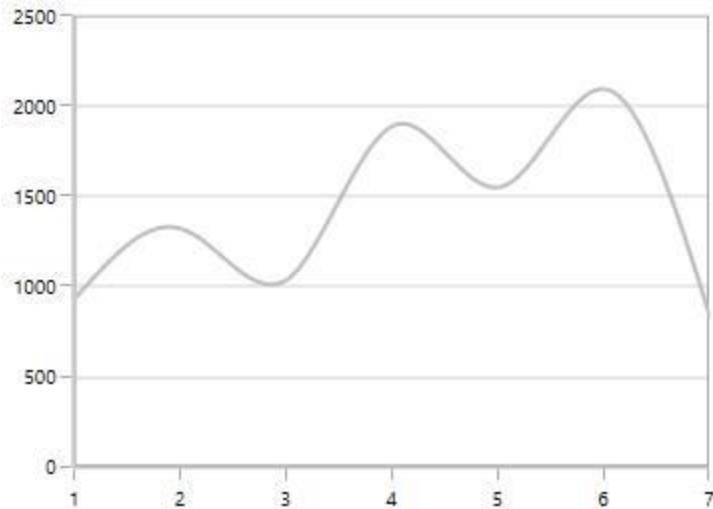
### XML

```
<syncfusion:SfChart.SecondaryAxis>  
  <syncfusion:NumericalAxis StartRangeFromZero="True">  
  </syncfusion:NumericalAxis>  
</syncfusion:SfChart.SecondaryAxis>
```

### C#

```
chart.SecondaryAxis = new NumericalAxis()  
{  
  StartRangeFromZero = true  
};
```





**Note:** By default, Range is calculated between the minimum and maximum value of the data points.

#### [CategoryAxis](#)

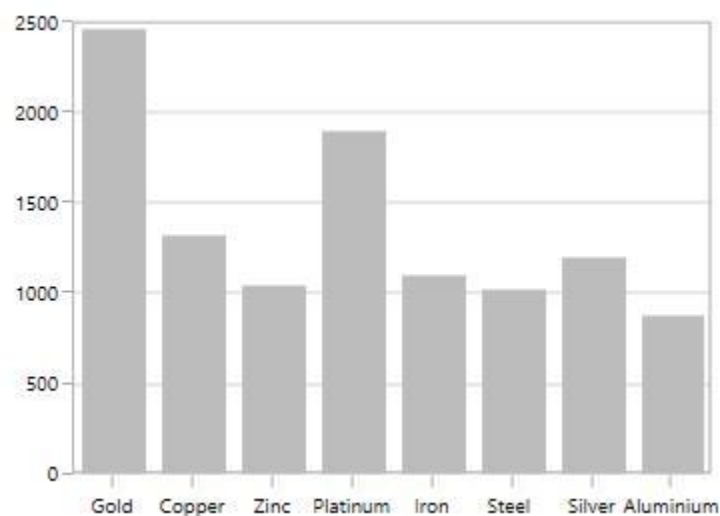
[CategoryAxis](#) is an indexed based axis that plots values based on the index of the data point collection. The points are equally spaced here. The following code example initializes the [CategoryAxis](#).

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:CategoryAxis >  
</syncfusion:CategoryAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new CategoryAxis();
```



#### LabelPlacement

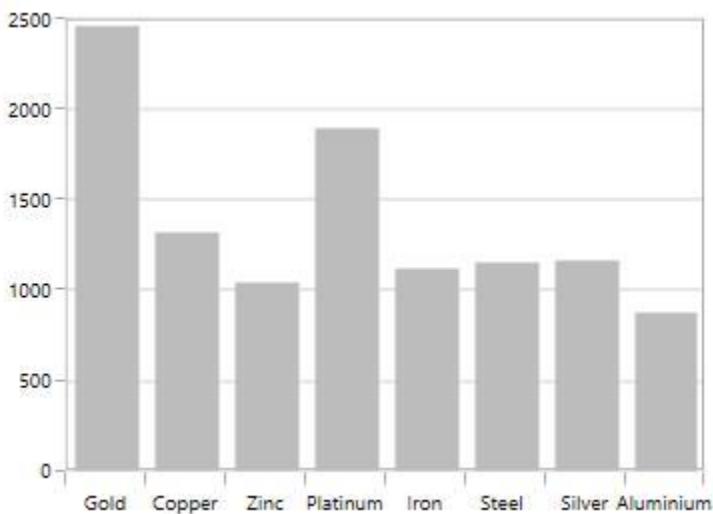
In [CategoryAxis](#), labels are placed based on tick lines using [LabelPlacement](#) property. By default the labels are placed [OnTicks](#). The following code example demonstrates placing the label between ticks in [CategoryAxis](#)

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:CategoryAxis LabelPlacement="BetweenTicks">  
</syncfusion:CategoryAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new CategoryAxis()  
{  
    LabelPlacement = LabelPlacement.BetweenTicks  
};
```



#### IsIndexed

By default, [CategoryAxis](#) plots the values based on the index of the data points. However, the [CategoryAxis](#) can be made to plot the data points based on its data, instead of index value by disabling the [IsIndexed](#) property of [CategoryAxis](#), and it is shown in the following code example.

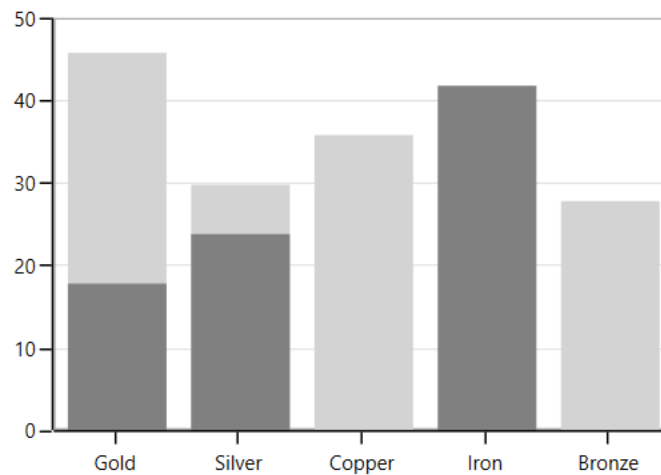
#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:CategoryAxis IsIndexed="False"/>  
</syncfusion:CategoryAxis>
```

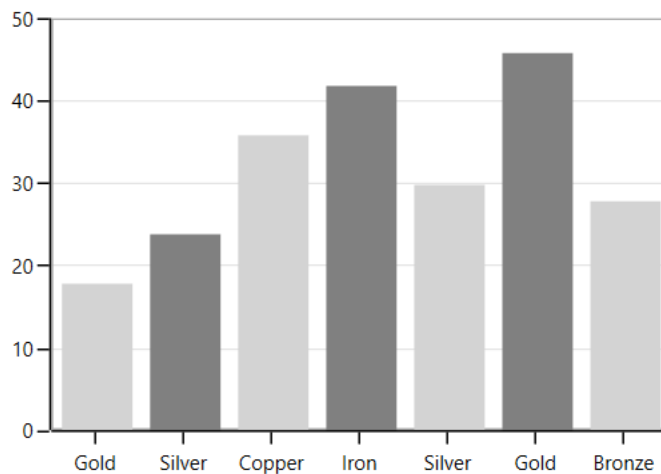
#### C#

```
chart.PrimaryAxis = new CategoryAxis()  
{  
    IsIndexed = false,  
};
```

The following screenshot illustrates the series with IsIndexed value as False.



By default, the IsIndexed property value is true, and it is shown in following screenshot.



**Note:** This feature is not applicable for Accumulation series, ErrorBarSeries, RadarSeries, and PolarSeries.

### AggregateFunctions

When the [IsIndexed](#) property of the [CategoryAxis](#) is disabled, the same index values(XValue) are grouped by [AggregateFunctions](#) property of the axis. The following are the types of [AggregateFunctions](#),

- [Average](#)
- [Count](#)
- [Max](#)
- [Min](#)
- [None](#)
- [Sum](#)

The default value of [AggregateFunctions](#) is None, and it is shown in the following code example.

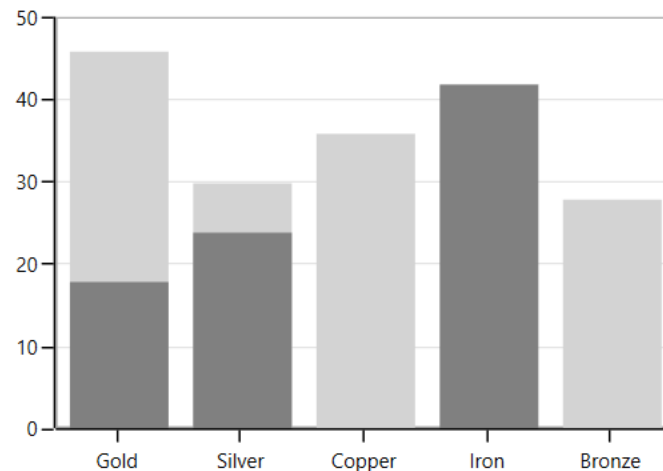
### XML

```
<syncfusion:SfChart.PrimaryAxis>
```

```
<syncfusion:CategoryAxis IsIndexed="False"/>
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    IsIndexed = false,
};
```



### Average

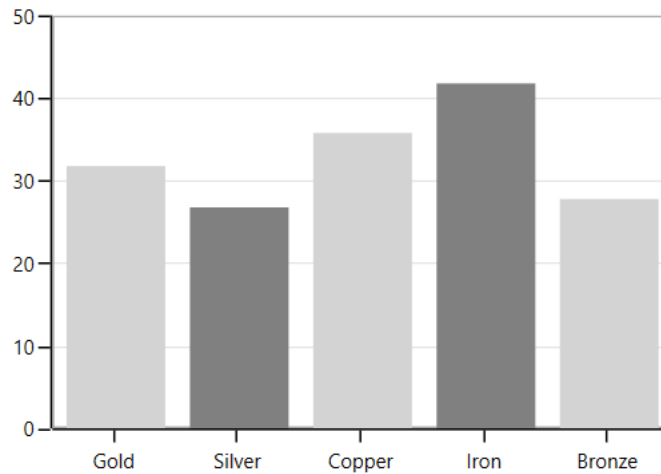
The following code example illustrates the axis with [AggregateFunctions](#) is Average,

### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:CategoryAxis IsIndexed="False" AggregateFunctions="Average"/>
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    IsIndexed = false,
    AggregateFunctions = AggregateFunctions.Average
};
```



### Count

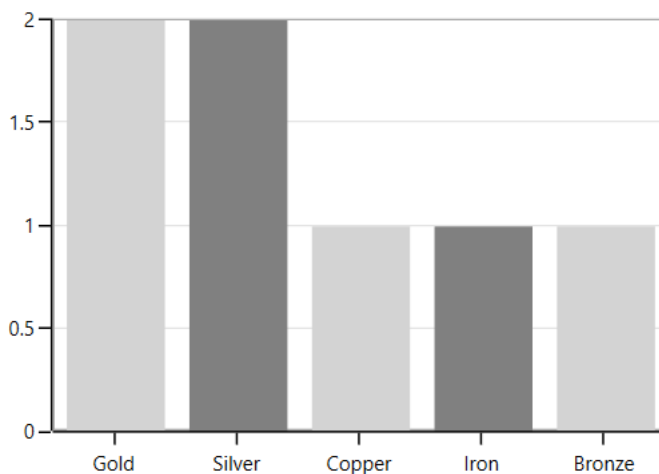
The following code example illustrates the axis with [AggregateFunctions](#) is Count,

#### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:CategoryAxis IsIndexed="False" AggregateFunctions="Count"/>
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    IsIndexed = false,
    AggregateFunctions = AggregateFunctions.Count
};
```



### Max

The following code example illustrates the axis with [AggregateFunctions](#) is Max,

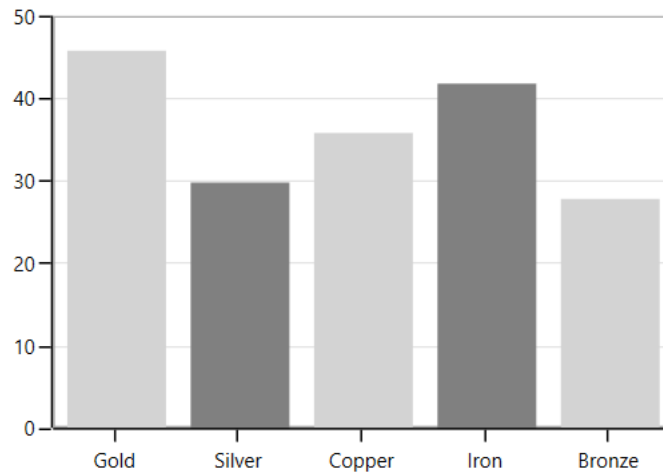
#### XML

```
<syncfusion:SfChart.PrimaryAxis>
```

```
<syncfusion:CategoryAxis IsIndexed="False" AggregateFunctions="Max"/>
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    IsIndexed = false,
    AggregateFunctions = AggregateFunctions.Max
};
```



### Min

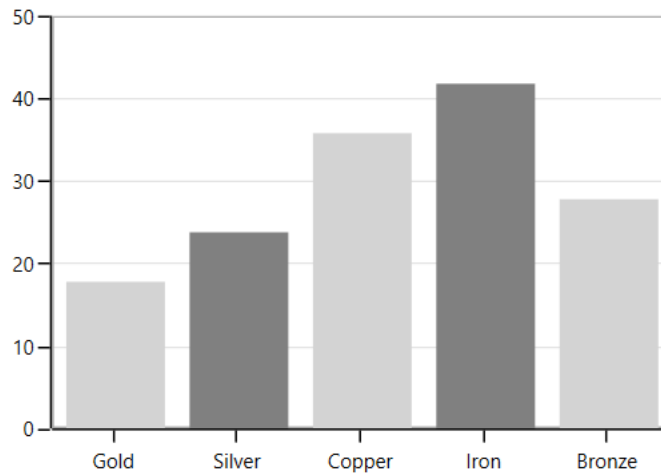
The following code example illustrates the axis with [AggregateFunctions](#) is Min,

### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:CategoryAxis IsIndexed="False" AggregateFunctions="Min"/>
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    IsIndexed = false,
    AggregateFunctions = AggregateFunctions.Min
};
```



### Sum

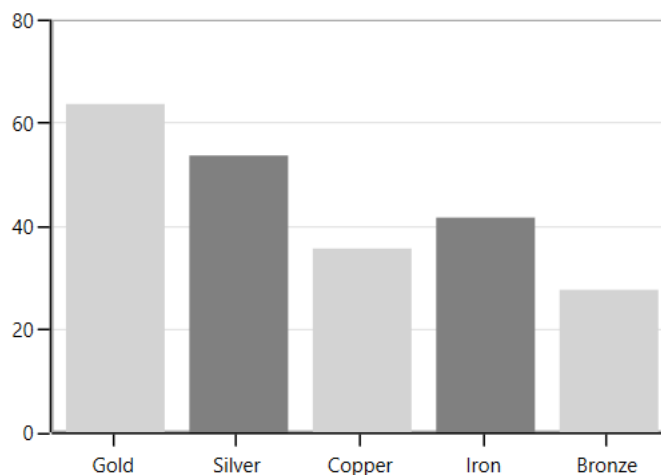
The following code example illustrates the axis with [AggregateFunctions](#) is Sum,

### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:CategoryAxis IsIndexed="False" AggregateFunctions="Sum"/>
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    IsIndexed = false,
    AggregateFunctions = AggregateFunctions.Sum
};
```



### [DateTimeAxis](#)

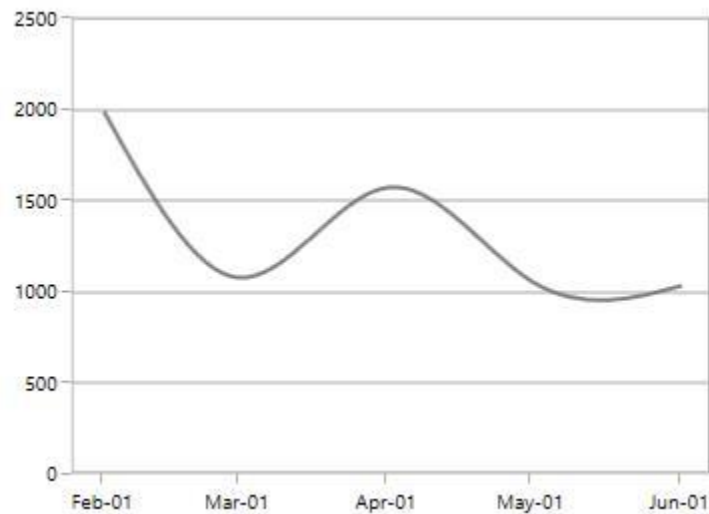
[DateTimeAxis](#) is used to plot DateTime values. The [DateTimeAxis](#) is widely used to make financial charts in places like the Stock Market, where index plotting is done every day.

### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:DateTimeAxis LabelFormat="MMM-dd"></syncfusion:DateTimeAxis>
</syncfusion:SfChart.PrimaryAxis>
```

**C#**

```
chart.PrimaryAxis = new DateTimeAxis()
{
    LabelFormat = "MMM-dd"
};
```

**Customizing the Range**

[Minimum](#) and [Maximum](#) properties behavior is same as in `NumericalAxis` instead of setting numerical value, you have to set date time values.

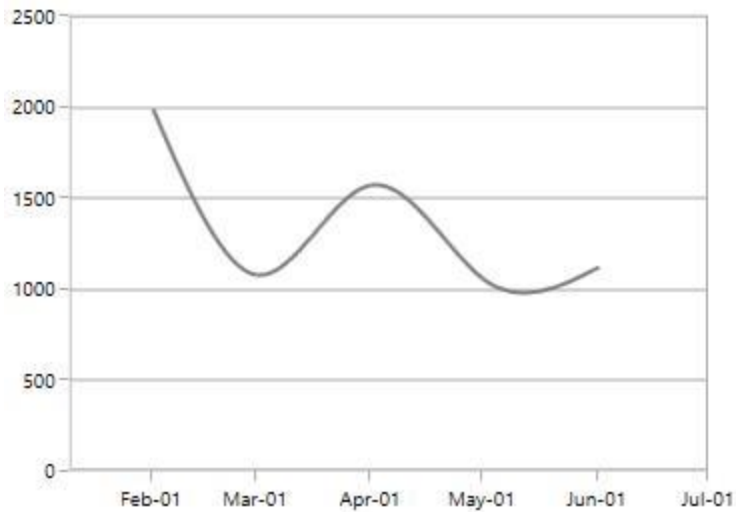
**XML**

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:DateTimeAxis Minimum="2015/01/10" Maximum="2015/07/01"
LabelFormat="MMM-dd"
IntervalType="Months" Interval="1">
</syncfusion:DateTimeAxis>
</syncfusion:SfChart.PrimaryAxis>
```

**C#**

```
chart.PrimaryAxis = new DateTimeAxis()
{
    Minimum = new DateTime(2015, 01, 10),
    Maximum = new DateTime(2015, 07, 01),
    LabelFormat = "MMM-dd",
    IntervalType = DateTimeIntervalType.Months,
    Interval = 1
};
```





### Business Hours Range Calculation

SfChart provides support to plot only the business hours in DateTimeAxis. This support is enabled by setting [EnableBusinessHours](#) property to true.

The following properties are used for business hour range calculation

- [OpenTime](#)- Represents the open working time of a day.
- [CloseTime](#)- Represents the close working time of a day.
- [WorkingDays](#)- Represents the working [days](#) of a week.

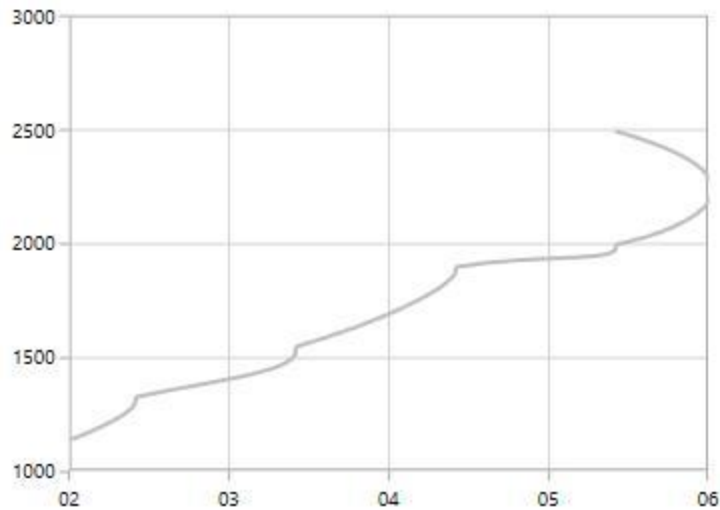
The following code snippet demonstrates the business hours support in DateTimeAxis

#### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:DateTimeAxis EnableBusinessHours="True" OpenTime="9"
CloseTime="24"
WorkingDays="Friday, Saturday, Sunday, Monday, Tuesday, Wednesday, Sunday">
</syncfusion:DateTimeAxis>
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new DateTimeAxis()
{
    EnableBusinessHours = true,
    OpenTime = 9,
    CloseTime = 24,
    WorkingDays = Day.Friday | Day.Saturday | Day.Sunday |
    Day.Monday | Day.Tuesday | Day.Wednesday | Day.Sunday
};
```



### *DateTimeCategoryAxis*

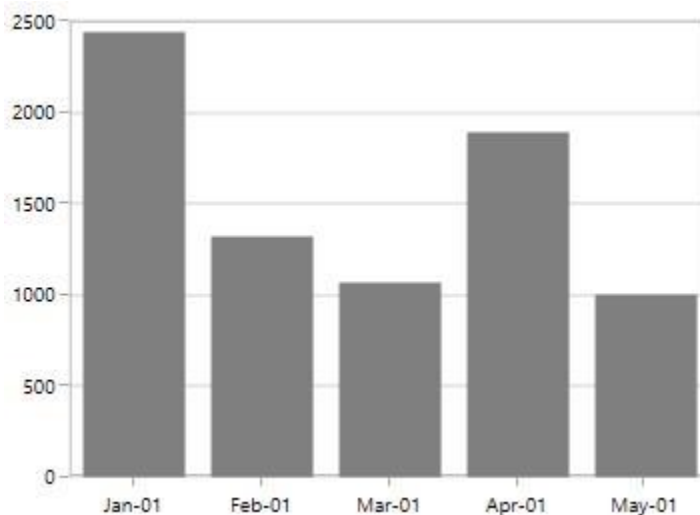
[DateTimeCategoryAxis](#) is a special type of axis used mainly with financial series. All the data points are plotted with equal spaces, similar to [CategoryAxis](#), thereby removing space for missing dates. [Interval](#) and range for the axis are calculated similar to [DateTimeAxis](#). There are no visual gaps between points, even when the difference between two points is more than a year.

### **XML**

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:DateTimeCategoryAxis LabelFormat="MMM-dd" >
</syncfusion:DateTimeCategoryAxis>
</syncfusion:SfChart.PrimaryAxis>
```

### **C#**

```
chart.PrimaryAxis = new DateTimeCategoryAxis()
{
    LabelFormat = "MMM-dd"
};
```



### *TimeSpan Axis*

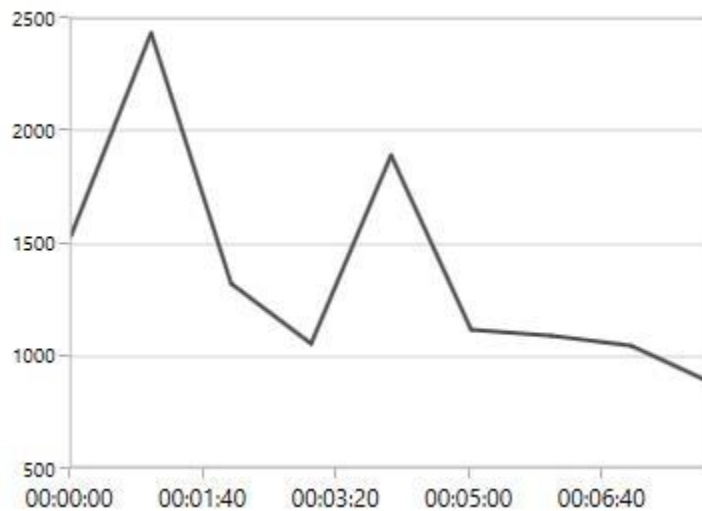
[TimeSpanAxis](#) is used to plot the time span values in the [PrimaryAxis](#). [TimeSpanAxis](#) has the advantage of plotting data with milliseconds difference. The limitation of [TimeSpanAxis](#) is that it can only accept timespan values (hh:mm:ss) and date time values are not accepted.

### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:TimeSpanAxis >
</syncfusion:TimeSpanAxis>
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new TimeSpanAxis();
```



### Customizing the Range

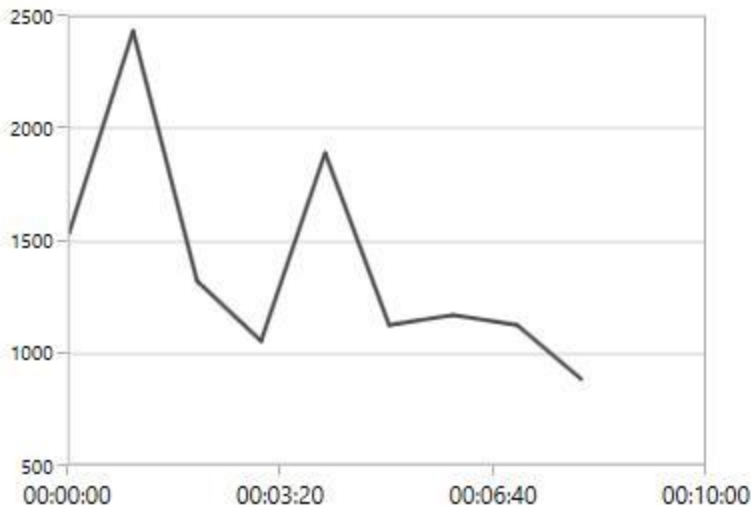
The following code snippet demonstrates the [Minimum](#) and [Maximum](#) properties for [TimeSpanAxis](#).

### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:TimeSpanAxis Minimum="00:00:00" Maximum="00:10:00">
</syncfusion:TimeSpanAxis>
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new TimeSpanAxis()
{
    Minimum = new TimeSpan(00, 00, 00),
    Maximum = new TimeSpan(00, 10, 00)
};
```



### [LogarithmicAxis](#)

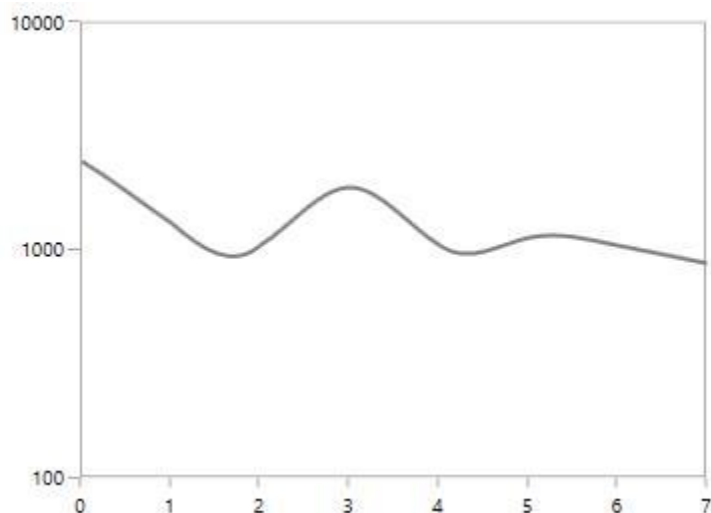
[LogarithmicAxis](#) is used to plot the logarithmic scale for the chart. The Logarithmic values will be plotted based on the logarithmic base value as 10.

### XML

```
<syncfusion:SfChart.SecondaryAxis>
  <syncfusion:LogarithmicAxis />
</syncfusion:SfChart.SecondaryAxis>
```

### C#

```
chart.SecondaryAxis = new LogarithmicAxis();
```



### Logarithmic Base

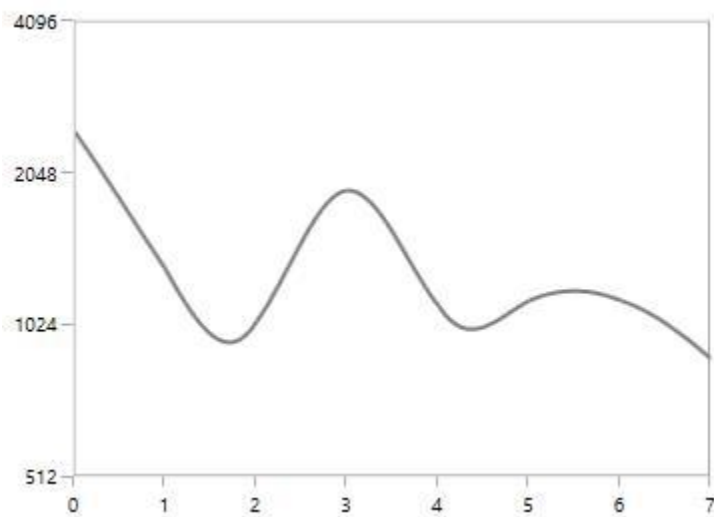
You can also change the base for logarithmic values. By default the logarithmic values is calculated the base from 10. The following code example demonstrates the logarithmic values in y axis calculated from base 2.

**XML**

```
<syncfusion:SfChart.SecondaryAxis>  
<syncfusion:LogarithmicAxis LogarithmicBase="2">  
</syncfusion:LogarithmicAxis>  
</syncfusion:SfChart.SecondaryAxis>
```

**C#**

```
chart.SecondaryAxis = new LogarithmicAxis()  
{  
    LogarithmicBase = 2  
};
```

**Customizing the Range**

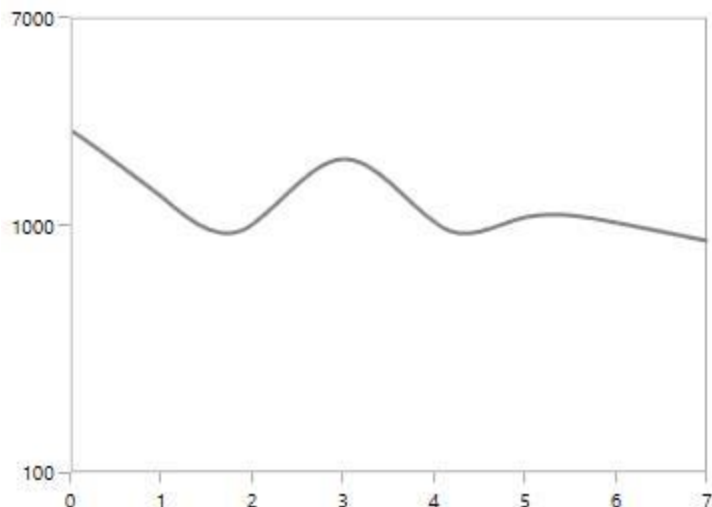
The following code snippet demonstrates the range customization of [LogarithmicAxis](#) based on [Minimum](#) and [Maximum](#) properties.

**XML**

```
<syncfusion:SfChart.SecondaryAxis>  
<syncfusion:LogarithmicAxis Minimum="100" Maximum="7000" >  
</syncfusion:LogarithmicAxis>  
</syncfusion:SfChart.SecondaryAxis>
```

**C#**

```
chart.SecondaryAxis = new LogarithmicAxis()  
{  
    Minimum = 100,  
    Maximum = 7000  
};
```



The following property is common for all types of axes,

- **IncludeStripLineRange**

### IncludeStripLineRange

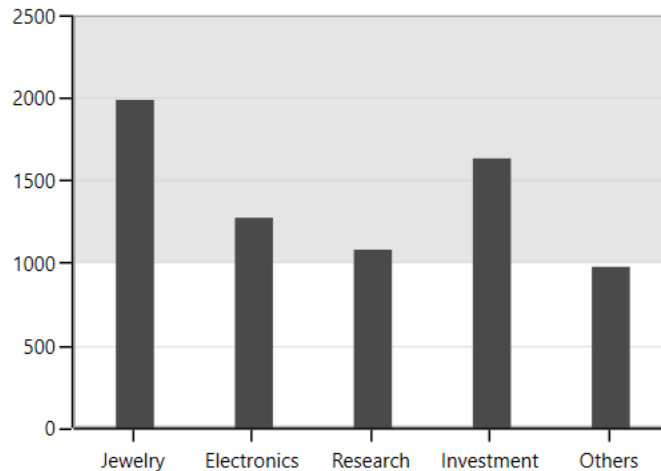
By default, [striplines](#) are drawn up to the chart axis range, if the provided values (Start and Width) exceed the actual range of chart axis. It can be avoided by using [IncludeStripLineRange](#) property. While enabling [IncludeStripLineRange](#) property, the axis range will be extended up to the provided stripline range. The property can be set as shown in the following code example.

### XML

```
<Chart:SfChart.SecondaryAxis>
  <Chart:NumericalAxis IncludeStripLineRange="True" >
    <Chart:NumericalAxis.StripLines>
      <Chart:ChartStripLine Start="1000"
        Width="1500"
        Background="LightGray"
        Opacity="0.6"/>
    </Chart:NumericalAxis.StripLines>
  </Chart:NumericalAxis>
</Chart:SfChart.SecondaryAxis>
```

### C#

```
NumericalAxis axis = new NumericalAxis();
axis.IncludeStripLineRange = true;
ChartStripLine stripline = new ChartStripLine()
{
    Start = 1000,
    Width = 1500,
    Background = new SolidColorBrush(Colors.LightGray),
    Opacity = 0.6
};
axis.StripLines.Add(stripline);
```



### IncludeAnnotationRange

By default, [annotations](#) are drawn up to the chart axis range, if the provided values (X1, X2, Y1, and Y2) exceed the actual range of [chart axis](#). It can be avoided by using [IncludeAnnotationRange](#) property. While enabling [IncludeAnnotationRange](#) property, the axis range will be extended up to the provided annotation range.

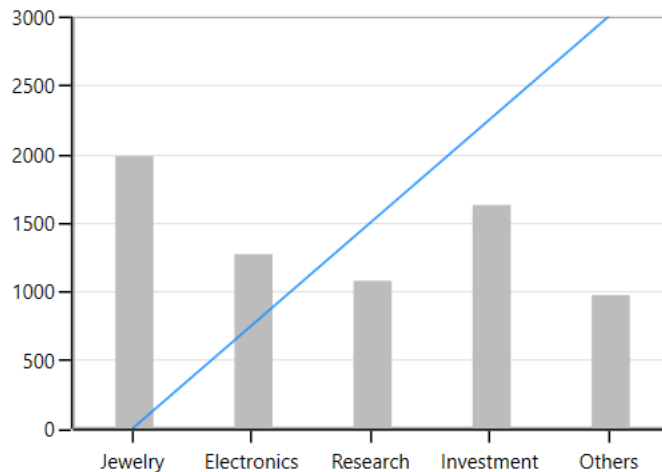
The property is applicable only for linear axes, and it can be set as shown in the following code example.

### XML

```
<Chart:SfChart.SecondaryAxis>
<Chart:NumericalAxis IncludeAnnotationRange="True" />
</Chart:SfChart.SecondaryAxis>
<Chart:SfChart.Annotations>
<Chart:LineAnnotation X1="0" X2="4" Y1="0" Y2="3000" />
</Chart:SfChart.Annotations>
```

### C#

```
chart.SecondaryAxis = new NumericalAxis()
{
    IncludeAnnotationRange = true
};
LineAnnotation annotation = new LineAnnotation();
annotation.X1 = 0;
annotation.X2 = 4;
annotation.Y1 = 0;
annotation.Y2 = 3000;
chart.Annotations.Add(annotation);
```



### Inverting axis

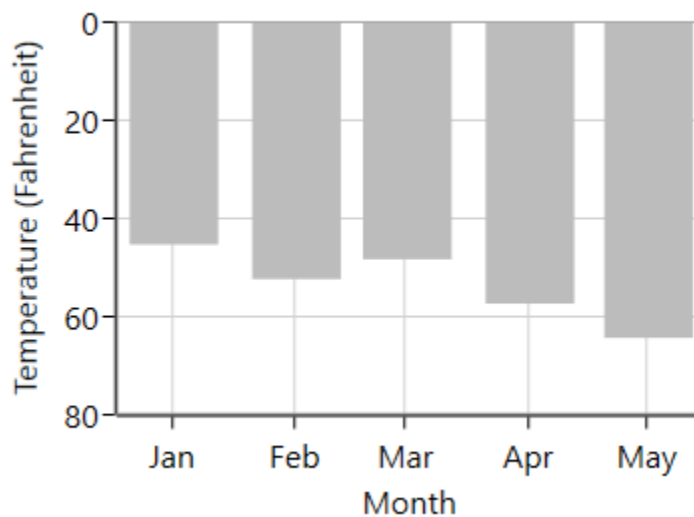
Axis can be inverted using the [IsInversed](#) property. The default value of this property is `False`.

### XML

```
<chart:SfChart.SecondaryAxis>
  <chart:NumericalAxis IsInversed="True"/>
</chart:SfChart.SecondaryAxis>
```

### C#

```
this.Chart.SecondaryAxis.IsInversed = true;
```



### Customizing the Intervals

[ChartAxis](#) calculates the range and intervals automatically based on the data points. The axis range and interval can be defined using the [Minimum](#), [Maximum](#) and [Interval](#) properties.

### NumericalAxis



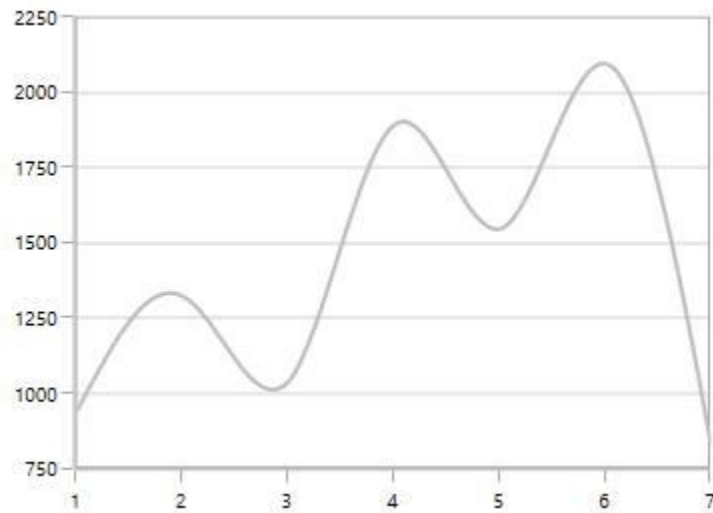
The following code snippet demonstrates the [Interval](#) customization in NumericalAxis.

#### XML

```
<syncfusion:SfChart.SecondaryAxis>  
<syncfusion:NumericalAxis Interval="250">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.SecondaryAxis>
```

#### C#

```
chart.SecondaryAxis = new NumericalAxis()  
{  
    Interval = 250  
};
```



#### CategoryAxis

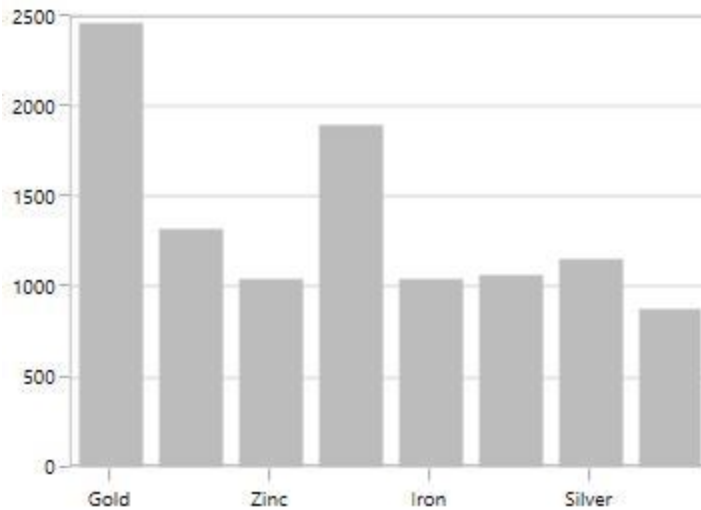
The following code snippet demonstrates the [Interval](#) customization in CategoryAxis.

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:CategoryAxis Interval="2">  
</syncfusion:CategoryAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new CategoryAxis()  
{  
    Interval = 2  
};
```



### DateTimeAxis

The DateTimeAxis [Interval](#) value corresponds to the type specified in the [IntervalType](#) property.

For instance, if the [Interval](#) is set as 2 and [IntervalType](#) is set as [Days](#), the labels are plotted for every two days. The following are the options for [IntervalType](#) property

Auto

- Days
- Hours
- Milliseconds
- Minutes
- Months
- Seconds
- Years

The default [IntervalType](#) of a [DateTimeAxis](#) is Auto. It calculates the type automatically and the interval, accordingly.

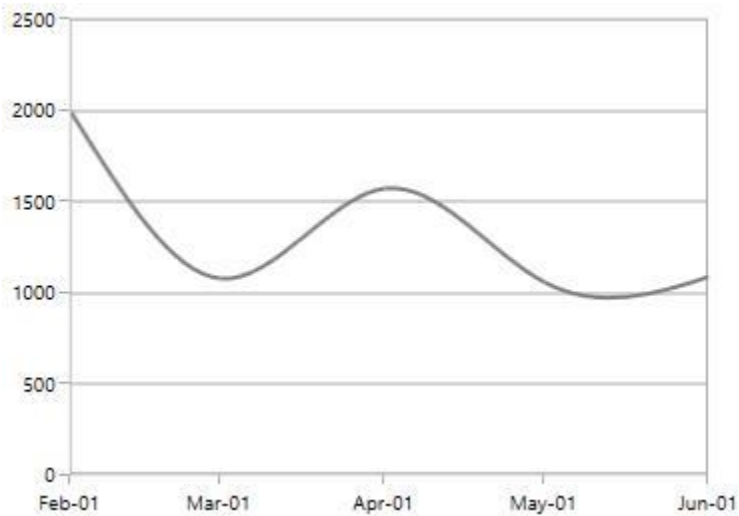
The following code snippet demonstrates the DateTimeAxis having one month interval.

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis Interval="1" IntervalType="Months"  
LabelFormat="MMM-dd"></syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    Interval = 1,  
    IntervalType = DateTimeIntervalType.Months,  
    LabelFormat = "MMM-dd"  
};
```



### DesiredIntervalsCount

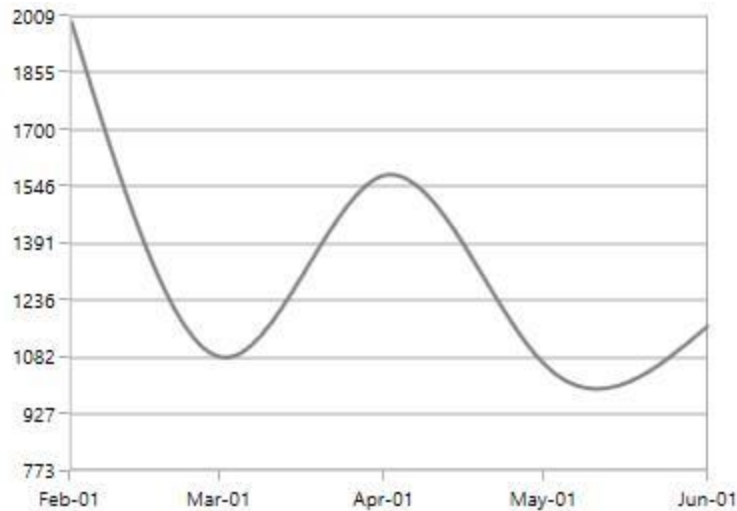
[DesiredIntervalsCount](#) property is used to specify the count of the axis labels between the first and last label. The following sample code defines the [DesiredIntervalsCount](#) property.

#### XML

```
<syncfusion:SfChart.SecondaryAxis>  
<syncfusion:NumericalAxis DesiredIntervalsCount="7">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.SecondaryAxis>
```

#### C#

```
chart.SecondaryAxis = new NumericalAxis()  
{  
    DesiredIntervalsCount = 7  
};
```



### Maximum Labels

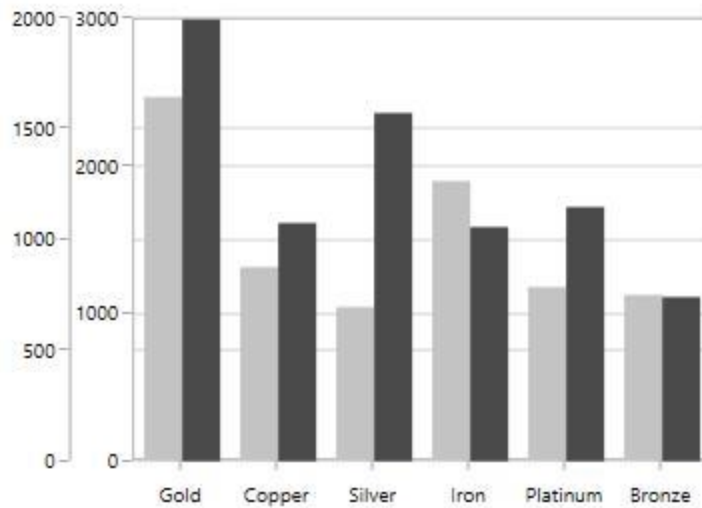
[MaximumLabels](#) property defines the count of the axis labels in the 100 pixels.

### XML

```
<syncfusion:SfChart.SecondaryAxis>
  <syncfusion:NumericalAxis MaximumLabels="2">
  </syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>
<syncfusion:ColumnSeries ItemsSource="{Binding Demands}"
  XBindingPath="Demand" YBindingPath="Year2010">
  <syncfusion:ColumnSeries.YAxis>
  <syncfusion:NumericalAxis MaximumLabels="2" >
  </syncfusion:NumericalAxis>
  </syncfusion:ColumnSeries.YAxis>
</syncfusion:ColumnSeries>
```

### C#

```
chart.SecondaryAxis = new NumericalAxis()
{
  MaximumLabels = 2
};
NumericalAxis axis = new NumericalAxis() { MaximumLabels = 2 };
ColumnSeries series = new ColumnSeries()
{
  ItemsSource = new ViewModel().Demands,
  XBindingPath = "Demand",
  YBindingPath = "Year2010",
  YAxis = axis
};
chart.Series.Add(series);
```



### Apply Padding to the Range

The [NumericalAxis](#) and [DateTimeAxis](#) have a [RangePadding](#) property that can be used to add padding to the range of a chart's axes.

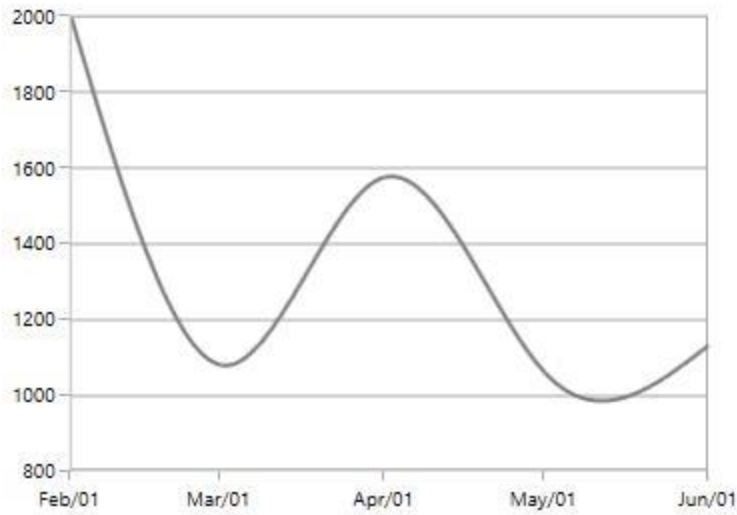
#### *DateTimeRangePadding*

The [RangePadding](#) types available in the [DateTimeAxis](#) are:

- Auto
- Additional
- None
- Round
- RoundStart
- RoundEnd
- PrependInterval
- AppendInterval

#### **Auto**

By default the date time range padding is [Auto](#).



### Additional

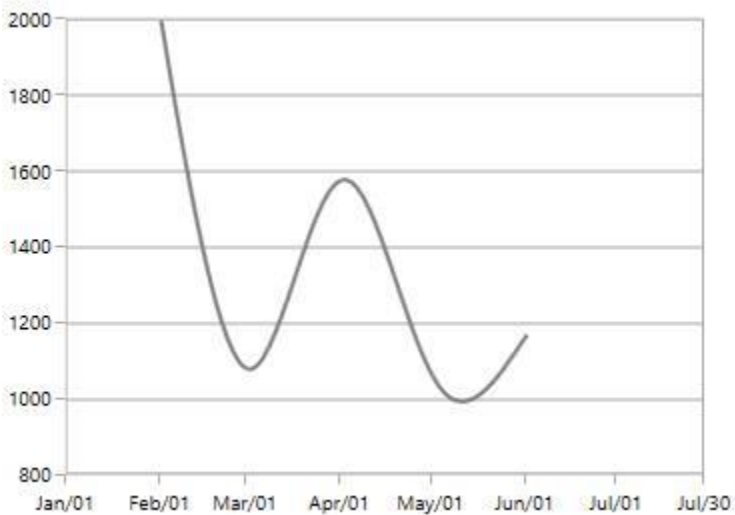
When [RangePadding](#) for [DateTimeAxis](#), the DateTime interval of the axis is added as padding, as shown in the following screenshot.

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis RangePadding="Additional">  
</syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    RangePadding = DateTimeRangePadding.Additional  
};
```



## Round

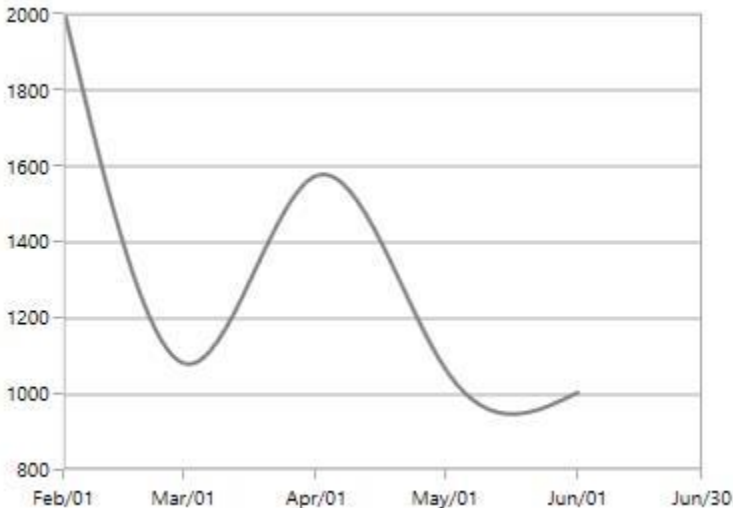
When [RangePadding](#) for [DateTimeAxis](#) is set to [Round](#), the range of the chart axis is rounded off to the nearest possible DateTime value, as shown in the following screenshot.

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis RangePadding="Round">  
</syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    RangePadding = DateTimeRangePadding.Round  
};
```



## None

When the [RangePadding](#) for a [DateTimeAxis](#) is [None](#).

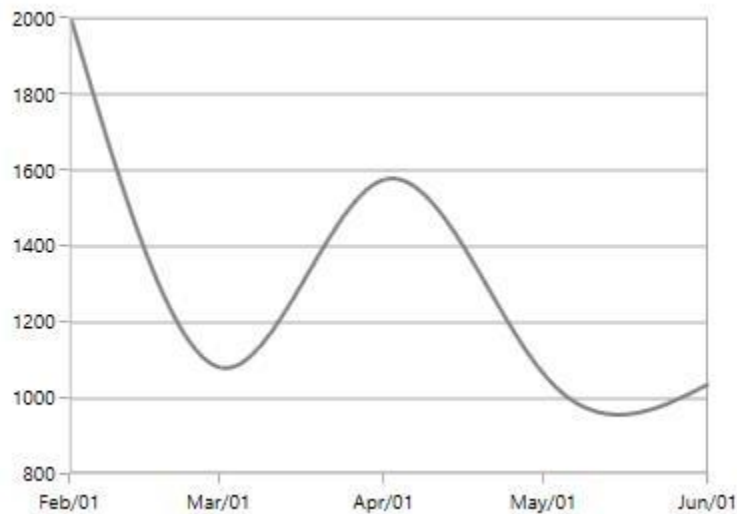
The following screenshot demonstrates a chart's x-axis with [RangePadding](#) set to [None](#).

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis RangePadding="None">  
</syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    RangePadding = DateTimeRangePadding.None  
};
```



### RoundStart

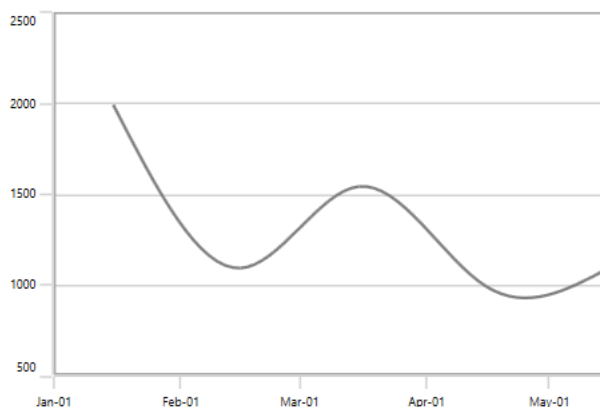
When [RangePadding](#) for [DateTimeAxis](#) is set to [RoundStart](#), the range of the chart axis is rounded in the start off to the nearest possible DateTime value, as shown in the following screenshot.

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis RangePadding="RoundStart">  
</syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    RangePadding = DateTimeRangePadding.RoundStart  
};
```



### RoundEnd



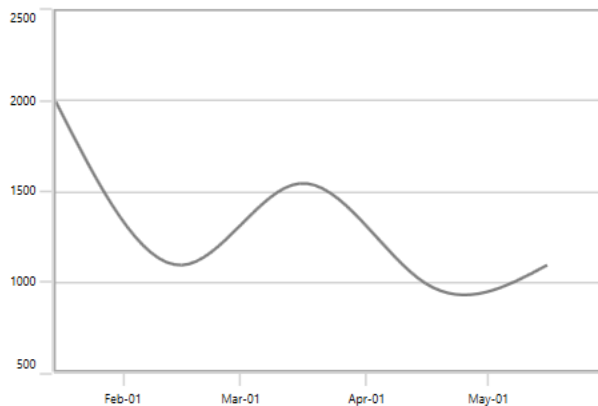
When [RangePadding](#) for [DateTimeAxis](#) is set to [RoundEnd](#), the range of the chart axis is rounded in the end off to the nearest possible DateTime value, as shown in the following screenshot.

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis RangePadding="RoundEnd">  
</syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    RangePadding = DateTimeRangePadding.RoundEnd  
};
```



#### PrependInterval

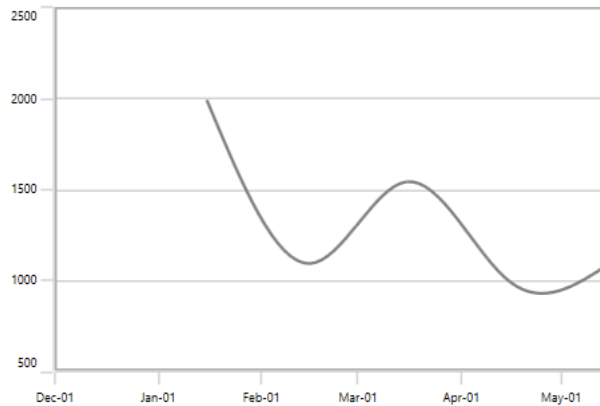
When [RangePadding](#) for [DateTimeAxis](#) is set to [PrependInterval](#), the DateTime interval of the axis is added in the start as padding, as shown in the following screenshot.

#### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis RangePadding="PrependInterval">  
</syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    RangePadding = DateTimeRangePadding.PrependInterval  
};
```



### AppendInterval

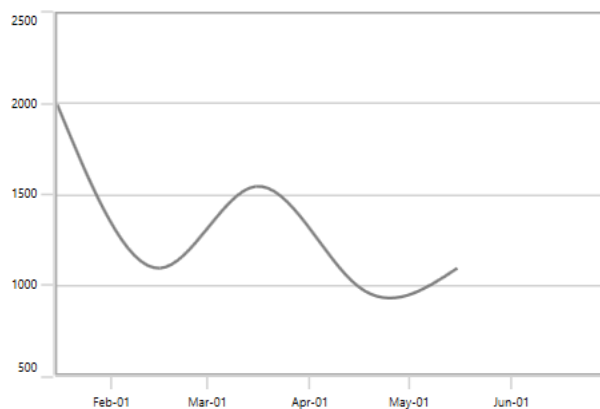
When [RangePadding](#) for [DateTimeAxis](#) is set to [AppendInterval](#), the DateTime interval of the axis is added in the end as padding, as shown in the following screenshot.

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:DateTimeAxis RangePadding="AppendInterval">  
</syncfusion:DateTimeAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new DateTimeAxis()  
{  
    RangePadding = DateTimeRangePadding.AppendInterval  
};
```



### [NumericalRangePadding](#)

The following types are available for [NumericalAxis](#):

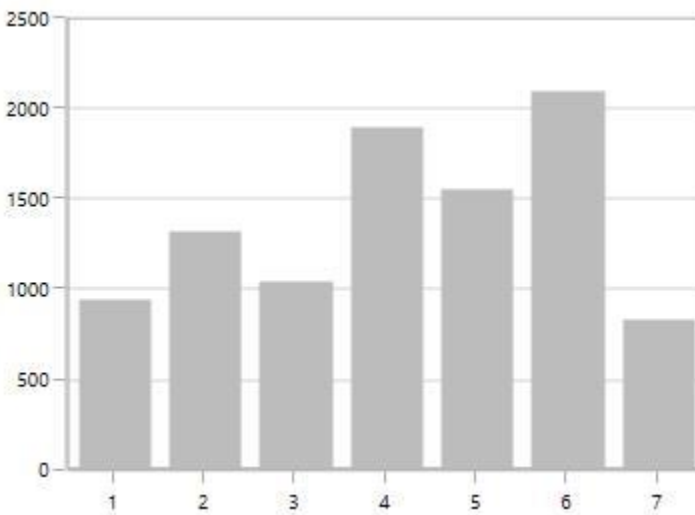
- Additional
- None
- Normal
- Round

- Auto
- RoundStart
- RoundEnd
- PrependInterval
- AppendInterval

### Round

By default, the default [RangePadding](#) value for [PrimaryAxis](#) is Auto and for [SecondaryAxis](#), the default value is [Round](#).

The following screenshot illustrates a chart's y-axis with [RangePadding](#) set to [Round](#).



### Normal

[Normal RangePadding](#) for a [NumericalAxis](#) is used mostly for the y-axis to have padding based on the Range calculation.

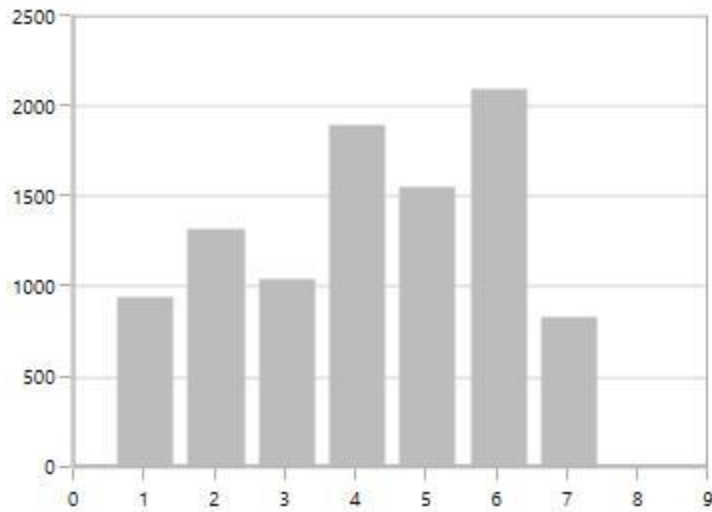
The following screenshot illustrates a chart's y-axis with [RangePadding](#) set to [Normal](#).

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis RangePadding="Normal">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    RangePadding = NumericalPadding.Additional  
};
```



### Additional

If [RangePadding](#) for [NumericalAxis](#) is set to [Additional](#), the interval of the axis is added as padding.

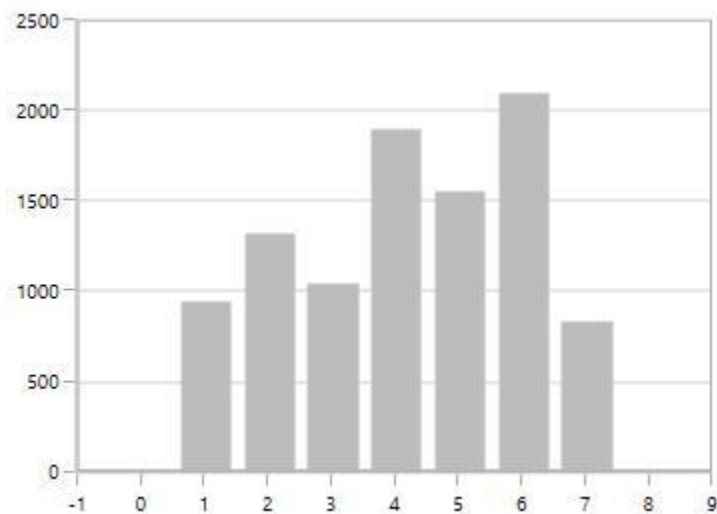
The following screenshot demonstrates a chart's x-axis with [RangePadding](#) set to [Additional](#).

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis RangePadding="Additional">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    RangePadding = NumericalPadding.Additional  
};
```



## None

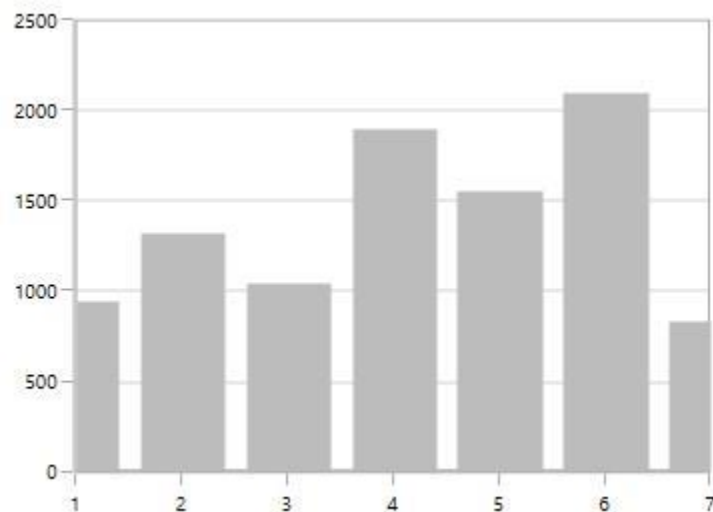
The following screenshot demonstrates [RangePadding](#) as [None](#), where no padding is applied for the axis.

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis RangePadding="None">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    RangePadding = NumericalPadding.None  
};
```



## RoundStart

If [RangePadding](#) for [NumericalAxis](#) is set to [RoundStart](#), rounds the range of the chart axis in the start to the nearest possible value.

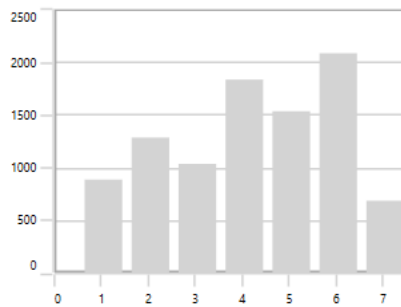
The following screenshot demonstrates a chart's x-axis with [RangePadding](#) set to [RoundStart](#).

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis RangePadding="RoundStart">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    RangePadding = NumericalPadding.RoundStart  
};
```



### RoundEnd

If [RangePadding](#) for [NumericalAxis](#) is set to [RoundEnd](#), rounds the range of the chart axis in the end to the nearest possible value.

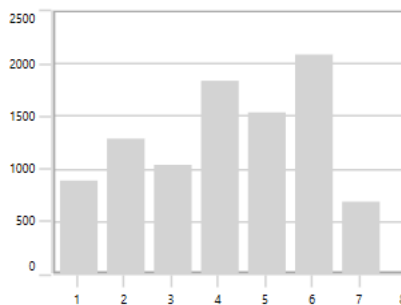
The following screenshot demonstrates a chart's x-axis with [RangePadding](#) set to [RoundEnd](#).

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis RangePadding="RoundEnd">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    RangePadding = NumericalPadding.RoundEnd  
};
```



### PrependInterval

If [RangePadding](#) for [NumericalAxis](#) is set to [PrependInterval](#), the interval of the axis is added in the start as padding.

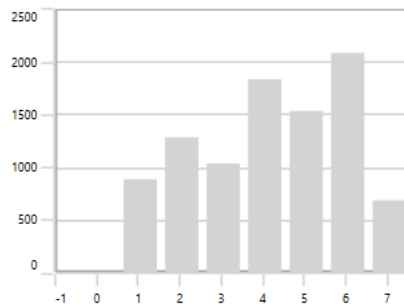
The following screenshot demonstrates a chart's x-axis with [RangePadding](#) set to [PrependInterval](#).

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis RangePadding="PrependInterval">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>
```

**C#**

```
chart.PrimaryAxis = new NumericalAxis()
{
    RangePadding = NumericalPadding.PrependInterval
};
```

**AppendInterval**

If [RangePadding](#) for [NumericalAxis](#) is set to [AppendInterval](#), the interval of the axis is added in the end as padding.

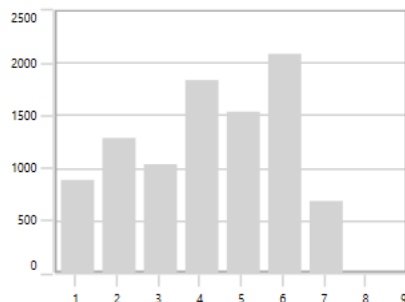
The following screenshot demonstrates a chart's x-axis with [RangePadding](#) set to [AppendInterval](#).

**XML**

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:NumericalAxis RangePadding="AppendInterval">
</syncfusion:NumericalAxis>
</syncfusion:SfChart.PrimaryAxis>
```

**C#**

```
chart.PrimaryAxis = new NumericalAxis()
{
    RangePadding = NumericalPadding.AppendInterval
};
```

**Applying Padding to the Axis**

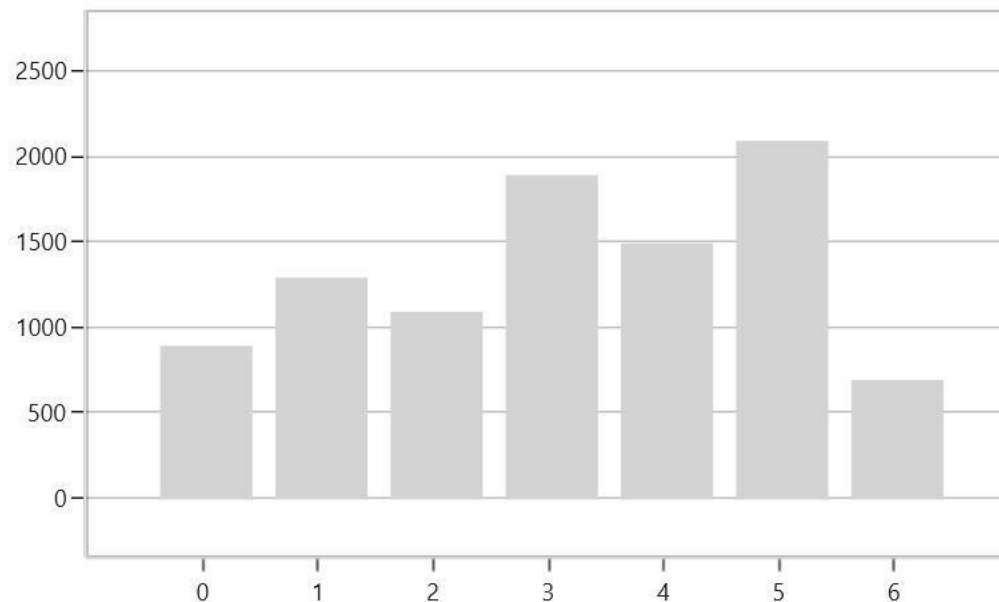
[PlotOffset](#) property is used to provide padding to the axis. The following code snippet demonstrates the padding applied to both x and y axes.

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis PlotOffset="30">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>  
<syncfusion:SfChart.SecondaryAxis>  
<syncfusion:NumericalAxis PlotOffset="30">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.SecondaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    PlotOffset = 30  
};  
chart.SecondaryAxis = new NumericalAxis()  
{  
    PlotOffset = 30  
};
```



#### [PlotOffsetStart](#)

[PlotOffsetStart](#) property is used to provide padding to the axis at start position. The following code snippet demonstrates the padding applied to start position for both x and y axes.

### XML

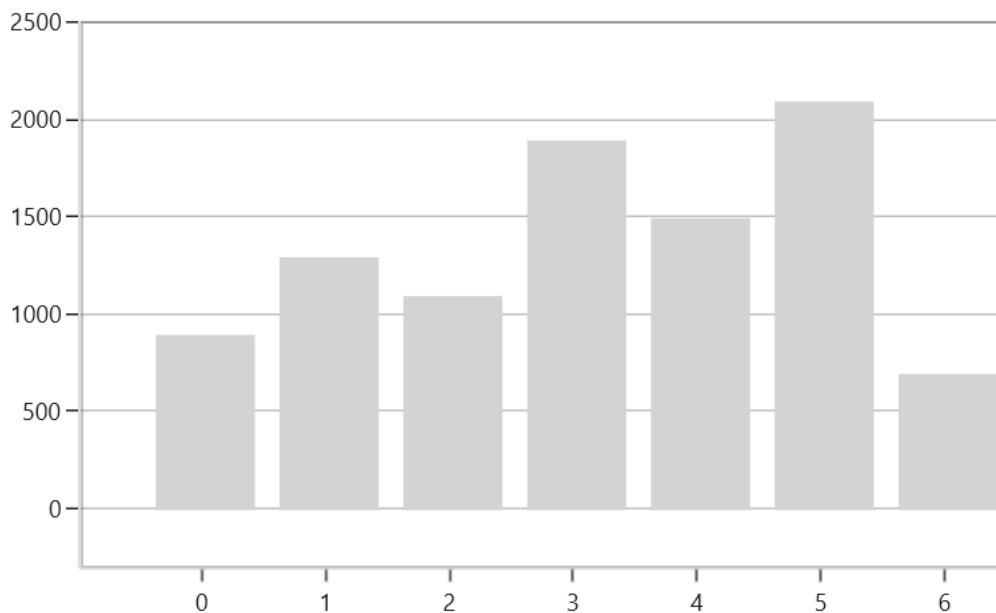
```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis PlotOffsetStart="30">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>  
<syncfusion:SfChart.SecondaryAxis>
```



```
<syncfusion:NumericalAxis PlotOffsetStart="30">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.SecondaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()  
{  
    PlotOffsetStart = 30  
};  
chart.SecondaryAxis = new NumericalAxis()  
{  
    PlotOffsetStart = 30  
};
```



### *PlotOffsetEnd*

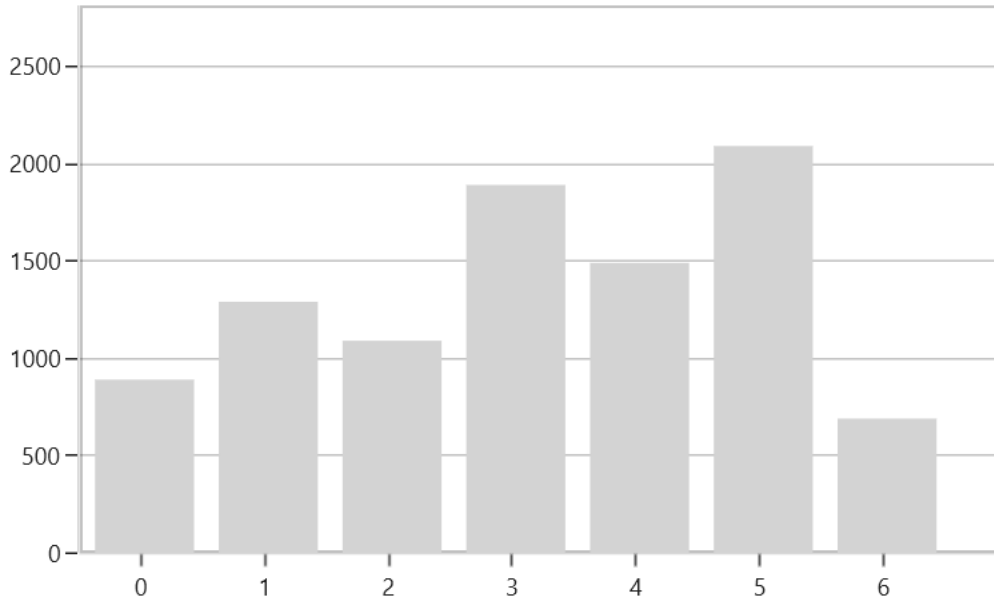
[PlotOffsetEnd](#) property is used to provide padding to the axis at end position. The following code snippet demonstrates the padding applied to end position for both x and y axes.

### XML

```
<syncfusion:SfChart.PrimaryAxis>  
<syncfusion:NumericalAxis PlotOffsetEnd="30">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.PrimaryAxis>  
<syncfusion:SfChart.SecondaryAxis>  
<syncfusion:NumericalAxis PlotOffsetEnd="30">  
</syncfusion:NumericalAxis>  
</syncfusion:SfChart.SecondaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()
{
    PlotOffsetEnd = 30
};
chart.SecondaryAxis = new NumericalAxis()
{
    PlotOffsetEnd = 30
};
```



### AutoScrollingDelta

[AutoScrollingDelta](#) is used to ensure whether the specified range of data is always visible in the chart. It always shows the recently added data points at the end, and the scrolling will be reset to the end of the range whenever a new point is added.

By activating the [EnableScrollBar](#) property of the axis or by adding [ChartZoomPanBehavior](#) to the chart, you can scroll to the previous data points.

### AutoScrollingDeltaType

In [DateTimeAxis](#), you can apply auto scrolling delta value in [Years](#), [Months](#), [Days](#), [Hours](#), [Minutes](#), [Seconds](#) and [Milliseconds](#) by setting the [AutoScrollingDeltaType](#) property. The default value of this property is [Auto](#), and the delta will be calculated automatically based on range.

### XML

```
<chart:SfChart.PrimaryAxis>
<chart:DateTimeAxis AutoScrollingDelta = "3" AutoScrollingDeltaType =
"Days">
</chart:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new DateTimeAxis()
```

```
{
    AutoScrollingDelta = 3,
    AutoScrollingDeltaType = DateTimeIntervalType.Days
};
```

### *AutoScrollingMode*

The [AutoScrollingMode](#) property is used to determine whether the axis should be scrolled from the start position or end position. The default value of this property is [End](#).

### **XML**

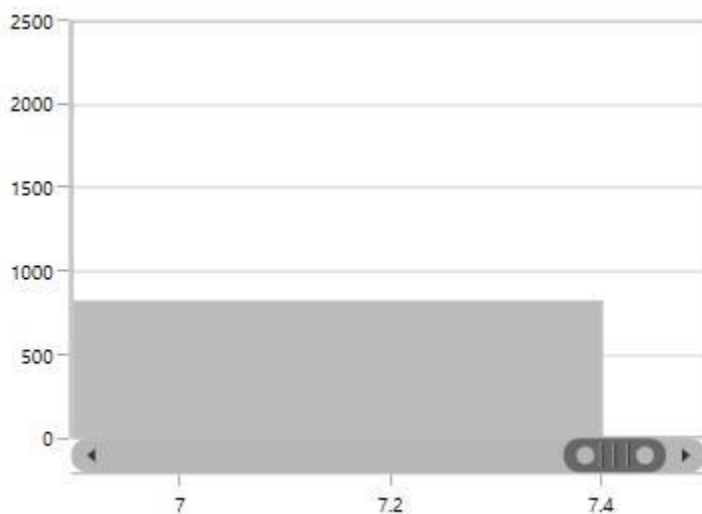
```
<chart:SfChart.PrimaryAxis>
<chart:DateTimeAxis AutoScrollingDelta = "3" AutoScrollingMode = "Start">
</chart:SfChart.PrimaryAxis>
```

### **C#**

```
chart.PrimaryAxis = new DateTimeAxis()
{
    AutoScrollingDelta = 3,
    AutoScrollingMode = ChartAutoScrollingMode.Start
};
```

### Auto Interval Calculation on Zooming

[EnableAutoIntervalOnZooming](#) property is used to maintain the interval even it is in zooming state only if we set the interval to the axis. Default value of this property is true. While zooming based on the auto range padding the interval will be calculated.



If you set [EnableAutoIntervalOnZooming](#) as False, the intervals will be calculated on the interval based on the axis while zooming.

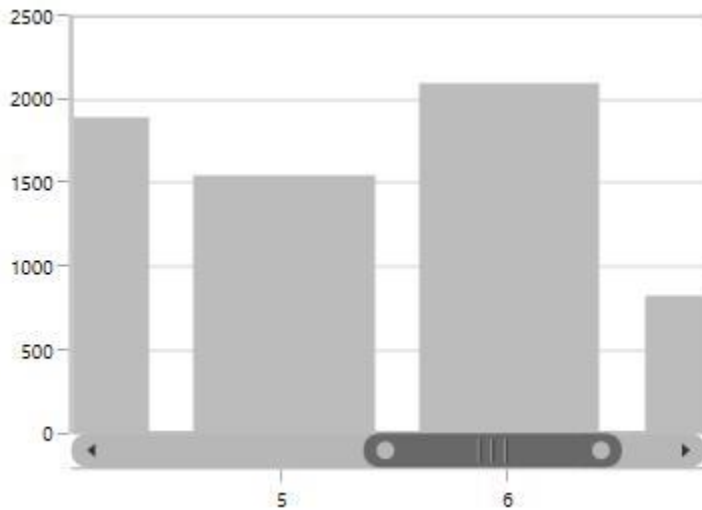
### **XML**

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:NumericalAxis EnableScrollBar="True" Interval="1"
EnableAutoIntervalOnZooming="False">
</syncfusion:NumericalAxis>
```

```
</syncfusion:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new NumericalAxis()
{
    EnableScrollBar = true,
    EnableAutoIntervalOnZooming = false,
    Interval = 1
};
```



### Multiple Axes

SfChart provides a way to arrange multiple series inside the same chart area, giving the chart more space than x-axis and y-axis. These axes can be arranged in a stacking order or in a side by side pattern.

By default, all the series are plotted based on primary and secondary axis. You can add more axes by adding additional axis to the series. There are two properties [XAxis](#) and [YAxis](#) in all the series type which is used to provide Multiple axes support, except [AccumulationSeries](#).

### XML

```
<syncfusion:ColumnSeries ItemsSource="{Binding Demands}"
    XBindingPath="Demand" YBindingPath="Year2011">
</syncfusion:ColumnSeries>
<syncfusion:LineSeries ItemsSource="{Binding Demands}"
    XBindingPath="Date" YBindingPath="Year2011">
<syncfusion:LineSeries.XAxis>
<syncfusion:DateTimeAxis />
</syncfusion:LineSeries.XAxis>
<syncfusion:LineSeries.YAxis>
<syncfusion:NumericalAxis/>
</syncfusion:LineSeries.YAxis>
</syncfusion:LineSeries>
```

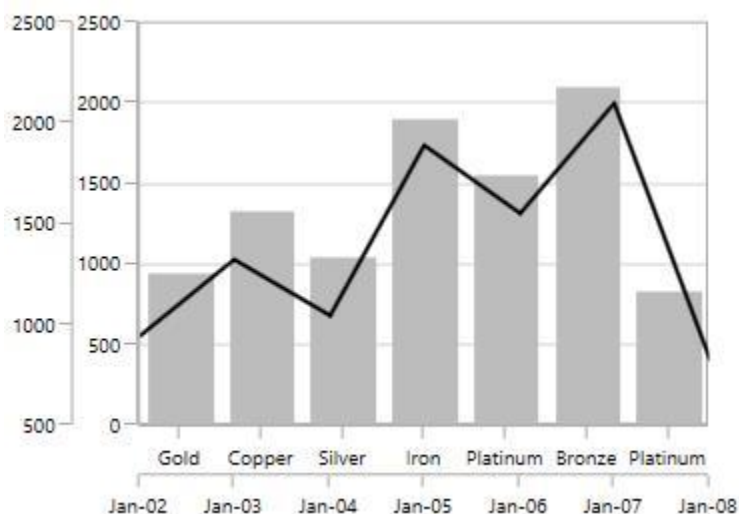
### C#

```
ColumnSeries series1 = new ColumnSeries()
```

```

{
    ItemsSource = new ViewModel().Demands,
    XBindingPath = "Demand",
    YBindingPath = "Year2011"
};
LineSeries series2 = new LineSeries()
{
    ItemsSource = new ViewModel().Demands,
    XBindingPath = "Date",
    YBindingPath = "Year2011",
};
series2.XAxis = new DateTimeAxis()
{
    Header = "Additional X Axis"
};
series2.YAxis = new NumericalAxis()
{
    Header = "Additional Y Axis"
};
chart.Series.Add(series1);
chart.Series.Add(series2);

```



In the above screenshot, the LineSeries is plotted based on additional X & Y axes, and ColumnSeries (or remaining series) is plotted based on the primary and secondary axes.

### Multi-level Labels

[Axis](#) can be customized with multiple levels of label by using its [MultiLevelLabels](#) property. These labels are placed based on the provided [Start](#) and [End](#) range values. You can add any number of labels to an axis. The following code example illustrates how to set a multilevel label.

### XML

```

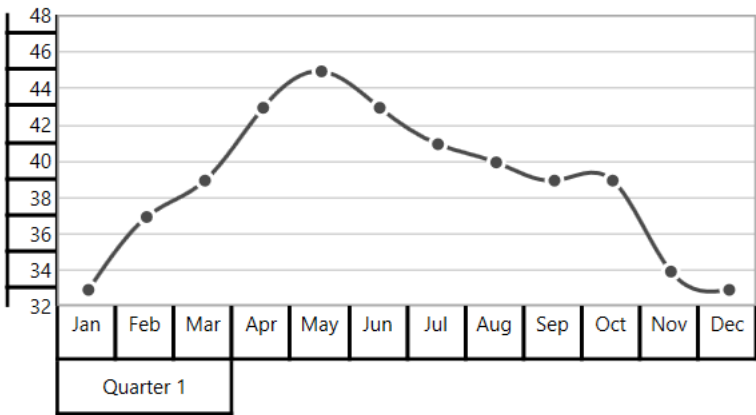
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis ShowLabelBorder="True">
<chart:CategoryAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="-0.5" End="2.5" Text="Quarter 1" />
</chart:CategoryAxis.MultiLevelLabels>
</chart:CategoryAxis>

```

```
</chart:SfChart.PrimaryAxis>
```

C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    ShowLabelBorder = true,
};
ChartMultiLevelLabel label = new ChartMultiLevelLabel()
{
    Start = -0.5,
    End = 2.5,
    Text = "Quarter 1"
};
chart.PrimaryAxis.MultiLevelLabels.Add(label);
```



Regarding Start and End Properties

[Start](#) and [End](#) properties of [ChartMultiLevelLabel](#) are type of objects. You can provide the start and end values for a multi-level label based on its Axis type. It is described in the following table.

S.No	Axis Type	Start/End value	Example
1	CategoryAxis	Index-Based	Start=0(zeroth index position) End = 1(first index position)
2	DateTimeCategoryAxis	Index-Based	Start = 0(zeroth index position) End = 1(first index position)
3	NumericalAxis	Value-Based	Start= 5( Value) End= 10( Value)
4	LogarithmicAxis	Value-Based	Start= 10(Value) End= 1000(Value)
5	DateTimeAxis	Value-Based	Start = "2017/01/01" End="2017/01/02"
6	TimeSpanAxis	Value-Based	Start = "00:00:01" End="00:00:05"

Customizing multi-level labels

Border Customization

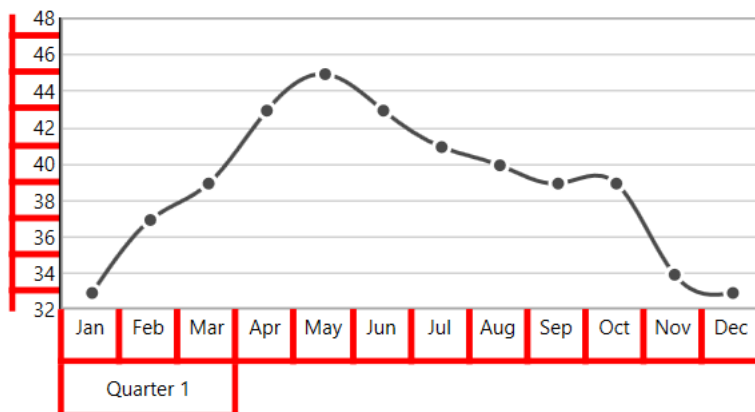
[ChartMultiLevelLabel](#) border width and color can be customized with [LabelBorderWidth](#) and [LabelBorderBrush](#) properties of chart axis. It can be set as shown in the following code example.

#### XML

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis LabelBorderBrush="Red" LabelBorderWidth="3"
ShowLabelBorder="True">
<chart:CategoryAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="-0.5" End="2.5" Text="Quarter 1" />
</chart:CategoryAxis.MultiLevelLabels>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
```

#### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    LabelBorderWidth = 3,
    ShowLabelBorder = true,
    LabelBorderBrush = new SolidColorBrush(Colors.Red),
};
ChartMultiLevelLabel label = new ChartMultiLevelLabel()
{
    Start = -0.5,
    End = 2.5,
    Text = "Quarter 1",
    BorderWidth = 4
};
chart.PrimaryAxis.MultiLevelLabels.Add(label);
```



#### Border Type

[Chart Axis](#) provides support to various types of border for [ChartMultiLevelLabels](#). It can be applied by using its [MultiLevelLabelsBorderType](#) property. The default [MultiLevelLabelsBorderType](#) is [Rectangle](#). The another supported border types are [Brace](#), [None](#) and [WithoutTopAndBottomBorder](#).

#### Rectangle

#### XML

```
<chart:SfChart.PrimaryAxis>
```

```

<chart:CategoryAxis ShowLabelBorder="True">
<chart:CategoryAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="-0.5" End="2.5" Text="Quarter 1" />
<chart:ChartMultiLevelLabel Start="2.5" End="5.5" Text="Quarter 2"/>
<chart:ChartMultiLevelLabel Start="5.5" End="8.5" Text="Quarter 3"/>
<chart:ChartMultiLevelLabel Start="8.5" End="11.5" Text="Quarter 4"/>
</chart:CategoryAxis.MultiLevelLabels>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis ShowLabelBorder="True">
<chart:NumericalAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="32" End="36" Text="Low"/>
<chart:ChartMultiLevelLabel Start="36" End="42" Text="Medium"/>
<chart:ChartMultiLevelLabel Start="42" End="48" Text="High"/>
</chart:NumericalAxis.MultiLevelLabels>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>

```

**C#**

```

chart.PrimaryAxis = new CategoryAxis()
{
    ShowLabelBorder = true,
};
ChartMultiLevelLabel label1 = new ChartMultiLevelLabel()
{
    Start = -0.5,
    End = 2.5,
    Text = "Quarter 1",
};
chart.PrimaryAxis.MultiLevelLabels.Add(label1);
ChartMultiLevelLabel label2 = new ChartMultiLevelLabel()
{
    Start = 2.5,
    End = 5.5,
    Text = "Quarter 2"
};
chart.PrimaryAxis.MultiLevelLabels.Add(label2);
ChartMultiLevelLabel label3 = new ChartMultiLevelLabel()
{
    Start = 5.5,
    End = 8.5,
    Text = "Quarter 3"
};
chart.PrimaryAxis.MultiLevelLabels.Add(label3);
ChartMultiLevelLabel label4 = new ChartMultiLevelLabel()
{
    Start = 8.5,
    End = 11.5,
    Text = "Quarter 4"
};
chart.PrimaryAxis.MultiLevelLabels.Add(label4);
chart.SecondaryAxis = new NumericalAxis()
{
    ShowLabelBorder = true,
};

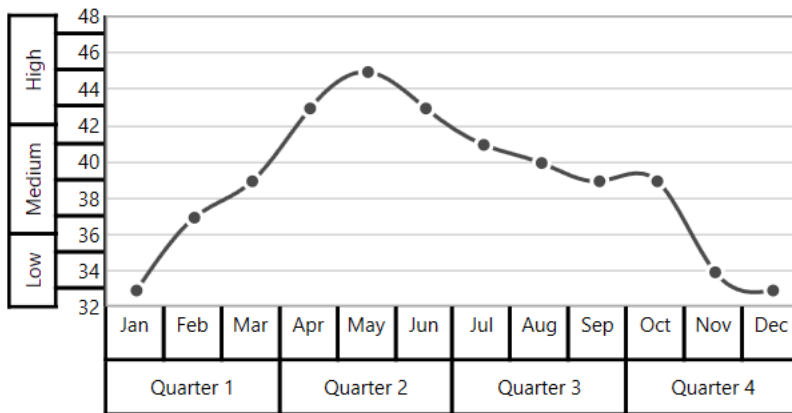
```



```

};
ChartMultiLevelLabel label5 = new ChartMultiLevelLabel()
{
    Start = 32,
    End = 36,
    Text = "Low"
};
chart.SecondaryAxis.MultiLevelLabels.Add(label5);
ChartMultiLevelLabel label6 = new ChartMultiLevelLabel()
{
    Start = 36,
    End = 42,
    Text = "Medium"
};
chart.SecondaryAxis.MultiLevelLabels.Add(label6);
ChartMultiLevelLabel label7 = new ChartMultiLevelLabel()
{
    Start = 42,
    End = 48,
    Text = "High"
};
chart.SecondaryAxis.MultiLevelLabels.Add(label7);

```



## Brace

### XML

```

<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis MultiLevelLabelsBorderType="Brace"
ShowLabelBorder="True">
<chart:CategoryAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="-0.5" End="2.5" Text="Quarter 1" />
<chart:ChartMultiLevelLabel Start="2.5" End="5.5" Text="Quarter 2" />
<chart:ChartMultiLevelLabel Start="5.5" End="8.5" Text="Quarter 3" />
<chart:ChartMultiLevelLabel Start="8.5" End="11.5" Text="Quarter 4" />
</chart:CategoryAxis.MultiLevelLabels>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis MultiLevelLabelsBorderType="Brace"
ShowLabelBorder="True">
<chart:NumericalAxis.MultiLevelLabels>

```

```

<chart:ChartMultiLevelLabel Start="32" End="36" Text="Low" />
<chart:ChartMultiLevelLabel Start="36" End="42" Text="Medium"/>
<chart:ChartMultiLevelLabel Start="42" End="48" Text="High" />
</chart:NumericalAxis.MultiLevelLabels>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>

```

**C#**

```

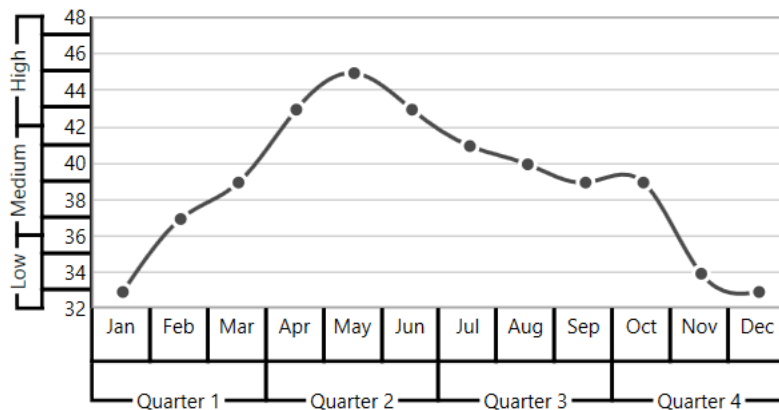
chart.PrimaryAxis = new CategoryAxis()
{
    ShowLabelBorder = true,
    MultiLevelLabelsBorderStyle = BorderType.Brace
};
ChartMultiLevelLabel label1 = new ChartMultiLevelLabel()
{
    Start = -0.5,
    End = 2.5,
    Text = "Quarter 1",
};
chart.PrimaryAxis.MultiLevelLabels.Add(label1);
ChartMultiLevelLabel label2 = new ChartMultiLevelLabel()
{
    Start = 2.5,
    End = 5.5,
    Text = "Quarter 2",
};
chart.PrimaryAxis.MultiLevelLabels.Add(label2);
ChartMultiLevelLabel label3 = new ChartMultiLevelLabel()
{
    Start = 5.5,
    End = 8.5,
    Text = "Quarter 3",
};
chart.PrimaryAxis.MultiLevelLabels.Add(label3);
ChartMultiLevelLabel label4 = new ChartMultiLevelLabel()
{
    Start = 8.5,
    End = 11.5,
    Text = "Quarter 4",
};
chart.PrimaryAxis.MultiLevelLabels.Add(label4);
chart.SecondaryAxis = new NumericalAxis()
{
    ShowLabelBorder = true,
    MultiLevelLabelsBorderStyle = BorderType.Brace
};
ChartMultiLevelLabel label5 = new ChartMultiLevelLabel()
{
    Start = 32,
    End = 36,
    Text = "Low",
};
chart.SecondaryAxis.MultiLevelLabels.Add(label5);
ChartMultiLevelLabel label6 = new ChartMultiLevelLabel()
{

```

```

Start = 36,
End = 42,
Text = "Medium",
};
chart.SecondaryAxis.MultiLevelLabels.Add(label6);
ChartMultiLevelLabel label7 = new ChartMultiLevelLabel()
{
    Start = 42,
    End = 48,
    Text = "High",
};
chart.SecondaryAxis.MultiLevelLabels.Add(label7);

```



None

**XML**

```

<chart:SfChart.PrimaryAxis MultiLevelLabelsBorderType="None"
ShowLabelBorder="True">
<chart:CategoryAxis>
<chart:CategoryAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="-0.5" End="2.5" Text="Quarter 1" />
<chart:ChartMultiLevelLabel Start="2.5" End="5.5" Text="Quarter 2" />
<chart:ChartMultiLevelLabel Start="5.5" End="8.5" Text="Quarter 3" />
<chart:ChartMultiLevelLabel Start="8.5" End="11.5" Text="Quarter 4" />
</chart:CategoryAxis.MultiLevelLabels>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis MultiLevelLabelsBorderType="None"
ShowLabelBorder="True">
<chart:NumericalAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="32" End="36" Text="Low" />
<chart:ChartMultiLevelLabel Start="36" End="42" Text="Medium" />
<chart:ChartMultiLevelLabel Start="42" End="48" Text="High" />
</chart:NumericalAxis.MultiLevelLabels>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>

```

**C#**

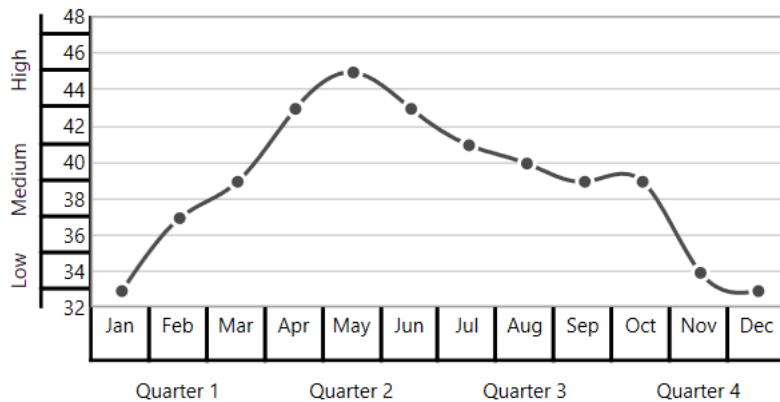
```

chart.PrimaryAxis = new CategoryAxis()

```

```
{
    ShowLabelBorder = true,
    MultiLevelLabelsBorderType = BorderType.None
};
ChartMultiLevelLabel label1 = new ChartMultiLevelLabel()
{
    Start = -0.5,
    End = 2.5,
    Text = "Quarter 1"
};
chart.PrimaryAxis.MultiLevelLabels.Add(label1);
ChartMultiLevelLabel label2 = new ChartMultiLevelLabel()
{
    Start = 2.5,
    End = 5.5,
    Text = "Quarter 2"
};
chart.PrimaryAxis.MultiLevelLabels.Add(label2);
ChartMultiLevelLabel label3 = new ChartMultiLevelLabel()
{
    Start = 5.5,
    End = 8.5,
    Text = "Quarter 3"
};
chart.PrimaryAxis.MultiLevelLabels.Add(label3);
ChartMultiLevelLabel label4 = new ChartMultiLevelLabel()
{
    Start = 8.5,
    End = 11.5,
    Text = "Quarter 4"
};
chart.PrimaryAxis.MultiLevelLabels.Add(label4);
chart.SecondaryAxis = new NumericalAxis()
{
    ShowLabelBorder = true,
    MultiLevelLabelsBorderType = BorderType.None
};
ChartMultiLevelLabel label5 = new ChartMultiLevelLabel()
{
    Start = 32,
    End = 36,
    Text = "Low"
};
chart.SecondaryAxis.MultiLevelLabels.Add(label5);
ChartMultiLevelLabel label6 = new ChartMultiLevelLabel()
{
    Start = 36,
    End = 42,
    Text = "Medium"
};
chart.SecondaryAxis.MultiLevelLabels.Add(label6);
ChartMultiLevelLabel label7 = new ChartMultiLevelLabel()
{
    Start = 42,
    End = 48,
    Text = "High"
};
};
```

```
chart.SecondaryAxis.MultiLevelLabels.Add(label7);
```



### WithoutTopAndBottomBorder

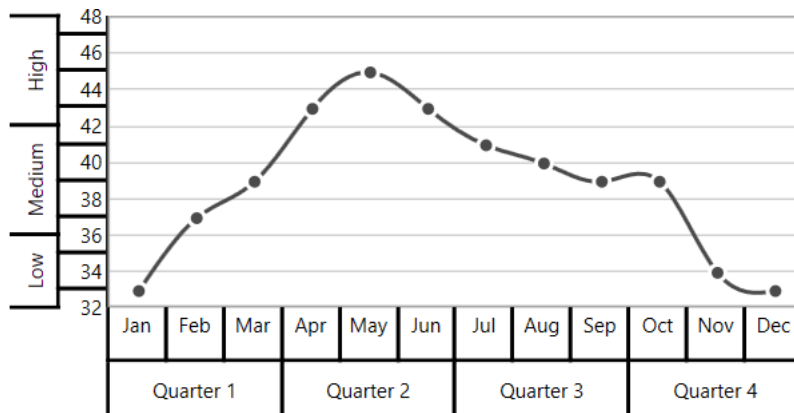
#### XML

```
<chart:SfChart.PrimaryAxis>
  <chart:CategoryAxis ShowLabelBorder="True"
    MultiLevelLabelsBorderType="WithoutTopAndBottomBorder">
    <chart:CategoryAxis.MultiLevelLabels>
      <chart:ChartMultiLevelLabel Start="-0.5" End="2.5" Text="Quarter 1"/>
      <chart:ChartMultiLevelLabel Start="2.5" End="5.5" Text="Quarter 2" />
      <chart:ChartMultiLevelLabel Start="5.5" End="8.5" Text="Quarter 3" />
      <chart:ChartMultiLevelLabel Start="8.5" End="11.5" Text="Quarter 4" />
    </chart:CategoryAxis.MultiLevelLabels>
    </chart:CategoryAxis>
  </chart:SfChart.PrimaryAxis>
  <chart:SfChart.SecondaryAxis>
    <chart:NumericalAxis ShowLabelBorder="True"
      MultiLevelLabelsBorderType="WithoutTopAndBottomBorder">
      <chart:NumericalAxis.MultiLevelLabels>
        <chart:ChartMultiLevelLabel Start="32" End="36" Text="Low" />
        <chart:ChartMultiLevelLabel Start="36" End="42" Text="Medium"/>
        <chart:ChartMultiLevelLabel Start="42" End="48" Text="High"/>
      </chart:NumericalAxis.MultiLevelLabels>
    </chart:NumericalAxis>
  </chart:SfChart.SecondaryAxis>
```

#### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    ShowLabelBorder = true,
    MultiLevelLabelsBorderType = BorderType.WithoutTopAndBottomBorder
};
ChartMultiLevelLabel label1 = new ChartMultiLevelLabel()
{
    Start = -0.5,
    End = 2.5,
    Text = "Quarter 1",
};
chart.PrimaryAxis.MultiLevelLabels.Add(label1);
```

```
ChartMultiLevelLabel label2 = new ChartMultiLevelLabel()
{
    Start = 2.5,
    End = 5.5,
    Text = "Quarter 2",
};
chart.PrimaryAxis.MultiLevelLabels.Add(label2);
ChartMultiLevelLabel label3 = new ChartMultiLevelLabel()
{
    Start = 5.5,
    End = 8.5,
    Text = "Quarter 3",
};
chart.PrimaryAxis.MultiLevelLabels.Add(label3);
ChartMultiLevelLabel label4 = new ChartMultiLevelLabel()
{
    Start = 8.5,
    End = 11.5,
    Text = "Quarter 4",
};
chart.PrimaryAxis.MultiLevelLabels.Add(label4);
chart.SecondaryAxis = new NumericalAxis()
{
    ShowLabelBorder = true,
    MultiLevelLabelsBorderStyle = BorderType.WithoutTopAndBottomBorder
};
ChartMultiLevelLabel label5 = new ChartMultiLevelLabel()
{
    Start = 32,
    End = 36,
    Text = "Low",
};
chart.SecondaryAxis.MultiLevelLabels.Add(label5);
ChartMultiLevelLabel label6 = new ChartMultiLevelLabel()
{
    Start = 36,
    End = 42,
    Text = "Medium",
};
chart.SecondaryAxis.MultiLevelLabels.Add(label6);
ChartMultiLevelLabel label7 = new ChartMultiLevelLabel()
{
    Start = 42,
    End = 48,
    Text = "High",
};
chart.SecondaryAxis.MultiLevelLabels.Add(label7);
```



### Text Customization

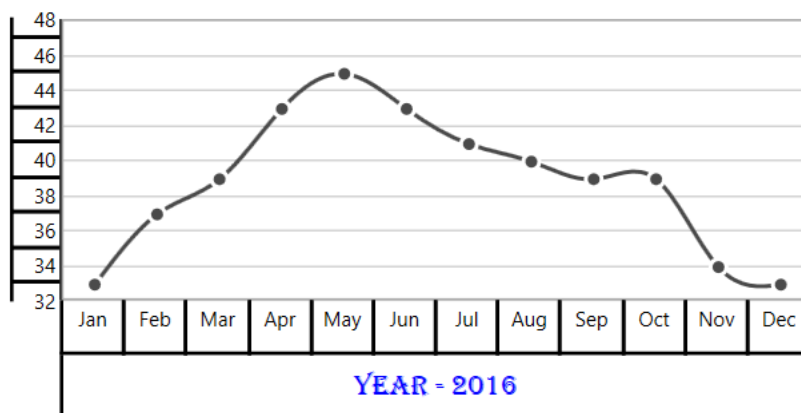
[ChartMultiLevelLabel](#) text can be customized with its [FontSize](#), [FontFamily](#) and [Foreground](#) properties. It is shown in following code example.

### XML

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis ShowLabelBorder="True">
<chart:CategoryAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="-0.5" End="11.5" FontFamily="Algerian"
Foreground="Blue" FontSize="14" Text="Year - 2016"/>
</chart:CategoryAxis.MultiLevelLabels>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    ShowLabelBorder = true,
};
ChartMultiLevelLabel label = new ChartMultiLevelLabel()
{
    Start = -0.5,
    End = 11.5,
    Text = "Year - 2016",
    Foreground = new SolidColorBrush(Colors.Blue),
    FontSize = 14,
    FontFamily = new FontFamily("Algerian")
};
chart.PrimaryAxis.MultiLevelLabels.Add(label);
```



### Label Alignment

The text of [ChartMultiLevelLabel](#) can be aligned with its [LabelAlignment](#) property. The default value of [LabelAlignment](#) property is Center.

### Center

### XML

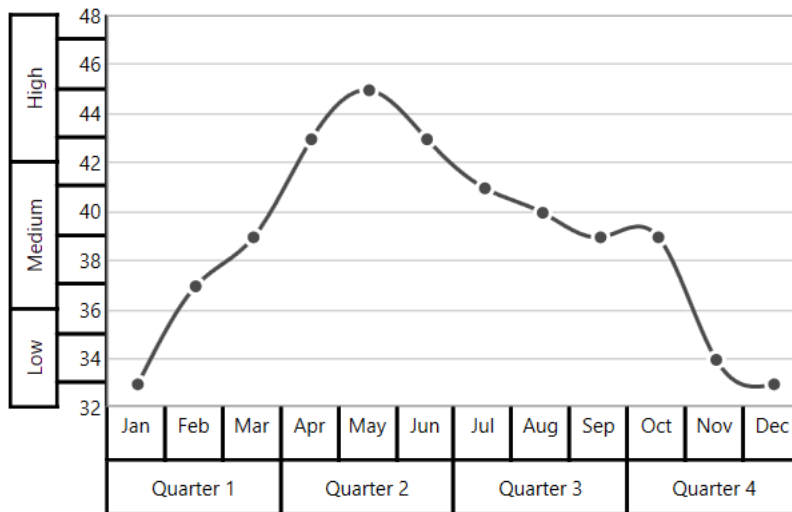
```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis ShowLabelBorder="True">
<chart:CategoryAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="-0.5" End="2.5" Text="Quarter 1" />
<chart:ChartMultiLevelLabel Start="2.5" End="5.5" Text="Quarter 2"/>
<chart:ChartMultiLevelLabel Start="5.5" End="8.5" Text="Quarter 3"/>
<chart:ChartMultiLevelLabel Start="8.5" End="11.5" Text="Quarter 4"/>
</chart:CategoryAxis.MultiLevelLabels>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis ShowLabelBorder="True">
<chart:NumericalAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="32" End="36" Text="Low"/>
<chart:ChartMultiLevelLabel Start="36" End="42" Text="Medium"/>
<chart:ChartMultiLevelLabel Start="42" End="48" Text="High"/>
</chart:NumericalAxis.MultiLevelLabels>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    ShowLabelBorder = true,
};
ChartMultiLevelLabel label1 = new ChartMultiLevelLabel()
{
    Start = -0.5,
    End = 2.5,
    Text = "Quarter 1",
};
chart.PrimaryAxis.MultiLevelLabels.Add(label1);
ChartMultiLevelLabel label2 = new ChartMultiLevelLabel()
```



```
{
    Start = 2.5,
    End = 5.5,
    Text = "Quarter 2"
};
chart.PrimaryAxis.MultiLevelLabels.Add(label2);
ChartMultiLevelLabel label3 = new ChartMultiLevelLabel()
{
    Start = 5.5,
    End = 8.5,
    Text = "Quarter 3"
};
chart.PrimaryAxis.MultiLevelLabels.Add(label3);
ChartMultiLevelLabel label4 = new ChartMultiLevelLabel()
{
    Start = 8.5,
    End = 11.5,
    Text = "Quarter 4"
};
chart.PrimaryAxis.MultiLevelLabels.Add(label4);
chart.SecondaryAxis = new NumericalAxis()
{
    ShowLabelBorder = true,
};
ChartMultiLevelLabel label5 = new ChartMultiLevelLabel()
{
    Start = 32,
    End = 36,
    Text = "Low"
};
chart.SecondaryAxis.MultiLevelLabels.Add(label5);
ChartMultiLevelLabel label6 = new ChartMultiLevelLabel()
{
    Start = 36,
    End = 42,
    Text = "Medium"
};
chart.SecondaryAxis.MultiLevelLabels.Add(label6);
ChartMultiLevelLabel label7 = new ChartMultiLevelLabel()
{
    Start = 42,
    End = 48,
    Text = "High"
};
chart.SecondaryAxis.MultiLevelLabels.Add(label7);
```



Near

XML

```

<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis ShowLabelBorder="True">
<chart:CategoryAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="-0.5" End="2.5" Text="Quarter 1"
LabelAlignment="Near" />
<chart:ChartMultiLevelLabel Start="2.5" End="5.5" Text="Quarter 2"
LabelAlignment="Near"/>
<chart:ChartMultiLevelLabel Start="5.5" End="8.5" Text="Quarter 3"
LabelAlignment="Near"/>
<chart:ChartMultiLevelLabel Start="8.5" End="11.5" Text="Quarter 4"
LabelAlignment="Near"/>
</chart:CategoryAxis.MultiLevelLabels>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis ShowLabelBorder="True">
<chart:NumericalAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="32" End="36" Text="Low"
LabelAlignment="Near"/>
<chart:ChartMultiLevelLabel Start="36" End="42" Text="Medium"
LabelAlignment="Near"/>
<chart:ChartMultiLevelLabel Start="42" End="48" Text="High"
LabelAlignment="Near"/>
</chart:NumericalAxis.MultiLevelLabels>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>

```

C#

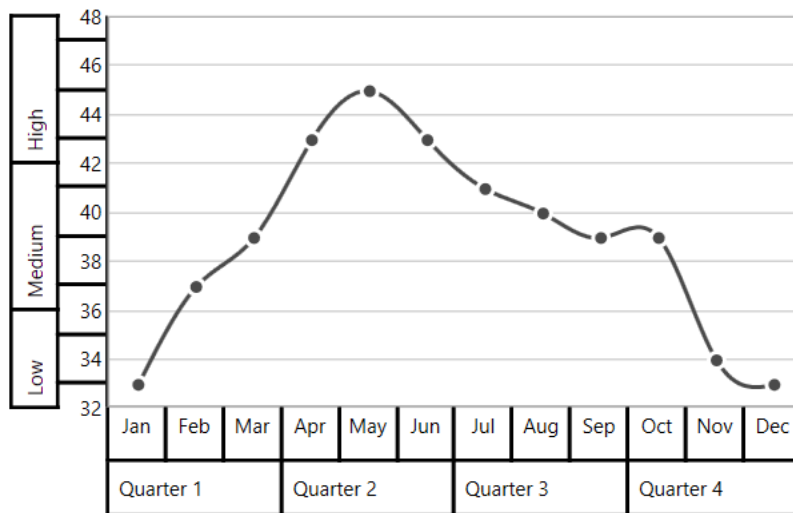
```

chart.PrimaryAxis = new CategoryAxis()
{
    ShowLabelBorder = true,
};
ChartMultiLevelLabel label1 = new ChartMultiLevelLabel()
{

```

```
Start = -0.5,
End = 2.5,
Text = "Quarter 1",
LabelAlignment = LabelAlignment.Near
};
chart.PrimaryAxis.MultiLevelLabels.Add(label1);
ChartMultiLevelLabel label2 = new ChartMultiLevelLabel()
{
    Start = 2.5,
    End = 5.5,
    Text = "Quarter 2",
    LabelAlignment = LabelAlignment.Near
};
chart.PrimaryAxis.MultiLevelLabels.Add(label2);
ChartMultiLevelLabel label3 = new ChartMultiLevelLabel()
{
    Start = 5.5,
    End = 8.5,
    Text = "Quarter 3",
    LabelAlignment = LabelAlignment.Near
};
chart.PrimaryAxis.MultiLevelLabels.Add(label3);
ChartMultiLevelLabel label4 = new ChartMultiLevelLabel()
{
    Start = 8.5,
    End = 11.5,
    Text = "Quarter 4",
    LabelAlignment = LabelAlignment.Near
};
chart.PrimaryAxis.MultiLevelLabels.Add(label4);
chart.SecondaryAxis = new NumericalAxis()
{
    ShowLabelBorder = true,
};
ChartMultiLevelLabel label5 = new ChartMultiLevelLabel()
{
    Start = 32,
    End = 36,
    Text = "Low",
    LabelAlignment = LabelAlignment.Near
};
chart.SecondaryAxis.MultiLevelLabels.Add(label5);
ChartMultiLevelLabel label6 = new ChartMultiLevelLabel()
{
    Start = 36,
    End = 42,
    Text = "Medium",
    LabelAlignment = LabelAlignment.Near
};
chart.SecondaryAxis.MultiLevelLabels.Add(label6);
ChartMultiLevelLabel label7 = new ChartMultiLevelLabel()
{
    Start = 42,
    End = 48,
    Text = "High",
    LabelAlignment = LabelAlignment.Near
};
```

```
chart.SecondaryAxis.MultiLevelLabels.Add(label7);
```



Far

### XML

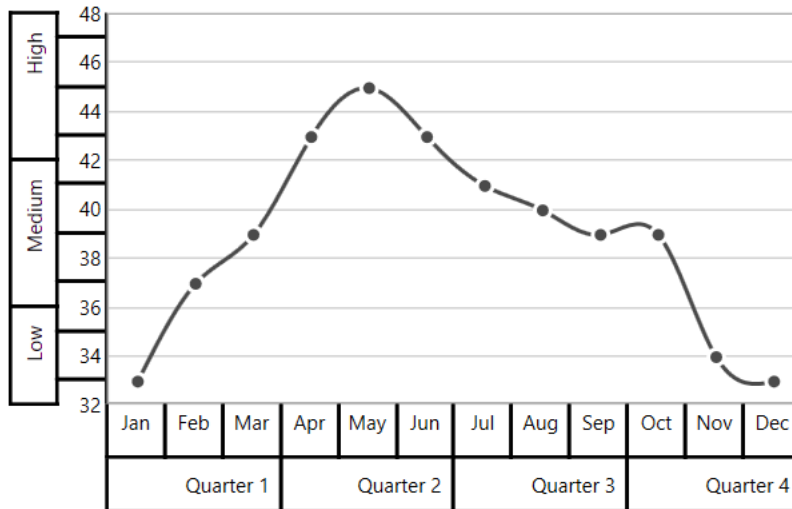
```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis ShowLabelBorder="True">
<chart:CategoryAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="-0.5" End="2.5" Text="Quarter 1"
LabelAlignment="Far" />
<chart:ChartMultiLevelLabel Start="2.5" End="5.5" Text="Quarter 2"
LabelAlignment="Far"/>
<chart:ChartMultiLevelLabel Start="5.5" End="8.5" Text="Quarter 3"
LabelAlignment="Far"/>
<chart:ChartMultiLevelLabel Start="8.5" End="11.5" Text="Quarter 4"
LabelAlignment="Far"/>
</chart:CategoryAxis.MultiLevelLabels>
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis ShowLabelBorder="True">
<chart:NumericalAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="32" End="36" Text="Low"
LabelAlignment="Far"/>
<chart:ChartMultiLevelLabel Start="36" End="42" Text="Medium"
LabelAlignment="Far"/>
<chart:ChartMultiLevelLabel Start="42" End="48" Text="High"
LabelAlignment="Far"/>
</chart:NumericalAxis.MultiLevelLabels>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    ShowLabelBorder = true,
```

```
};
ChartMultiLevelLabel label1 = new ChartMultiLevelLabel()
{
    Start = -0.5,
    End = 2.5,
    Text = "Quarter 1",
    LabelAlignment = LabelAlignment.Near
};
chart.PrimaryAxis.MultiLevelLabels.Add(label1);
ChartMultiLevelLabel label2 = new ChartMultiLevelLabel()
{
    Start = 2.5,
    End = 5.5,
    Text = "Quarter 2",
    LabelAlignment = LabelAlignment.Near
};
chart.PrimaryAxis.MultiLevelLabels.Add(label2);
ChartMultiLevelLabel label3 = new ChartMultiLevelLabel()
{
    Start = 5.5,
    End = 8.5,
    Text = "Quarter 3",
    LabelAlignment = LabelAlignment.Near
};
chart.PrimaryAxis.MultiLevelLabels.Add(label3);
ChartMultiLevelLabel label4 = new ChartMultiLevelLabel()
{
    Start = 8.5,
    End = 11.5,
    Text = "Quarter 4",
    LabelAlignment = LabelAlignment.Near
};
chart.PrimaryAxis.MultiLevelLabels.Add(label4);
chart.SecondaryAxis = new NumericalAxis()
{
    ShowLabelBorder = true,
};
ChartMultiLevelLabel label5 = new ChartMultiLevelLabel()
{
    Start = 32,
    End = 36,
    Text = "Low",
    LabelAlignment = LabelAlignment.Near
};
chart.SecondaryAxis.MultiLevelLabels.Add(label5);
ChartMultiLevelLabel label6 = new ChartMultiLevelLabel()
{
    Start = 36,
    End = 42,
    Text = "Medium",
    LabelAlignment = LabelAlignment.Near
};
chart.SecondaryAxis.MultiLevelLabels.Add(label6);
ChartMultiLevelLabel label7 = new ChartMultiLevelLabel()
{
    Start = 42,
    End = 48,
```

```
Text = "High",
LabelAlignment = LabelAlignment.Near
};
chart.SecondaryAxis.MultiLevelLabels.Add(label7);
```



The text of [ChartMultiLevelLabel](#) will be trimmed automatically when the text width exceeds the width of [ChartMultiLevelLabel](#), and it is shown below.

### XML

```
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis ShowLabelBorder="True">
<chart:NumericalAxis.MultiLevelLabels>
<chart:ChartMultiLevelLabel Start="32" End="36" Text="Low Temperature"/>
<chart:ChartMultiLevelLabel Start="36" End="42" Text="Medium Temperature"/>
<chart:ChartMultiLevelLabel Start="42" End="48" Text="High Temperature"/>
</chart:NumericalAxis.MultiLevelLabels>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>
```

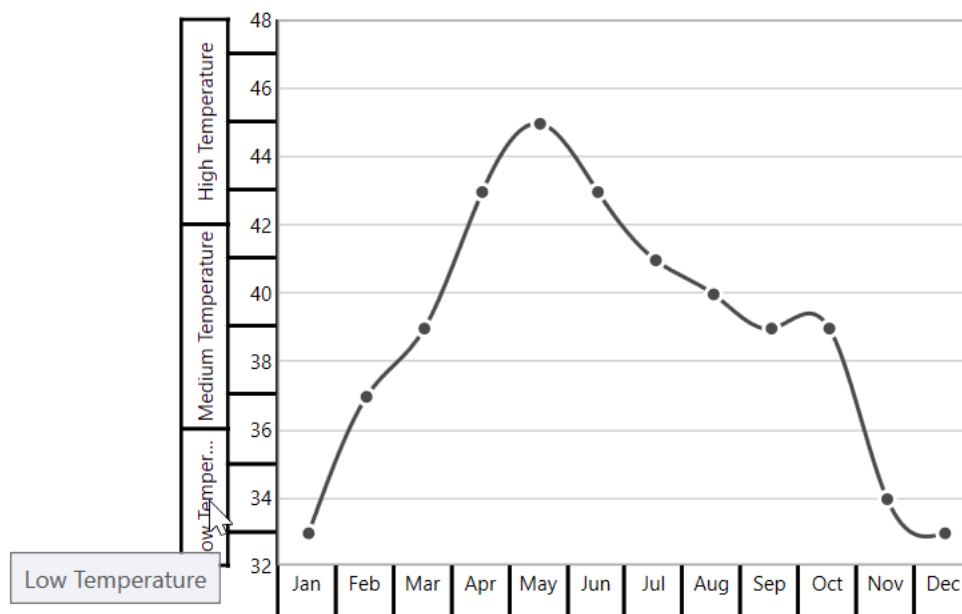
### C#

```
chart.SecondaryAxis = new NumericalAxis()
{
    ShowLabelBorder = true,
};
ChartMultiLevelLabel label5 = new ChartMultiLevelLabel()
{
    Start = 32,
    End = 36,
    Text = "Low Temperature"
};
chart.SecondaryAxis.MultiLevelLabels.Add(label5);
ChartMultiLevelLabel label6 = new ChartMultiLevelLabel()
{
    Start = 36,
    End = 42,
    Text = "Medium Temperature"
};
```

```

};
chart.SecondaryAxis.MultiLevelLabels.Add(label6);
ChartMultiLevelLabel label7 = new ChartMultiLevelLabel()
{
    Start = 42,
    End = 48,
    Text = "High Temperature"
};
chart.SecondaryAxis.MultiLevelLabels.Add(label7);

```



## Events

### *ActualRangeChanged*

The [ActualRangeChanged](#) event occurs when an axis range is changed. This argument contains the following information.

- [ActualMinimum](#) - Gets or sets the actual minimum value of axis.
- [ActualMaximum](#) - Gets or sets the actual maximum value of axis.
- [VisibleMinimum](#) - Gets or sets the visible minimum value of axis.
- [VisibleMaximum](#) - Gets or sets the visible maximum value of axis.
- [ActualInterval](#) - Gets the actual interval of axis.

### *LabelCreated*

The [LabelCreated](#) event occurs when the axis label is created. This argument contains [AxisLabel](#) of [ChartAxisLabel](#), which contains the following properties.

- [LabelContent](#) - Gets or sets the content of label.
- [Position](#) - Gets or sets the position of label.

### *AxisBoundsChanged*

The [AxisBoundsChanged](#) event occurs when the bounds of the axis are changed. This argument contains the following information.

- [NewBounds](#) - Gets the new axis bounds.
- [OldBounds](#) - Gets the old axis bounds.

#### *LabelClicked*

The [LabelClicked](#) event is triggered when labels are clicked. Supports for 2D axis. The argument contains [AxisLabel](#) of [ChartAxisLabel](#), which contains the following properties.

- [LabelContent](#) - Gets the content of label.
- [Position](#) - Gets the position of label.
- [PrefixLabelTemplate](#) - Gets the prefix template of label.
- [PostfixLabelTemplate](#) - Gets the postfix template of label.

---

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

---

See also

[How to get axis range in chart](#)

[How to wrap the multi-level axis label's text in WPF Chart](#)

[How to use the DateTimeAxis as SecondaryAxis in WPF Chart](#)

[How to apply the custom labels in WPF Logarithmic Axis](#)

[How to customize label formats of date-time axis during the interval transitions](#)

[How to rotate axis label in the Chart](#)

[How to hide axis in chart](#)

[How to position the primary and secondary axes as the center of chart](#)

[How to display the axis labels in a particular format](#)

[How to display the axis labels for all datapoints](#)

[How to define ticker labels of custom axis](#)

[How to display striplines in DateTimeAxis of WPF Chart](#)

[How to set the custom labels with auto range for axis](#)

#### Legend in WPF Charts (SfChart)

[Legend](#) provides metadata which helps for identifying elements in chart like [chart series](#), [technical indicators](#), and [trendlines](#).

You can define the legend using the following code example.

#### **XML**

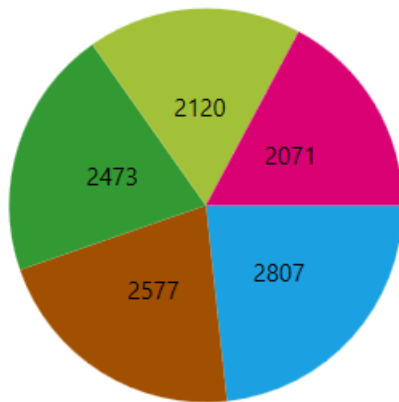
```
<chart:SfChart.Legend>  
<chart:ChartLegend />  
</chart:SfChart.Legend>
```

#### **C#**

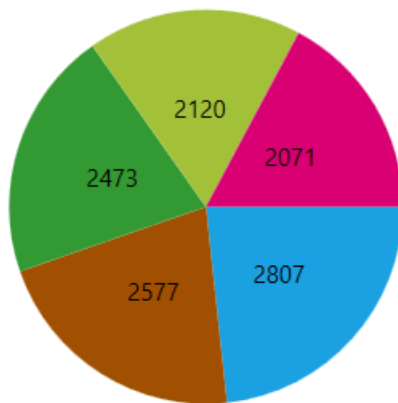
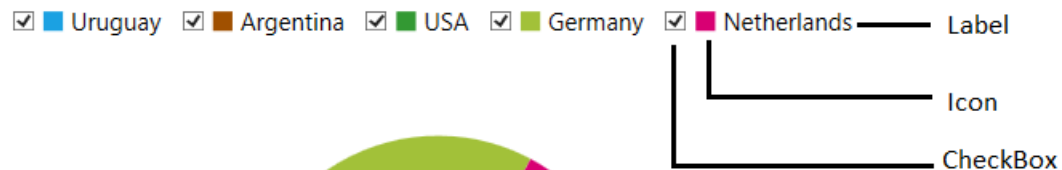


```
SfChart chart = new SfChart();
chart.Legend = new ChartLegend();
```

■ Uruguay ■ Argentina ■ USA ■ Germany ■ Netherlands



Each legend composed of the following parts:



### Legend Icon

Represents the symbol associated with each legend item. By default, the legend icon is [Rectangle](#).

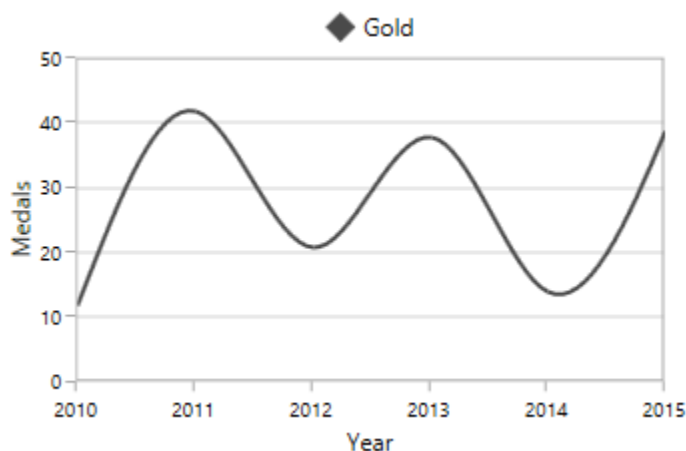
This can be customized using the [LegendIcon](#) property in any series as in below code snippet:

### XML

```
<chart:SfChart.Legend>
<chart:ChartLegend/>
</chart:ChartLegend>
<chart:SplineSeries XBindingPath="Year" Label="Gold"
ItemsSource="{Binding List}" YBindingPath="India"
LegendIcon="Diamond" />
```

**C#**

```
chart.Legend = new ChartLegend();
SplineSeries splineSeries = new SplineSeries()
{
    Label = "Gold",
    ItemsSource = new ViewModel().List,
    XBindingPath = "Year",
    YBindingPath = "India",
    LegendIcon = ChartLegendIcon.Diamond
};
chart.Series.Add(splineSeries);
```



The following properties are used to customize the legend icons.

- [IconWidth](#)-Gets or sets the double value that represents the legend icon(s) width.
- [IconHeight](#)-Gets or sets the double value that represents that legend icon(s) height.
- [IconVisibility](#)-Gets or sets the Visibility of the legend icon.
- [ItemMargin](#)-Gets or sets the margin for the legend items.
- [CornerRadius](#)-Gets or sets the corner radius of the legend.

The following code example illustrates the customization of legend icon.

**XML**

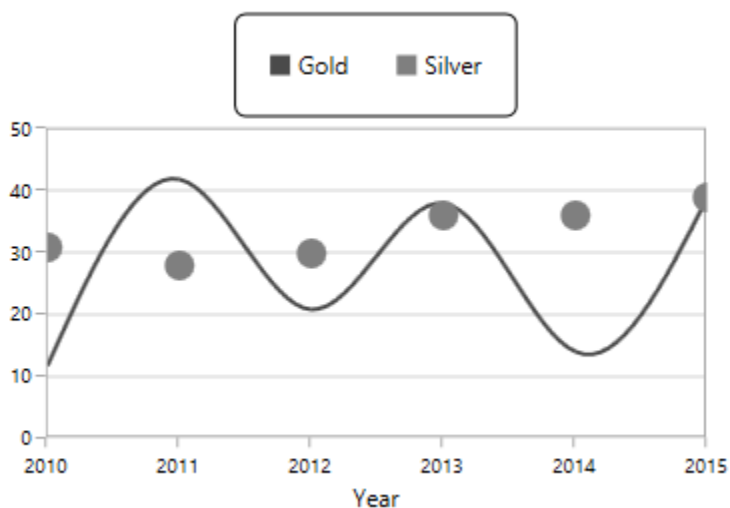
```
<chart:SfChart.Legend>
<chart:ChartLegend IconHeight="10" IconWidth="10"
Margin="0,0,0,5"
HorizontalAlignment="Center"
VerticalAlignment="Center"
DockPosition="Top"
BorderBrush="Black" BorderThickness="1"
IconVisibility="Visible" CornerRadius="5"
ItemMargin="10">
</chart:ChartLegend>
</chart:SfChart.Legend>
```

**C#**

```

chart.Legend = new ChartLegend()
{
    IconHeight = 10,
    IconWidth = 10,
    Margin = new Thickness(0, 0, 0, 5),
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center,
    DockPosition = ChartDock.Top,
    IconVisibility = Visibility.Visible,
    CornerRadius = new CornerRadius(5),
    ItemMargin = new Thickness(10),
    BorderThickness = new Thickness(1),
    BorderBrush = new SolidColorBrush(Colors.Black)
};

```



The visibility of the legend icon can be changed by setting [IconVisibility](#) property in ChartLegend.

### XML

```

<chart:SfChart.Legend>
<chart:ChartLegend IconHeight="8" IconWidth="8"
IconVisibility="Collapsed" />
</chart:SfChart.Legend>

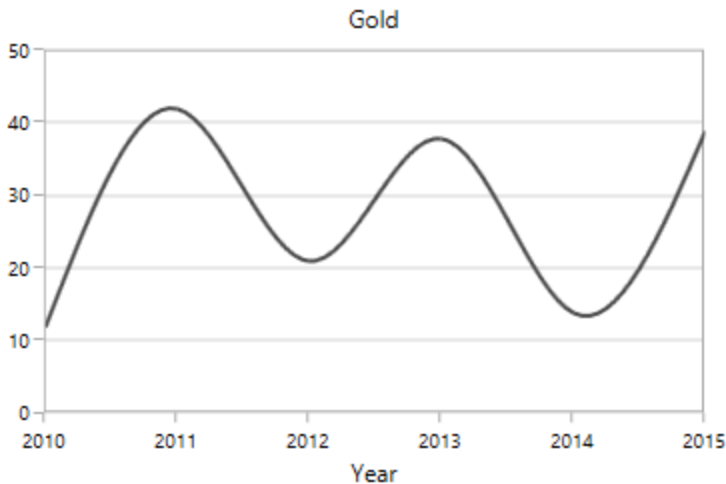
```

### C#

```

chart.Legend = new ChartLegend()
{
    IconHeight = 8,
    IconWidth = 8,
    IconVisibility = Visibility.Collapsed,
};

```



### Custom Legend Icon

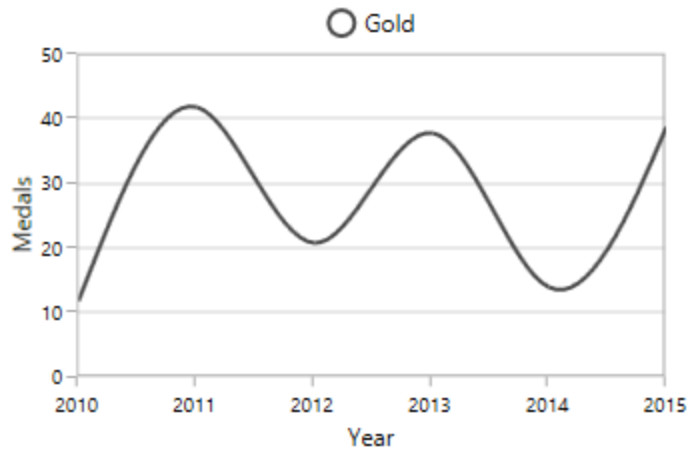
We can add custom icon for the legend using [LegendIconTemplate](#) property in ChartSeries as in below example.

#### XML

```
<syncfusion:SfChart x:Name="chart">
  <syncfusion:SfChart.Resources>
    <DataTemplate x:Key="iconTemplate">
      <Ellipse Height="15" Width="15" Fill="White"
        Stroke="#4a4a4a" StrokeThickness="2"/>
    </DataTemplate>
  </syncfusion:SfChart.Resources>
  <syncfusion:SfChart.Legend>
    <syncfusion:ChartLegend/>
  </syncfusion:SfChart.Legend>
  <syncfusion:SplineSeries XBindingPath="Year" Label="Gold"
    ItemsSource="{Binding List}"
    YBindingPath="India"
    LegendIconTemplate="{StaticResource iconTemplate}">
  </syncfusion:SplineSeries>
</syncfusion:SfChart>
```

#### C#

```
SplineSeries series = new SplineSeries()
{
    ItemsSource = new ViewModel().List,
    XBindingPath = "Year",
    YBindingPath = "India",
    LegendIconTemplate = chart.Resources["iconTemplate"] as DataTemplate
};
chart.Series.Add(series);
```



### Label

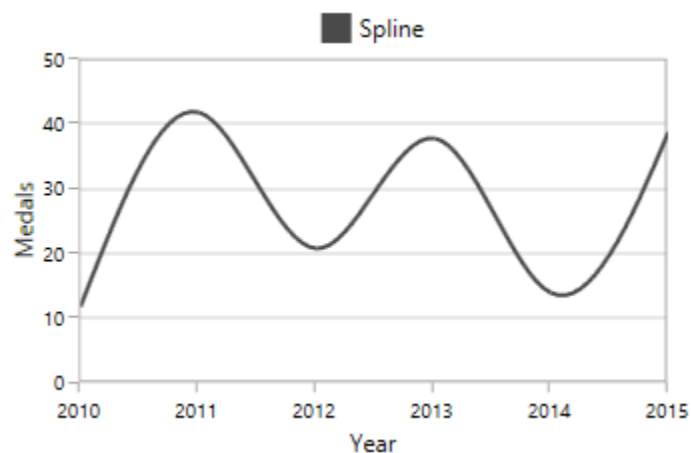
This allows us to specify the label for each series which is to be displayed in legend label.

### XML

```
<chart:SfChart.Legend>
<chart:ChartLegend>
</chart:ChartLegend>
</chart:SfChart.Legend>
<chart:SplineSeries XBindingPath="Year" Label="Spline"
ItemsSource="{Binding List}" YBindingPath="India"/>
```

### C#

```
chart.Legend = new ChartLegend();
SplineSeries splineSeries = new SplineSeries()
{
    Label = "Spline",
    ItemsSource = new ViewModel().List,
    XBindingPath = "Year",
    YBindingPath = "India",
};
chart.Series.Add(splineSeries);
```



## Checkbox

Used to view or collapse the associated series. By default, the `CheckboxVisibility` is *Collapsed*.

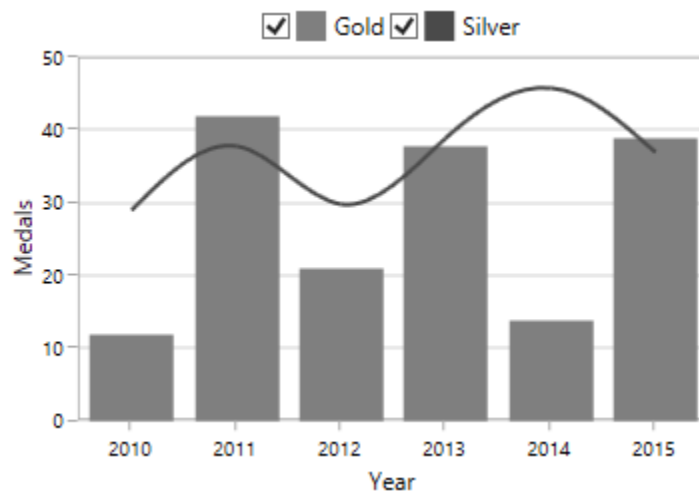
We can enable it by using the [CheckBoxVisibility](#) property as in below code example:

## XML

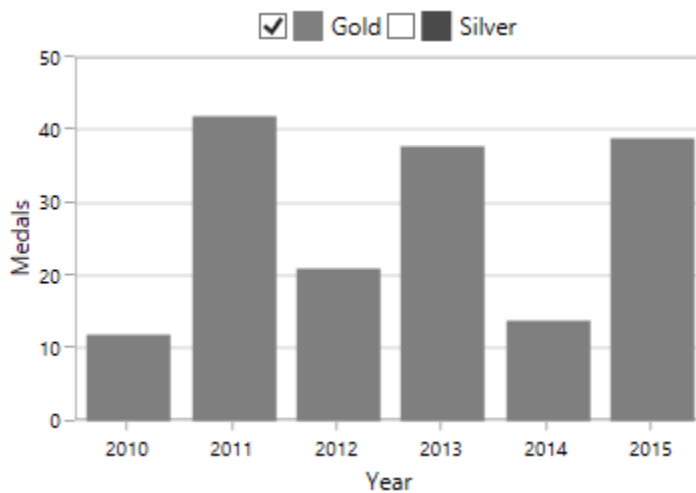
```
<chart:SfChart.Legend>
<chart:ChartLegend CheckBoxVisibility="Visible" />
</chart:SfChart.Legend>
```

## C#

```
chart.Legend = new ChartLegend()
{
    CheckBoxVisibility = Visibility.Visible
};
```



The series can be collapsed by unchecking the `CheckBox` as below:



### ToggleSeriesVisibility

[ToggleSeriesVisibility](#) is used to view or collapse the associated [series](#), by clicking on its legend item. By default, [ToggleSeriesVisibility](#) property is *False*.

We can enable the [ToggleSeriesVisibility](#) property as in below code example:

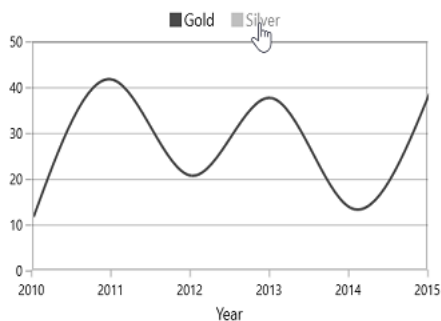
### XML

```
<chart:SfChart.Legend>
<chart:ChartLegend ToggleSeriesVisibility="True" />
</chart:SfChart.Legend>
```

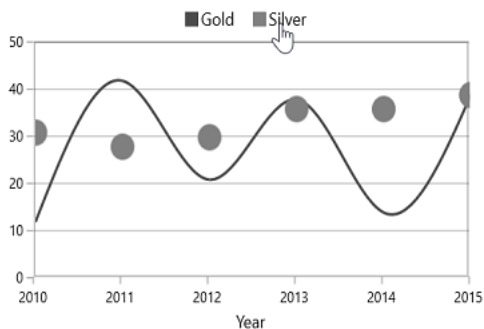
### C#

```
chart.Legend = new ChartLegend()
{
    ToggleSeriesVisibility = true
};
```

The [series](#) can be collapsed, by clicking on the respective legend item,



We can also view the associated [series](#), by clicking on its disabled legend item,



### Positioning the Legend

#### Legend Position

This allows us to position the legends [Inside](#) or [Outside](#) of the chart area (plotting area).

By default, it will be displayed outside and positioned at top (using [DockPosition](#)) of the chart area.

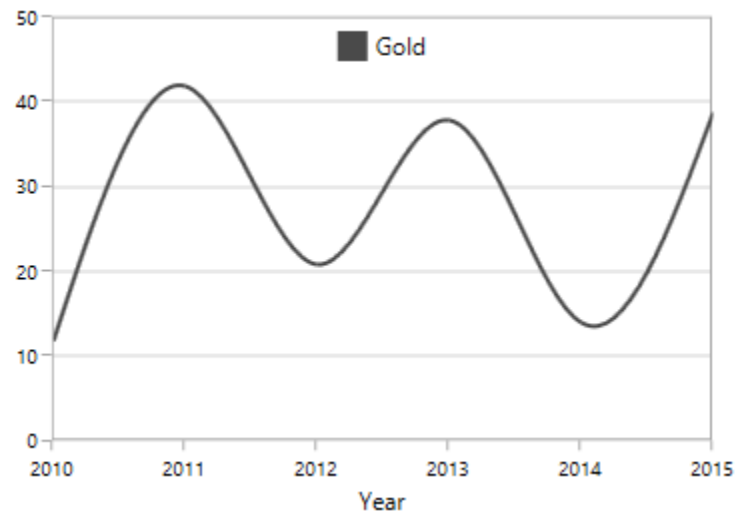
### XML

```
<chart:SfChart.Legend>
<chart:ChartLegend LegendPosition="Inside" />
</chart:SfChart.Legend>
```

```
</chart:SfChart.Legend>
```

### C#

```
chart.Legend = new ChartLegend()  
{  
    LegendPosition = LegendPosition.Inside  
};
```



### Docking

Legends can be docked left, right, and top or bottom around the chart area using [DockPosition](#) property.

By default, the ChartLegend is docked at the top of the chart as mentioned earlier.

To display the legend at the bottom, you can set the [DockPosition](#) as [Bottom](#) as in below code snippet.

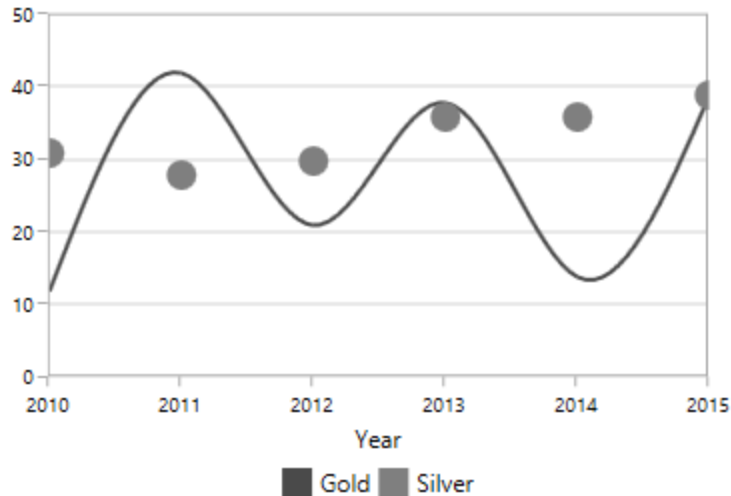
### XML

```
<chart:SfChart.Legend>  
<chart:ChartLegend DockPosition="Bottom"/>  
</chart:SfChart.Legend>
```

### C#

```
chart.Legend = new ChartLegend()  
{  
    DockPosition = ChartDock.Bottom  
};
```





### Floating Legends

To position the legend at any arbitrary location inside chart, we need to set [DockPosition](#) as **Floating** and provide its relative position using [OffsetX](#) and [OffsetY](#) properties.

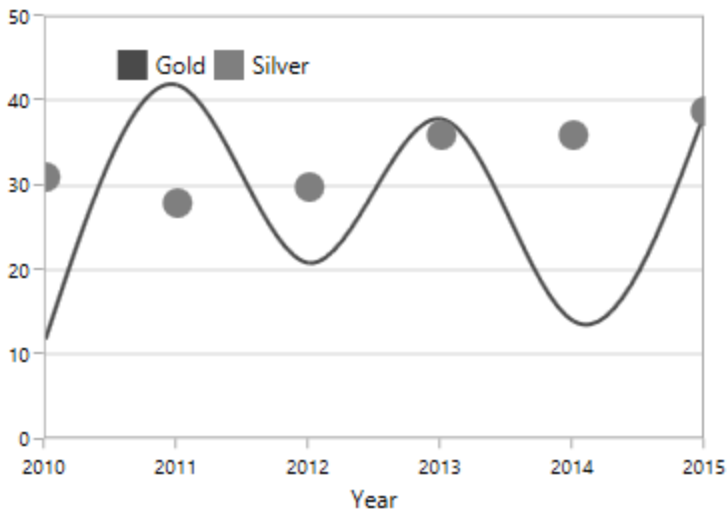
Offset specifies x or y distance from origin.

### XML

```
<chart:SfChart.Legend>  
<chart:ChartLegend DockPosition="Floating" OffsetX="30" OffsetY="10"/>  
</chart:SfChart.Legend>
```

### C#

```
chart.Legend = new ChartLegend()  
{  
    DockPosition = ChartDock.Floating,  
    OffsetX = 30,  
    OffsetY = 10  
};
```



### Legend Header

Chart provides support to add any UIElement as a header for legend items.

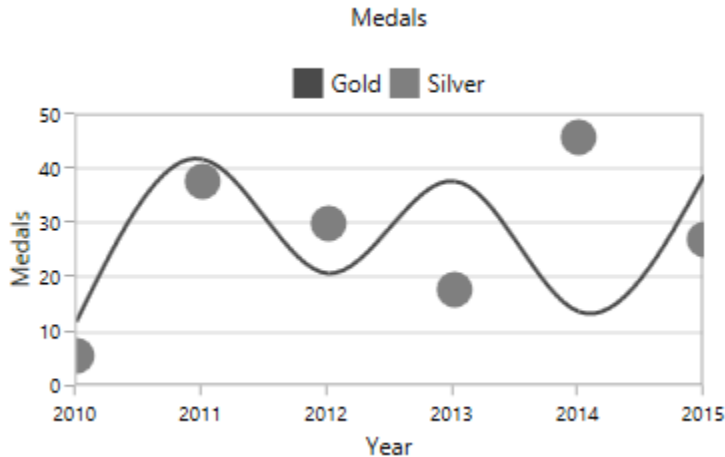
You can define the [Header](#) for legend using the following code example.

### XML

```
<chart:SfChart.Legend>
  <chart:ChartLegend>
    <chart:ChartLegend.Header>
      <TextBlock Text="Medals" VerticalAlignment="Center"
        HorizontalAlignment="Center" Margin="15"/>
    </chart:ChartLegend.Header>
  </chart:ChartLegend>
</chart:SfChart.Legend>
```

### C#

```
ChartLegend legend = new ChartLegend();
TextBlock textBlock = new TextBlock()
{
    Text = "Medals",
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center,
    Margin = new Thickness(15)
};
legend.Header = textBlock;
chart.Legend = legend;
```



### Multiple Legends

Chart control supports showing the legend in multiple panels, to view the legend clearly when multiple areas and greater numbers of chart series are present.

The following code example shows how to create multiple legends in a single chart.

#### XML

```
<chart:SfChart.Legend>
<chart:ChartLegendCollection>
<chart:ChartLegend chart:SfChart.Column="0"/>
<chart:ChartLegend chart:SfChart.Column="1"/>
</chart:ChartLegendCollection>
</chart:SfChart.Legend>
<chart:ColumnSeries Interior="#4a4a4a" Label="Legend1"
ItemsSource="{Binding SneakersDetail}" XBindingPath="Brand"
YBindingPath="ItemsCount1"/>
<chart:SplineSeries Label="Legend2" ItemsSource="{Binding SneakersDetail}"
XBindingPath="Brand" YBindingPath="ItemsCount"/>
<chart:SplineSeries.XAxis>
<chart:CategoryAxis chart:SfChart.Column="1">
</chart:CategoryAxis>
</chart:SplineSeries.XAxis>
</chart:SplineSeries>
```

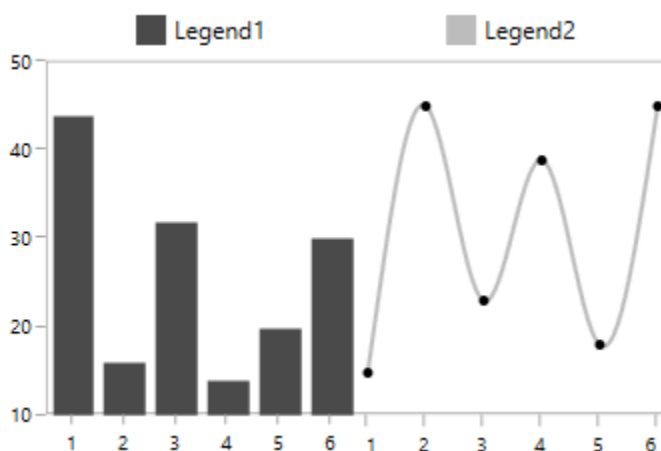
#### C#

```
ChartLegendCollection legendCollection = new ChartLegendCollection();
ChartLegend legend1 = new ChartLegend();
SfChart.SetColumn(legend1, 0);
ChartLegend legend2 = new ChartLegend();
SfChart.SetColumn(legend2, 1);
legendCollection.Add(legend1);
legendCollection.Add(legend2);
chart.Legend = legendCollection;
ColumnSeries columnSeries = new ColumnSeries()
{
    Label = "Legend1",
    ItemsSource = new ViewModel().SneakersDetail,
    XBindingPath = "Brand",
```

```

YBindingPath = "ItemCount1",
Interior = new SolidColorBrush(Color.FromRgb(0x4a, 0x4a, 0x4a)),
};
CategoryAxis axis = new CategoryAxis();
SfChart.SetColumn(axis, 1);
SplineSeries splineSeries = new SplineSeries()
{
    Label = "Legend1",
    ItemsSource = new ViewModel().SneakersDetail,
    XBindingPath = "Brand",
    YBindingPath = "ItemCount",
    XAxis = axis
};
chart.Series.Add(columnSeries);
chart.Series.Add(splineSeries);

```



### Legends for Accumulation Series

For the series like Pie, Doughnut, Funnel and Pyramid, legends will be generated for all the data points. But for remaining series, each legend corresponds to each series. By default, the [Interior](#) color of the segment (data point) is applied to the legend icon.

The following code snippets explains how the legends displaying for accumulation series.

### XML

```

<chart:SfChart.Legend>
<chart:ChartLegend />
</chart:SfChart.Legend>
<chart:PieSeries XBindingPath="Category"
ItemsSource="{Binding Tax}" YBindingPath="Percentage"/>

```

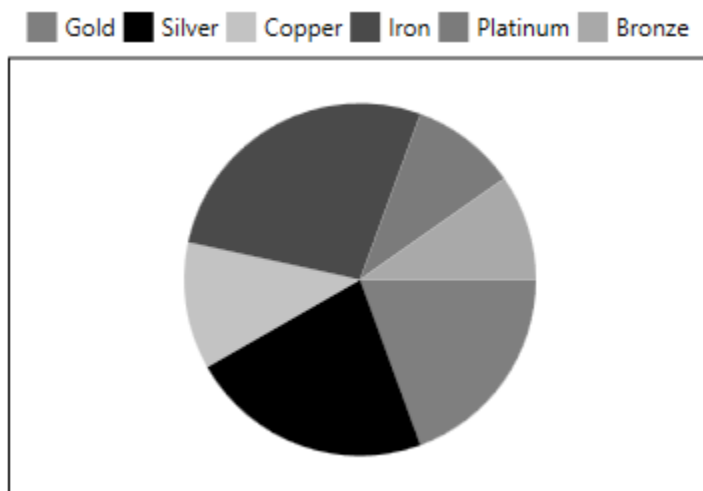
### C#

```

chart.Legend = new ChartLegend();
PieSeries pieSeries = new PieSeries()
{
    ItemsSource = new ViewModel().Tax,
    XBindingPath = "Category",
    YBindingPath = "Percentage"
};

```

```
chart.Series.Add(pieSeries);
```



**Note:** Here Legend 'Label' will be the x value of the Pie chart.

#### Series visibility on legend

We can limit the number of series and trendlines to be displayed in chart using [VisibilityOnLegend](#) property as shown in below example.

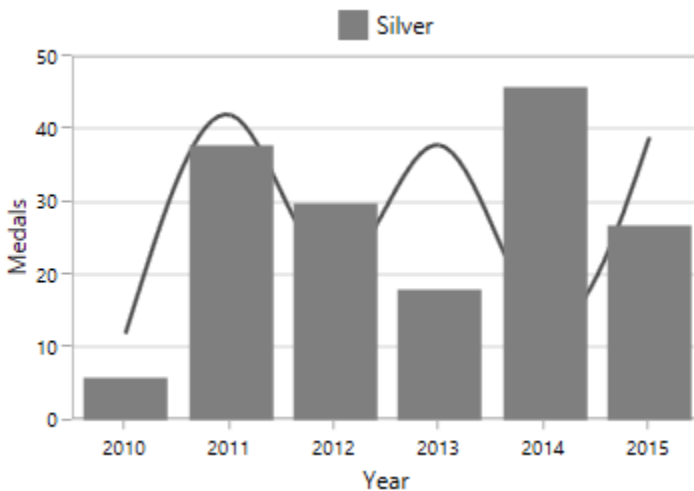
#### XML

```
<chart:SfChart.Legend>
<chart:ChartLegend>
</chart:ChartLegend>
</chart:SfChart.Legend>
<chart:SplineSeries XBindingPath="Year" Label="Gold"
VisibilityOnLegend="Collapsed"
ItemsSource="{Binding List}" YBindingPath="India">
</chart:SplineSeries>
<chart:ColumnSeries XBindingPath="Year"
VisibilityOnLegend="Visible"
Label="Silver" YBindingPath="America"
ItemsSource="{Binding List}" />
```

#### C#

```
chart.Legend = new ChartLegend();
SplineSeries splineSeries = new SplineSeries()
{
    Label = "Gold",
    ItemsSource = new ViewModel().List,
    XBindingPath = "Year",
    YBindingPath = "India",
    VisibilityOnLegend = Visibility.Collapsed
};
ColumnSeries columnSeries = new ColumnSeries()
{
    Label = "Silver",
```

```
ItemsSource = new ViewModel().List,  
XBindingPath = "Year",  
YBindingPath = "America",  
VisibilityOnLegend = Visibility.Visible  
};  
chart.Series.Add(splineSeries);  
chart.Series.Add(columnSeries);
```



### Legend Orientation

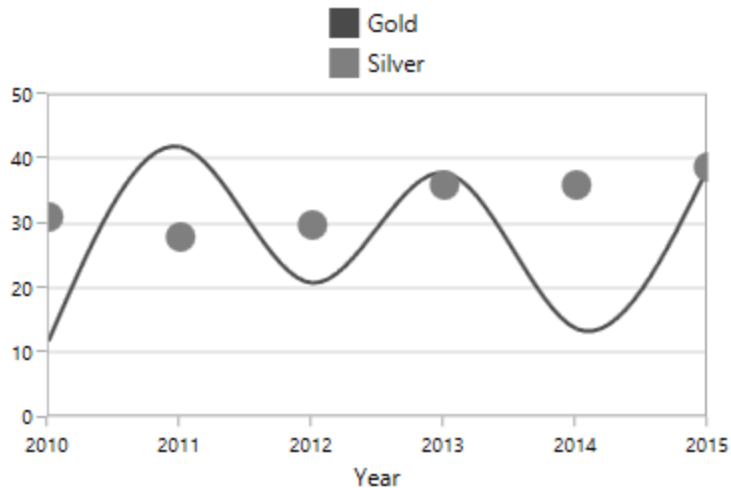
Orientation of the Legend can be vertical or horizontal. By default the [Orientation](#) is [Horizontal](#).

### XML

```
<chart:SfChart.Legend>  
<chart:ChartLegend Orientation="Vertical"/>  
</chart:SfChart.Legend>
```

### C#

```
chart.Legend = new ChartLegend()  
{  
    Orientation = ChartOrientation.Vertical  
};
```



## Customization

### ItemTemplate

You can customize each legend item using `ItemTemplate` property in `ChartLegend` as in below code snippet:

#### XML

```
<syncfusion:SfChart x:Name="chart">
  <syncfusion:SfChart.Resources>
    <local:ImageConverter x:Key="imageConverter"/>
    <DataTemplate x:Key="itemTemplate">
      <Grid Margin="10,0,10,0" >
        <Grid.RowDefinitions>
          <RowDefinition/>
          <RowDefinition/>
        </Grid.RowDefinitions>
        <Image Width="30" Height="15"
          Source="{Binding Converter={StaticResource imageConverter}}"/>
        <TextBlock HorizontalAlignment="Center" FontSize="12"
          Grid.Row="1" Foreground="Black"
          FontWeight="SemiBold" Text="{Binding Label}">
        </TextBlock>
      </Grid>
    </DataTemplate>
  </syncfusion:SfChart.Resources>
  <syncfusion:SfChart.Legend>
    <syncfusion:ChartLegend ItemTemplate="{StaticResource itemTemplate}"/>
  </syncfusion:SfChart.Legend>
</syncfusion:SfChart>
```

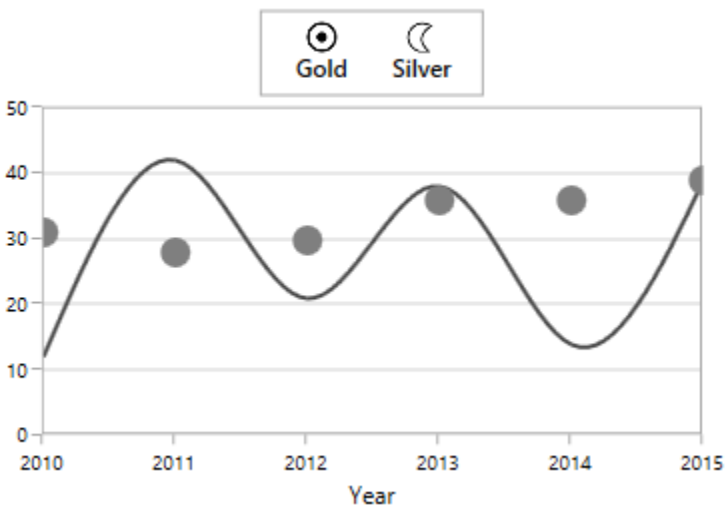
#### C#

```
SfChart chart = new SfChart();
chart.Legend = new ChartLegend()
{
    ItemTemplate = chart.Resources["itemTemplate"] as DataTemplate
};
public class ImageConverter:IValueConverter
```

```

{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        LegendItem item = value as LegendItem;
        if (item.Label == "Gold")
            return new BitmapImage(new
                Uri(("gold_symb.png"), UriKind.RelativeOrAbsolute));
        else
            return new BitmapImage(new Uri(("silver_symb.png"),
                UriKind.RelativeOrAbsolute));
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        return value;
    }
}

```



### Customizing Legend Items Layout

When there is more number of legends, the legend exceeds the chart will be cropped, as it arranged horizontally. To avoid the cropping we can change the existing arrangement layout (one which arrange each legend items horizontally) using `ItemsPanel` property as in below code snippet:

#### XML

```

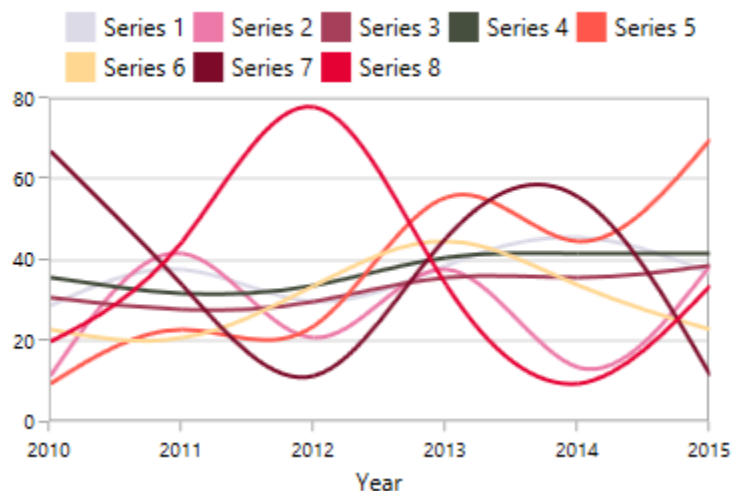
<syncfusion:SfChart x:Name="chart">
    <syncfusion:SfChart.Resources>
        <ItemsPanelTemplate x:Key="itemPanelTemplate">
            <WrapPanel/>
        </ItemsPanelTemplate>
    </syncfusion:SfChart.Resources>
    <syncfusion:SfChart.Legend>
        <syncfusion:ChartLegend ItemsPanel="{StaticResource itemPanelTemplate}"/>
    </syncfusion:SfChart.Legend>
</syncfusion:SfChart>

```



**C#**

```
SfChart chart = new SfChart();
chart.Legend = new ChartLegend()
{
    ItemsPanel = chart.Resources["itemPanelTemplate"] as ItemsPanelTemplate
};
```

**Troubleshooting**

The legend item is not showing because the chart view is running in the background thread with the timespan problem while updating the chart element dynamically. You can resolve this by using the `Dispatcher.BeginInvoke()` method. Please refer to the following code sample.

**C#**

```
ChartWindow chartWindow = new ChartWindow();
chartWindow.DataContext = measurementChart;
chartWindow.Content = new AutoChart();
chartWindow.Content = new SfChart()
{
    . . .
}
Action action = chartWindow.Show;
Application.Current.Dispatcher.BeginInvoke(action, Array.Empty<object>());
```

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

See also

[How to customize the legend Icon based on series appearance in WPF Chart](#)

[How to achieve the draggable legend in WPF Chart](#)

[How to wrap the text in the WPF Chart legend](#)

[How to control the visibility of all series with a single legend item in WPF Chart](#)

[How to create custom legend items in WPF Chart](#)

[How to get a notification when the legend items are clicked in WPF Chart](#)

[How to add multiple legend items in scroll viewer](#)

[How to format the legend text](#)

[How to set or modify the label of the each legend](#)

[How to customize the legends position](#)

[How to customize the icons of the legends in Chart](#)

### Series in WPF Charts (SfChart)

ChartSeries is the visual representation of the data. SfChart offers many types of series ranging from LineSeries to FinancialSeries like HiLo and Candle. Based on your requirements and specifications, any type of Series can be added for data visualization.

The following APIs are common for the most of the series types:

- [XBindingPath](#) – A string property that represents the X values for the series.
- [YBindingPath](#) – A string property that represents the Y values for the series.
- [Stroke](#) – Represents the brush for the series outline.
- [StrokeThickness](#) – Represents the thickness of the series outline.
- [Interior](#) – Represents the brush to fill the series.
- [Palette](#) – Used to define the set of pre-defined or custom colors for the series.
- [IsSeriesVisible](#) – A bool property, which is used to enable or disable the series visibility.

---

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configure with built-in support for creating stunning visual effects.

---

### Fast in WPF Charts (SfChart)

SfChart includes functionality for fast-plotting more than 10 fast chart types. Each chart type is easily configurable with built-in support for creating stunning visual effects.

- Fast Series - Segments of the series is rendered using a polyline segment.
- Fast Bitmap Series - Segments of the series is rendered using WritableBitmap.

---

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configure with built-in support for creating stunning visual effects.

---

### Sorting in WPF Charts (SfChart)

Chart provides the support for sorting the data point either in ascending or descending based on X or Y axis.

#### *Enable Sorting*

This [IsSortData](#) property is used to enable the sorting in series.

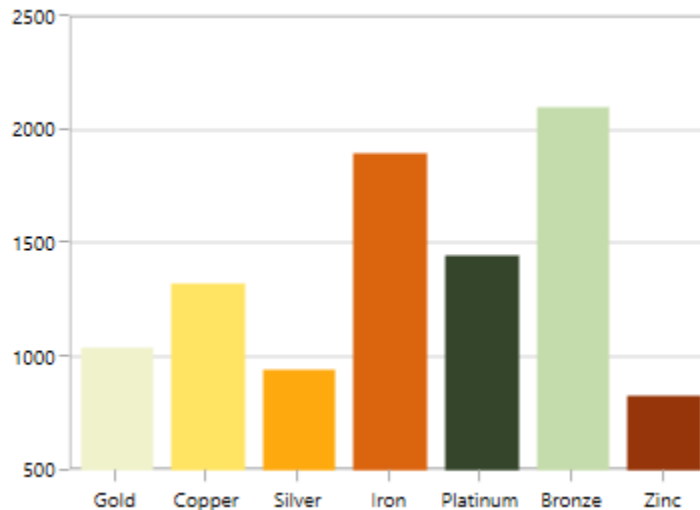
#### *Changing sorting direction*

The [SortDirection](#) property defines the direction of sorting either in [Ascending](#) or [Descending](#) based on x or y value.

*Changing sorting axis*

This [SortBy](#) property decides whether sorting should be done based on [X](#) or [Y](#) values.

The following example illustrates a simple chart (without apply sorting):



Sorting for category(non-linear) axis

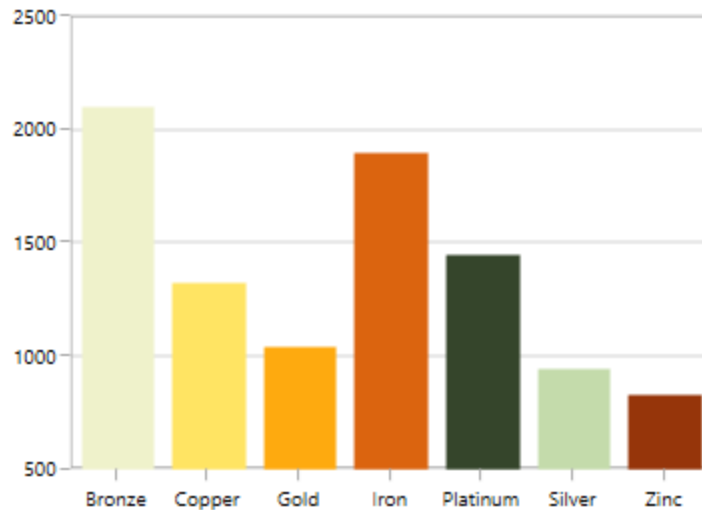
**Sorting x axis in ascending order:**

**XML**

```
<syncfusion:ColumnSeries IsSortData="True" SortBy="X"
SortDirection="Ascending"
ItemsSource="{Binding Demands}" Interior="#4A4A4A"
XBindingPath="Demand" YBindingPath="Year2011"/>
```

**C#**

```
ColumnSeries columnSeries = new ColumnSeries()
{
    IsSortData = true,
    SortBy = SortingAxis.X,
    SortDirection = Direction.Ascending,
    ItemsSource = new ViewModel().Demands,
    XBindingPath = "Demand",
    YBindingPath = "Year2011",
    Interior = new SolidColorBrush(Color.FromRgb(0x4A, 0x4A, 0x4A))
};
chart.Series.Add(columnSeries);
```



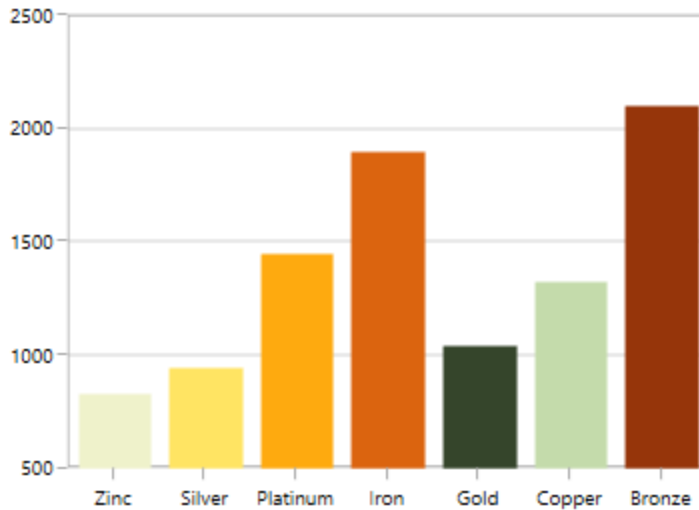
Sorting x axis in descending order:

#### XML

```
<syncfusion:ColumnSeries IsSortData="True" SortBy="X"
SortDirection="Descending"
ItemsSource="{Binding Demands}" Interior="#4A4A4A"
XBindingPath="Demand" YBindingPath="Year2011"/>
```

#### C#

```
ColumnSeries columnSeries = new ColumnSeries()
{
    IsSortData = true,
    SortBy = SortingAxis.X,
    SortDirection = Direction.Descending,
    ItemsSource = new ViewModel().Demands,
    XBindingPath = "Demand",
    YBindingPath = "Year2011",
    Interior = new SolidColorBrush(Color.FromRgb(0x4A, 0x4A, 0x4A))
};
chart.Series.Add(columnSeries);
```



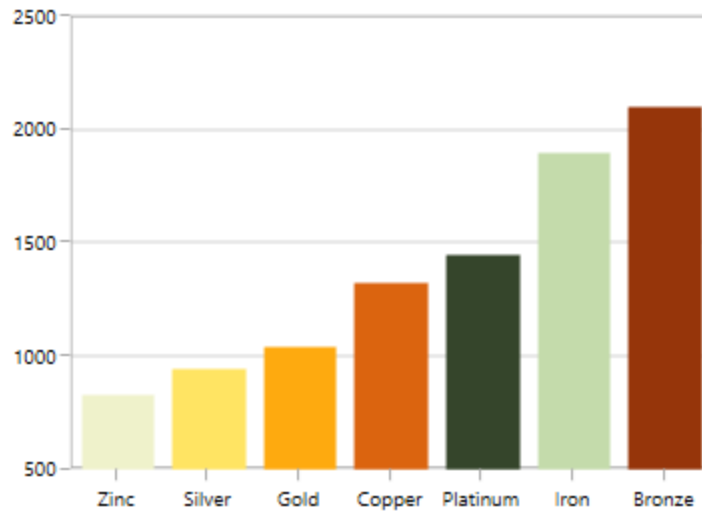
Sorting y axis in ascending order:

#### XML

```
<syncfusion:ColumnSeries IsSortData="True" SortBy="Y"
SortDirection="Ascending"
ItemsSource="{Binding Demands}" Interior="#4A4A4A"
XBindingPath="Demand" YBindingPath="Year2011"/>
```

#### C#

```
ColumnSeries columnSeries = new ColumnSeries()
{
    IsSortData = true,
    SortBy = SortingAxis.Y,
    SortDirection = Direction.Ascending,
    ItemsSource = new ViewModel().Demands,
    XBindingPath = "Demand",
    YBindingPath = "Year2011",
    Interior = new SolidColorBrush(Color.FromRgb(0x4A, 0x4A, 0x4A))
};
chart.Series.Add(columnSeries);
```



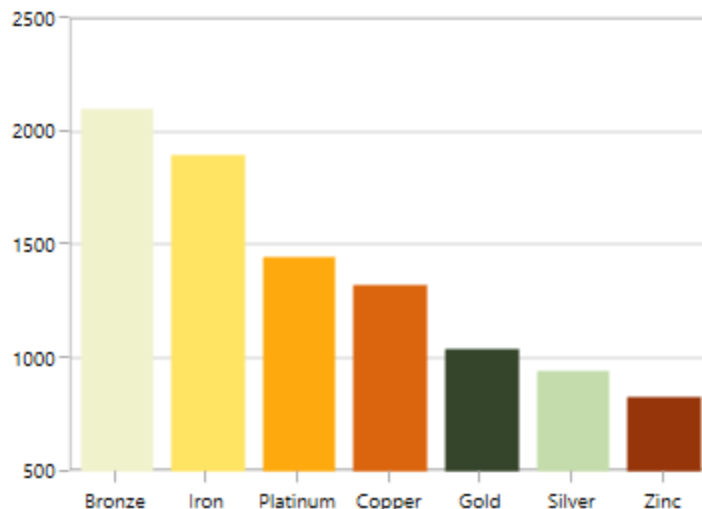
Sorting y axis in descending order:

#### XML

```
<syncfusion:ColumnSeries IsSortData="True" SortBy="Y"
SortDirection="Descending"
ItemsSource="{Binding Demands}" Interior="#4A4A4A"
XBindingPath="Demand" YBindingPath="Year2011"/>
```

#### C#

```
ColumnSeries columnSeries = new ColumnSeries()
{
    IsSortData = true,
    SortBy = SortingAxis.X,
    SortDirection = Direction.Descending,
    ItemsSource = new ViewModel().Demands,
    XBindingPath = "Demand",
    YBindingPath = "Year2011",
    Interior = new SolidColorBrush(Color.FromRgb(0x4A, 0x4A, 0x4A))
};
chart.Series.Add(columnSeries);
```



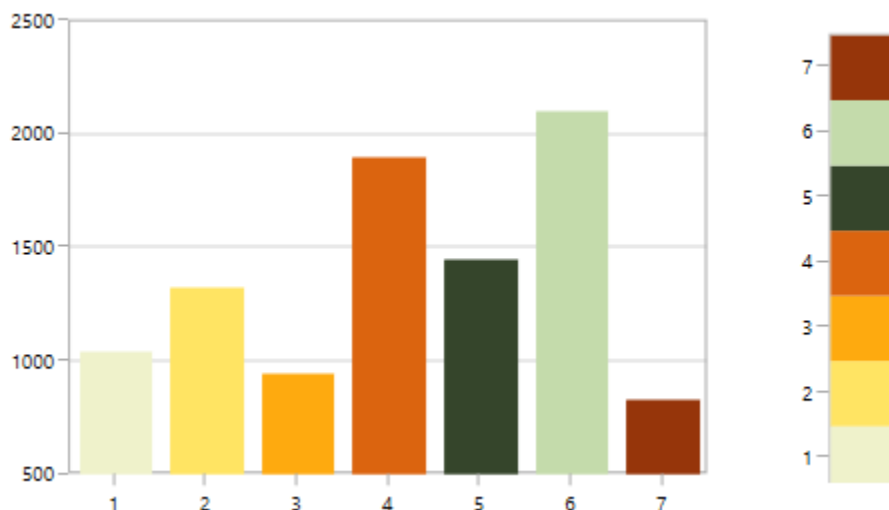
**Note:** This feature is primarily applicable for indexed (non-linear) axis like CategoryAxis. For linear axis like NumericalAxis, only the order of rendering will be sorted. i.e., the order in which the data point is being rendered.

#### Sorting for linear axis

As mentioned above, the sorting for the linear axis is different from CategoryAxis. Here the rendering order of the data point (x or y) will be sorted.

This will be useful especially when we have one or more values added in same data point. Also this rendering order will be captured by applying Palette to each point.

The following example illustrates a simple chart having AutumnBrights palette (without apply sorting):



#### Sorting x axis in ascending order

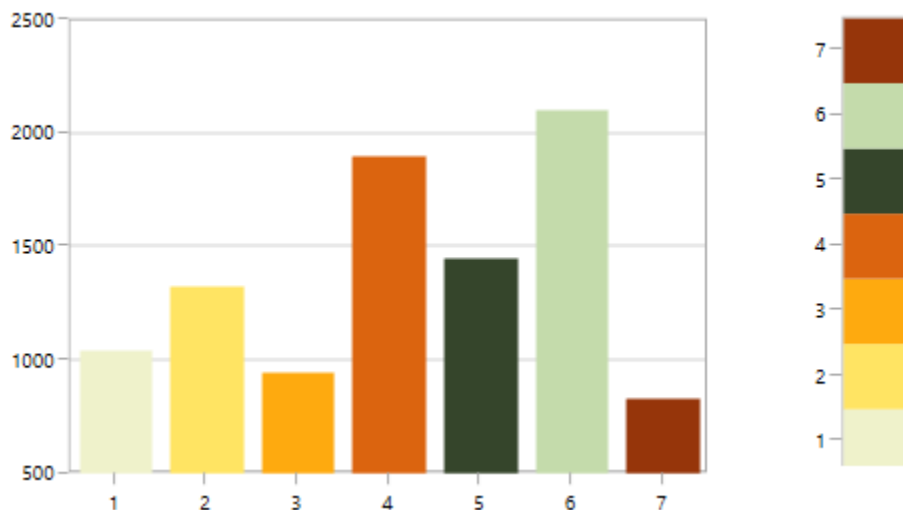
##### XML

```
<syncfusion:ColumnSeries IsSortData="True" SortBy="X"
    Palette="AutumnBrights"
    SortDirection="Ascending">
```

```
ItemsSource="{Binding Demands}"
XBindingPath="Position" YBindingPath="Year2011"/>
```

**C#**

```
ColumnSeries columnSeries = new ColumnSeries()
{
    IsSortData = true,
    SortBy = SortingAxis.X,
    SortDirection = Direction.Ascending,
    Palette = ChartColorPalette.AutumnBrights,
    ItemsSource = new ViewModel().Demands,
    XBindingPath = "Position",
    YBindingPath = "Year2011",
    Interior = new SolidColorBrush(Color.FromRgb(0x4A, 0x4A, 0x4A))
};
chart.Series.Add(columnSeries);
```

**Sorting x axis in descending order****XML**

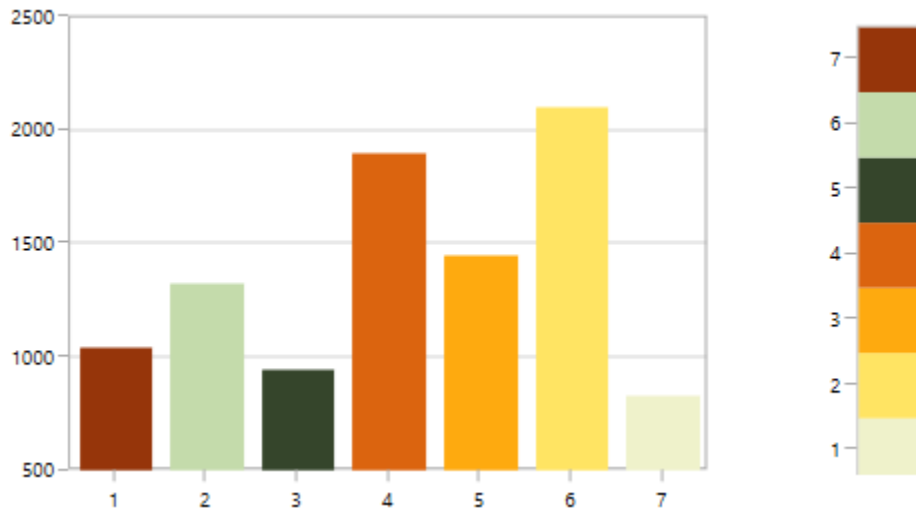
```
<syncfusion:ColumnSeries IsSortData="True" SortBy="X"
    Palette="AutumnBrights"
    SortDirection="Descending"
    ItemsSource="{Binding Demands}"
    XBindingPath="Position" YBindingPath="Year2011"/>
```

**C#**

```
ColumnSeries columnSeries = new ColumnSeries()
{
    IsSortData = true,
    SortBy = SortingAxis.X,
    SortDirection = Direction.Descending,
    Palette = ChartColorPalette.AutumnBrights,
```



```
ItemsSource = new ViewModel().Demands,
XBindingPath = "Position",
YBindingPath = "Year2011",
Interior = new SolidColorBrush(Color.FromRgb(0x4A, 0x4A, 0x4A))
};
chart.Series.Add(columnSeries);
```



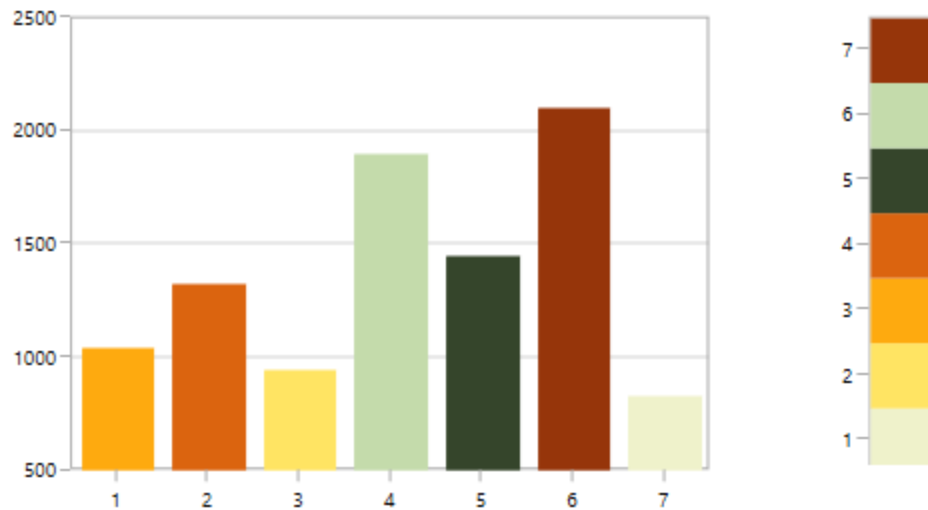
Sorting y axis in ascending order

#### XML

```
<syncfusion:ColumnSeries IsSortData="True" SortBy="Y"
Palette="AutumnBrights"
SortDirection="Ascending"
ItemsSource="{Binding Demands}"
XBindingPath="Position" YBindingPath="Year2011"/>
```

#### C#

```
ColumnSeries columnSeries = new ColumnSeries()
{
    IsSortData = true,
    SortBy = SortingAxis.Y,
    SortDirection = Direction.Ascending,
    Palette = ChartColorPalette.AutumnBrights,
    ItemsSource = new ViewModel().Demands,
    XBindingPath = "Position",
    YBindingPath = "Year2011",
    Interior = new SolidColorBrush(Color.FromRgb(0x4A, 0x4A, 0x4A))
};
chart.Series.Add(columnSeries);
```



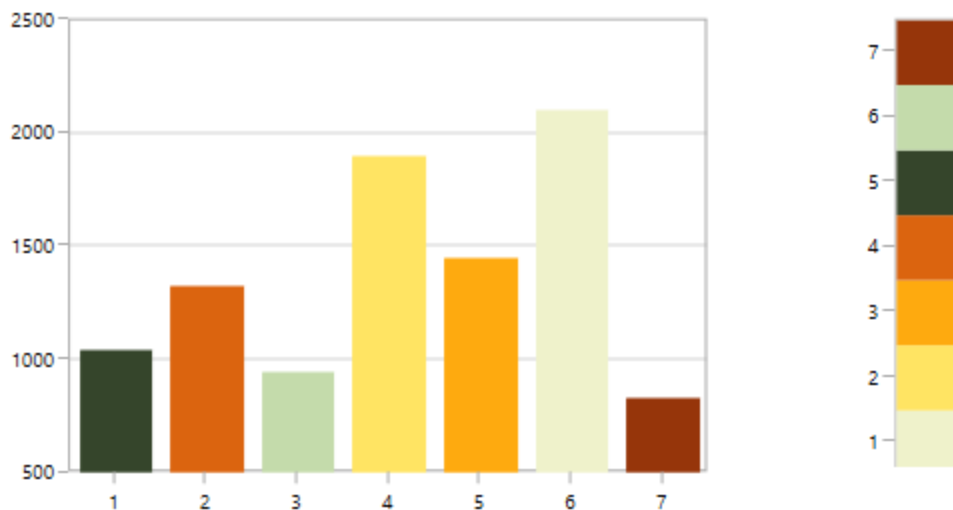
Sorting y axis in descending order

#### XML

```
<syncfusion:ColumnSeries IsSortData="True" SortBy="Y"
    Palette="AutumnBrights"
    SortDirection="Descending"
    ItemsSource="{Binding Demands}"
    XBindingPath="Position" YBindingPath="Year2011"/>
```

#### C#

```
ColumnSeries columnSeries = new ColumnSeries()
{
    IsSortData = true,
    SortBy = SortingAxis.Y,
    SortDirection = Direction.Descending,
    Palette = ChartColorPalette.AutumnBrights,
    ItemsSource = new ViewModel().Demands,
    XBindingPath = "Position",
    YBindingPath = "Year2011",
    Interior = new SolidColorBrush(Color.FromRgb(0x4A, 0x4A, 0x4A))
};
chart.Series.Add(columnSeries);
```



**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

### Data Binding in WPF Charts (SfChart)

SfChart offers ItemsSource property to bind various datasource ranges from simple collection property to complex properties.

*Binding a simple collection to the chart*

#### XML

```
<syncfusion:SfChart >
  <syncfusion:LineSeries
    ItemsSource="{Binding Demands}"
    XBindingPath="Demand"
    YBindingPath="Year2010">
  </syncfusion:LineSeries>
</syncfusion:SfChart>
```

#### C#

```
SfChart chart = new SfChart();
LineSeries lineSeries = new LineSeries()
{
    ItemsSource = new ViewModel().Demands,
    XBindingPath = "Demand",
    YBindingPath = "Year2010",
};
chart.Series.Add(lineSeries);
public class GoldDemand
{
    public string Demand { get; set; }
    public double Year2010 { get; set; }
    public double Year2011 { get; set; }
}
public sealed partial class MainPage : Page
```

```

{
    public MainPage()
    {
        this.InitializeComponent();
        this.Demands = new ObservableCollection<GoldDemand>
        {
            new GoldDemand() {Demand = "Jewelry", Year2010 = 1998.0, Year2011 = 2361.2},
            new GoldDemand() {Demand = "Electronics", Year2010 = 1284.0, Year2011 = 1328.0},
            new GoldDemand() {Demand = "Research", Year2010 = 1090.5, Year2011 = 1032.0},
            new GoldDemand() {Demand = "Investment", Year2010 = 1643.0, Year2011 = 1898.0},
            new GoldDemand() {Demand = "Bank Purchases", Year2010 = 987.0, Year2011 = 887.0},
            new GoldDemand() {Demand = "Others", Year2010 = 1090.5, Year2011 = 1032.0},
            new GoldDemand() {Demand = "Investment", Year2010 = 1643.0, Year2011 = 1898.0},
            new GoldDemand() {Demand = "Bank Purchases", Year2010 = 987.0, Year2011 = 887.0},
            new GoldDemand() {Demand = "Electronics", Year2010 = 1284.0, Year2011 = 1328.0},
            new GoldDemand() {Demand = "Research", Year2010 = 1090.5, Year2011 = 1032.0},
            new GoldDemand() {Demand = "Investment", Year2010 = 1643.0, Year2011 = 1898.0},
            new GoldDemand() {Demand = "Bank Purchases", Year2010 = 987.0, Year2011 = 887.0}
        };
        DataContext = this;
    }
    public ObservableCollection<GoldDemand> Demands { get; set; }
}

```

### *Binding complex property to the chart*

The complex property binding feature enables you to access nested object reference property values to render the chart segment.

### **XML**

```

<syncfusion:LineSeries ItemsSource="{Binding DataWithMulData}" XBindingPath
="StadiumObject.CupDetailsObj.CupName" YBindingPath="StadiumObject.NumSeats"
/>

```

### **C#**

```

StadiumDetails stadiumDetails = new StadiumDetails();
LineSeries series = new LineSeries()
{
    ItemsSource = new ViewModel().DataWithMulData,
    XBindingPath = "stadiumDetails.CupDetailsObj.CupName",
    YBindingPath = "stadiumDetails.NumSeats",
    Interior = new SolidColorBrush(Color.FromRgb(0xBC, 0xBC, 0xBC))
};
chart.Series.Add(series);

```

```

public class StadiumDetails
{
    public string PlaceName { get; set; }
    public int NumSeats { get; set; }
    public int Price { get; set; }
    public CupDetails CupDetailsObj { get; set; }
}
public class CupDetails
{
    public string CupName { get; set; }
}
public class DataPointWithMulData
{
    public string Name { get; set; }
    public StadiumDetails StadiumObject { get; set; }
}

```

#### *Binding array property to the chart*

The SfChart supports array values for the XBindingPath and YBindingPath. XBindingPath and YBindingPath are bound with the property name in the corresponding index value. You can bind the same property with different index values.

The following code example demonstrates how to bind the array values for the XBindingPath and YBindingPath.

#### **XML**

```

<chart:SfChart>
<chart:ColumnSeries x:Name="series" ItemsSource="{Binding Brands}"
XBindingPath="Brand[1]" YBindingPath="Count[0]" >
</chart:ColumnSeries>
</chart:SfChart>

```

#### **C#**

```

public class Model
{
    public string[] Brand { get; set; }
    public double[] Count { get; set; }
}
public class ViewModel
{
    public ViewModel()
    {
        Brands = new ObservableCollection<Model>();
        Brands.Add(new Model() { Brand = new string[] { "Reebok", "Adidas" }, Count
= new double[] { 34, 23 } });
        Brands.Add(new Model() { Brand = new string[] { "Benz", "Audi" }, Count
= new double[] { 50, 20 } });
        Brands.Add(new Model() { Brand = new string[] { "iPhone", "Nokia" }, Count
= new double[] { 24, 30 } });
        Brands.Add(new Model() { Brand = new string[] { "Lenovo", "Acer" }, Count
= new double[] { 38, 23 } });
        Brands.Add(new Model() { Brand = new string[] { "Fastrack", "Titan" }, Count
= new double[] { 27, 29 } });
    }
}

```

```

}
public ObservableCollection<Model> Brands { get; set; }
}
private void CreateChart()
{
    ViewModel view = new ViewModel();
    SfChart chart = new SfChart();
    ColumnSeries series = new ColumnSeries();
    series.ItemsSource = view.Brands;
    series.XBindingPath = "Brand[1]";
    series.YBindingPath = "Count[0]";
    chart.Series.Add(series);
    grid.Children.Add(chart);
}

```

### *Listening Property Changes*

You can notify the [XBindingPath](#) and [YBindingPath](#) properties changes by setting [ListenPropertyChange](#) as true as shown in the below code snippet.

### **XML**

```

<chart:ScatterSeries ScatterWidth="20" ScatterHeight="20" Label="Coal"
ListenPropertyChange="True"
ItemsSource="{Binding EnergyProductions}" Interior="#BCBCBC"
XBindingPath="ID" YBindingPath="Coal">
</chart:ScatterSeries>

```

### **C#**

```

ScatterSeries series = new ScatterSeries()
{
    ItemsSource = new ViewModel().EnergyProductions,
    XBindingPath = "ID",
    YBindingPath = "Coal",
    ScatterWidth = 20,
    ScatterHeight = 20,
    Label = "Coal",
    ListenPropertyChange=true,
    Interior = new SolidColorBrush(Color.FromRgb(0xBC, 0xBC, 0xBC))
};
chart.Series.Add(series);

```

Also, When enabling this property to the series you need to implements [INotifyPropertyChanged](#) to the underlying data object to make the chart listen to the property changes of your data object.

**Note:** By default, the property change was disabled. So the dynamic updates will not get reflect in chart. You need to enable this property.

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to knows various chart types and how to easily configured with built-in support for creating stunning visual effects.

See also

[How to bind a list of Tuple in WPF Charts](#)

[How to bind the underlying DataTable model to the DataMarker Template in WPF Charts](#)

[How to bind the SQL Database to WPF Charts](#)

[How to bind the JSON data in WPF Chart](#)

[How to bind KeyValuePair collection in WPF Chart](#)

[How to bind the series collection property using MVVM pattern](#)

[How to bind data table in the Chart](#)

[How to create a real time Chart using MVVM in WPF](#)

[How to bind the array property in Chart](#)

[How to generate dynamic number of series based on common items source](#)

[How to manage the empty values \(NaN\) in Chart](#)

## Data Markers in WPF Charts (SfChart)

Chart adornments (Data Markers) are used to display values related to a chart segment element. Values from data point(x, y) or other custom properties from a data source can be displayed.

Each adornment can be represented by the following:

- Label - Displays the segment label content at the (X, Y) point.
- Marker- Displays the desired symbol at the (X, Y) point.
- ConnectorLine - Line used to connect the (X, Y) point and the label element.

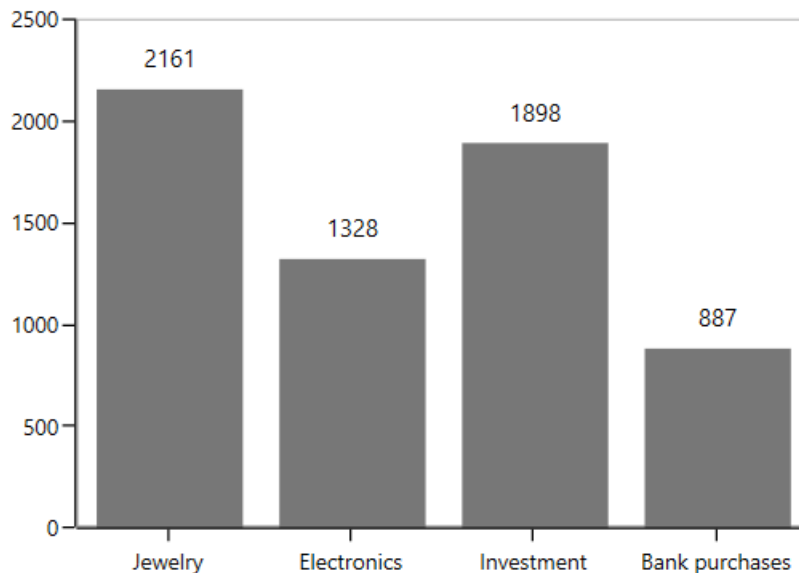
The following code example illustrates how to initialize the adornment.

### XML

```
<syncfusion:ColumnSeries Interior="#777777" ItemsSource="{Binding Demands}"
XBindingPath="Category" YBindingPath="Value">
  <syncfusion:ColumnSeries.AdornmentsInfo>
    <syncfusion:ChartAdornmentInfo></syncfusion:ChartAdornmentInfo>
  </syncfusion:ColumnSeries.AdornmentsInfo>
</syncfusion:ColumnSeries>
```

### C#

```
ColumnSeries series = new ColumnSeries()
{
    ItemsSource = ViewModel().Demands,
    XBindingPath = "Demands",
    YBindingPath = "Value",
    Interior = new SolidColorBrush(Color.FromRgb(0x77, 0x77, 0x77))
};
ChartAdornmentInfo adornmentInfo = new ChartAdornmentInfo();
series.AdornmentsInfo = adornmentInfo;
chart.Series.Add(series);
```



See also

[How to show different data marker based on the value in the WPF Chart](#)

[How to show custom data marker in the WPF Chart](#)

[How to rotate text in adornment](#)

[How to display the labels inside segments](#)

[How to bind the underlying DataTable model to the DataMarker Template in WPF Charts](#)

### Annotations in WPF Charts (SfChart)

SfChart supports Annotations, which allows you to mark the specific area of interest in the chart area. You can draw custom shapes, also text and images can be added using Annotations.

The following annotations are supported in SfChart

- [Text Annotation](#)
- [Shape Annotation](#)
- [Image Annotation](#)

### Adding Annotation

You can create an instance for any type of Annotation and add it to [Annotations](#) collection. Here for instance, the EllipseAnnotation is added.

### XML

```
<chart:SfChart.Annotations>  
<chart:EllipseAnnotation X1="1.5" Y1="20" Text="Ellipse" X2="3" Y2="23" >  
</chart:EllipseAnnotation>  
</chart:SfChart.Annotations>
```

### C#

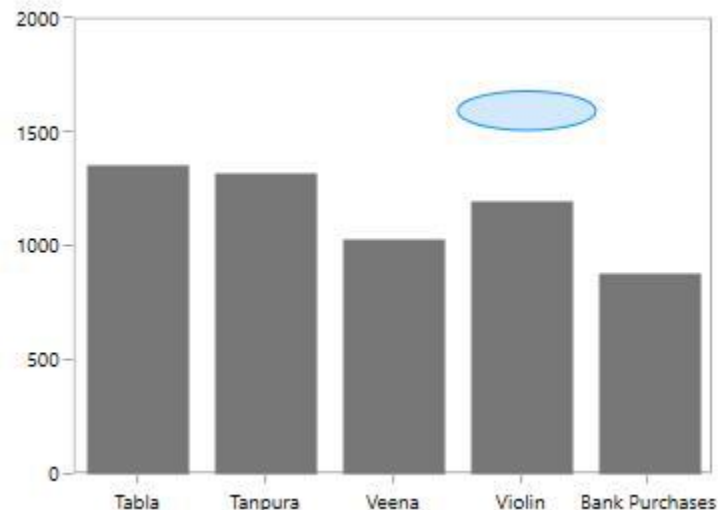
```
EllipseAnnotation annotation=new EllipseAnnotation()  
{
```



```

X1 = 1.5, Y1 = 20,
X2 = 3, Y2 = 23,
Text = "Ellipse"
};
chart.Annotations.Add(annotation);

```



### Positioning the Annotation

Annotations can be positioned in plot area based on [X1](#) and [Y1](#) properties and for image and shape annotations you need to specify [X2](#) and [Y2](#) properties. These X and Y values can be specified with axis units or pixel units and this can be identified using [CoordinateUnit](#) property.

### Positioning based on CoordinateUnit as Axis

To position the annotation based on axis, set the X1 and Y1, X2 and Y2 properties based on axis range values, if needed, set the CoordinateUnit value as [Axis](#).

### XML

```

<chart:RectangleAnnotation X1="1.25" Y1="1300" FontSize="10"
FontFamily="Calibri" Text="Axis Value" X2="2.25" Y2="1500"
CoordinateUnit="Axis">
</chart:RectangleAnnotation>

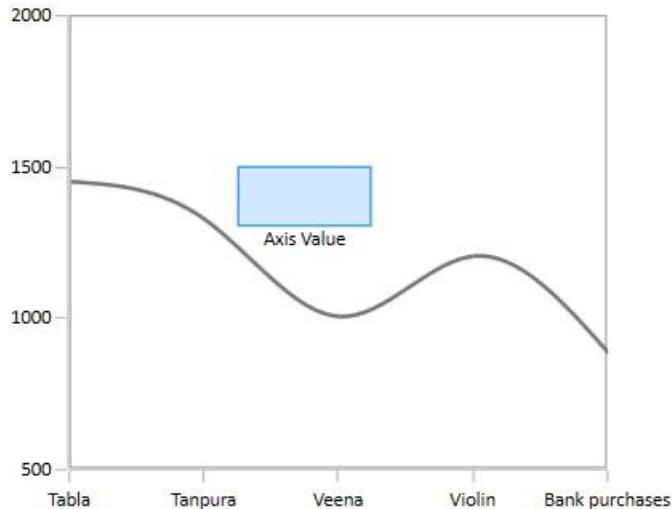
```

### C#

```

RectangleAnnotation annotation=new RectangleAnnotation()
{
X1 = 1.25, Y1 = 1300,
X2 = 2.25, Y2 = 1500,
Text = "Axis Value",
CoordinateUnit=CoordinateUnit.Axis,
Text="Axis Value"
};
chart.Annotations.Add(annotation);

```



### Positioning based on CoordinateUnit as Pixels

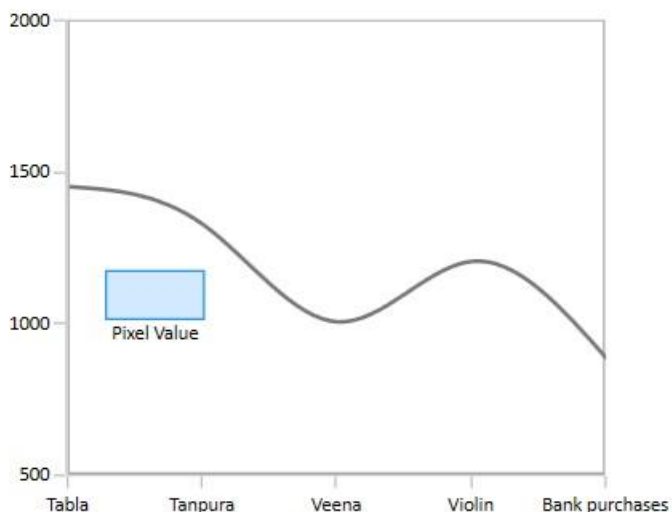
To position based on the pixel values you have to set the `CoordinateUnit` as `Pixels` and the pixel values in `X1` and `Y1`, `X2` and `Y2` properties in `Annotation`.

#### XML

```
<chart:RectangleAnnotation X1="50" Y1="150" Text="Pixel Value" X2="100"
Y2="125" CoordinateUnit="Pixel">
</chart:RectangleAnnotation>
```

#### C#

```
RectangleAnnotation annotation=new RectangleAnnotation()
{
    X1 = 50, Y1 = 150,
    X2 = 100, Y2 = 125,
    CoordinateUnit=CoordinateUnit.Pixel,
    Text="Pixel Value"
};
chart.Annotations.Add(annotation);
```



### Adding Annotation for MultipleAxes

You can also add annotation for a particular axis when there is multiple axes using [XAxisName](#) and [YAxisName](#) properties as in the below code snippet.

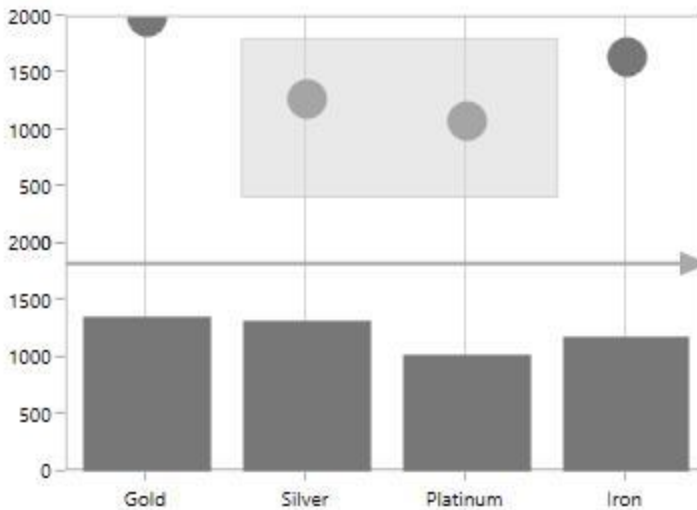
#### XML

```
<chart:SfChart Width="400" Height="400" BorderBrush="Transparent">
  <chart:SfChart.RowDefinitions>
    <chart:ChartRowDefinition></chart:ChartRowDefinition>
    <chart:ChartRowDefinition></chart:ChartRowDefinition>
  </chart:SfChart.RowDefinitions>
  <!--PrimaryAxis-->
  <chart:SfChart.PrimaryAxis>
    <chart:CategoryAxis FontSize="11" ShowGridLines="False"/>
  </chart:SfChart.PrimaryAxis>
  <chart:SfChart.SecondaryAxis>
    <chart:NumericalAxis x:Name="ColumnAxis" Maximum="2000" FontSize="11"
      Interval="500" chart:ChartBase.Row="0" ShowGridLines="False"/>
  </chart:SfChart.SecondaryAxis>
  <chart:SfChart.Annotations>
    <chart:AnnotationCollection>
      <chart:RectangleAnnotation YAxisName="ScatterAxis" Fill="LightGray"
        Stroke="DarkGray" Opacity="0.5" X1="0.5" Y1="900" X2="2.5" Y2="1600" >
      </chart:RectangleAnnotation>
      <chart:HorizontalLineAnnotation YAxisName="ColumnAxis" Stroke="DarkGray"
        X1="-0.5" X2="3.5" Y1="1700"
        LineCap="Arrow"></chart:HorizontalLineAnnotation>
    </chart:AnnotationCollection>
  </chart:SfChart.Annotations>
  <chart:ColumnSeries Interior="#777777" ItemsSource="{Binding
    CategoricalData}" XBindingPath="Category" YBindingPath="Plastic">
  </chart:ColumnSeries>
  <chart:ScatterSeries Interior="#777777" ItemsSource="{Binding
    CategoricalData}" XBindingPath="Category" YBindingPath="Plastic">
    <chart:ScatterSeries.YAxis>
      <chart:NumericalAxis x:Name="ScatterAxis" Maximum="2000" FontSize="11"
        Interval="500" chart:ChartBase.Row="1" ShowGridLines="False"/>
    </chart:ScatterSeries.YAxis>
  </chart:ScatterSeries>
</chart:SfChart>
```

#### C#

```
SfChart chart = new SfChart();
chart.RowDefinitions.Add(new ChartRowDefinition());
chart.RowDefinitions.Add(new ChartRowDefinition());
chart.PrimaryAxis = new CategoryAxis();
chart.SecondaryAxis = new NumericalAxis();
ChartBase.SetRow(chart.SecondaryAxis, 0);
HorizontalLineAnnotation annotation = new HorizontalLineAnnotation()
{
    X1 = -0.5,
    Y1 = 1700,
    X2 = 3.5,
    YAxisName = "ColumnAxis",
    LineCap=LineCap.Arrow,
}
```

```
Stroke=new SolidColorBrush(Colors.DarkGray)
};
RectangleAnnotation rect = new RectangleAnnotation()
{
    YAxisName = "ScatterAxis",
    Fill = new SolidColorBrush(Colors.LightGray),
    Stroke = new SolidColorBrush(Colors.DarkGray),
    Opacity = 0.5,
    X1 = 0.5,
    Y1 = 900,
    X2 = 2.5,
    Y2 = 1600
};
ColumnSeries columnSeries = new ColumnSeries()
{
    ItemsSource = new CategoricalViewModel().CategoricalData,
    XBindingPath = "Category",
    YBindingPath = "Plastic",
    Interior = new SolidColorBrush(Color.FromRgb(0x77, 0x77, 0x77))
};
ScatterSeries scatterSeries = new ScatterSeries()
{
    ItemsSource = new CategoricalViewModel().CategoricalData,
    XBindingPath = "Category",
    YBindingPath = "Plastic",
    Interior = new SolidColorBrush(Color.FromRgb(0x77, 0x77, 0x77))
};
NumericalAxis axis = new NumericalAxis()
{
    Name = "ScatterAxis",
    Maximum = 2000,
    FontSize = 11,
    Interval = 500,
    ShowGridLines = false
};
scatterSeries.YAxis = axis;
ChartBase.SetRow(axis, 1);
chart.Series.Add(columnSeries);
chart.Series.Add(scatterSeries);
```



### Text Annotation

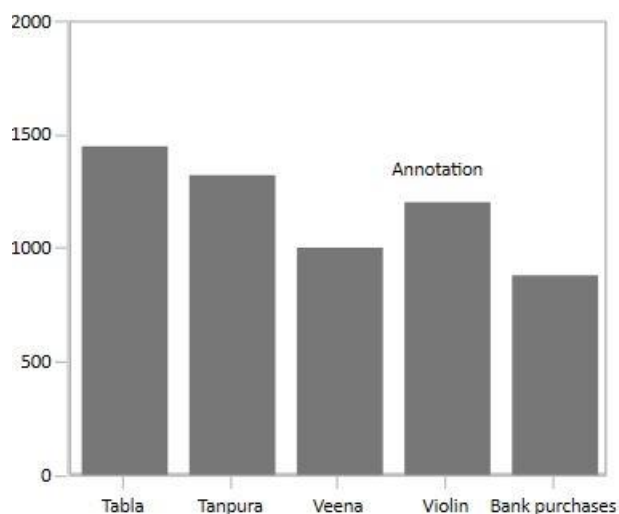
[TextAnnotations](#) are used to add simple with help of '[Text](#)' property in specific points over the chart area.

### XML

```
<chart:SfChart.Annotations>
<chart:TextAnnotation Text="Annotation" X1="2.5" Y1="1400" >
</chart:TextAnnotation>
</chart:SfChart.Annotations>
```

### C#

```
TextAnnotation annotation=new TextAnnotation()
{
    X1 = 2.5,
    Y1 = 1400,
    Text="Annotation"
};
chart.Annotations.Add(annotation);
```



### Customizing Text Annotation

SfChart provides you with an editing option for the text in any annotations. When text annotation is enabled editing, if we click the text annotation it switches to edit mode which provide easy way of customizing the text at run time.

The following properties are used to customize the text:

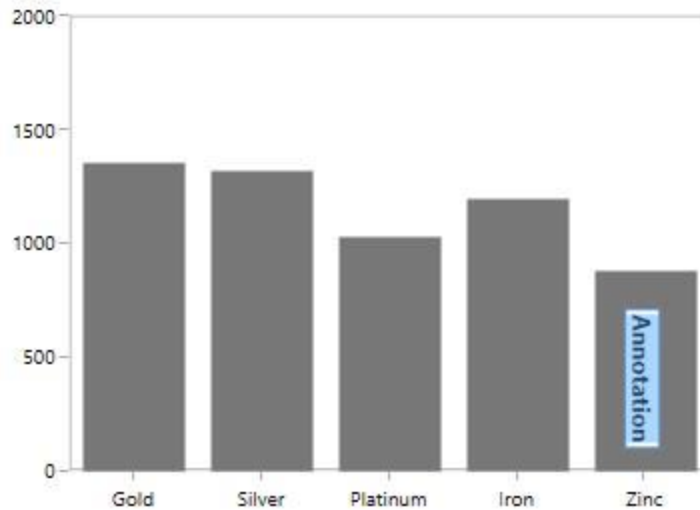
- [EnableEditing](#) - Used to define whether the text in TextAnnotation can be edited or not.
- [Angle](#) - Used to get or set the angle for rotating the Annotation.
- ['EnableClipping'](#) - Used to define whether annotation should clip while crossing with boundary.
- [Foreground](#) - Used to change the text color.
- [FontSize](#) - An int value that represents the font size of the annotation text.
- [FontFamily](#) - Represents the font family of the annotation text.
- [FontStyle](#) - Represents the font style of the annotation text.
- [FontWeight](#) - Represents the font weight of the annotation text.
- [FontStretch](#) - Represents the font stretch for the annotation description.

### XML

```
<chart:TextAnnotation Angle="90" EnableEditing="True"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch" FontWeight="Bold"
Foreground="Black" Text="Annotation" X1="3.5" Y1="500" >
```

### C#

```
TextAnnotation annotation = new TextAnnotation()
{
    X1 = 3.5,
    Y1 = 500,
    Text = "Annotation",
    EnableEditing=true,
    Foreground=new SolidColorBrush(Colors.Black),
    FontStyle=FontStyles.Bold,
    HorizontalAlignment=HorizontalAlignment.Stretch,
    VerticalAlignment=VerticalAlignment.Stretch
};
chart.Annotations.Add(annotation);
```



### Shape Annotation

[ShapeAnnotation](#) allows you to add annotations in the form of shapes such as rectangle, ellipse, horizontal line and vertical line at the specific area of interest, in the chart area.

- [EllipseAnnotation](#) - Used to draw a circle or an ellipse over the chart area.
- [RectangleAnnotation](#) - Used to draw a rectangle over the chart area.
- [LineAnnotation](#) - Used to draw a line over the chart area.
- [VerticalLineAnnotation](#) - Used to draw a vertical line across the chart area.
- [HorizontalLineAnnotation](#) - Used to add a horizontal line across the chart area.

The following API's are commonly used in all ShapeAnnotation:

- [Fill](#) - Represents the brush inside the Shape Annotation.
- [X2](#) - Represents the X2 Coordinate of the Shape Annotation.
- [Y2](#) - Represents the Y2 Coordinate of the Shape Annotation.
- [CanDrag](#) - A Boolean value that represent to drag the Annotation.
- [CanResize](#) - A Boolean value that represent to resize the Annotation.

### Ellipse Annotation

The '[EllipseAnnotation](#)' is used to draw an oval or a circle in specific points over the chart area.

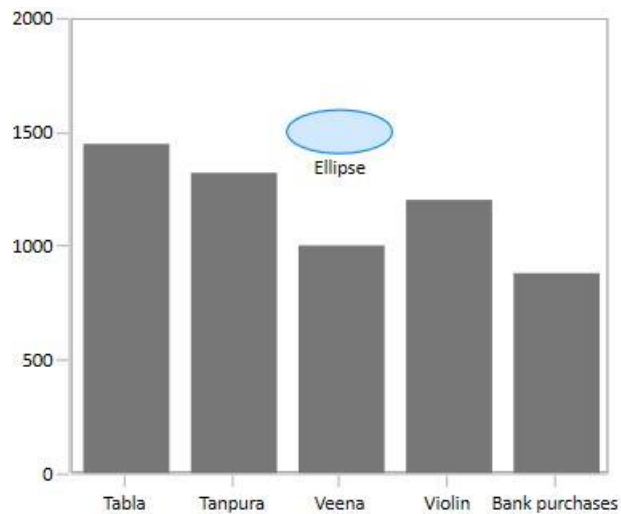
#### XML

```
<chart:EllipseAnnotation X1="1.5" Y1="1400" X2="2.5" Y2="1600"
Text="Ellipse">
</chart:EllipseAnnotation>
```

#### C#

```
EllipseAnnotation ellipse = new EllipseAnnotation()
{
    X1 = 1.5,
    Y1 = 1400,
    X2 = 2.5,
    Y2 = 1600,
```

```
Text = "Ellipse"  
};
```



#### *Rectangle Annotation*

The '[RectangleAnnotation](#)' is used to draw a rectangle or a square in specific points over the chart area.

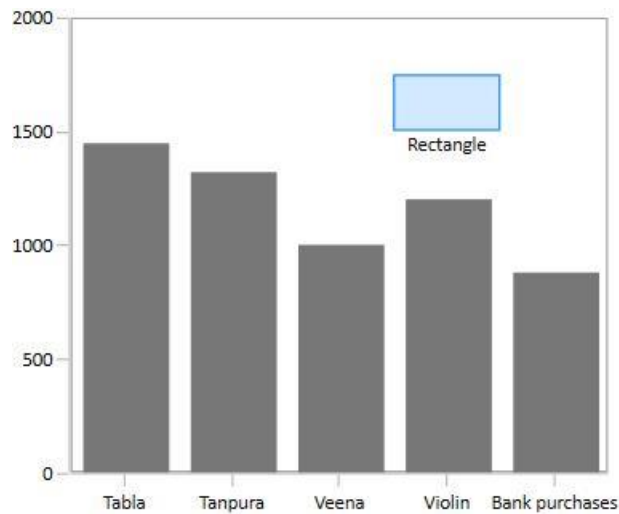
#### **XML**

```
<chart:RectangleAnnotation X1="2.5" Y1="1500" X2="3.5" Y2="1750"  
Text="Rectangle">  
</chart:RectangleAnnotation>
```

#### **C#**

```
RectangleAnnotation rectangle = new RectangleAnnotation()  
{  
    X1 = 2.5,  
    Y1 = 1500,  
    X2 = 3.5,  
    Y2 = 1750,  
    Text = "Rectangle"  
};
```





### Line Annotation

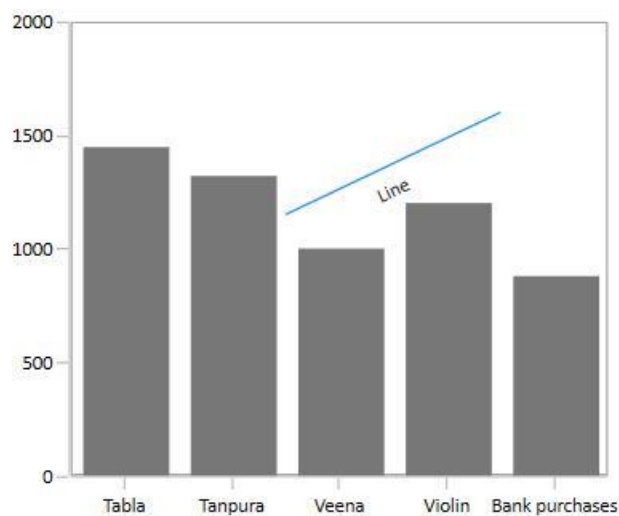
The '[LineAnnotation](#)' is used to draw a line in specific points over the chart area.

### XML

```
<chart:LineAnnotation X1="1.5" Y1="1150" X2="3.5" Y2="1600" Text="Line">
</chart:LineAnnotation>
```

### C#

```
LineAnnotation line = new LineAnnotation()
{
    X1 = 1.5,
    Y1 = 1150,
    X2 = 3.5,
    Y2 = 1600,
    Text = "Line"
};
```



*Vertical and Horizontal line annotation*

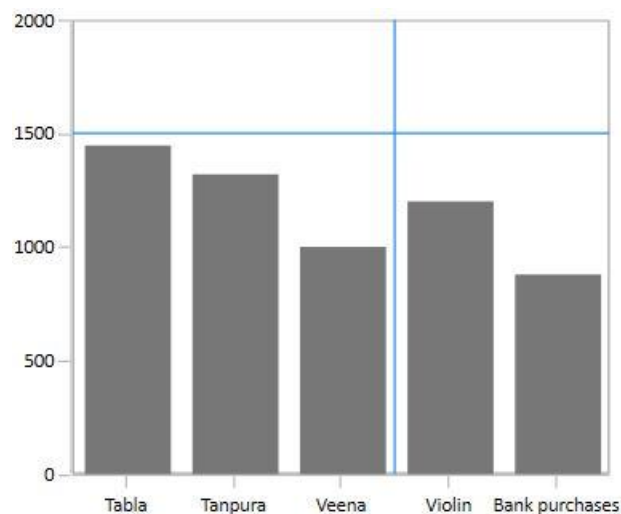
The '[VerticalLineAnnotation](#)' and [HorizontalLineAnnotation](#) are used to draw vertical and horizontal lines in specific points over the chart area.

**XML**

```
<chart:HorizontalLineAnnotation X1="-0.5" Y1="1500" X2="4.5">
</chart:HorizontalLineAnnotation>
<chart:VerticalLineAnnotation X1="2.5"></chart:VerticalLineAnnotation>
```

**C#**

```
HorizontalLineAnnotation hor = new HorizontalLineAnnotation()
{
    X1 = -0.5,
    Y1 = 1500,
    X2 = 4.5,
};
VerticalLineAnnotation ver = new VerticalLineAnnotation()
{
    X1 = 2.5
};
```

*Customizing Line Annotation*

The appearance of the LineAnnotation, VerticalLineAnnotation and HorizontalLineAnnotation can be customized with use of following properties.

- [GrabExtent](#) - Used to extent the hit visible area while performing dragging and resizing.
- [ShowLine](#) - Used to collapse the visibility of the line annotation.
- [LineCap](#)
- [ShowAxisLabel](#) - Used to display the axis labels in which the line is placed

**Adding arrow to line annotation**

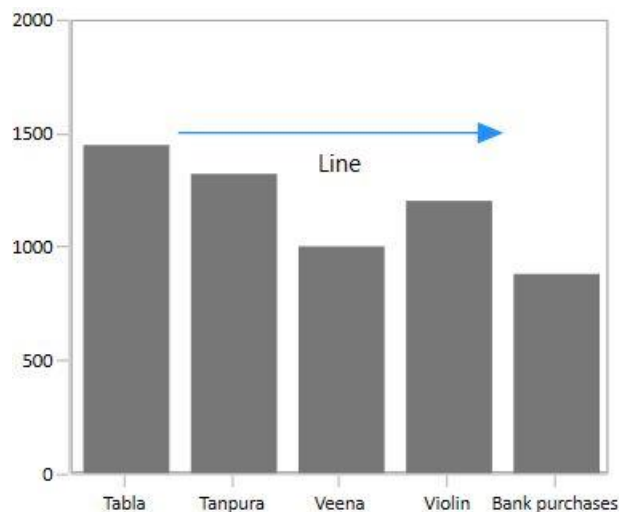
To display single headed arrow, set the '[LineCap](#)' property to '[Arrow](#)'. The default value of the '[LineCap](#)' property is '[None](#)'.

**XML**

```
<chart:LineAnnotation X1="0.5" Y1="1500" X2="3.5" Y2="1500" LineCap="Arrow"
Text="Line">
</chart:LineAnnotation>
```

**C#**

```
LineAnnotation ellipse = new LineAnnotation()
{
    X1 = 0.5,
    Y1 = 1500,
    X2 = 3.5,
    Y2 = 1500,
    Text = "Line",
    LineCap=LineCap.Arrow
};
```

**Displaying Axis Labels for LineAnnotation**

[VerticalLineAnnotation](#) and [HorizontalLineAnnotation](#) also displays the axis labels in which the line is placed. This feature can be enabled by setting [ShowAxisLabel](#) property to true as in the below code snippet.

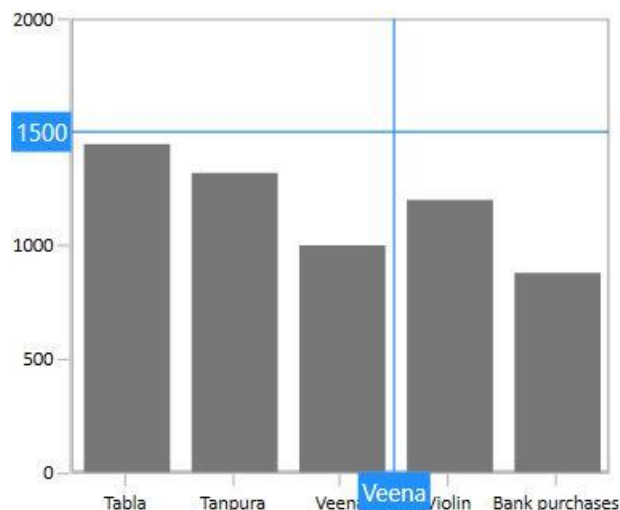
**XML**

```
<chart:HorizontalLineAnnotation ShowAxisLabel="True" X1="-0.5" Y1="1500"
X2="4.5">
</chart:HorizontalLineAnnotation>
<chart:VerticalLineAnnotation ShowAxisLabel="True"
X1="2.5"></chart:VerticalLineAnnotation>
```

**C#**

```
HorizontalLineAnnotation hor = new HorizontalLineAnnotation()
{
    X1 = -0.5,
    Y1 = 1500,
```

```
X2 = 4.5,
ShowAxisLabel=true
};
VerticalLineAnnotation ver = new VerticalLineAnnotation()
{
X1 = 2.5,
ShowAxisLabel=true
};
```



Also, axis label can be customized the default appearance using [AxisLabelTemplate](#) property.

#### *Adding text in shape annotation*

For all the shape annotations, the text can be displayed by using the [Text](#) property.

#### **Customizing text in shape annotation**

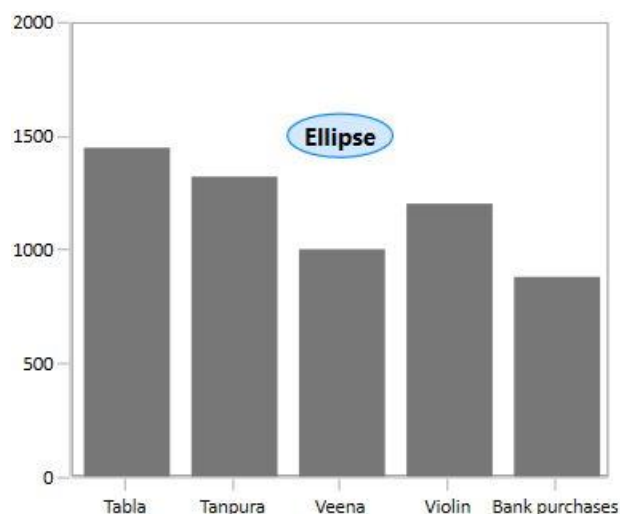
The text alignment can be changed using [HorizontalTextAlignment](#) and [VerticalTextAlignment](#) properties.

#### **XML**

```
<chart:EllipseAnnotation X1="1.5" Y1="1400" X2="2.5" Y2="1600"
FontWeight="Bold" HorizontalTextAlignment="Center"
VerticalTextAlignment="Center" Text="Ellipse">
</chart:EllipseAnnotation>
```

#### **C#**

```
EllipseAnnotation ellipse = new EllipseAnnotation()
{
X1 = 1.5,
Y1 = 1400,
X2 = 2.5,
Y2 = 1600,
HorizontalTextAlignment =HorizontalAlignment.Center,
VerticalTextAlignment = VerticalAlignment.Center,
FontStyle=FontStyles.Bold,
Text = "Ellipse"
};
```



**Note:** [HorizontalTextAlignment](#) and [VerticalTextAlignment](#) properties are not applicable for [TextAnnotation](#).

#### Customizing the Shape Annotation

SfChart allows customization of shape annotation using the following properties.

- [Stroke](#) - Represents the brush for the annotation outline.
- [StrokeThickness](#) - Represents the thickness of the annotation outline.
- [StrokeDashArray](#) - Represents the DashArray of the annotation stroke.
- [StrokeDashCap](#) - Represents the DashCap of the annotation stroke.
- [StrokeDashOffset](#) - Represents the DashOffset of the annotation stroke.
- [StrokeEndLineCap](#) - Represents the end line cap of the annotation stroke.
- [StrokeLineJoin](#) - Represents the line join of the annotation outline.
- [StrokeMiterLimit](#) - Represents the limit on the ratio of the miter length to half of the annotation shape.

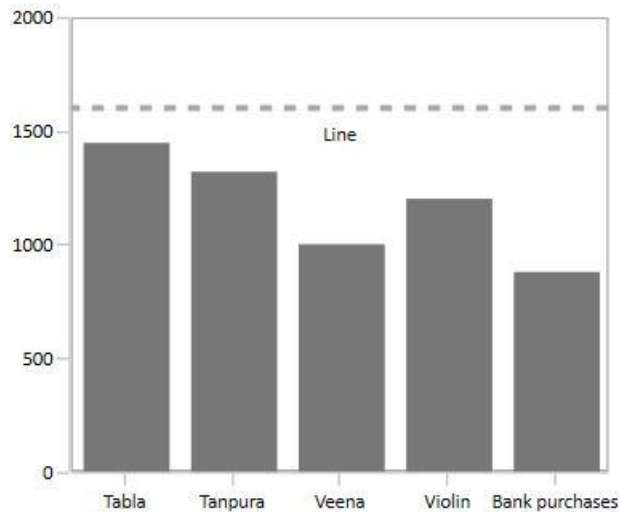
#### XML

```
<chart:HorizontalLineAnnotation X1="-0.5" StrokeThickness="3"
StrokeDashCap="Square" StrokeEndLineCap="Square" StrokeStartLineCap="Square"
Stroke="DarkGray" Fill="LightGray" StrokeDashArray="1,3" Y1="1600" X2="4.5"
Y2="1600" Text="Line">
</chart:HorizontalLineAnnotation>
```

#### C#

```
SfChart chart = new SfChart();
HorizontalLineAnnotation annotation = new HorizontalLineAnnotation()
{
    X1 = -0.5,
    X2 = 4.5,
    Y1 = 1500,
    StrokeThickness = 3,
    Stroke = new SolidColorBrush(Colors.DarkGray),
    Fill = new SolidColorBrush(Colors.LightGray),
```

```
StrokeDashArray = new DoubleCollection() { 1, 3 },
StrokeStartLineCap = PenLineCap.Square,
StrokeEndLineCap = PenLineCap.Square,
StrokeDashCap = PenLineCap.Round
};
chart.Annotations.Add(annotation);
```



### Image Annotation

SfChart provides support to add images as Annotation over the chart area, using the class [ImageAnnotation](#).

The following API's are used in ImageAnnotation.

- [Angle](#) – An integer value that represents the rotation angle for the text in Annotation.
- [ImageSource](#) - Represents the source from where the image must be added.
- [X2](#) - Represents the X2 Coordinate of the Annotation.
- [Y2](#) - Represents the Y2 Coordinate of the Annotation.

### XML

```
<syncfusion:ImageAnnotation Text="Annotation"
HorizontalTextAlignment="Center"
VerticalTextAlignment="Top" ImageSource="Images\Graduate.png" X1="2.5"
Y1="1200" X2="3.6" Y2="1700" >
</syncfusion:ImageAnnotation>
```

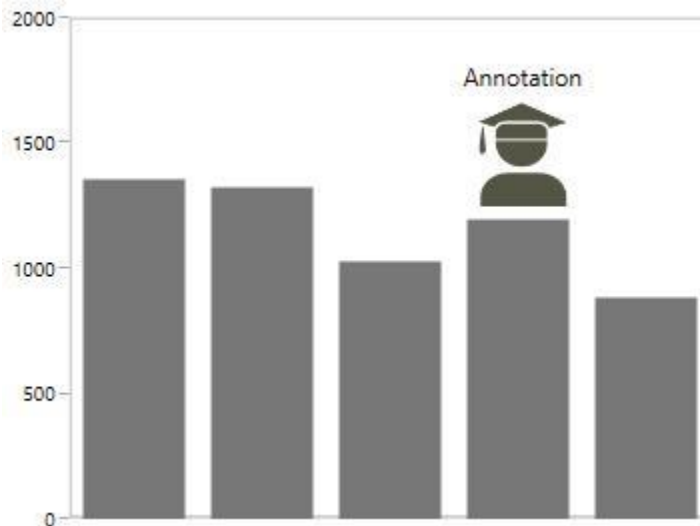
### C#

```
ImageAnnotation annotation = new ImageAnnotation()
{
    Text = "Annotation",
    HorizontalTextAlignment = HorizontalAlignment.Center,
    VerticalTextAlignment = VerticalAlignment.Top,
    X1 = 2.5,
    Y1 = 1200,
    X2 = 3.6,
    Y2 = "1700",
```

```

ImageSource = new BitmapImage(new Uri(@"Images\Graduate.png",
UriKind.RelativeOrAbsolute))
};

```



### Interaction

SfChart provides dragging and resizing support for [ShapeAnnotations](#).

The following API's are used for dragging and resizing the annotation

- [CanDrag](#)- A Boolean value that allows the annotation to drag.
- [CanResize](#)- A Boolean value that allows the annotation to resize.
- [DraggingMode](#)- Represents the dragging direction for the annotation.
- [ResizingMode](#)- Represents the resizing direction for the annotation.

### Dragging the Annotation

The following code example demonstrates the dragging the rectangle annotation.

#### XML

```

<chart:RectangleAnnotation X1="0.6" CanDrag="True" X2="2.2" Y2="1500"
Y1="1800"
Stroke="DarkGray" Fill="LightGray" Opacity="0.5">
</chart:RectangleAnnotation>

```

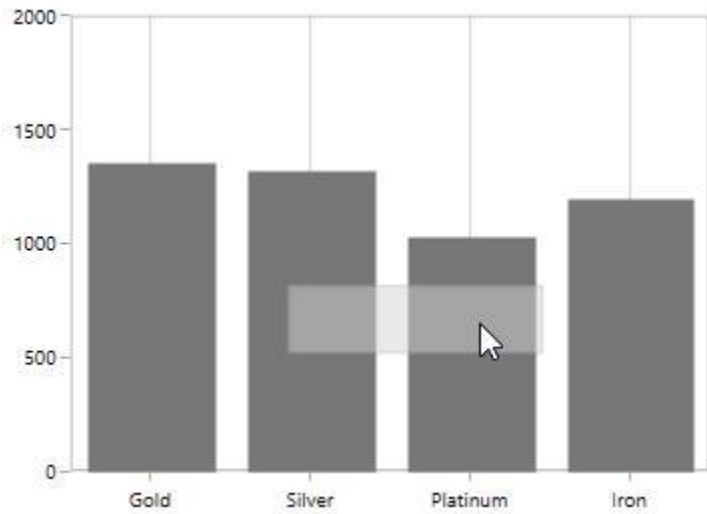
#### C#

```

RectangleAnnotation an = new RectangleAnnotation()
{
    X1 = 0.6,
    Y1 = 1800,
    X2 = 2.2,
    Y2 = 1500,
    Fill = new SolidColorBrush(Colors.LightGray),
    Stroke = new SolidColorBrush(Colors.DarkGray),
    CanDrag = true,
    Opacity = 0.5
}

```

```
};
```



Also, the direction of dragging can be customized by using [DraggingMode](#) property.

### Resizing the Annotation

You can resize the annotation by enabling [CanResize](#) property to True as in the below code snippet.

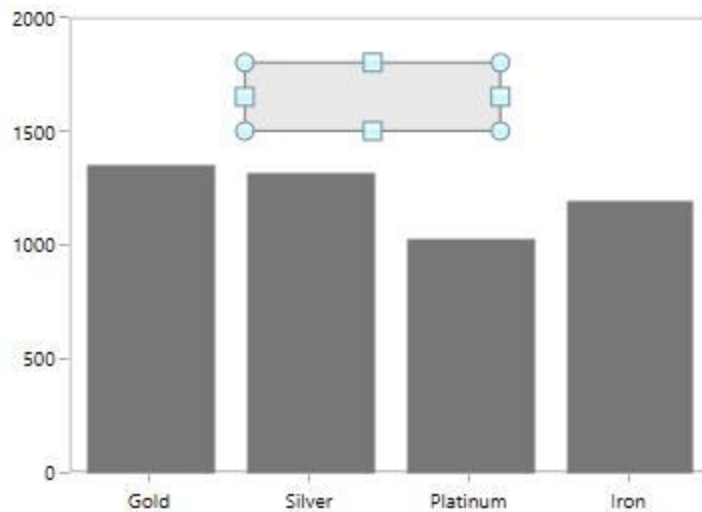
#### XML

```
<chart:RectangleAnnotation X1="0.6" CanResize="True" X2="2.2" Y2="1500"  
Y1="1800"  
Stroke="DarkGray" Fill="LightGray" Opacity="0.5" >  
</chart:RectangleAnnotation>
```

#### C#

```
RectangleAnnotation an = new RectangleAnnotation()  
{  
    X1 = 0.6,  
    Y1 = 1800,  
    X2 = 2.2,  
    Y2 = 1500,  
    Fill = new SolidColorBrush(Colors.LightGray),  
    Stroke = new SolidColorBrush(Colors.DarkGray),  
    CanDrag = true,  
    Opacity = 0.5  
};
```





Also, the direction of resizing can be customized by using [ResizingMode](#) property.

### ToolTip

SfChart provides support to view the tooltip when mouse hovered on the annotation. To view the tooltip you have to enable the [ShowToolTip](#) property. By default for tooltip there is no content, you have to set the content for the tooltip in [ToolTipContent](#) property.

- [ToolTipPlacement](#) - Used to position the Tooltip with top, bottom, left or right side of the cursor.
- [ToolTipTemplate](#) - Used to customize the default appearance of the Tooltip.

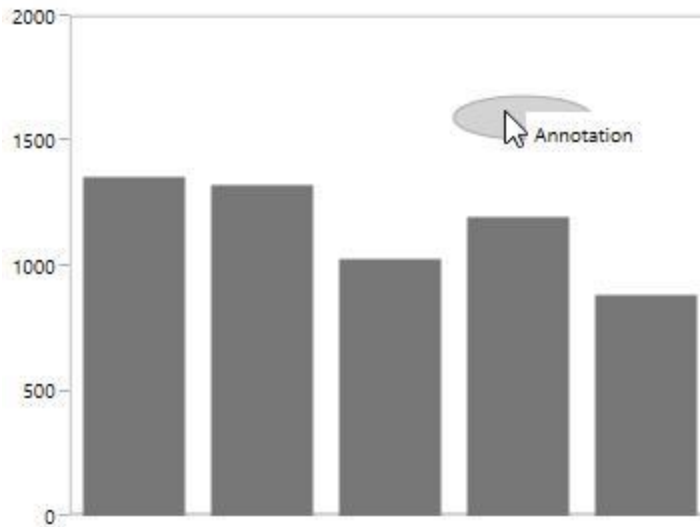
The following code example demonstrates the default tooltip.

### XML

```
<chart:EllipseAnnotation X1="2.5" Y1="1500" X2="3.6" Y2="1680"
ShowToolTip="True" ToolTipContent="Annotation">
</chart:EllipseAnnotation>
```

### C#

```
SfChart chart = new SfChart();
EllipseAnnotation annotation=new EllipseAnnotation ()
{
    X1 = 2.5,
    Y1 = 1500,
    X2 = 3.6,
    Y2 = 1680,
    Stroke = new SolidColorBrush(Colors.DarkGray),
    Fill = new SolidColorBrush (Colors.LightGray),
    ShowToolTip = true ,
    ToolTipContent = "Annotation"
};
chart.Annotations.Add(annotation);
```



### ToolTipTemplate

The default appearance of the Tooltip can be changed using [ToolTipTemplate](#) property as in the below code snippet.

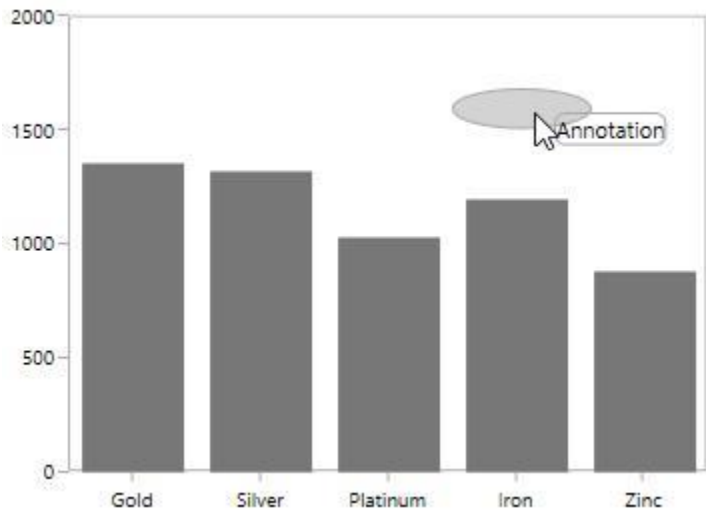
#### XML

```
<syncfusion:SfChart x:Name="chart">
  <syncfusion:SfChart.Resources>
    <DataTemplate x:Key="tooltipTemplate">
      <Border CornerRadius="5" BorderBrush="DarkGray" BorderThickness="1">
        <TextBlock FontSize="11" Text="Annotation"
          Foreground="Black"/>
      </Border>
    </DataTemplate>
  </syncfusion:SfChart.Resources>
  <syncfusion:SfChart.Annotations>
    <syncfusion:EllipseAnnotation X1="2.5" Y1="1500" Stroke="DarkGray"
      Fill="LightGray" ShowToolTip="True" X2="3.6" Y2="1680"
      ToolTipTemplate="{StaticResource tooltipTemplate}">
    </syncfusion:EllipseAnnotation>
  </syncfusion:SfChart.Annotations>
</syncfusion:SfChart>
```

#### C#

```
SfChart chart = new SfChart();
EllipseAnnotation annotation=new EllipseAnnotation ()
{
    X1 = 2.5,
    Y1 = 1500,
    X2 = 3.6,
    Y2 = 1680,
    Stroke = new SolidColorBrush(Colors.DarkGray),
    Fill = new SolidColorBrush (Colors.LightGray),
    ShowToolTip = true ,
    ToolTipTemplate = chart.Resources["tooltipTemplate"] as DataTemplate
};
```

```
chart.Annotations.Add(annotation);
```



### Annotation Clipping

SfChart allows you to clip the annotation if the annotation crosses the boundary by setting [EnableClipping](#) property to True as in the below code snippet.

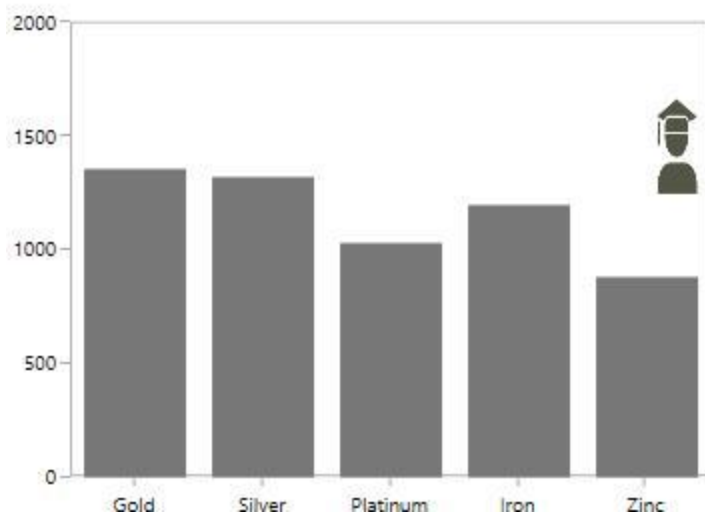
#### XML

```
<chart:SfChart.Annotations>  
<chart:ImageAnnotation ImageSource="Images\Graduate.png" X1="6" Y1="16"  
X2="9" Y2="18" EnableClipping="True"/>  
</chart:SfChart.Annotations>
```

#### C#

```
SfChart chart = new SfChart();  
ImageAnnotation image = new ImageAnnotation()  
{  
    X1 = 6,  
    Y1 = 16,  
    X2 = 9,  
    Y2 = 18,  
    ImageSource = new BitmapImage(new Uri("Images\Graduate.png",  
UriKind.RelativeOrAbsolute)),  
    EnableClipping=true  
};  
chart.Annotations.Add(image);
```

The following screenshot explains that even when x value is provided out of bounds the image annotation is placed inside the chart area.



### Annotation based on axis

The value of X1, X2, Y1, and Y2 properties of annotation will differ based on the axis type. The following table illustrates how to set the values for X1 and X2 properties of annotation based on the corresponding primary axis.

Sl.No	Axis Type	X1 and X2 values	Example
1	CategoryAxis	Index based	X1 = 2, X2 = 3 (start point = 2nd index™s value and end point 3rd index value)
2	DateTimeCategoryAxis	Index based	X1 = 2, X2 = 3 (start point = 2nd index™s value and end point 3rd index value)
3	DateTimeAxis	Value based	X1 = "2015/01/31", X2 = "2015/02/01"
4	TimeSpanAxis	Value based	X1= 00:00:40 X2=00:00:50
5	Logarithmic Axis	Value based	X1= 50(XValue) X2=50(XValue)
6	Numerical Axis	Value based	X1= 10(XValue) X2=15(XValue)

### DateTimeAxis

The corresponding DateTime value will be given as values for X1 and X2 properties.

### XML

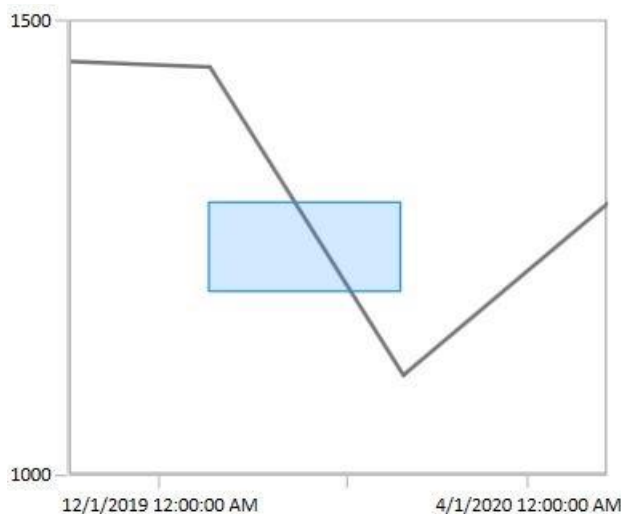
```
<chart:SfChart.PrimaryAxis>
<chart:DateTimeAxis />
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis />
</chart:SfChart.SecondaryAxis>
<chart:SfChart.Annotations>
<chart:RectangleAnnotation X1="2019/12/17" Y1="1200" X2="2020/02/19"
Y2="1300" >
</chart:RectangleAnnotation>
</chart:SfChart.Annotations>
```

**C#**

```

SfChart chart = new SfChart();
chart.PrimaryAxis = new DateTimeAxis();
chart.SecondaryAxis = new NumericalAxis();
RectangleAnnotation annotation = new RectangleAnnotation()
{
    X1 = new DateTime(2019, 12, 17), X2 = new DateTime(2020, 02, 19), Y1 = 1200, Y2
    = 1300,
};
chart.Annotations.Add(annotation);

```

**Events**

SfChart provides the following events in [Annotation](#).

- [Selected](#) - Occurs when the annotation is selected.
- [UnSelected](#) - Occurs when annotation is deselected.
- [DragStarted](#) - Occurs at the start of the dragging.
- [DragDelta](#) - Occurs when the drag takes place.
- [DragCompleted](#) - Occurs when the dragging is completed. You can cancel the dragging by using Cancel argument.
- [DragEnter](#) - Occurs when the cursor is moved over the annotation for dragging.
- [DragLeave](#) - Occurs when the cursor leaves the annotation after dragging.
- [MouseDown](#) - Occurs when any mouse button is pressed while the pointer is over the annotation.
- [MouseUp](#) - Occurs when any mouse button is released while the pointer is over the annotation.
- [MouseLeftButtonDown](#) - Occurs when the left mouse button is pressed while the mouse pointer is over the annotation.
- [MouseRightButtonDown](#) - Occurs when the right mouse button is pressed while the mouse pointer is over the annotation.
- [MouseRightButtonUp](#) - Occurs when the right mouse button is released while the mouse pointer is over the annotation.
- [MouseMove](#) - Occurs when the mouse pointer moves while over the annotation.

- [MouseLeave](#) - Occurs when the mouse pointer leaves the bounds of the annotation.

---

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configure with built-in support for creating stunning visual effects.

---

See also

[How to display trackball label or tooltip over chart annotation](#)

[How to add annotations by using MVVM binding](#)

[How to change the cursor of the annotation](#)

[How to bind the ViewModel property to content template of a TextAnnotation](#)

[How to draw horizontal line in chart](#)

## Vertical Charts in WPF Charts (SfChart)

SfChart provides support for vertical charts. You can plot vertical chart for any chart using [IsTransposed](#) and [OpposedPosition](#) properties.

### OpposedPosition

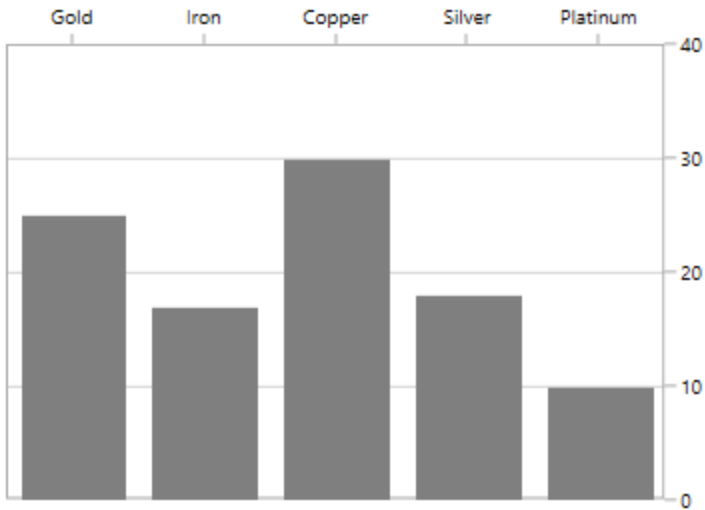
Allows to position the axis in the opposite direction to the default position. The following code example illustrates placing the primary and secondary axes in opposite direction.

### XML

```
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis    OpposedPosition="True" >
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis Minimum="0" Maximum="40" Interval="10"
OpposedPosition="True"/>
</chart:SfChart.SecondaryAxis>
```

### C#

```
chart.PrimaryAxis = new CategoryAxis()
{
    OpposedPosition = true
};
chart.SecondaryAxis = new NumericalAxis()
{
    Minimum = 0,
    Maximum = 40,
    Interval = 10,
    OpposedPosition = true
};
```



### IsTransposed

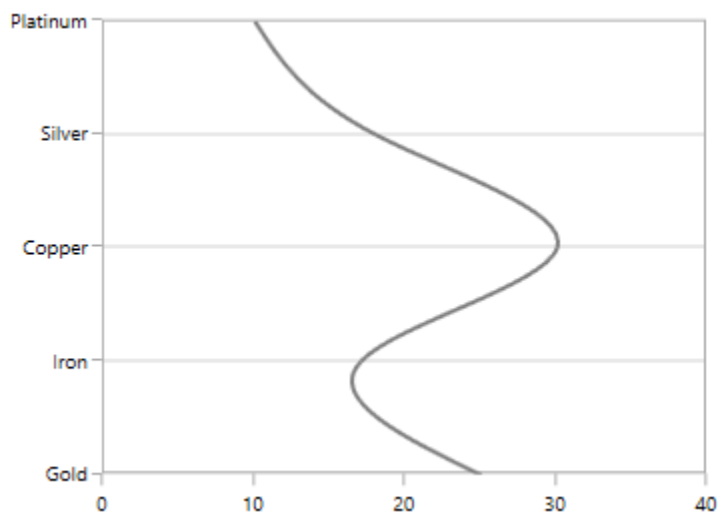
This property used to switch the plotting of the series to vertical.

### XML

```
<chart:LineSeries IsTransposed="True"
ItemsSource="{Binding SneakersDetail}" XBindingPath="Brand"
YBindingPath="ItemCount" >
```

### C#

```
LineSeries series = new LineSeries()
{
    IsTransposed = true,
    ItemsSource = new ViewModel().SneakersDetail,
    XBindingPath = "Brand",
    YBindingPath = "ItemCount"
};
```



The following example demonstrates the vertical charts.

### XML

```
<chart:SfChart>
<chart:SfChart.ColumnDefinitions>
<chart:ChartColumnDefinition />
<chart:ChartColumnDefinition/>
</chart:SfChart.ColumnDefinitions>
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis ShowGridLines="False" >
</chart:CategoryAxis>
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis/>
</chart:SfChart.SecondaryAxis>
<chart:LineSeries IsTransposed="True"
ItemsSource="{Binding SneakersDetail}" XBindingPath="Brand"
YBindingPath="ItemsCount" >
<chart:LineSeries.AdornmentsInfo>
<chart:ChartAdornmentInfo ShowMarker="True" Symbol="Ellipse"
SymbolHeight="10" SymbolInterior="#7f7f7f" SymbolWidth="10">
</chart:ChartAdornmentInfo>
</chart:LineSeries.AdornmentsInfo>
</chart:LineSeries>
<chart:LineSeries Interior="DarkGray" IsTransposed="True"
ItemsSource="{Binding SneakersDetail}" XBindingPath="Brand"
YBindingPath="postion" >
<chart:LineSeries.AdornmentsInfo>
<chart:ChartAdornmentInfo ShowLabel="False" ShowMarker="True"
Symbol="Ellipse" SymbolHeight="10"
SymbolInterior="DarkGray" SymbolWidth="10"></chart:ChartAdornmentInfo>
</chart:LineSeries.AdornmentsInfo>
<chart:LineSeries.YAxis>
<chart:NumericalAxis />
</chart:LineSeries.YAxis>
</chart:LineSeries>
</chart:SfChart>
```

### C#

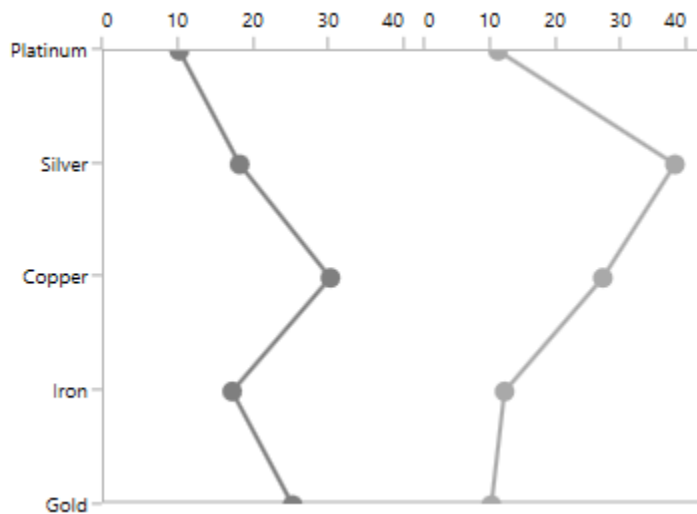
```
SfChart chart = new SfChart();
chart.ColumnDefinitions.Add(new ChartColumnDefinition());
chart.ColumnDefinitions.Add(new ChartColumnDefinition());
chart.PrimaryAxis = new CategoryAxis()
{
    ShowGridLines = true
};
NumericalAxis axis = new NumericalAxis();
chart.SecondaryAxis = axis;
ChartBase.SetColumn(axis, 1);
LineSeries series1 = new LineSeries()
{
    IsTransposed = true,
    ItemsSource = new ViewModel().SneakersDetail,
    XBindingPath = "Brand",
    YBindingPath = "ItemsCount"
```



```

};
ChartAdornmentInfo adornmentInfo1 = new ChartAdornmentInfo()
{
    ShowMarker = true,
    Symbol = ChartSymbol.Ellipse,
    SymbolHeight = 10,
    SymbolWidth = 10,
    SymbolInterior = new SolidColorBrush(Color.FromRgb(0x7f, 0x7f, 0x7f)),
};
LineSeries series2 = new LineSeries()
{
    IsTransposed = true,
    Interior = new SolidColorBrush(Colors.DarkGray),
    ItemsSource = new ViewModel().SneakersDetail,
    XBindingPath = "Brand",
    YBindingPath = "position",
    YAxis = new NumericalAxis()
};
ChartAdornmentInfo adornmentInfo2 = new ChartAdornmentInfo()
{
    ShowLabel = false,
    ShowMarker = true,
    Symbol = ChartSymbol.Ellipse,
    SymbolHeight = 10,
    SymbolWidth = 10,
    SymbolInterior = new SolidColorBrush(Colors.DarkGray),
};
series1.AdornmentsInfo = adornmentInfo1;
series2.AdornmentsInfo = adornmentInfo2;
ChartBase.SetColumn(series1, 0);
ChartBase.SetColumn(series2, 1);
chart.Series.Add(series1);
chart.Series.Add(series2);

```



**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

## Striplines in WPF Charts (SfChart)

SfChart allows you to add Striplines to the chart, which shades the specific region or range in the plot area background at regular or custom intervals.

### Positioning the Striplines

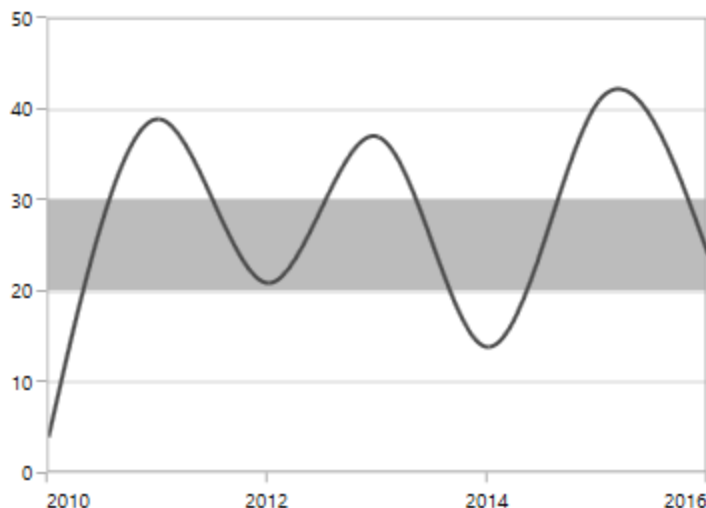
SfChart provides [Start](#) and [Width](#) property for defining the Stripline start and end range. These values correspond to the axis values (or range). The Stripline can be filled using brush set in [Background](#) property.

### XML

```
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis >
<syncfusion:NumericalAxis.StripLines>
<syncfusion:ChartStripLine Start="20" Width="10" Background="#BCBCBC"/>
</syncfusion:NumericalAxis.StripLines>
</syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>
```

### C#

```
NumericalAxis axis = new NumericalAxis();
ChartStripLine stripline = new ChartStripLine()
{
    Start = 20, Width = 10,
    Background = new SolidColorBrush(Color.FromRgb(0xBC, 0xBC, 0xBC))
};
axis.StripLines.Add(stripline);
chart.SecondaryAxis = axis;
```



### Position based on device coordinates

You can specify the stripline width in pixel by enabling [IsPixelWidth](#) Boolean property. By default, this property value is false.

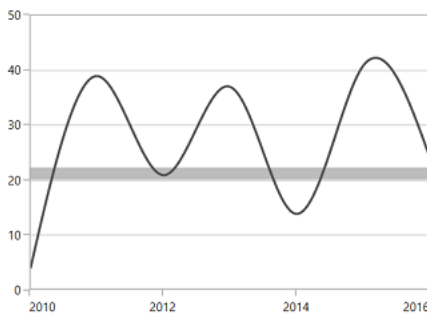
The following code example illustrates the positioning of stripline based on pixels.

### XML

```
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis >
<syncfusion:NumericalAxis.StripLines>
<syncfusion:ChartStripLine Start="20" Width="10"
IsPixelWidth="True" Background="#BCBCBC"/>
</syncfusion:NumericalAxis.StripLines>
</syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>
```

**C#**

```
NumericalAxis axis = new NumericalAxis();
ChartStripLine stripline = new ChartStripLine()
{
    Start = 20, Width = 10,
    IsPixelWidth = true,
    Background = new SolidColorBrush(Color.FromRgb(0xBC, 0xBC, 0xBC))
};
axis.StripLines.Add(stripline);
chart.SecondaryAxis = axis;
```

**Label**

We can define any text inside the stripline using [Label](#) property. Also SfChart provides various customization options for this label like alignment, templates, etc.

The [LabelHorizontalAlignment](#) and [LabelVerticalAlignment](#) property can be used for positioning the labels inside the stripline.

**XML**

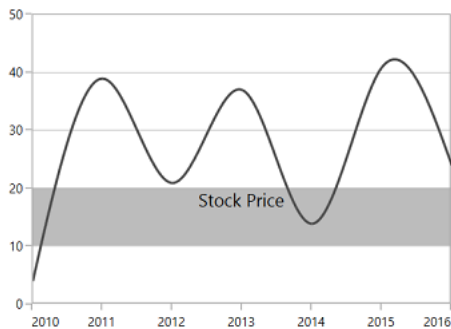
```
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis>
<syncfusion:NumericalAxis.StripLines>
<syncfusion:ChartStripLine Start="10" Width="10"
Label="Stock Price"
LabelHorizontalAlignment="Center"
LabelVerticalAlignment="Top"
Background="#BCBCBC"/>
</syncfusion:NumericalAxis.StripLines>
</syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>
```

**C#**

```

NumericalAxis axis = new NumericalAxis();
ChartStripLine stripline = new ChartStripLine()
{
    Start = 20, Width = 10,
    Label = "Stock Price",
    LabelHorizontalAlignment = HorizontalAlignment.Center,
    LabelVerticalAlignment = VerticalAlignment.Top,
    Background = new SolidColorBrush(Color.FromRgb(0xBC, 0xBC, 0xBC))
};
axis.StripLines.Add(stripline);
chart.SecondaryAxis = axis;

```



#### Rotating the label

The label can be rotated to the specified angle using [LabelAngle](#) property. The following code example explains the rotation of stripline label:

#### XML

```

<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis>
<syncfusion:NumericalAxis.StripLines>
<syncfusion:ChartStripLine Start="10" Width="20"
Label="Stock Price"
LabelAngle="-45"
Background="#BCBCBC"/>
</syncfusion:NumericalAxis.StripLines>
</syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>

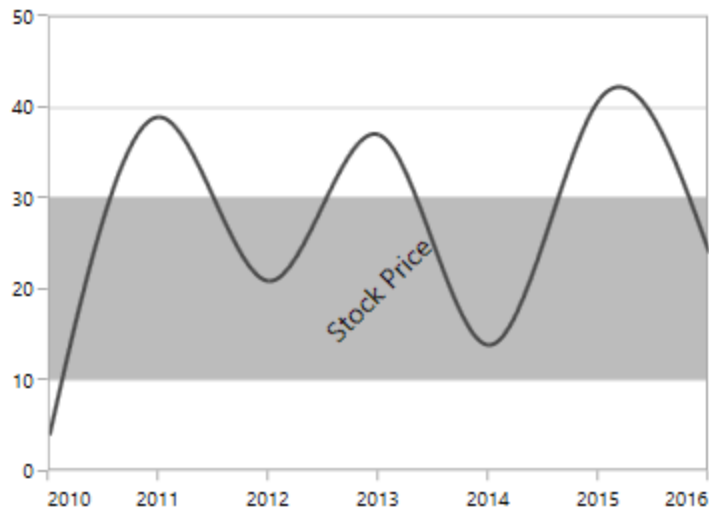
```

#### C#

```

NumericalAxis axis = new NumericalAxis();
ChartStripLine stripline = new ChartStripLine()
{
    Start = 20, Width = 10,
    Label = "Stock Price",
    LabelAngle = -45,
    Background = new SolidColorBrush(Color.FromRgb(0xBC, 0xBC, 0xBC))
};
axis.StripLines.Add(stripline);
chart.SecondaryAxis = axis;

```



**Note:** Here, Start and Width of the stripline as adjusted based on the rotation angle. Stripline won't adjust its range based on the stripline angle.

*Template support for the label*

[LabelTemplate](#) property allows you to define the data template for the stripline label like the following code example.

#### XML

```
<syncfusion:SfChart x:Name="chart">
  <syncfusion:SfChart.Resources>
    <DataTemplate x:Key="labelTemplate">
      <Border Background="Gray" CornerRadius="5" >
        <TextBlock Text="{Binding }" Foreground="White"
          FontStyle="Normal" FontSize="10"
          FontWeight="Bold" Margin="3"/>
      </Border>
    </DataTemplate>
  </syncfusion:SfChart.Resources>
  <syncfusion:SfChart.PrimaryAxis>
    <syncfusion:NumericalAxis>
      <syncfusion:NumericalAxis.StripLines>
        <syncfusion:ChartStripLine Width="20" Start="10"
          Label="Stock Price"
          Background="#C3C3C3"
          LabelTemplate="{StaticResource labelTemplate}"/>
      </syncfusion:NumericalAxis.StripLines>
    </syncfusion:NumericalAxis>
  </syncfusion:SfChart.PrimaryAxis>
</syncfusion:SfChart>
```

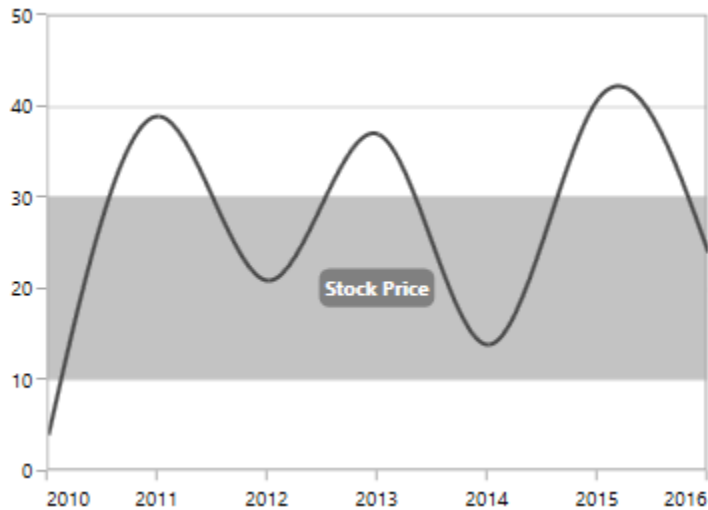
#### C#

```
SfChart chart = new SfChart();
chart.PrimaryAxis = new NumericalAxis();
ChartStripLine stripline = new ChartStripLine()
{
    Width = 20,
```

```

Start = 10,
Label = "Stock Price",
Background = new SolidColorBrush(Color.FromRgb(0xC3, 0XC3, 0XC3)),
LabelTemplate = chart.Resources["labelTemplate"] as DataTemplate
};
chart.PrimaryAxis.StripLines.Add(stripline);

```



### Multiple Striplines

You can add multiple number of striplines in the same axis like the following code example,

### XML

```

<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis >
<syncfusion:NumericalAxis.StripLines>
<syncfusion:ChartStripLine
SegmentStartValue="0"
Width="10"
Start="0" Label="Low" Background="#C3C3C3"/>
<syncfusion:ChartStripLine
Width="10" Start="20"
Label="Average" Background="#C3C3C3"/>
<syncfusion:ChartStripLine
Width="10"
Start="40" Label="High" Background="#C3C3C3"/>
</syncfusion:NumericalAxis.StripLines>
</syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>

```

### C#

```

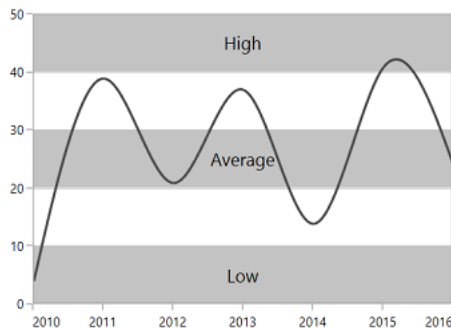
NumericalAxis axis = new NumericalAxis();
ChartStripLine stripline1 = new ChartStripLine()
{
SegmentStartValue = 0, Start = 0,
Width = 10, Label = "Low",
Background = new SolidColorBrush(Color.FromRgb(0xC3, 0xC3, 0xC3))
};

```

```

ChartStripLine stripline2 = new ChartStripLine()
{
    Start = 20, Width = 10,
    Label = "Average",
    Background = new SolidColorBrush(Color.FromRgb(0xC3, 0xC3, 0xC3))
};
ChartStripLine stripline3 = new ChartStripLine()
{
    Start = 40, Width = 10,
    Label = "High",
    Background = new SolidColorBrush(Color.FromRgb(0xC3, 0xC3, 0xC3))
};
axis.StripLines.Add(stripline1);
axis.StripLines.Add(stripline2);
axis.StripLines.Add(stripline3);
chart.SecondaryAxis = axis;

```



If you want to repeat the same type of stripline at regular intervals, SfChart provides two properties [RepeatEvery](#) and [RepeatUntil](#).

**Note:** This can be used to fill plot area background alternatively.

### XML

```

<syncfusion:NumericalAxis.StripLines>
<syncfusion:ChartStripLine RepeatEvery="20" RepeatUntil="50"
Width="10" Start="0" Background="#C3C3C3"/>
<syncfusion:ChartStripLine RepeatEvery="20" RepeatUntil="50"
Width="10" Start="10" Background="#A3A3A3"/>
</syncfusion:NumericalAxis.StripLines>

```

### C#

```

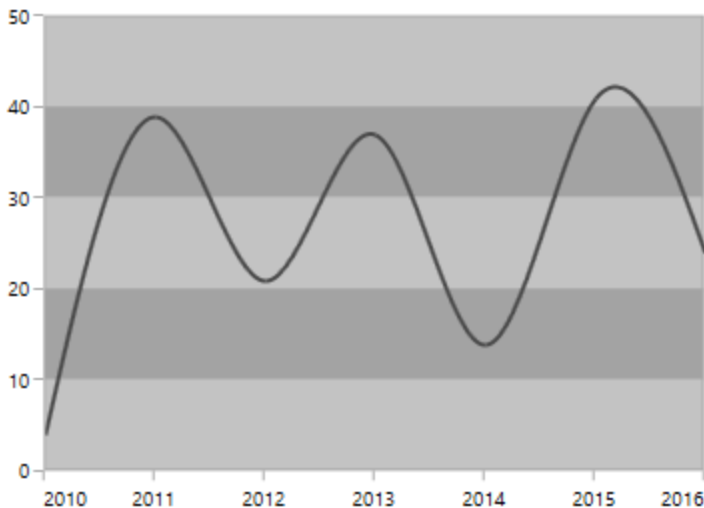
NumericalAxis axis = new NumericalAxis();
ChartStripLine stripline1 = new ChartStripLine()
{
    RepeatEvery = 20,
    RepeatUntil = 50,
    Start = 0, Width = 10,
    Background = new SolidColorBrush(Color.FromRgb(0xC3, 0xC3, 0xC3))
};
ChartStripLine stripline2 = new ChartStripLine()
{

```

```

RepeatEvery = 20,
RepeatUntil = 50,
Start = 10, Width = 10,
Background = new SolidColorBrush(Color.FromRgb(0xA3, 0xA3, 0xA3))
};
axis.StripLines.Add(stripline1);
axis.StripLines.Add(stripline2);
chart.SecondaryAxis = axis;

```



### Segmented Stripline

Striplines can also be placed in a particular region with respect to segment. You can enable the segment striplines using [IsSegmented](#) property.

So the start and end value of this type of striplines can be defined using [SegmentStartValue](#) and [SegmentEndValue](#) property.

The following code example demonstrates segmented striplines.

### XML

```

<syncfusion:SfChart.SecondaryAxis>
  <syncfusion:NumericalAxis >
    <syncfusion:NumericalAxis.StripLines>
      <syncfusion:ChartStripLine
        IsSegmented="True" SegmentStartValue="0"
        Width="10" SegmentEndValue="2"
        SegmentAxisName="Segment1"
        Start="0" Label="Low" Background="#C3C3C3"/>
      <syncfusion:ChartStripLine
        IsSegmented="True" SegmentStartValue="2"
        SegmentAxisName="Segment2"
        Width="10" SegmentEndValue="4" Start="20"
        Label="Average" Background="#C3C3C3"/>
      <syncfusion:ChartStripLine
        IsSegmented="True" SegmentStartValue="4"
        Width="10" SegmentEndValue="6"
        SegmentAxisName="Segment3"

```



```

Start="40" Label="High" Background="#C3C3C3"/>
</syncfusion:NumericalAxis.StripLines>
</syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>

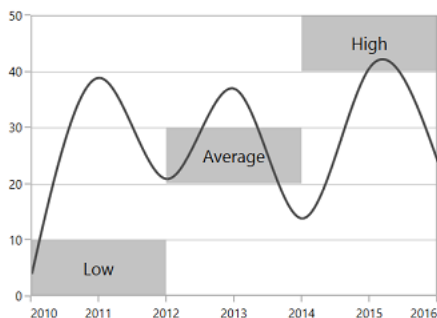
```

**C#**

```

NumericalAxis axis = new NumericalAxis();
ChartStripLine stripline1 = new ChartStripLine()
{
    IsSegmented = true,
    SegmentStartValue = 0,
    SegmentEndValue = 2,
    SegmentAxisName = "Segment1",
    Start = 0, Width = 10,
    Label = "Low",
    Background = new SolidColorBrush(Color.FromRgb(0xC3, 0xC3, 0xC3))
};
ChartStripLine stripline2 = new ChartStripLine()
{
    IsSegmented = true,
    SegmentStartValue = 2,
    SegmentEndValue = 4,
    SegmentAxisName = "Segment2",
    Start = 20, Width = 10,
    Label = "Average",
    Background = new SolidColorBrush(Color.FromRgb(0xC3, 0xC3, 0xC3))
};
ChartStripLine stripline3 = new ChartStripLine()
{
    IsSegmented = true,
    SegmentStartValue = 4,
    SegmentEndValue = 6,
    SegmentAxisName = "Segment3",
    Start = 40, Width = 10,
    Label = "High",
    Background = new SolidColorBrush(Color.FromRgb(0xC3, 0xC3, 0xC3))
};
axis.StripLines.Add(stripline1);
axis.StripLines.Add(stripline2);
axis.StripLines.Add(stripline3);
chart.SecondaryAxis = axis;

```



### Customization

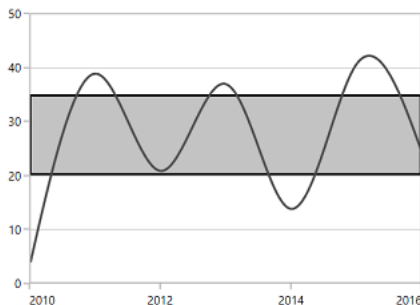
The background, border brush and border thickness of the stripline can be modified using [Background](#), [BorderBrush](#) and [BorderThickness](#) properties as in the following code example.

#### XML

```
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis >
<syncfusion:NumericalAxis.StripLines>
<syncfusion:ChartStripLine Start="20" Width="15"
BorderBrush="Black"
BorderThickness="2"
Background="#C3C3C3"/>
</syncfusion:NumericalAxis.StripLines>
</syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>
```

#### C#

```
NumericalAxis axis = new NumericalAxis();
ChartStripLine stripline = new ChartStripLine()
{
    Start = 20, Width = 15,
    BorderThickness = new Thickness(2),
    BorderBrush = new SolidColorBrush(Colors.Black),
    Background = new SolidColorBrush(Color.FromRgb(0xC3, 0xC3, 0xC3))
};
axis.StripLines.Add(stripline);
chart.SecondaryAxis = axis;
```



### Transparency

You can set the transparency for the striplines using `Opacity` property as in the following code snippets.

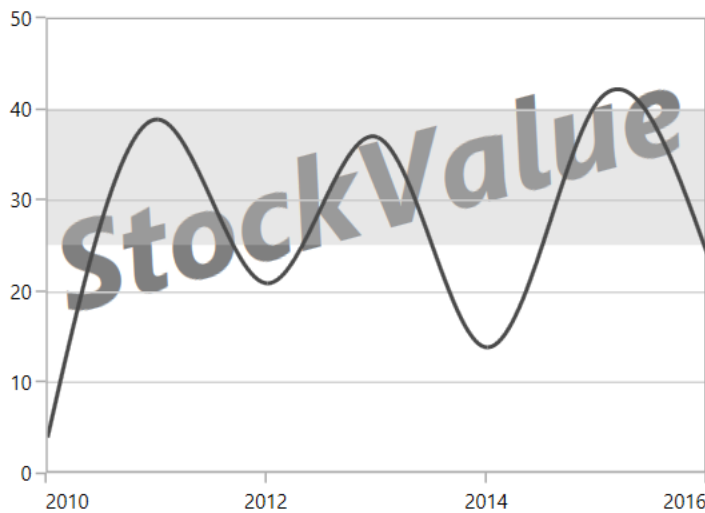
#### XML

```
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis >
<syncfusion:NumericalAxis.StripLines>
<syncfusion:ChartStripLine Start="25" Width="15"
Opacity="0.4" Background="#C3C3C3"/>
</syncfusion:NumericalAxis.StripLines>
</syncfusion:NumericalAxis>
</syncfusion:SfChart.SecondaryAxis>
```

```
<syncfusion:SfChart.Watermark>
<syncfusion:Watermark >
<syncfusion:Watermark.Content>
<TextBlock Text="StockValue">
<TextBlock.RenderTransform>
<RotateTransform Angle="345"/>
</TextBlock.RenderTransform>
</TextBlock>
</syncfusion:Watermark.Content>
</syncfusion:SfChart.Watermark>
```

## C#

```
NumericalAxis axis = new NumericalAxis();
ChartStripLine stripline = new ChartStripLine()
{
    Start = 25, Width = 15,
    Opacity = 0.4,
    Background = new SolidColorBrush(Color.FromRgb(0xC3, 0xC3, 0xC3))
};
axis.StripLines.Add(stripline);
chart.SecondaryAxis = axis;
chart.Watermark = new Watermark()
{
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center
};
TextBlock textBlock = new TextBlock();
textBlock.Text = "StockValue";
textBlock.FontSize = 70;
textBlock.RenderTransform = new RotateTransform() { Angle = 345 };
chart.Watermark.Content = textBlock;
```



---

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configure with built-in support for creating stunning visual effects.

---

See also

[How to display striplines in DateTimeAxis of WPF Chart](#)

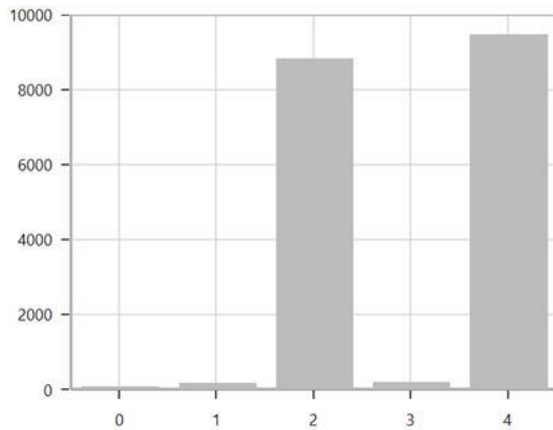
### Scale Breaks in WPF Charts (SfChart)

Scale break is a stripe drawn in the chart area to denote the break in the continuity of data points. Scale breaks are useful when there is a large difference in the data points. Scale break allows you to have different ranges on the same axis to visualize the data effectively.

#### Positioning the Breaks

SfChart provides [Start](#) and [End](#) properties for defining the scale break range (ranges that need to be skipped). These values are based on axis values.

The following image has data points with both greater and smaller magnitude, but the segments with smaller values are not visualized properly.



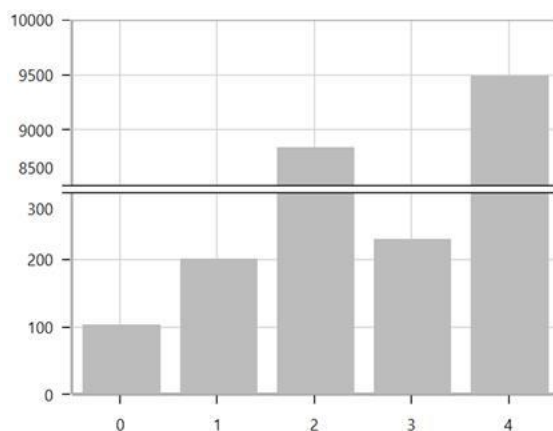
Applying scale breaks helps in proper visualization of all the data points.

### XML

```
<chart:SfChart.SecondaryAxis>  
  <chart:NumericalAxis>  
    <chart:NumericalAxis.AxisScaleBreaks>  
      <chart:ChartAxisScaleBreak Start="300"  
        End="8500">  
      </chart:ChartAxisScaleBreak>  
    </chart:NumericalAxis>  
  </chart:SfChart.SecondaryAxis>
```

### C#

```
NumericalAxis axis = new NumericalAxis();  
ChartAxisScaleBreak scaleBreak = new ChartAxisScaleBreak();  
scaleBreak.Start = 300;  
scaleBreak.End = 8500;  
axis.AxisScaleBreaks.Add(scaleBreak);  
chart.SecondaryAxis = axis;
```



### Break Position Customization

For the defined break range, its position in the chart area can be customized using the [BreakPosition](#) property in numerical axis.

Break position is determined based on the following factors:

#### *Data Count*

Based on the number of data points that fall in axis ranges (other than break range) scale break will be positioned.

In the below image the range [0,350] contains maximum number of data compared to the range [8000, 10000] hence the break is positioned in such a way that allocates more space to the range [0,350].

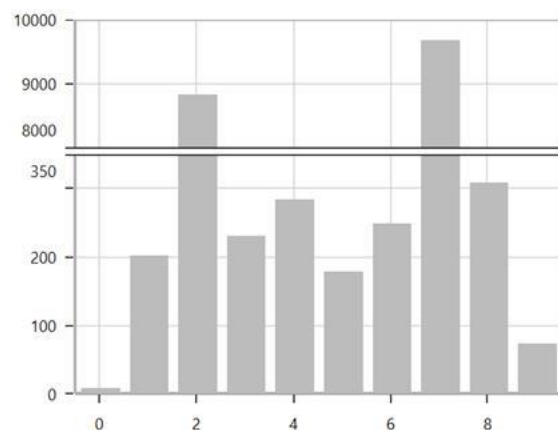
Range [0,350] takes nearly 4/5th of the axis height and the range [8000,10000] takes 1/5th.

#### **XML**

```
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis x:Name="axis" BreakPosition="DataCount">
<chart:NumericalAxis.AxisScaleBreaks>
<chart:ChartAxisScaleBreak Start="350"
End="8000">
</chart:ChartAxisScaleBreak>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>
```

#### **C#**

```
NumericalAxis axis = new NumericalAxis();
axis.BreakPosition = ScaleBreakPosition.DataCount;
ChartAxisScaleBreak scaleBreak = new ChartAxisScaleBreak();
scaleBreak.Start = 350;
scaleBreak.End = 8000;
axis.AxisScaleBreaks.Add(scaleBreak);
chart.SecondaryAxis = axis;
```



#### *Scale*

[Scale](#) option allows you to position the breaks based on the delta of each axis range relative to the other.

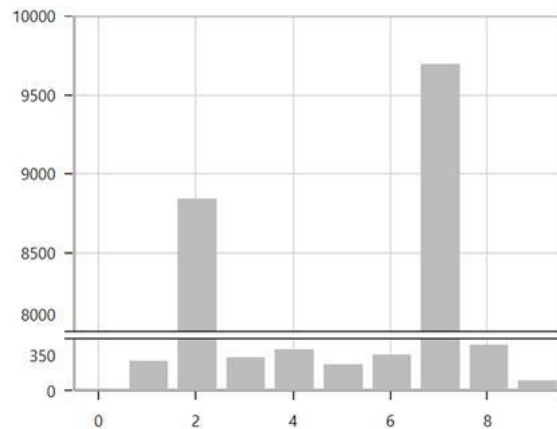
#### **XML**

```
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis x:Name="axis" BreakPosition="Scale">
```

```
<chart:NumericalAxis.AxisScaleBreaks>
<chart:ChartAxisScaleBreak Start="350"
End="8000">
</chart:ChartAxisScaleBreak>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>
```

**C#**

```
NumericalAxis axis = new NumericalAxis();
axis.BreakPosition = ScaleBreakPosition.Scale;
ChartAxisScaleBreak scaleBreak = new ChartAxisScaleBreak();
scaleBreak.Start = 350;
scaleBreak.End = 8000;
axis.AxisScaleBreaks.Add(scaleBreak);
chart.SecondaryAxis = axis;
```

*Percent*

Percent option allows to position the breaks at the specified percentage of the axis available height.

Percentage can be specified using [BreakPercent](#) property. Default BreakPercent value is 50.

In the below image, each break is given percent value as 50. First break is positioned at the 50% of the axis, and the next break is positioned at 50% of the remaining space and so on.

**XML**

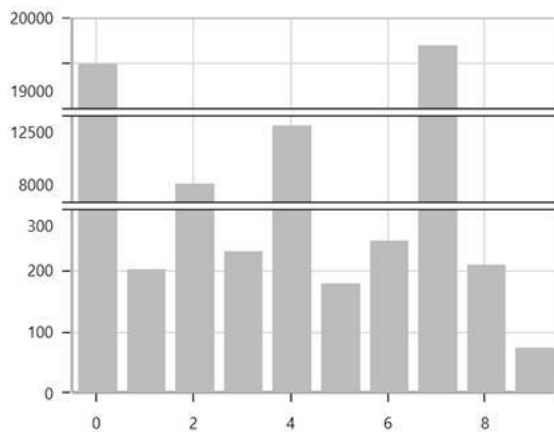
```
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis x:Name="axis" BreakPosition="Percent ">
<chart:NumericalAxis.AxisScaleBreaks>
<chart:ChartAxisScaleBreak Start="300"
End="8000" BreakPercent="50">
</chart:ChartAxisScaleBreak>
<chart:ChartAxisScaleBreak Start="12500"
End="19000" BreakPercent="50">
</chart:ChartAxisScaleBreak>
</chart:NumericalAxis.AxisScaleBreaks>
</chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>
```

**C#**

```

NumericalAxis axis = new NumericalAxis();
axis.BreakPosition = ScaleBreakPosition.Percent;
ChartAxisScaleBreak scaleBreak1 = new ChartAxisScaleBreak();
scaleBreak1.Start = 300;
scaleBreak1.End = 8000;
scaleBreak1.BreakPercent = 50;
axis.AxisScaleBreaks.Add(scaleBreak1);
ChartAxisScaleBreak scaleBreak2 = new ChartAxisScaleBreak();
scaleBreak2.Start = 12500;
scaleBreak2.End = 19000;
scaleBreak2.BreakPercent = 50;
axis.AxisScaleBreaks.Add(scaleBreak2);
chart.SecondaryAxis = axis;

```



### Multiple Breaks

Multiple breaks can be included in the chart.

### XML

```

<chart:SfChart.SecondaryAxis>
  <chart:NumericalAxis>
    <chart:NumericalAxis.AxisScaleBreaks>
      <chart:ChartAxisScaleBreak Start="300"
        End="8000">
      </chart:ChartAxisScaleBreak>
      <chart:ChartAxisScaleBreak Start="12500"
        End="19000">
      </chart:ChartAxisScaleBreak>
    </chart:NumericalAxis.AxisScaleBreaks>
  </chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>

```

### C#

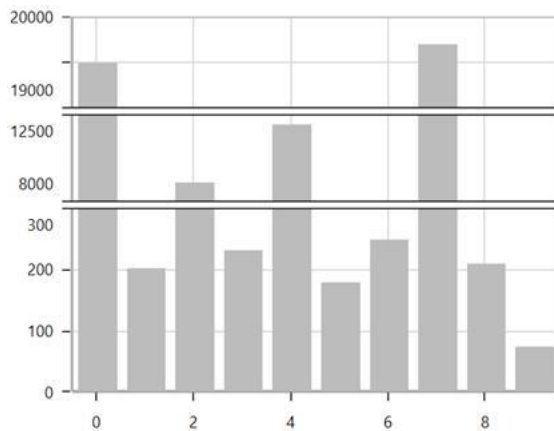
```

NumericalAxis axis = new NumericalAxis();
ChartAxisScaleBreak scaleBreak1 = new ChartAxisScaleBreak();
scaleBreak1.Start = 300;
scaleBreak1.End = 8000;
axis.AxisScaleBreaks.Add(scaleBreak1);
ChartAxisScaleBreak scaleBreak2 = new ChartAxisScaleBreak();
scaleBreak2.Start = 12500;

```



```
scaleBreak2.End = 19000;
axis.AxisScaleBreaks.Add(scaleBreak2);
chart.SecondaryAxis = axis;
```



### Customization

The following are the customizing options for scale break.

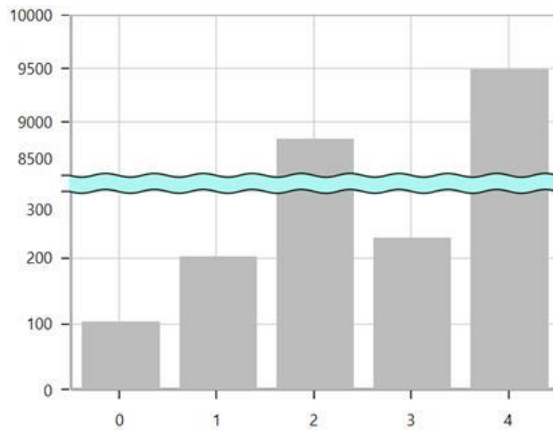
Line type such as [Wave](#) or [StraightLine](#), background, spacing, stroke, stroke thickness of the scale break can be customized using [LineType](#), [Fill](#), [BreakSpacing](#), [Stroke](#), [StrokeThickness](#) properties respectively.

### XML

```
<chart:SfChart.SecondaryAxis>
  <chart:NumericalAxis>
    <chart:NumericalAxis.AxisScaleBreaks>
      <chart:ChartAxisScaleBreak Start="300" BreakSpacing="12"
        End="8500" LineType="Wave"
        Fill="PaleTurquoise"
        Stroke="Black" StrokeThickness="1.2" >
      </chart:ChartAxisScaleBreak>
    </chart:NumericalAxis.AxisScaleBreaks>
  </chart:NumericalAxis>
</chart:SfChart.SecondaryAxis>
```

### C#

```
NumericalAxis axis = new NumericalAxis();
ChartAxisScaleBreak scaleBreak = new ChartAxisScaleBreak();
scaleBreak.Start = 300;
scaleBreak.End = 8500;
scaleBreak.LineType = BreakLineType.Wave;
scaleBreak.BreakSpacing = 12;
scaleBreak.Fill = new SolidColorBrush(Colors.PaleTurquoise);
scaleBreak.Stroke = new SolidColorBrush(Colors.Black);
scaleBreak.StrokeThickness = 1.2;
axis.AxisScaleBreaks.Add(scaleBreak);
chart.SecondaryAxis = axis;
```



**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

### Technical Indicators in WPF Charts (SfChart)

Technical indicators are the base of technical analysis, which are used to determine the future market trends.

#### Adding Technical Indicators to the Chart

Technical indicator merely an another type meta series. The following steps illustrates how to add the technical indicators to the chart:

##### Initializing Indicator

Create the instance for any technical indicator and add it to the [TechnicalIndicators](#) collection.

Here for instance, the [Accumulation Distribution](#) is added.

#### XML

```
<chart:SfChart.TechnicalIndicators>
  <chart:AccumulationDistributionIndicator>
</chart:AccumulationDistributionIndicator>
</chart:SfChart.TechnicalIndicators>
```

#### C#

```
AccumulationDistributionIndicator indicator = new
AccumulationDistributionIndicator();
chart.TechnicalIndicators.Add(indicator);
```

##### Binding the Data

Next you need to bind the property path for the [Open](#), [High](#), [Low](#) and [Close](#) along with x value binding property.

#### XML

```
<chart:SfChart.TechnicalIndicators>
  <chart:AccumulationDistributionIndicator Open="Open" Close="Close"
  High="High" Low="Low">
</chart:AccumulationDistributionIndicator>
</chart:SfChart.TechnicalIndicators>
```

**C#**

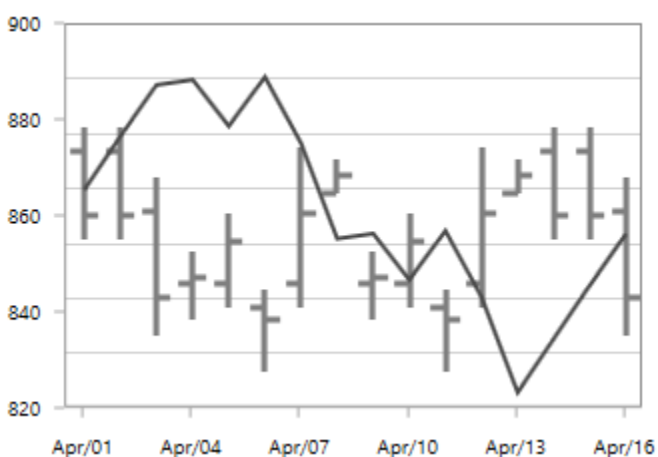
```
AccumulationDistributionIndicator indicator = new
AccumulationDistributionIndicator()
{
    Open = "Open", Close = "Close",
    High = "High", Low = "Low"
};
chart.TechnicalIndicators.Add(indicator);
```

*Specifying the ItemsSource***XML**

```
<chart:SfChart.TechnicalIndicators>
<chart:AccumulationDistributionIndicator Open="Open" Close="Close"
High="High"
Low="Low" ItemsSource="{Binding StockPriceDetails}"
XBindingPath="Date">
</chart:AccumulationDistributionIndicator>
</chart:SfChart.TechnicalIndicators>
```

**C#**

```
AccumulationDistributionIndicator indicator = new
AccumulationDistributionIndicator()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    Open = "Open", Close = "Close",
    High = "High", Low = "Low"
};
chart.TechnicalIndicators.Add(indicator);
```



The following sections covers all the different types of technical indicators available in SfChart.

Most of the indicators are having the [Period](#) and [SignalLineColor](#) properties as common, in which Period property indicates the moving average period and the SignalLineColor defines the color for the respective indicator line.

### Average True Range

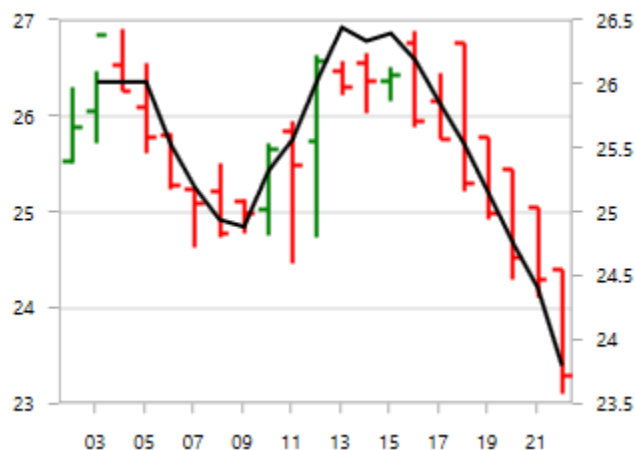
You can define the [AverageTrueRangeIndicator](#) using the following code example

#### XML

```
<chart:SfChart.TechnicalIndicators>
<chart:AverageTrueRangeIndicator ItemsSource="{Binding ViewModel1}"
Period="3" XBindingPath="Date" Volume="Volume"
SignalLineColor="Black" High="High" Low="Low"
Open="Open" Close="Close"/ >
</chart:SfChart.TechnicalIndicators>
```

#### C#

```
AverageTrueRangeIndicator indicator = new AverageTrueRangeIndicator()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    Open = "Open", Close = "Close",
    High = "High", Low = "Low",
    Volume = "Volume", Period = 3,
    SignalLineColor = new SolidColorBrush(Colors.Black)
};
chart.TechnicalIndicators.Add(indicator);
```



### Simple Average

The following code example demonstrates the usage of [SimpleAverageIndicator](#).

#### XML

```
<chart:SfChart.TechnicalIndicators>
<chart:SimpleAverageIndicator ItemsSource="{Binding ViewModel1}"
Period="3"
SignalLineColor="Black" XBindingPath="Date"
Volume="Volume"
High="High" Low="Low" Open="Open" Close="Close" >
```

```
</chart:SimpleAverageIndicator >
</chart:SfChart.TechnicalIndicators>
```

**C#**

```
SimpleAverageIndicator indicator = new SimpleAverageIndicator()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    Open = "Open", Close = "Close",
    High = "High", Low = "Low",
    Volume = "Volume", Period = 3,
    SignalLineColor = new SolidColorBrush(Colors.Black)
};
chart.TechnicalIndicators.Add(indicator);
```

**RSI**

The Relative Strength Index(RSI) indicators are having additional two lines other than signal line, which indicate the overbought and oversold region.

The [UpperLineColor](#) property is used to define the color for the line indicating overbought region and the [LowerLineColor](#) property is used to define the color for the line indicating oversold region.

To define the [RSITechnicalIndicator](#), you can use the following code example:

**XML**

```
<chart:SfChart.TechnicalIndicators>
<chart:RSITechnicalIndicator
    ItemsSource="{Binding ViewModel1}" Period="3"
    SignalLineColor="Black" XBindingPath="Date" Volume="Volume"
    UpperLineColor="Blue" LowerLineColor="Red"
    High="High" Low="Low" Open="Open" Close="Close"/>
</chart:SfChart.TechnicalIndicators>
```

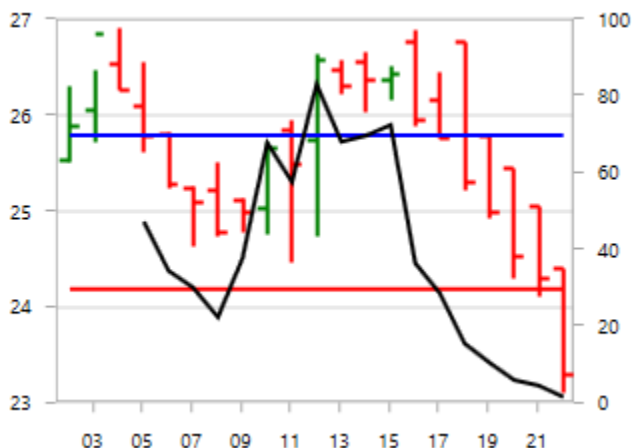
**C#**

```
RSITechnicalIndicator indicator = new RSITechnicalIndicator()
{
```

```

ItemsSource = new ViewModel().StockPriceDetails,
XBindingPath = "Date",
Open = "Open", Close = "Close",
High = "High", Low = "Low",
Volume = "Volume", Period = 3,
SignalLineColor = new SolidColorBrush(Colors.Black),
UpperLineColor = new SolidColorBrush(Colors.Blue),
LowerLineColor = new SolidColorBrush(Colors.Red)
};
chart.TechnicalIndicators.Add(indicator);

```



### Momentum

This indicator is having two lines momentum line and center line. No signal line is in this indicator. You can define momentum technical indicator using the following code example.

The [MomentumLineColor](#) property and [CenterLineColor](#) property are used to define the color for the momentum and center line respectively.

### XML

```

<chart:SfChart.TechnicalIndicators>
<chart:MomentumTechnicalIndicator ItemsSource="{Binding ViewModel1}"
Period="3" CenterLineColor="Blue" XBindingPath="Date"
Volume="Volume" MomentumLineColor="Black"
High="High" Low="Low" Open="Open" Close="Close"/ >
</chart:SfChart.TechnicalIndicators>

```

### C#

```

MomentumTechnicalIndicator indicator = new MomentumTechnicalIndicator()
{
ItemsSource = new ViewModel().StockPriceDetails,
XBindingPath = "Date",
Open = "Open", Close = "Close",
High = "High", Low = "Low",
Volume = "Volume", Period = 3,
MomentumLineColor = new SolidColorBrush(Colors.Black),
CenterLineColor = new SolidColorBrush(Colors.Red)
};

```

```
chart.TechnicalIndicators.Add(indicator);
```



### Stochastic

This indicator is used to measure the range and momentum of price movements. It contains [KPeriod](#) and [DPeriod](#) property defining the 'K' percentage and 'D' percentage respectively. No signal line in this indicator.

The [UpperLineColor](#), [LowerLineColor](#) and [PeriodLineColor](#) property are used to define the brushes for the Stochastic indicator lines.

You can define stochastic technical indicator using the following code example:

### XML

```
<chart:SfChart.TechnicalIndicators>
<chart:StochasticTechnicalIndicator ItemsSource="{Binding ViewModel1}"
Period="3" SignalLineColor="Black" KPeriod="8" DPeriod="5"
XBindingPath="Date" Volume="Volume" UpperLineColor="Blue"
LowerLineColor="LightBlue" PeriodLineColor="Blue"
High="High" Low="Low" Open="Open" Close="Close"/>
</chart:SfChart.TechnicalIndicators>
```

### C#

```
StochasticTechnicalIndicator indicator = new StochasticTechnicalIndicator()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date", Volume = "Volume",
    Open = "Open", Close = "Close",
    High = "High", Low = "Low",
    Period = 3, KPeriod = 8, DPeriod = 5,
    SignalLineColor = new SolidColorBrush(Colors.Black),
    PeriodLineColor = new SolidColorBrush(Colors.Red),
    UpperLineColor = new SolidColorBrush(Colors.Blue),
    LowerLineColor = new SolidColorBrush(Colors.Purple)
};
chart.TechnicalIndicators.Add(indicator);
```



### Exponential Average

The [ExponentialAverageIndicator](#) is similar to [SimpleAverageIndicator](#) and this can be defined using the following code examples.

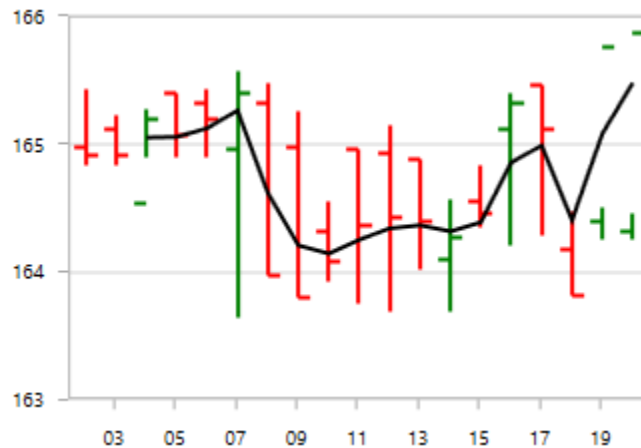
#### XML

```
<chart:SfChart.TechnicalIndicators>
<chart:ExponentialAverageIndicator ItemsSource="{Binding ViewModel1}"
Period="3" XBindingPath="Date" Volume="Volume"
SignalLineColor="Black" High="High" Low="Low"
Open="Open" Close="Close"/ >
</chart:SfChart.TechnicalIndicators>
```

#### C#

```
ExponentialAverageIndicator indicator = new ExponentialAverageIndicator()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    Open = "Open", Close = "Close",
    High = "High", Low = "Low",
    Volume = "Volume", Period = 3,
    SignalLineColor = new SolidColorBrush(Colors.Black)
};
chart.TechnicalIndicators.Add(indicator);
```





### Triangular Average

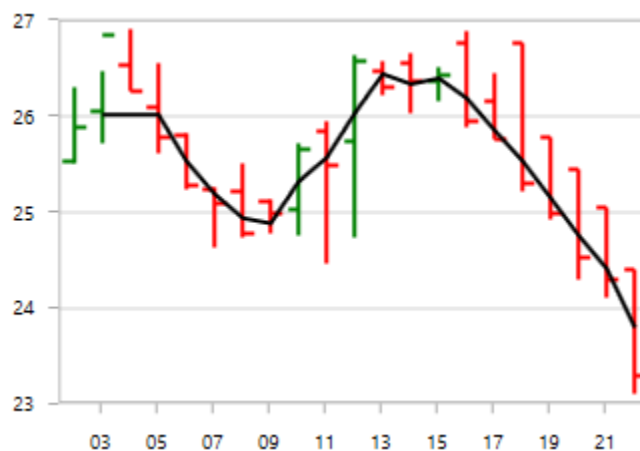
The [TriangularAverageIndicator](#) can be defined as in the following code example.

### XML

```
<chart:SfChart.TechnicalIndicators>
<chart:TriangularAverageIndicator ItemsSource="{Binding ViewModel1}"
Period="3" XBindingPath="Date" Volume="Volume"
SignalLineColor="Black" High="High" Low="Low"
Open="Open" Close="Close"/ >
</chart:SfChart.TechnicalIndicators>
```

### C#

```
TriangularAverageIndicator indicator = new TriangularAverageIndicator()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    Open = "Open", Close = "Close",
    High = "High", Low = "Low",
    Volume = "Volume", Period = 3,
    SignalLineColor = new SolidColorBrush(Colors.Black)
};
chart.TechnicalIndicators.Add(indicator);
```



### Accumulation Distribution

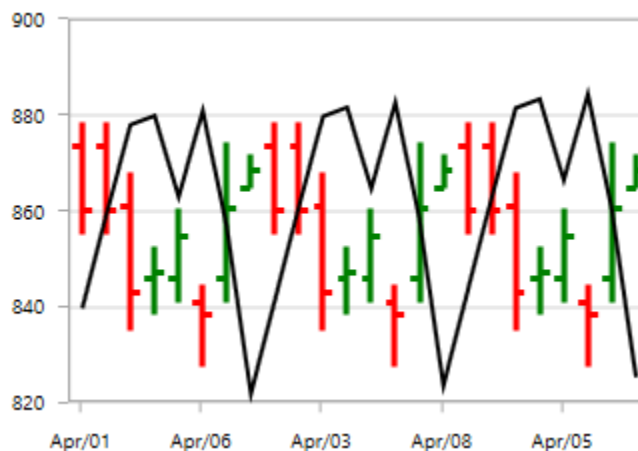
The following code example help you to add [AccumulationDistributionIndicator](#).

#### XML

```
<chart:SfChart.TechnicalIndicators>
<chart:AccumulationDistributionIndicator
ItemsSource="{Binding ViewModel1}"
XBindingPath="Date" Volume="Volume"
SignalLineColor="Black" High="High" Low="Low"
Open="Open" Close="Close" >
</chart:AccumulationDistributionIndicator >
</chart:SfChart.TechnicalIndicators>
```

#### C#

```
AccumulationDistributionIndicator indicator = new
AccumulationDistributionIndicator()
{
ItemsSource = new ViewModel().StockPriceDetails,
XBindingPath = "Date",
Open = "Open", Close = "Close",
High = "High", Low = "Low",
Volume = "Volume", Period = 3,
SignalLineColor = new SolidColorBrush(Colors.Black)
};
chart.TechnicalIndicators.Add(indicator);
```



### Bollinger Band

This indicator also having [UpperLineColor](#), [LowerLineColor](#) and [SignalLineColor](#) property for defining the brushes for the indicator lines.

You can define the [BollingerBandIndicator](#) using the following code example:

#### XML

```
<chart:SfChart.TechnicalIndicators>
<chart:BollingerBandIndicator
ItemsSource="{Binding ViewModel1}" Period="3"
UpperLineColor="Blue" LowerLineColor="Red"
```

```

XBindingPath="Date" Volume="Volume" SignalLineColor="Black"
High="High" Low="Low" Open="Open" Close="Close"/>
</chart:SfChart.TechnicalIndicators>

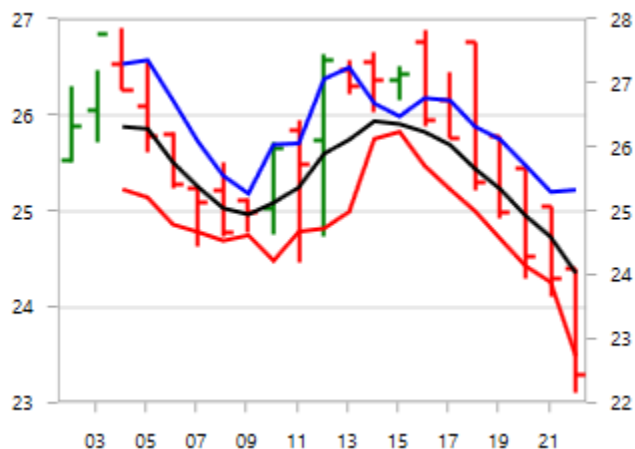
```

## C#

```

BollingerBandIndicator indicator = new BollingerBandIndicator()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    Open = "Open", Close = "Close",
    High = "High", Low = "Low",
    Volume = "Volume", Period = 3,
    SignalLineColor = new SolidColorBrush(Colors.Black),
    UpperLineColor = new SolidColorBrush(Colors.Blue),
    LowerLineColor = new SolidColorBrush(Colors.Red)
};
chart.TechnicalIndicators.Add(indicator);

```



## MACD

This is mostly using indicator having [ShortPeriod](#) and [LongPeriod](#) for defining the motion of the indicator.

Other than signal line, MACD is having convergence and divergence line. The brushes for these lines can be defined using [ConvergenceLineColor](#) and [DivergenceLineColor](#).

Also you can draw [Line](#), [Histogram](#) MACD or [Both](#) using the [Type](#) property, which defines the type of MACD to be drawn.

You can define the [MACDTechnicalIndicator](#) using the following code example:

## XML

```

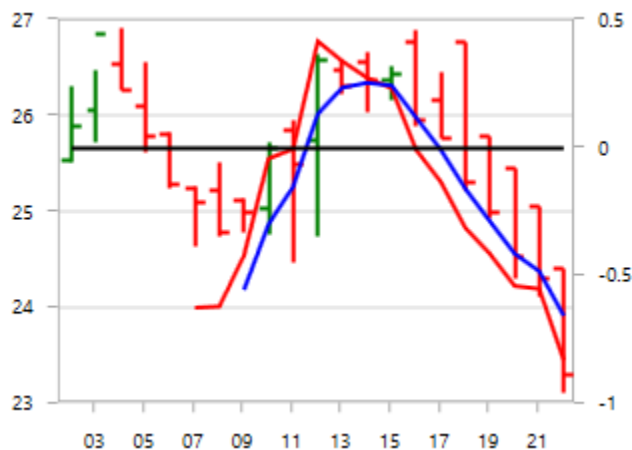
<chart:SfChart.TechnicalIndicators>
<chart:MACDTechnicalIndicator ItemsSource="{Binding ViewModel1}"
Type="Line" ShortPeriod="2" Period="3" LongPeriod="6"
ConvergenceLineColor="Red" DivergenceLineColor="Blue"
XBindingPath="Date" Volume="Volume" SignalLineColor="Black"
High="High" Low="Low" Open="Open" Close="Close" >
</chart:MACDTechnicalIndicator >

```

```
</chart:SfChart.TechnicalIndicators>
```

## C#

```
MACDTechnicalIndicator indicator = new MACDTechnicalIndicator()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date", Volume = "Volume",
    Open = "Open", Close = "Close",
    High = "High", Low = "Low",
    Period = 3, ShortPeriod = 2, LongPeriod = 6,
    Type = MACDType.Line,
    SignalLineColor = new SolidColorBrush(Colors.Black),
    ConvergenceLineColor = new SolidColorBrush(Colors.Red),
    DivergenceLineColor = new SolidColorBrush(Colors.Blue),
};
chart.TechnicalIndicators.Add(indicator);
```



**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configure with built-in support for creating stunning visual effects.

## Trendlines in WPF Charts (SfChart)

Trendlines are used to analyze and display the trends in the data graphically. It is built on the assumptions based on current and past price trends.

The following code examples illustrate how to add trend lines to the chart.

## XML

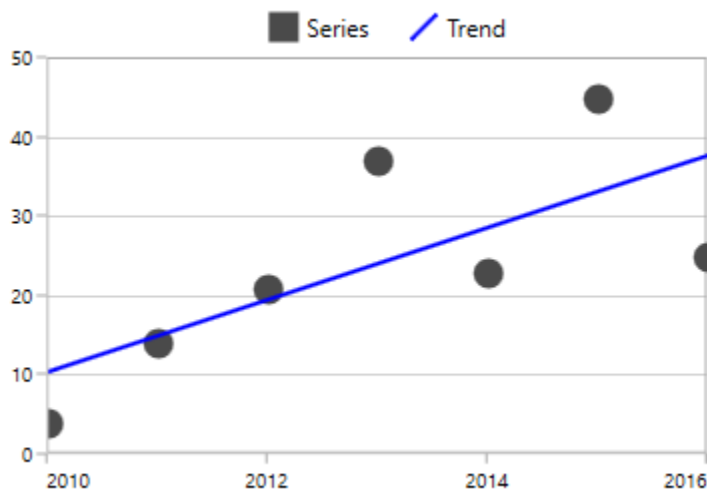
```
<syncfusion:ScatterSeries Interior="#4A4A4A" XBindingPath="Year"
Label="Series"
ItemsSource="{Binding List}" YBindingPath="India"
ScatterHeight="15" ScatterWidth="15">
<syncfusion:ScatterSeries.Trendlines>
<syncfusion:Trendline Label="Trend" />
</syncfusion:ScatterSeries.Trendlines>
</syncfusion:ScatterSeries>
```

**C#**

```

ScatterSeries scatterSeries = new ScatterSeries()
{
    ItemsSource = new ViewModel().List,
    XBindingPath = "Year",
    YBindingPath = "India",
    ScatterHeight = 15,
    ScatterWidth = 15,
    Interior = new SolidColorBrush(Color.FromRgb(0x4A, 0x4A, 0x4A)),
    Label = "Series"
};
Trendline trendline = new Trendline()
{
    Label = "Trend"
};
scatterSeries.Trendlines.Add(trendline);
chart.Series.Add(scatterSeries);

```



You can get the [Slope](#) and [Intercept](#) of the drawn trend line.

The visibility of the trend line is defined using [IsTrendlineVisible](#) property as in the following code examples.

**XML**

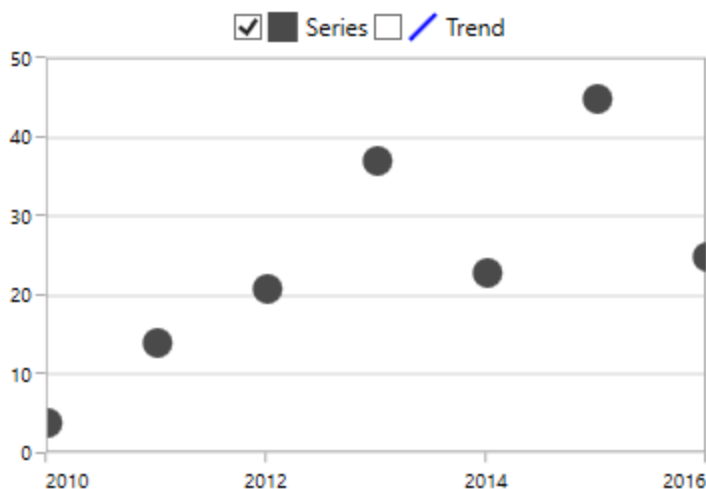
```

<syncfusion:ScatterSeries Interior="#4A4A4A" XBindingPath="Year"
Label="Series"
ItemsSource="{Binding List}" YBindingPath="India"
ScatterHeight="15" ScatterWidth="15">
<syncfusion:ScatterSeries.Trendlines>
<syncfusion:Trendline Label="Trend" IsTrendlineVisible="False"/>
</syncfusion:ScatterSeries.Trendlines>
</syncfusion:ScatterSeries>

```

**C#**

```
ScatterSeries scatterSeries = new ScatterSeries()
{
    ItemsSource = new ViewModel().List,
    XBindingPath = "Year",
    YBindingPath = "India",
    ScatterHeight = 15,
    ScatterWidth = 15,
    Interior = new SolidColorBrush(Color.FromRgb(0x4A, 0x4A, 0x4A)),
    Label = "Series"
};
Trendline trendline = new Trendline()
{
    Label = "Trend",
    IsTrendlineVisible = true
};
scatterSeries.Trendlines.Add(trendline);
chart.Series.Add(scatterSeries);
```



**Note:** Here we have enabled the [CheckBoxVisibility](#) for the Legend. The CheckBox state indicates that trendline is not visible. You can enable trendline dynamically using this checkbox.

### Types of Trendlines

SfChart supports the following type of Trendlines.

- [Linear](#)
- [Exponential](#)
- [Power](#)
- [Logarithmic](#)
- [Polynomial](#)

#### Linear

This shows when something is increasing or decreasing at a steady rate. This is the default trend line to be drawn for the chart.

The linear trend line will be calculated using the below formula:

$yValue = Intercept + Slope * xValue$  ( where  $xValue$  is underlying x value).

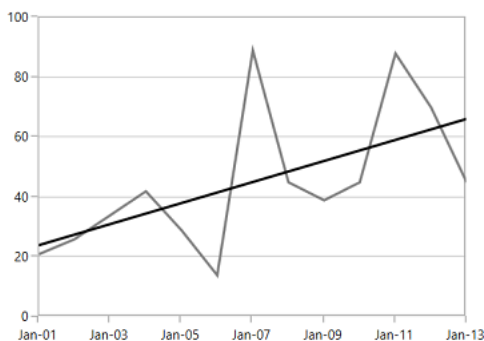
The following is the code example of this trend line.

### XML

```
<syncfusion:FastLineSeries XBindingPath="Date"
YBindingPath="Value"
Interior="#7F7F7F"
ItemsSource="{Binding StockPriceDetails}">
<syncfusion:FastLineSeries.Trendlines>
<syncfusion:Trendline Stroke="Black" Type="Linear"/>
</syncfusion:FastLineSeries.Trendlines>
</syncfusion:FastLineSeries>
```

### C#

```
FastLineSeries fastSeries = new FastLineSeries()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    YBindingPath = "Value",
    Interior = new SolidColorBrush(Color.FromRgb(0x7f, 0x7f, 0x7f)),
    Label = "Stock Price"
};
Trendline trendline = new Trendline()
{
    Label = "Trend",
    Stroke = new SolidColorBrush(Colors.Black),
    Type = TrendlineType.Linear
};
fastSeries.Trendlines.Add(trendline);
chart.Series.Add(fastS
```



### Exponential

This trend line is used when there is a constant increasing or decreasing in values. This is a curved line.

The linear trend line will be calculated using the below formula:

$(Intercept * Math.Exp(Slope * xValue))$  (where  $xValue$  is underlying x value).

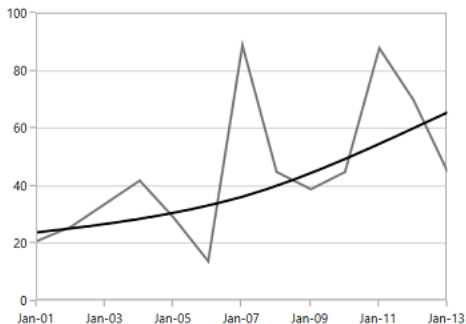
The following code example defines the exponential trendline.

**XML**

```
<syncfusion:FastLineSeries XBindingPath="Date"
YBindingPath="Value"
Interior="#7F7F7F"
ItemsSource="{Binding StockPriceDetails}">
<syncfusion:FastLineSeries.Trendlines>
<syncfusion:Trendline Stroke="Black" Type="Exponential"/>
</syncfusion:FastLineSeries.Trendlines>
</syncfusion:FastLineSeries>
```

**C#**

```
FastLineSeries fastSeries = new FastLineSeries()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    YBindingPath = "Value",
    Interior = new SolidColorBrush(Color.FromRgb(0x7f, 0x7f, 0x7f)),
    Label = "Stock Price"
};
Trendline trendline = new Trendline()
{
    Label = "Trend",
    Stroke = new SolidColorBrush(Colors.Black),
    Type = TrendlineType.Exponential
};
fastSeries.Trendlines.Add(trendline);
chart.Series.Add(fastS
```



**Note:** This is not recommended for the data values having zero and negative value.

*Power*

This trend line is used for comparing multiple sets of data that increase at specific rate.

This will be calculated using the following formula:

$(\text{Intercept} * \text{Math.Pow}(\text{xValue}, \text{Slope}))$  (Where xValue is underlying x value).

The following code example explains how to define the power trendline.

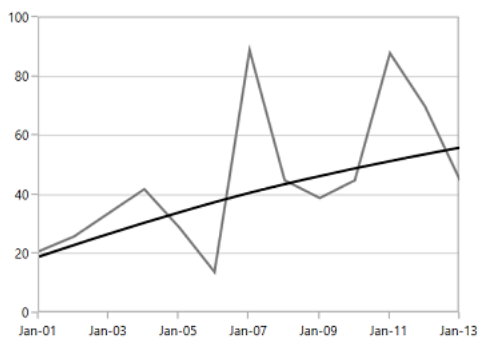
**XML**



```
<syncfusion:FastLineSeries XBindingPath="Date"
YBindingPath="Value"
Interior="#7F7F7F"
ItemsSource="{Binding StockPriceDetails}">
<syncfusion:FastLineSeries.Trendlines>
<syncfusion:Trendline Stroke="Black" Type="Power"/>
</syncfusion:FastLineSeries.Trendlines>
</syncfusion:FastLineSeries>
```

**C#**

```
FastLineSeries fastSeries = new FastLineSeries()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    YBindingPath = "Value",
    Interior = new SolidColorBrush(Color.FromRgb(0x7f, 0x7f, 0x7f)),
    Label = "Stock Price"
};
Trendline trendline = new Trendline()
{
    Label = "Trend",
    Stroke = new SolidColorBrush(Colors.Black),
    Type = TrendlineType.Power
};
fastSeries.Trendlines.Add(trendline);
chart.Series.Add(fastS
```

*Logarithmic*

This is used when there is a quick change in the data, either increasing or decreasing and then levels out.

The will be calculated using the below formula:

$\text{Intercept} + \text{Slope} * \text{Math.Log}(x\text{Value})$  (where xValue is underlying x value).

The following code example illustrates the use of logarithmic trend line.

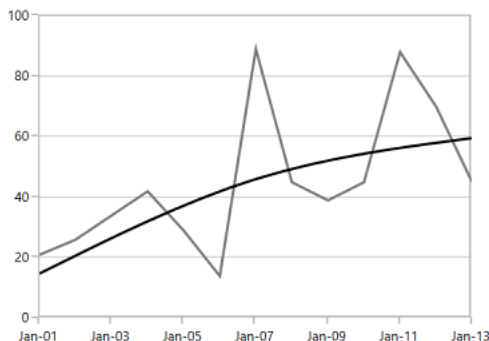
**XML**

```
<syncfusion:FastLineSeries XBindingPath="Date"
YBindingPath="Value"
Interior="#7F7F7F"
```

```
ItemsSource="{Binding StockPriceDetails}">
<syncfusion:FastLineSeries.Trendlines>
<syncfusion:Trendline Stroke="Black" Type="Logarithmic"/>
</syncfusion:FastLineSeries.Trendlines>
</syncfusion:FastLineSeries>
```

## C#

```
FastLineSeries fastSeries = new FastLineSeries()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    YBindingPath = "Value",
    Interior = new SolidColorBrush(Color.FromRgb(0x7f, 0x7f, 0x7f)),
    Label = "Stock Price"
};
Trendline trendline = new Trendline()
{
    Label = "Trend",
    Stroke = new SolidColorBrush(Colors.Black),
    Type = TrendlineType.Logarithmic
};
fastSeries.Trendlines.Add(trendline);
chart.Series.Add(fastS
```



## Polynomial

The polynomial trendline is a curved line that is used when there are more data fluctuations.

The polynomial trendline is calculated using the below formula:

$\text{PolynomialSlopes.Select} ( \text{value}, \text{index} ) \Rightarrow \text{value} * \text{Math.Pow} ( \text{xValue}, (\text{double}) \text{index} ) . \text{Sum} ()$  (where xValue is underlying x value)

To define the polynomial trendline, you can use the following code example.

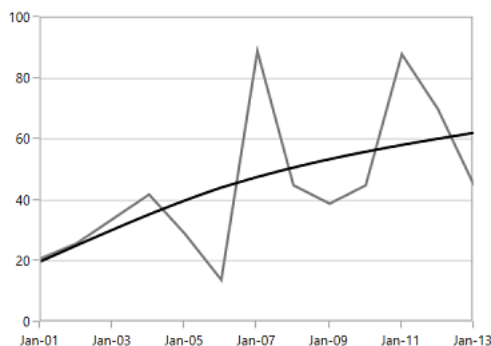
## XML

```
<syncfusion:FastLineSeries XBindingPath="Date"
YBindingPath="Value"
Interior="#7F7F7F"
ItemsSource="{Binding StockPriceDetails}">
```

```
<syncfusion:FastLineSeries.Trendlines>
<syncfusion:Trendline Stroke="Black" Type="Polynomial"/>
</syncfusion:FastLineSeries.Trendlines>
</syncfusion:FastLineSeries>
```

**C#**

```
FastLineSeries fastSeries = new FastLineSeries()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    YBindingPath = "Value",
    Interior = new SolidColorBrush(Color.FromRgb(0x7f, 0x7f, 0x7f)),
    Label = "Stock Price"
};
Trendline trendline = new Trendline()
{
    Label = "Trend",
    Stroke = new SolidColorBrush(Colors.Black),
    Type = TrendlineType.Polynomial
};
fastSeries.Trendlines.Add(trendline);
chart.Series.Add(fastS
```

**Polynomial Order**

You can set the Polynomial order for this trendline. Polynomial order calculates order based on the equation and this value should fall between 2 and 6.

**XML**

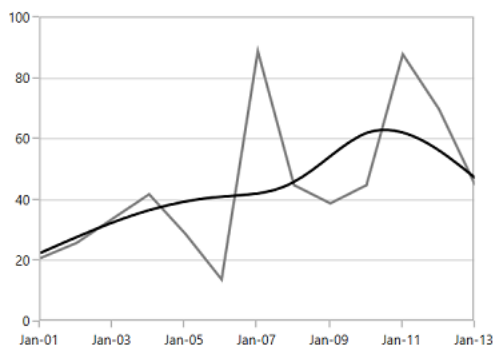
```
<syncfusion:FastLineSeries XBindingPath="Date"
YBindingPath="Value"
Interior="#7F7F7F"
ItemsSource="{Binding StockPriceDetails}">
<syncfusion:FastLineSeries.Trendlines>
<syncfusion:Trendline Stroke="Black" Type="Polynomial"
PolynomialOrder="5"/>
</syncfusion:FastLineSeries.Trendlines>
</syncfusion:FastLineSeries>
```

**C#**

```

FastLineSeries fastSeries = new FastLineSeries()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    YBindingPath = "Value",
    Interior = new SolidColorBrush(Color.FromRgb(0x7f, 0x7f, 0x7f)),
    Label = "Stock Price"
};
Trendline trendline = new Trendline()
{
    Label = "Trend",
    Stroke = new SolidColorBrush(Colors.Black),
    Type = TrendlineType.Polynomial,
    PolynomialOrder = 5
};
fastSeries.Trendlines.Add(trendline);
chart.Series.Add(fastSeries);

```



## Forecasting

Chart supports forecasting for the trendline, which is used to display trends about the future and the past.

The following two types of forecasting available in SfChart:

- Forward Forecasting
- Backward Forecasting

### Forward Forecast

For determining the future trends (in forward direction). The following code example explains the how to set the value for [ForwardForecast](#).

### XML

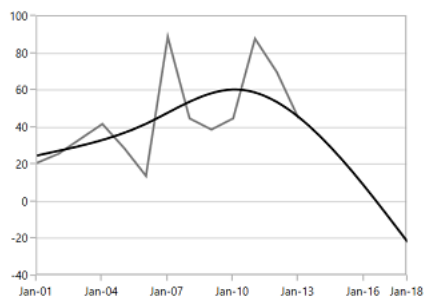
```

<syncfusion:FastLineSeries XBindingPath="Date" YBindingPath="Value"
    Label="Stock Price" Interior="#7F7F7F"
    ItemsSource="{Binding StockPriceDetails}">
    <syncfusion:FastLineSeries.Trendlines>
    <syncfusion:Trendline Stroke="Black"
        Type="Polynomial" PolynomialOrder="3" ForwardForecast="5" />
    </syncfusion:FastLineSeries.Trendlines>
</syncfusion:FastLineSeries>

```

**C#**

```
FastLineSeries fastSeries = new FastLineSeries()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    YBindingPath = "Value",
    Interior = new SolidColorBrush(Color.FromRgb(0x7f, 0x7f, 0x7f)),
    Label = "Stock Price"
};
Trendline trendline = new Trendline()
{
    Label = "Trend",
    Stroke = new SolidColorBrush(Colors.Black),
    Type = TrendlineType.Polynomial,
    PolynomialOrder = 3,
    ForwardForecast = 5
};
fastSeries.Trendlines.Add(trendline);
chart.Series.Add(fastSeries);
```

*Backward Forecast*

For determining the past trends (in backward direction). The following code example explains the how to set the value for [BackwardForecast](#).

**XML**

```
<syncfusion:FastLineSeries XBindingPath="Date" YBindingPath="Value"
Label="Stock Price" Interior="#7F7F7F"
ItemsSource="{Binding StockPriceDetails}">
<syncfusion:FastLineSeries.Trendlines>
<syncfusion:Trendline Stroke="Black"
Type="Polynomial" PolynomialOrder="3" BackwardForecast="5" />
</syncfusion:FastLineSeries.Trendlines>
</syncfusion:FastLineSeries>
```

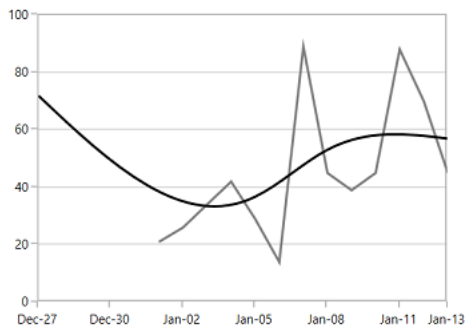
**C#**

```
FastLineSeries fastSeries = new FastLineSeries()
{
    ItemsSource = new ViewModel().StockPriceDetails,
```

```

XBindingPath = "Date",
YBindingPath = "Value",
Interior = new SolidColorBrush(Color.FromRgb(0x7f, 0x7f, 0x7f)),
Label = "Stock Price"
};
Trendline trendline = new Trendline()
{
    Label = "Trend",
    Stroke = new SolidColorBrush(Colors.Black),
    Type = TrendlineType.Polynomial,
    PolynomialOrder = 3,
    BackwardForecast = 5
};
fastSeries.Trendlines.Add(trendline);
chart.Series.Add(fastSeries);

```



### Customization

You can customize the trendline [Stroke](#), [StrokeThickness](#) and [StrokeDashArray](#) as in below code example.

### XML

```

<syncfusion:FastLineSeries XBindingPath="Date" YBindingPath="Value"
    Label="Stock Price" Interior="#7F7F7F"
    ItemsSource="{Binding StockPriceDetails}">
    <syncfusion:FastLineSeries.Trendlines>
    <syncfusion:Trendline Stroke="Black" Type="Linear"
        StrokeDashArray="4,4" StrokeThickness="2" />
    </syncfusion:FastLineSeries.Trendlines>
</syncfusion:FastLineSeries>

```

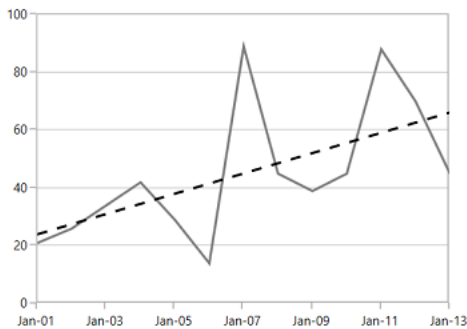
### C#

```

FastLineSeries fastSeries = new FastLineSeries()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    YBindingPath = "Value",
    Interior = new SolidColorBrush(Color.FromRgb(0x7f, 0x7f, 0x7f)),
    Label = "Stock Price"
};

```

```
Trendline trendline = new Trendline()
{
    Label = "Trend",
    Stroke = new SolidColorBrush(Colors.Black),
    Type = TrendlineType.Linear,
    StrokeThickness = 2,
    StrokeDashArray = new DoubleCollection() { 4, 4 }
};
fastSeries.Trendlines.Add(trendline);
chart.Series.Add(fastSeries);
```



**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configure with built-in support for creating stunning visual effects.

See also

[Chart with average line](#)

### Interactive Features in WPF Charts (SfChart)

SfChart provides interactive features such as tracking data points and resizing the scrollbar.

The following interactive features are supported in SfChart:

- Visual Data Editing
- Resizing Scrollbar
- CrossHair
- ToolTip
- TrackBall
- Zoom and Pan
- Selection

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configure with built-in support for creating stunning visual effects.

### Appearance in WPF Charts (SfChart)

SfChart supports various customizing and styling options that allow you to enrich the application.

## Palettes

SfChart provides options to apply different kinds of themes or palettes to your chart. You can define [Palette](#) either for the entire chart or for an individual series.

We have some predefined palette such as

- Metro
- AutumnBrights
- FloraHues
- Pineapple
- TomotoSpectrum
- RedChrome
- PurpleChrome
- BlueChrome
- GreenChrome
- Elite
- LightCandy
- SandyBeach

**Note:** Elite, SandyBeach and LightCandy palettes are not supported in the bitmap series types.

### Applying Palette to Series

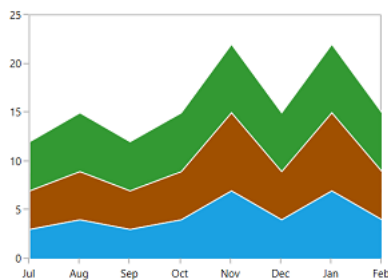
Each palette applies a set of predefined brushes to the series in a predefined order. The following code example shows you how to set the Metro Palette for the chart series.

#### XML

```
<chart:SfChart Height="250" Width="350" Palette="Metro" >
```

#### C#

```
chart.Palette = ChartColorPalette.Metro;
```



The following code example defined Palette as [BlueChrome](#).

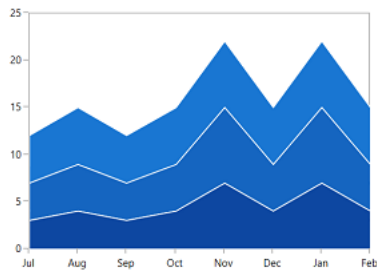
#### XML

```
<chart:SfChart Height="250" Width="350" Palette="BlueChrome">
```

#### C#



```
chart.Palette = ChartColorPalette.BlueChrome;
```



### Applying Palette to Segment

Each palette applies a set of predefined brushes to the segment in a predefined order. The following code example shows you how to set the Metro Palette for the chart series.

### XML

```
<chart:DoughnutSeries YBindingPath="Percentage" Palette="Metro"
XBindingPath="Category" ItemsSource="{Binding Tax}" />
```

### C#

```
DoughnutSeries series = new DoughnutSeries()
{
    ItemsSource = new ViewModel().Tax,
    XBindingPath = "Category",
    YBindingPath = "Percentage",
    Palette = ChartColorPalette.Metro
};
chart.Series.Add(series);
```



The following code example defined Palette as **AutumnBrights**.

### XML

```
<chart:DoughnutSeries YBindingPath="Percentage" Palette="AutumnBrights"
XBindingPath="Category" ItemsSource="{Binding Tax}" />
```

### C#

```
DoughnutSeries series = new DoughnutSeries()
{
    ItemsSource = new ViewModel().Tax,
    XBindingPath = "Category",
    YBindingPath = "Percentage",
    Palette = ChartColorPalette.AutumnBrights
};
chart.Series.Add(series);
```



**Note:** Metro palette is the default palette for both Series and Segment.

#### Custom Palette

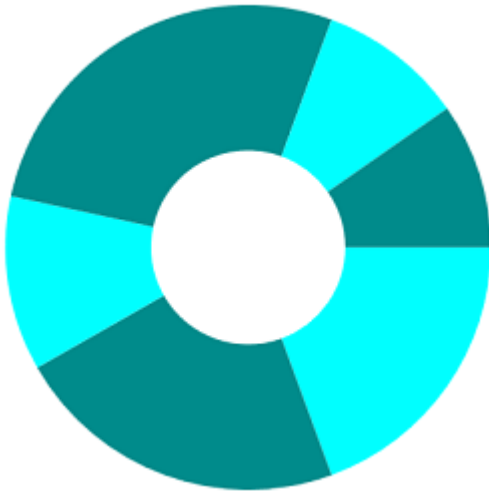
SfChart provides option which enables you to define your own color brushes with your preferred order for the Palette, using [ColorModel](#) as shown in the following code example.

#### XML

```
<chart:DoughnutSeries YBindingPath="Percentage" Palette="Custom"
XBindingPath="Category" ItemsSource="{Binding Tax}" >
  <chart:DoughnutSeries.ColorModel>
    <chart:ChartColorModel>
      <chart:ChartColorModel.CustomBrushes>
        <SolidColorBrush Color="Cyan"/>
        <SolidColorBrush Color="DarkCyan"/>
      </chart:ChartColorModel.CustomBrushes>
    </chart:ChartColorModel>
  </chart:DoughnutSeries.ColorModel>
</chart:DoughnutSeries>
```

#### C#

```
ChartColorModel colorModel = new ChartColorModel();
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.Cyan));
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.DarkCyan));
DoughnutSeries series = new DoughnutSeries()
{
    ItemsSource = new ViewModel().Tax,
    XBindingPath = "Category",
    YBindingPath = "Percentage",
    Palette = ChartColorPalette.Custom,
    ColorModel = colorModel
};
chart.Series.Add(series);
```



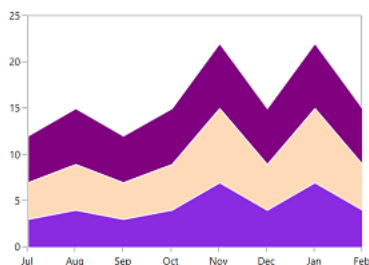
You can define the custom palette for series as in the below code example:

#### XML

```
<chart:SfChart Height="250" Width="350" Palette="Custom">
  <chart:SfChart.ColorModel>
    <chart:ChartColorModel>
      <chart:ChartColorModel.CustomBrushes>
        <SolidColorBrush Color="BlueViolet"/>
        <SolidColorBrush Color="PeachPuff"/>
        <SolidColorBrush Color="Purple"/>
      </chart:ChartColorModel.CustomBrushes>
    </chart:ChartColorModel>
  </chart:SfChart.ColorModel>
</chart:SfChart>
```

#### C#

```
chart.Palette = ChartColorPalette.Custom;
ChartColorModel colorModel = new ChartColorModel();
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.BlueViolet));
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.PeachPuff));
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.Purple));
chart.ColorModel = colorModel;
```



## Gradient Colors

Gradient colors for the chart series can be set by using the [Interior](#) or [ColorModel](#) property of the chart series with the help of [LinearGradientBrush](#) or [RadialGradientBrush](#).

The following code sample and screenshot illustrates how to apply the custom gradient colors for chart series using the [ColorModel](#) property.

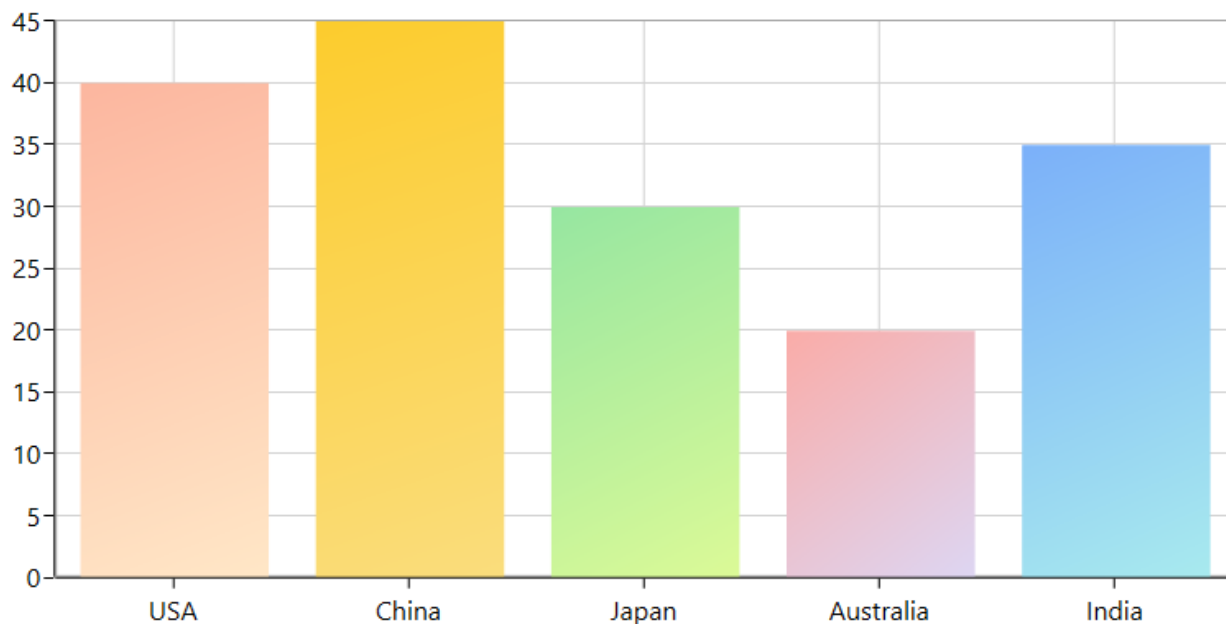
### XML

```
<chart:ColumnSeries XBindingPath="XValue"
YBindingPath="YValue"
ItemsSource="{Binding Data}"
Palette="Custom">
<chart:ColumnSeries.ColorModel>
<chart:ChartColorModel>
<chart:ChartColorModel.CustomBrushes>
<LinearGradientBrush>
<GradientStop Offset="1" Color="#FFE7C7" />
<GradientStop Offset="0" Color="#FCB69F" />
</LinearGradientBrush>
<LinearGradientBrush>
<GradientStop Offset="1" Color="#fadd7d" />
<GradientStop Offset="0" Color="#fccc2d" />
</LinearGradientBrush>
<LinearGradientBrush>
<GradientStop Offset="1" Color="#DCFA97" />
<GradientStop Offset="0" Color="#96E6A1" />
</LinearGradientBrush>
<LinearGradientBrush>
<GradientStop Offset="1" Color="#DDD6F3" />
<GradientStop Offset="0" Color="#FAACA8" />
</LinearGradientBrush>
<LinearGradientBrush>
<GradientStop Offset="1" Color="#A8EAE6" />
<GradientStop Offset="0" Color="#7BB0F9" />
</LinearGradientBrush>
</chart:ChartColorModel.CustomBrushes>
</chart:ChartColorModel>
</chart:ColumnSeries.ColorModel>
</chart:ColumnSeries>
```

### C#

```
SfChart chart = new SfChart();
...
ChartColorModel colorModel = new ChartColorModel();
LinearGradientBrush gradientColor1 = new LinearGradientBrush();
GradientStop stop1 = new GradientStop() { Offset = 1, Color =
Color.FromRgb(255, 231, 199) };
GradientStop stop2 = new GradientStop() { Offset = 0, Color =
Color.FromRgb(252, 182, 159) };
gradientColor1.GradientStops.Add(stop1);
gradientColor1.GradientStops.Add(stop2);
LinearGradientBrush gradientColor2 = new LinearGradientBrush();
stop1 = new GradientStop() { Offset = 1, Color = Color.FromRgb(250, 221,
125) };
```

```
stop2 = new GradientStop() { Offset = 0, Color = Color.FromRgb(252, 204, 45) };
gradientColor2.GradientStops.Add(stop1);
gradientColor2.GradientStops.Add(stop2);
...
colorModel.CustomBrushes.Add(gradientColor1);
colorModel.CustomBrushes.Add(gradientColor2);
...
ColumnSeries series = new ColumnSeries()
{
    ItemsSource = new ViewModel().Data,
    XBindingPath = "Element",
    YBindingPath = "YValue",
    Palette = ChartColorPalette.Custom,
    ColorModel = colorModel
};
chart.Series.Add(series);
```



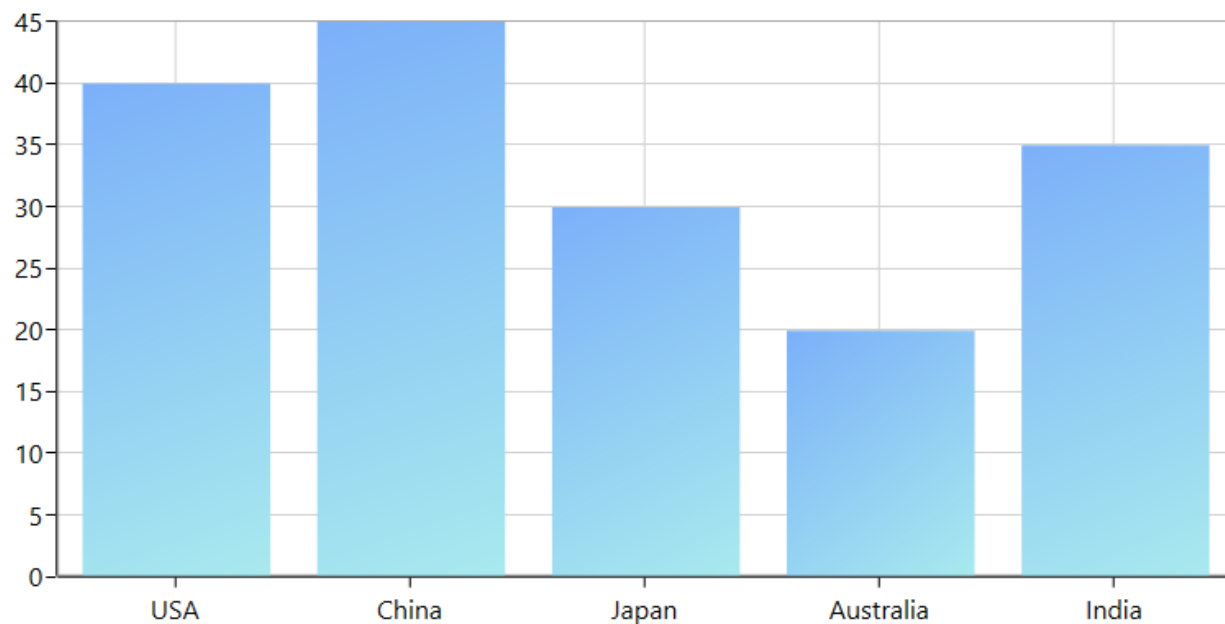
The following code sample and screenshot illustrates how to apply the gradient color using the [Interior](#) property of series with `LinearGradientBrush`.

#### XML

```
<chart:ColumnSeries XBindingPath="XValue"
YBindingPath="YValue"
ItemsSource="{Binding Data}">
  <chart:ColumnSeries.Interior>
    <LinearGradientBrush>
      <GradientStop Offset="1" Color="#A8EAE" />
      <GradientStop Offset="0" Color="#7BB0F9" />
    </LinearGradientBrush>
  </chart:ColumnSeries.Interior>
</chart:ColumnSeries>
```

**C#**

```
SfChart chart = new SfChart();  
...  
LinearGradientBrush gradientColor = new LinearGradientBrush();  
GradientStop stop1 = new GradientStop() { Offset = 1, Color =  
Color.FromRgb(168, 234, 238) };  
GradientStop stop2 = new GradientStop() { Offset = 0, Color =  
Color.FromRgb(123, 176, 249) };  
gradientColor.GradientStops.Add(stop1);  
gradientColor.GradientStops.Add(stop2);  
...  
ColumnSeries series = new ColumnSeries()  
{  
    ItemsSource = new ViewModel().Data,  
    XBindingPath = "Element",  
    YBindingPath = "YValue",  
    Interior = gradientColor,  
};  
chart.Series.Add(series);
```

**SegmentColorPath**

The color for the chart segments can be bound from its items source collection by using the [SegmentColorPath](#) property of series. The following code illustrates how to bind the color to the series with [SegmentColorPath](#) property.

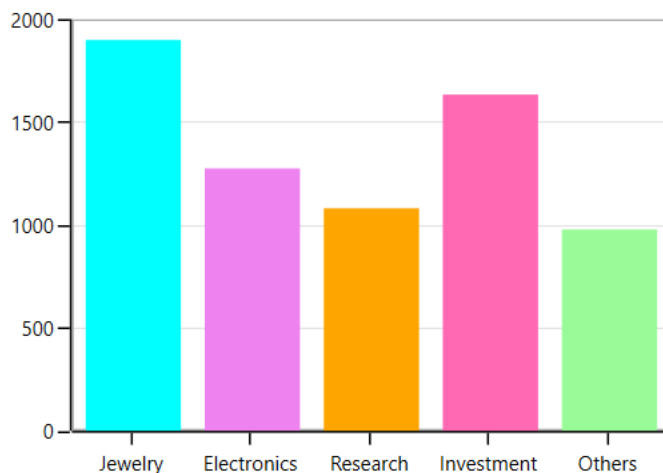
**XML**

```
<chart:ColumnSeries SegmentColorPath="ColorPath">  
</chart:ColumnSeries>
```

**C#**

```
ColumnSeries series = new ColumnSeries()
```

```
{
  SegmentColorPath = "SegmentColor"
};
```



**Note:** SegmentColorPath property is not applicable for Area, SplineArea, StepArea, RangeArea, FastLine, Candle, HiLoOpenClose, and CircularSeries (when the Polar and Radar DrawType is Area).

### Customize Legends

SfChart provides many options to customize the chart legends. Basically it is an ItemsControl. So, you can customize the ItemTemplate, ItemsPanel, etc.

The following code example demonstrates applying the palette color to the legend icon interior.

### XML

```
<chart:SfChart.Legend>
  <chart:ChartLegend DockPosition="Left" >
    <chart:ChartLegend.ItemTemplate>
      <DataTemplate>
        <StackPanel Orientation="Horizontal">
          <Grid Margin="20,0,0,0">
            <Grid.ColumnDefinitions>
              <ColumnDefinition/>
              <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Ellipse Width="10" Height="10"
              Fill="{Binding Interior}">
              <Ellipse.Effect>
                <DropShadowEffect Direction="315"
                  BlurRadius="0.8"
                  ShadowDepth="3"
                  Color="Black"/>
              </Ellipse.Effect>
            </Ellipse>
            <TextBlock Margin="5,0,0,0"
              FontSize="10"
              Grid.Column="1"
              Foreground="Black"
              Text="{Binding Label}"></TextBlock>
          </Grid>
```

```

</StackPanel>
</DataTemplate>
</chart:ChartLegend.ItemTemplate>
</chart:ChartLegend>
</chart:SfChart.Legend>

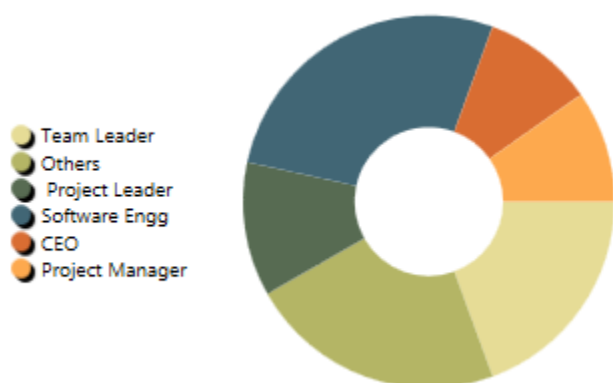
```

## C#

```

chart.Legend = new ChartLegend()
{
    DockPosition = ChartDock.Left,
    ItemTemplate = chart.Resources["itemTemplate"] as DataTemplate
};

```



If you are having more number of items in the legend, you can override the ItemsPanel and add ScrollViewer. So that you can able to scroll the legend items. Please refer [this](#) kb for more details

## Customize Tooltip

SfChart provides the option to define your own template for Tooltip. The following code example demonstrates the custom tooltip using the [TooltipTemplate](#) property.

## XML

```

<chart:BarSeries ItemsSource="{Binding CategoricalDatas}"
chart:ChartTooltip.ShowDuration="5000"
XBindingPath="Category" YBindingPath="Value"
ShowTooltip="True" Palette="FloraHues" >
<chart:BarSeries.TooltipTemplate>
<DataTemplate >
<Border BorderBrush="Black" BorderThickness="1">
<Grid Height="40">
<Grid.RowDefinitions>
<RowDefinition Height="0.5*"/>
<RowDefinition Height="0.5*"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition/>
<ColumnDefinition/>
<ColumnDefinition/>
</Grid.ColumnDefinitions>
<Rectangle Fill="White" Grid.RowSpan="2"
Grid.ColumnSpan="3" ></Rectangle>

```



```

<Image Grid.RowSpan="2" Grid.Column="0"
HorizontalAlignment="Left" Margin="3"
Source="{Binding Item.ImagePath}" ></Image>
<TextBlock Margin="3,3,6,3" Text="{Binding Item.Category}"
FontSize="10" Grid.Column="1"
Grid.ColumnSpan="2"
HorizontalAlignment="Left"
VerticalAlignment="Center" TextAlignment="Center"
Foreground="Black" />
<TextBlock VerticalAlignment="Center" Margin="3,3,6,3"
Grid.Column="2" Grid.Row="1"
TextAlignment="Left" Text="{Binding Item.Value}"
HorizontalAlignment="Left"
Foreground="Black" FontSize="10"/>
<TextBlock VerticalAlignment="Center"
Grid.Column="1" Grid.Row="1"
TextAlignment="Left" Text="Value:"
HorizontalAlignment="Left"
Foreground="Black" FontSize="10"/>
</Grid>
</Border>
</DataTemplate>
</chart:BarSeries.TooltipTemplate>
</chart:BarSeries>

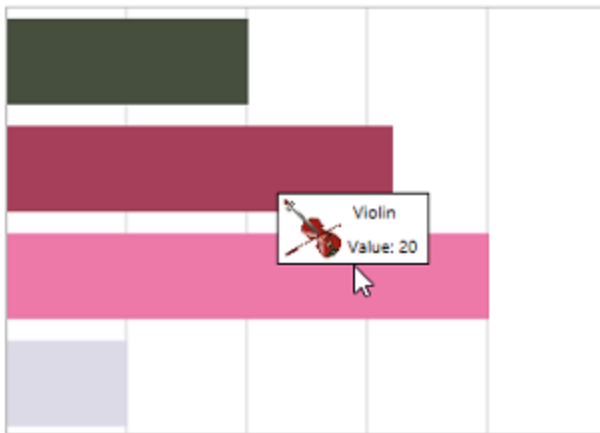
```

## C#

```

BarSeries series = new BarSeries()
{
    ItemsSource = new ViewModel().CategoricalDatas,
    XBindingPath = "Category",
    YBindingPath = "Value",
    Palette = ChartColorPalette.FlorasHues,
    ShowTooltip = true,
    TooltipTemplate = chart.Resources["tooltipTemplate"] as DataTemplate
};
ChartTooltip.SetShowDuration(series, 5000);
chart.Series.Add(series);

```



---

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

---

See also

[How to apply gradient colors for each series in WPF Chart](#)

[How to change colors of specific data points in the chart](#)

[How to set color for the series ColorModel property](#)

[How to add custom color model to series](#)

[How to define the fill color for each datapoint from ItemsSource](#)

### Animation in WPF Charts (SfChart)

SfChart allows you to animate the chart series on loading, and whenever the ItemsSource changes. Animation in chart can be enabled by setting the EnableAnimation property as True and defining the corresponding animation speed with AnimationDuration property.

The following types of series support Animation.

- Line
- Column
- Bar
- Area
- Scatter
- Bubble
- Spline
- Spline area
- Stacking column
- Stacking bar
- Stacking area
- Pie

The following APIs are used to customize the Animation.

- [EnableAnimation](#)-Represents a boolean value to enable the animation for series.
- [AnimationDuration](#)-Represents a TimeSpan value which sets animation speed for the chart.

The following example shows the Animation feature for chart series.

#### XML

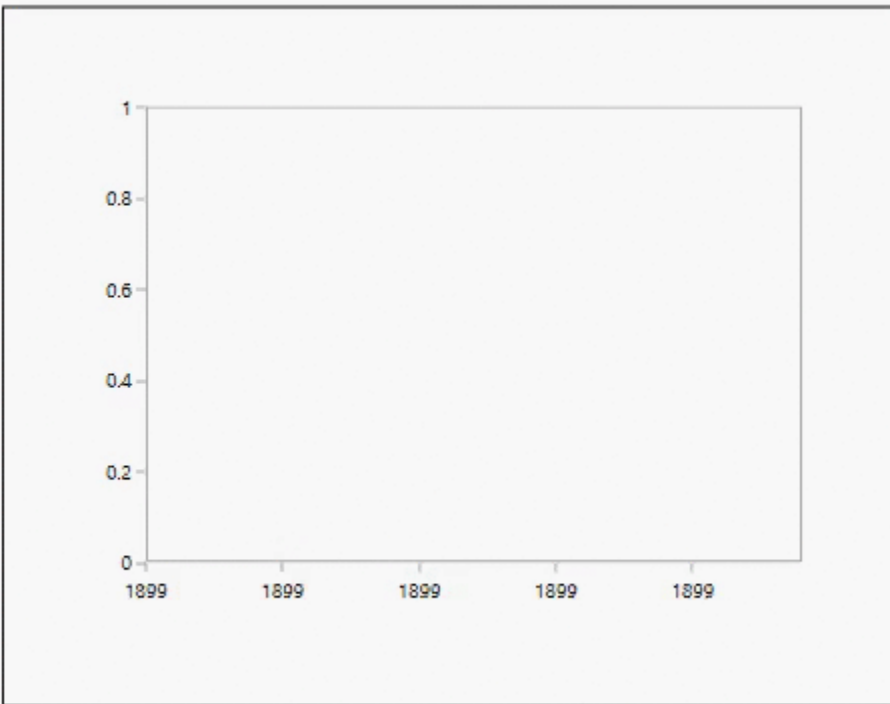
```
<syncfusion:SfChart>
  <syncfusion:ColumnSeries EnableAnimation="True" AnimationDuration="00:00:03"
  XBindingPath="Category" YBindingPath="Count" ItemsSource="{Binding}" />
</syncfusion:SfChart>
```

#### C#

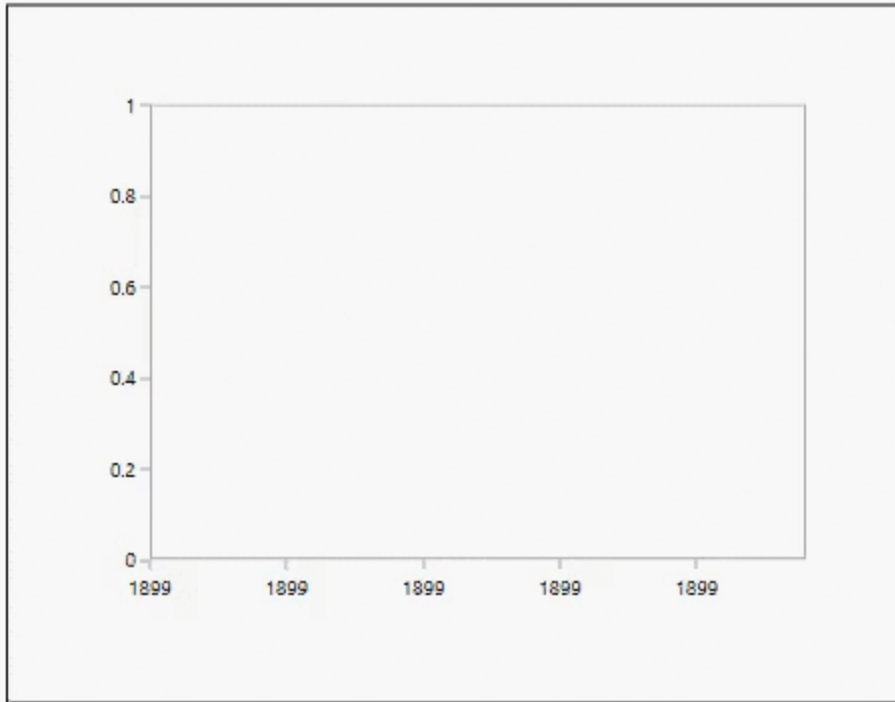
```
ColumnSeries columnSeries = new ColumnSeries()
{
```

```
ItemsSource = new ViewModel().Data,  
XBindingPath = "Category",  
YBindingPath = "Count",  
EnableAnimation = true,  
AnimationDuration = new TimeSpan(00, 00, 03)  
};  
chart.Series.Add(columnSeries);
```

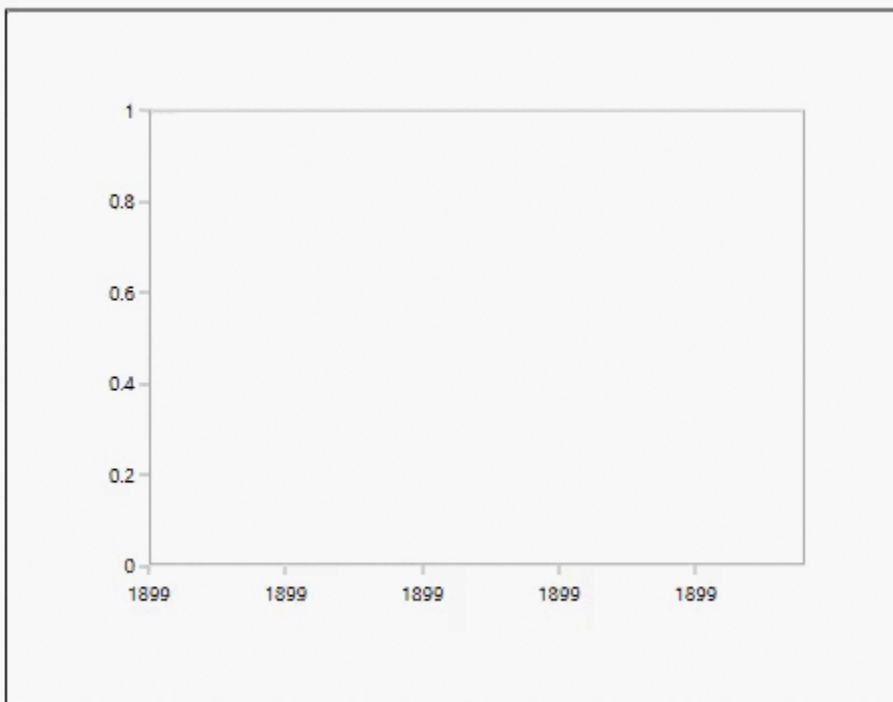
### Column Series



### SplineArea Series



### Scatter Series



---

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configure with built-in support for creating stunning visual effects.

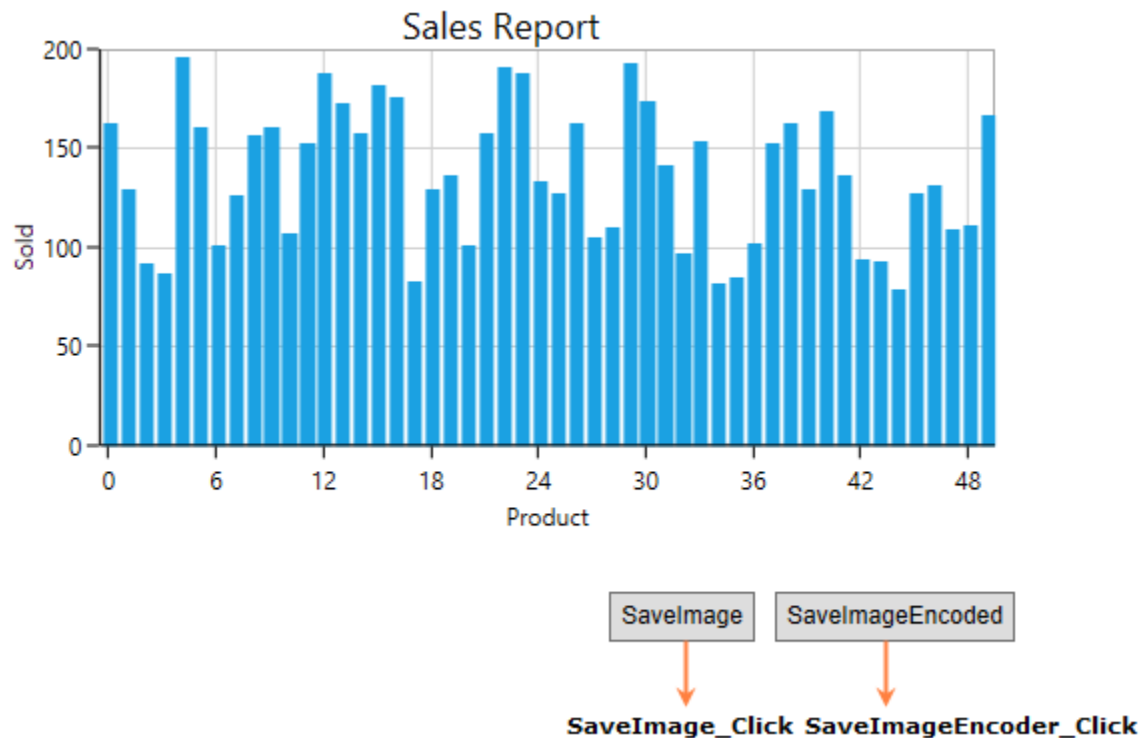
---

## Exporting in WPF Charts (SfChart)

Chart can be exported into image format. The following are the supported image formats:

- JPEG or JPG
- JPG-XR
- GIF
- PNG
- BMP
- TIFF

The following screenshot illustrates the chart, which has to be exported.



### Methods

Chart contains the following overloading methods for saving a chart as an image.

#### *Save(string filename)*

This method will export chart to the specified location with the given name. By default, i.e., if you didn't mention any specific location. It will be exported to "../bin/debug" location.

The following code examples illustrates the usage of this method:

#### **C#**

```
private void SaveImage_Click(object sender, RoutedEventArgs e)
{
    this.SampleChart.Save("ExportedChart"); //Save in Debug location
    this.SampleChart.Save("D:\\Pictures\\Test\\ExportedChart"); //Save in
    'D:\\Picture\\Test' location.
}
```

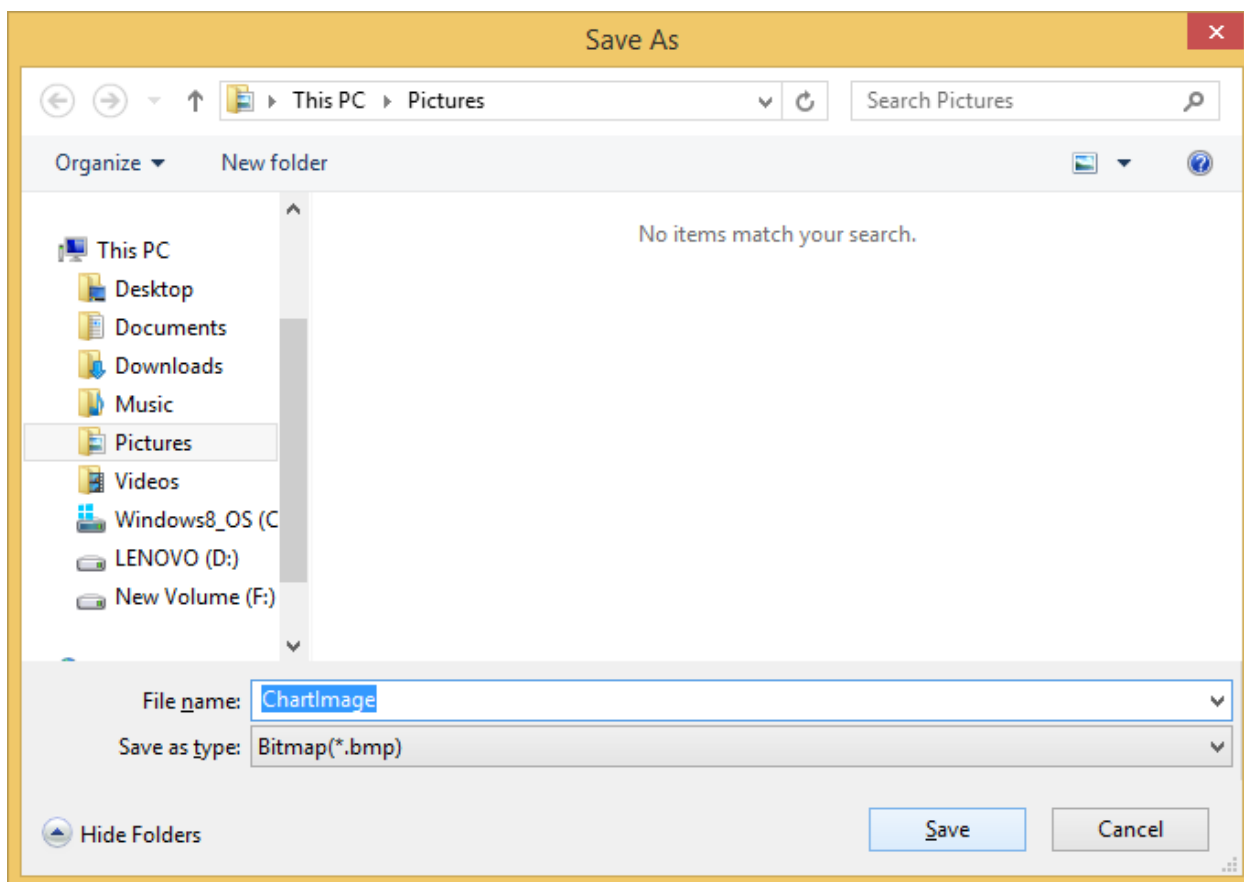
**Tips:** We can change the image formats in above code by changing its extension as .jpg, .tiff, etc.

*Save(Stream stream, BitmapEncoder imgEncoderID)*

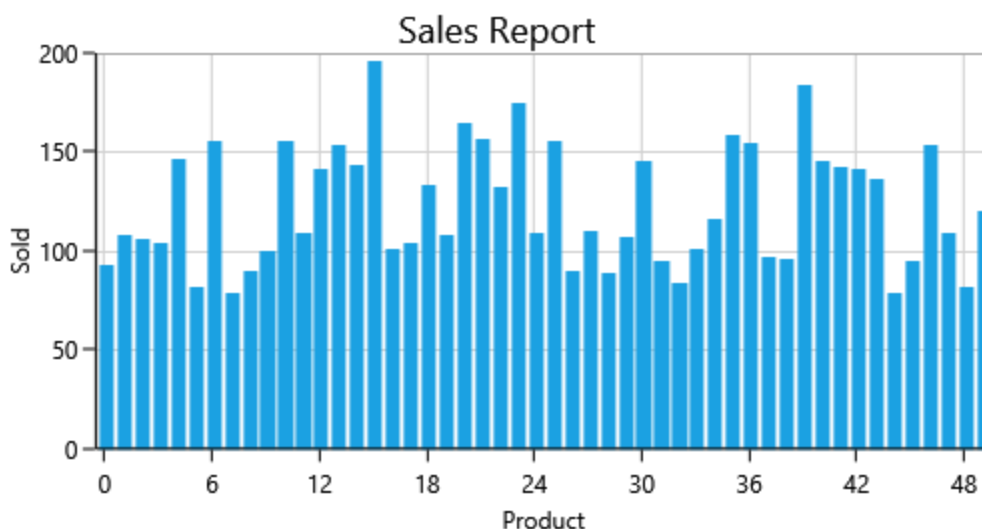
This helps to export the chart to any stream as in below code example.

**C#**

```
private void SaveImageEncoder_Click(object sender, RoutedEventArgs e)
{
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.Filter = "Bitmap (*.bmp) | *.bmp|JPEG (*.jpg,*.jpeg) | *.jpg;*.jpeg|Gif (*.gif) | *.gif|PNG (*.png) | *.png|TIFF (*.tif,*.tiff) | *.tif|All files (*.*) | *.*";
    if (sfd.ShowDialog() == true)
    {
        using (Stream fs = sfd.OpenFile())
        {
            SampleChart.Save(fs, new PngBitmapEncoder());
        }
    }
}
```



The image will be saved in the specified location in the SaveFileDialog.



*Export SfChart to image without rendering in UI*

You can export the chart to image without rendering in UI by setting the chart to **RootVisual** in **HwndSource** and passing **HwndSourceParameters** to the **HwndSource**. The following code snippet demonstrates this.

#### C#

```
static IntPtr ApplicationMessageFilter(IntPtr hwnd, int message, IntPtr
wParam, IntPtr lParam, ref bool handled)
{
    return IntPtr.Zero;
}

HwndSourceParameters sourceParameters = new HwndSourceParameters();
sourceParameters.HwndSourceHook = ApplicationMessageFilter;
HwndSource source = new HwndSource(sourceParameters);
source.RootVisual = chart;
//Save chart
chart.Save("Chart.png");
```

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

See also

[How to export chart as Image](#)

[How to print the chart](#)

[How to read image byte while rendering chart](#)

#### Printing in WPF Charts (SfChart)

SfChart supports printing that enables you to print the chart. The following method is used to print the chart.

`Print()`

This method will invoke a printing dialog window with set of printing options.

The following code example demonstrates the printing of chart in button click event:

#### XML

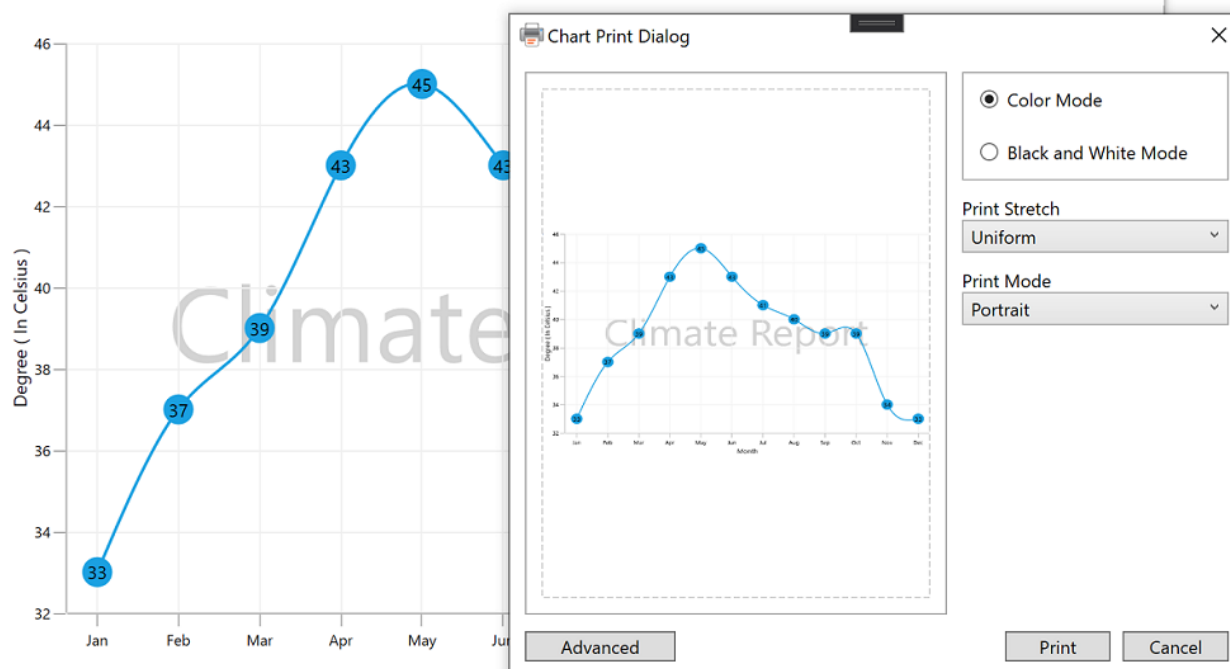
```
<chart:SfChart x:Name="ExportDemoChart" >
<chart:SfChart.Watermark>
<chart:Watermark Canvas.ZIndex="-1" HorizontalAlignment="Center"
VerticalAlignment="Center">
<chart:Watermark.Content>
<TextBlock Text="Climate Report" FontSize="60" Foreground="Gray"
Opacity="0.5"></TextBlock>
</chart:Watermark.Content>
</chart:Watermark>
</chart:SfChart.Watermark>
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis PlotOffset="20" Header="Month" />
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis Header="Degree ( In Celsius )"
RangePadding="Round"/>
</chart:SfChart.SecondaryAxis>
<!-- Add Series to the Chart-->
<chart:SplineSeries Label="Sports" ItemsSource="{Binding ClimateData}"
XBindingPath="Month" YBindingPath="Temperature">
<chart:SplineSeries.AdornmentsInfo>
<chart:ChartAdornmentInfo ShowMarker="True" Symbol="Ellipse"
ShowLabel="True"/>
</chart:SplineSeries.AdornmentsInfo>
</chart:SplineSeries>
</chart:SfChart>
```

#### C#

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    ExportDemoChart.Print();
}
```

The following image shows the printing dialog window upon invoking print method.





**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

### Watermark in WPF Charts (SfChart)

SfChart provides watermark support which is used to add text or images to the chart area. The major application of watermark is to define the copyright information of the user it belongs to.

This section is to help you understand how to use the [Watermark](#) in your chart.

#### Adding text watermark

You can add the text to chart background using the Content property of Watermark.

The following code example explains how to set your custom text as Watermark.

#### XML

```
<chart:SfChart.Watermark>
  <chart:Watermark VerticalAlignment="Center"
    HorizontalAlignment="Center" >
    <chart:Watermark.Content>
      <TextBlock Text="Metals"
        FontSize="70"
        Foreground="Black" >
      </TextBlock>
    </chart:Watermark.Content>
  </chart:Watermark>
</chart:SfChart.Watermark>
```

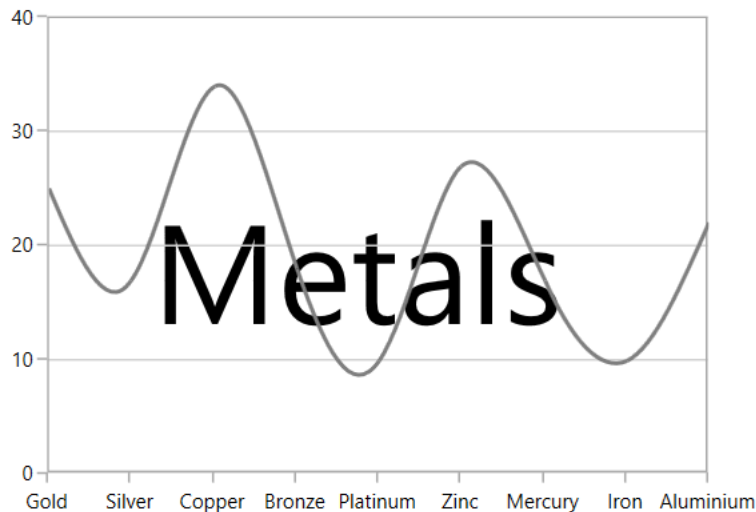
#### C#

```
chart.Watermark = new Watermark()
{
```

```

HorizontalAlignment = HorizontalAlignment.Center,
VerticalAlignment = VerticalAlignment.Center
};
chart.Watermark.Content = new TextBlock()
{
    Text = "Metals",
    FontSize = 70,
    Foreground = new SolidColorBrush(Colors.Black)
};

```



### Adding image watermark

You can also set images as Watermark as in below code snippet.

#### XML

```

<chart:SfChart.Watermark>
<chart:Watermark VerticalAlignment="Center"
HorizontalAlignment="Center" >
<chart:Watermark.Content>
<Image Source="demands.png" Height="175" Width="175"/>
</chart:Watermark.Content>
</chart:Watermark>
</chart:SfChart.Watermark>

```

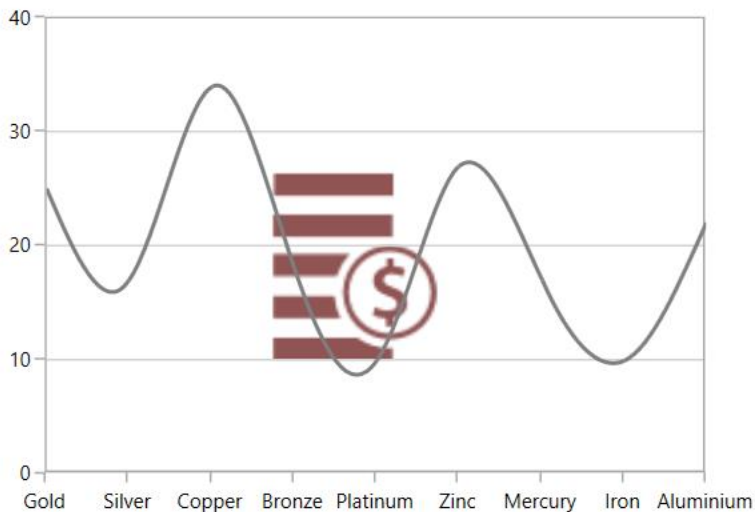
#### C#

```

chart.Watermark = new Watermark()
{
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center
};
chart.Watermark.Content = new Image()
{
    Height = 175,

```

```
Width = 175,  
Source = new BitmapImage(new Uri(@"demands.png",  
UriKind.RelativeOrAbsolute))  
};
```



**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

See also

[How to add watermark to chart](#)

### Performance in WPF Charts (SfChart)

- When your underlying data object implements `INotifyPropertyChanged`, you need to enable the `ListenPropertyChange` property of the series, to make the chart listen to the property changes of your data object. However enabling this property registers `PropertyChanged` event of every object in the data source. Due to this, chart's loading time is affected when there are a large number of points. By default, `ListenPropertyChange` is set to false in order to avoid the event registration unnecessarily.
- In case of a Line Series, when you have a large number of points to plot, you can make use of fast series types like `FastLineSeries` and `FastLineBitmapSeries`.
- In case of a Column Series, when you have large number of points to plot, you can make use of `FastColumnBitmapSeries`.
- You also have Fast Series types for financial charts like `HiLo` and `HiLoOpenClose` and they are named as `FastHiLoBitmapSeries` and `FastHiLoOpenCloseBitmapSeries` respectively.

**Note:** Stopped to provide support for the `DirectX` series. Instead of that, use `FastLineBitmapSeries` to render a large amount of data with good performance.

Deferred real-time updates

You can hold and resume the series updates in dynamic update scenarios using the below methods.

[SuspendSeriesNotification](#)

[ResumeSeriesNotification](#)

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

See also

[How to create a real time Chart using MVVM in WPF](#)

### Serialization in WPF Charts (SfChart)

SfChart provides the support for serializing and deserializing control. This section explains on how to serialize and deserialize SfChart.

Method Name	Description
Serialize()	Serialize the control and saves as the XML file in the parent folder.
Serialize(string filePath)	Serialize the control and saves as the XML file in the given file path.
Serialize(Stream stream)	Serialize the control and saves as the XML file in the given stream.
Deserialize()	Deserialize the XML file from the parent folder and returns the SfChart control object.
Deserialize(string filePath)	Deserialize the XML file from the given file path and returns the SfChart control object.
Deserialize(Stream stream)	Deserialize the XML file from the given stream and returns the SfChart control object.

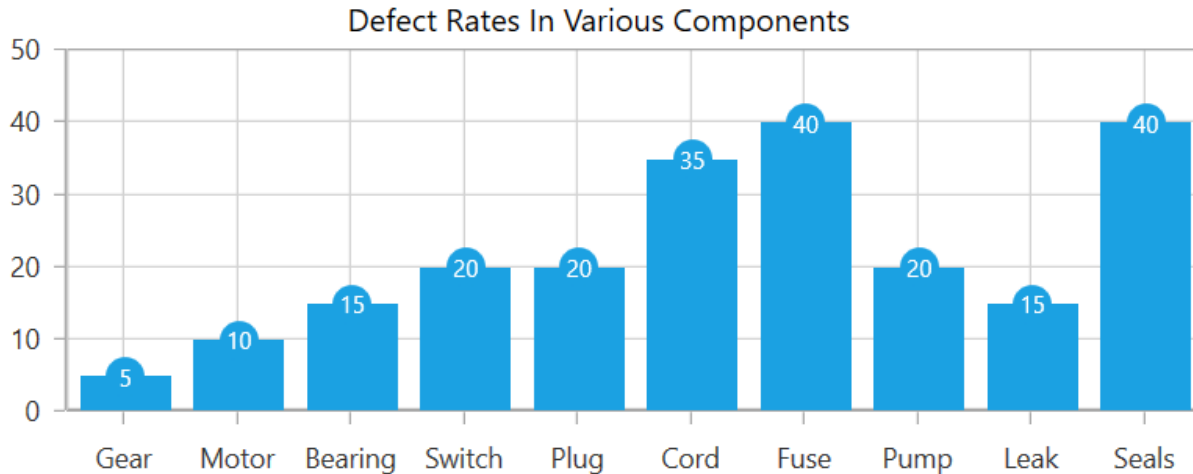
### XML

```
<Grid >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="350" />
<ColumnDefinition Width="350" />
</Grid.ColumnDefinitions>
<!--Serialize-->
<chart:SfChart Grid.Column="0" Margin="10" x:Name="chart">
<chart:SfChart.PrimaryAxis>
<chart:CategoryAxis />
</chart:SfChart.PrimaryAxis>
<chart:SfChart.SecondaryAxis>
<chart:NumericalAxis />
</chart:SfChart.SecondaryAxis>
<chart:ColumnSeries ItemsSource="{Binding CategoricalDatas}"
XBindingPath="Category"
YBindingPath="Value" EnableAnimation="True">
<chart:ColumnSeries.AdornmentsInfo>
```

```
<chart:ChartAdornmentInfo ShowLabel="True" ShowMarker="True"
Symbol="Ellipse" Foreground="White"/>
</chart:ColumnSeries.AdornmentsInfo>
</chart:ColumnSeries>
</chart:SfChart>
<!--Assign the DeSerialize Chart-->
<chart:SfChart Grid.Column="1" x:Name="deserializedChart"/>
</Grid>
```

## C#

```
//Action to Serialize the Chart
private void Button_Click(object sender, RoutedEventArgs e)
{
    chart.Serialize();
}
//Action to Deserialize the Chart
private void Load_Click(object sender, RoutedEventArgs e)
{
    deserializedChart = (SfChart)chart.Deserialize();
}
```



```
<SfChart Name="chart" Margin="10,10,10,10" xmlns="http://schemas.syncfusion.com/wpf" xmlns:av="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:s="clr-namespace:System;assembly=mscorlib">
  <SfChart.PrimaryAxis>
    <CategoryAxis ShowTrackBallInfo="True" Width="579.906666666667" av:Canvas.Left="36.0933333333333"
    av:Canvas.Top="185.1">
      <CategoryAxis.StripLines>
        <ChartStripLines />
      </CategoryAxis.StripLines>
      <CategoryAxis.MultiLevelLabels>
        <ChartMultiLevelLabels />
      </CategoryAxis.MultiLevelLabels>
      <CategoryAxis.AxisLineStyle>
        <av:Style TargetType="av:Line">
          <av:Style.Resources>
            <av:ResourceDictionary />
          </av:Style.Resources>
        </av:Style>
      </CategoryAxis.AxisLineStyle>
    </CategoryAxis>
  </SfChart.PrimaryAxis>
</SfChart>
```

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

### Coded UI in WPF Charts (SfChart)

Automated tests that drive your application through its user interface (UI) are known as **coded UI tests** (CUITs). These tests include functional testing of the UI controls. SfChart supports Coded UI automation that helps you create automated tests for inner elements and records the sequence of actions. While dragging the crosshair on UI elements, it shows the properties of the respective UI elements and you can also add assertion for each of the properties.

#### Levels

Levels	Description
Level 1	Record and Identify UI Elements when you perform mouse and keyboard actions.
Level 2	Provide Custom properties for UI Elements when you drag the crosshair on the UI Element.

Level 3	CUIT generates code from recorded session. Create specialized class to access Custom Properties so the generate code is simplified.
---------	---

## Requirements

Coded UI provides support only in,

- Visual Studio Premium
- Visual Studio Enterprise
- Visual Studio Ultimate

For more info about the platforms and configurations refer [here](#)

## Configuration

To test SfChart with CUITs, build the Extension Project and place it in the mentioned location. You can get the Extension Project of SfChart from [here](#).

1. Open the extension project and build it.
2. Get Syncfusion.SfChart.CUITExtension.WPF.dll from bin.

Place this assembly in below mentioned location, and install this assembly in GAC.

The above assembly must be placed into the following directory based on your Visual Studio version.

For Visual Studio 2010: C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\10.0\UITestExtensionPackages

For Visual Studio 2012: C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\11.0\UITestExtensionPackages

For Visual Studio 2013: C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\12.0\UITestExtensionPackages

---

**Note:** Syncfusion.SfChart.CUITExtension.WPF.dll need to be installed in GAC location. Please refer the MSDN link for [GAC](#) installation.

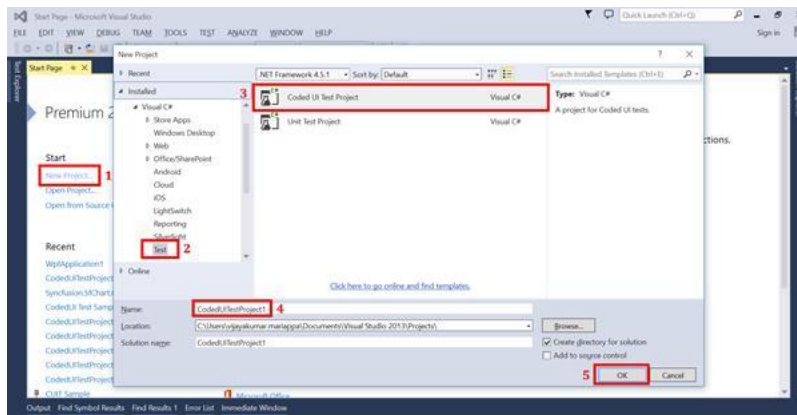
---

## Getting Started

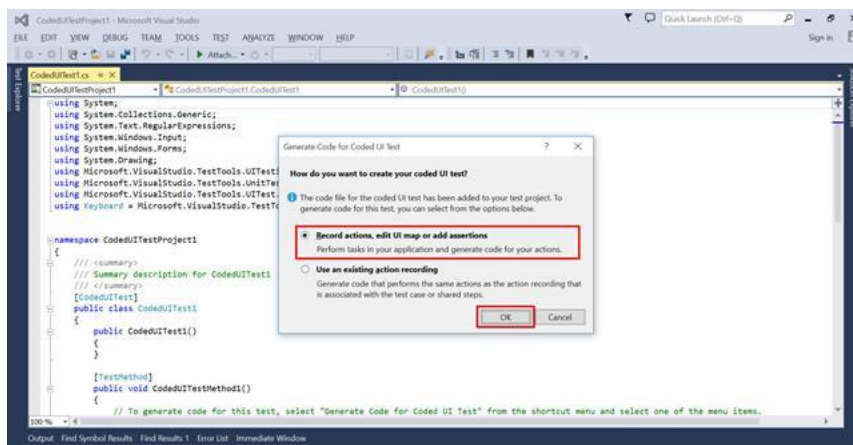
### Coded UI Project Creation

Run Visual Studio in administrator mode

Create a new Coded UI Test Project as shown in the following screenshot.



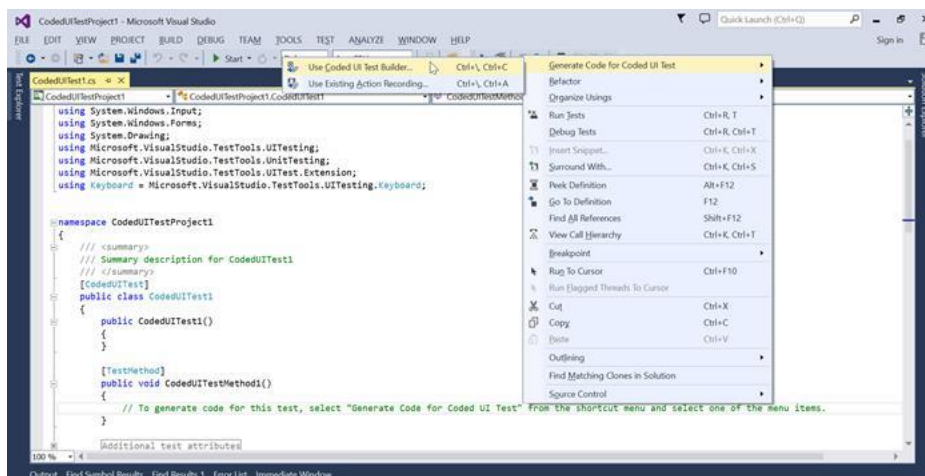
After you create a new Coded UI project, a CUIT file is added automatically and the Generate Code dialog box appears. In this, choose Record actions, edit UI map or add assertions.



Now Coded UI project Visual Studio gets minimized and CodedUITestBuilder appears in the bottom right corner of your window. You can record the actions by clicking Start Recording in CodedUITestBuilder.



You can also open the CodedUITestBuilder from existing Coded UI project by right clicking on the CodedUITestMethod1 in CUIT file and clicking the Generate Code for Coded UI Test.



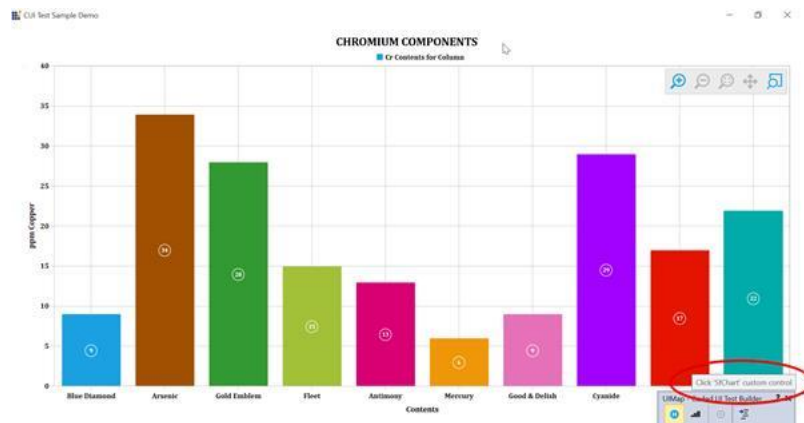


*Record and Generate code*

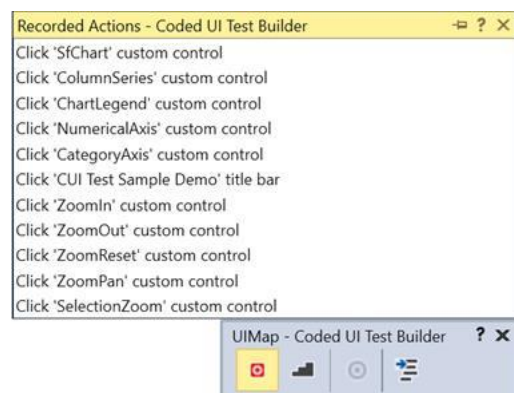
If you create a Coded UI Test project, UIMap – CodedUITestBuilder shown at bottom right corner.

Now you can record and perform actions in your application.

CodedUITestBuilder identify each actions and gives a tooltip message like below,



If you recorded by mistake, you can choose recorded steps to delete.



Once the record is completed, click the GenerateCode icon in CodedUITestBuilder for generate a test method. Then close the CodedUITestBuilder and you can see the generated code action as follows.

**C#**

```
public void RecordedMethod1 ()
{
    #region Variable Declarations
    WpfSfChart uISfChartCustom = this.UICUITestSampleDemoWindow.UISfChartCustom;
    WpfChartSeries uIColumnSeriesCustom =
        this.UICUITestSampleDemoWindow.UISfChartCustom.UIColumnSeriesCustom;
    WpfLegend uIChartLegendCustom =
        this.UICUITestSampleDemoWindow.UISfChartCustom.UIChartLegendCustom;
    WpfChartAxis uINumericalAxisCustom =
        this.UICUITestSampleDemoWindow.UISfChartCustom.UINumericalAxisCustom;
    WpfChartAxis uICategoryAxisCustom =
        this.UICUITestSampleDemoWindow.UISfChartCustom.UICategoryAxisCustom;
    WpfZoomingBehavior uIZoomInCustom =
        this.UICUITestSampleDemoWindow.UISfChartCustom.UIZoomInCustom;
    WpfZoomingBehavior uIZoomOutCustom =
        this.UICUITestSampleDemoWindow.UISfChartCustom.UIZoomOutCustom;
```

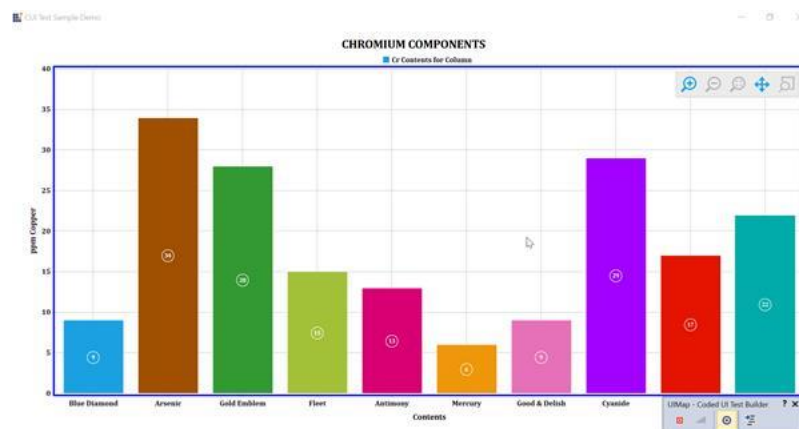
```

WpfZoomingBehavior uIZoomResetCustom =
    this.UICUITestSampleDemoWindow.UISfChartCustom.UIZoomResetCustom;
WpfZoomingBehavior uIZoomPanCustom =
    this.UICUITestSampleDemoWindow.UISfChartCustom.UIZoomPanCustom;
WpfZoomingBehavior uISelectionZoomCustom =
    this.UICUITestSampleDemoWindow.UISfChartCustom.UISelectionZoomCustom;
#endregion
}

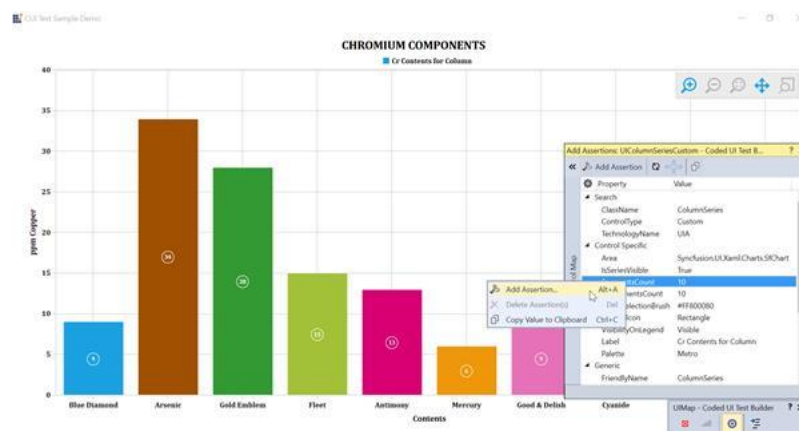
```

### Add Assertion

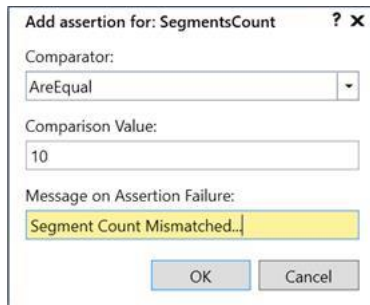
You can also create an assertion to check the modified chart/series properties value. Drag the crosshair to the chart series, and the Assertion window appears.



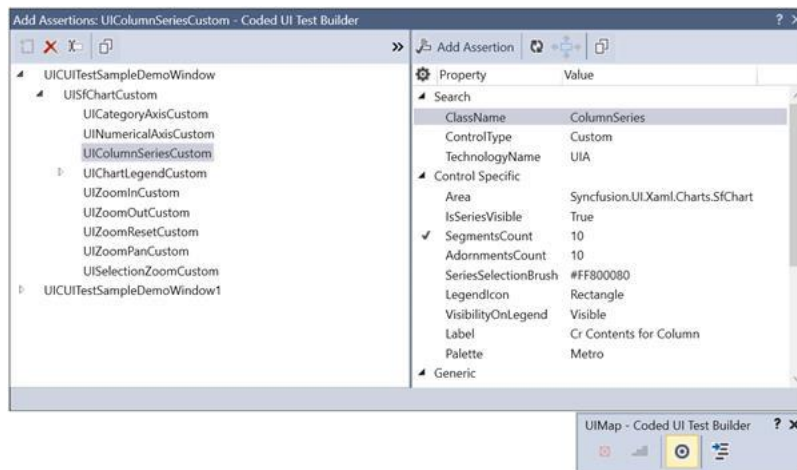
The series properties for control is now listed in the Assertion dialog box. You can add assertion by clicking the Generate Code button in CodedUITestBuilder.



Validate assertion property by using comparators. Also we can add assertion failure message.



Another way for adding assertion by clicking the cross hair icon in CodedUITestBuilder, here list out all identified controls and select the control we want to validate.



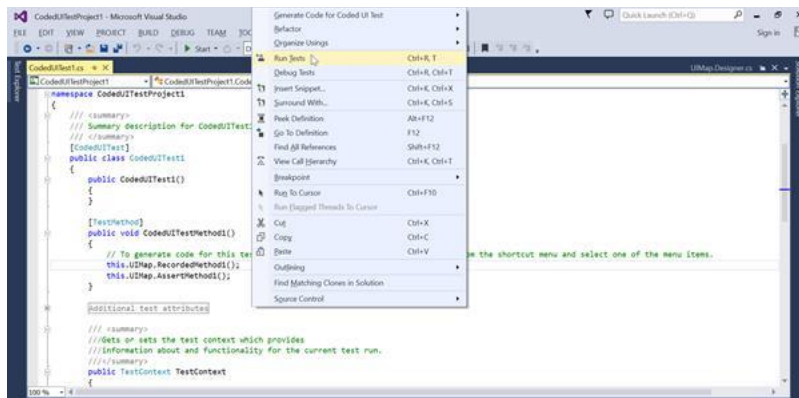
Click Generate code icon to generate assertion code then close the builder if validation is finish.

## C#

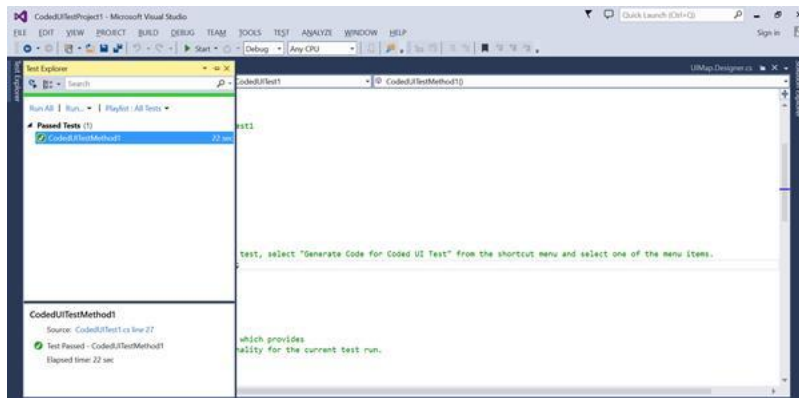
```
public void AssertMethod1 ()
{
    #region Variable Declarations
    WpfChartSeries uIColumnSeriesCustom =
    this.UICUITestSampleDemoWindow.UISfChartCustom.UIColumnSeriesCustom;
    #endregion
    // Verify that the 'SegmentsCount' property of 'ColumnSeries' custom control
    equals '10'
    Assert.AreEqual (this.AssertMethod1ExpectedValues.UIColumnSeriesCustomSegment
    sCount, uIColumnSeriesCustom.SegmentsCount, "Segemnts Count Mismatched...");
}
```

## Run Tests

After all tests and assertion are created, right-click on the Test method and click Run Tests to run the test as follows.



Open the test explorer and check test was passed or failed.



### Tables of Properties

The following properties are for each of the UI elements in SfChart,

UI Elements	Properties
SfChart	Header, PrimaryAxis, SecondaryAxis, Palette, AreaBackground, AreaBorderBrush, AreaBorderThickness, SideBySideSeriesPlacement, SeriesCount, BehaviorsCount, AnnotationsCount, TechnicalIndicatorsCount.
Axis	VisibleRange, Header, HeaderPosition, OpposedPosition, Orientation, ShowGridLines, IsInversed, ShowTrackBallInfo, LabelFormat, LabelsSource, LabelExtent, LabelsPosition, MaximumLabels, LabelsIntersectAction, LabelRotationAngle, EdgeLabelsDrawingMode, EdgeLabelsVisibilityMode, TickLinesPosition.
Series	Area, IsSeriesVisible, Interior, Stroke, StrokeThickness, SeriesSelectionBrush, LegendIcon, VisibilityOnLegend, Label, Palette, SegmentsCount, AdornmentsCount.
ZoomPan ToolBar	IconBackground, ToolBarIconHeight, ToolBarIconWidth, ToolBarIconMargin
Legend	Header, Orientation, DockPosition, LegendPosition, IconVisibility, CheckBoxVisibility, IconWidth, IconHeight, ItemMargin, Background, OffsetX, OffsetY.

Legend Item	IconVisibility, CheckBoxVisibility, VisibilityOnLegend, IsSeriesVisible, Label, IconWidth, IconHeight, ItemMargin.
-------------	--

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

## Migrating from Chart to SfChart in WPF Charts (SfChart)

SfChart is a new chart introduced in 11.1 version. SfChart is a very high performance chart enriched with several business features. This section helps you to identify equivalent Chart features/ APIs in SfChart.

### Adding Reference

Chart assembly Name: Syncfusion.Chart.Wpf assembly

SfChart assembly Name: Syncfusion.SfChart.WPF assembly

The following code example illustrates xmlns namespace for Chart. You can include the Syncfusion schema in WPF and both the charts are available in the WPF schema.

#### Chart

##### XML

```
xmlns:syncfusion="https://schemas.syncfusion.com/wpf"
```

#### SfChart

##### XML

```
xmlns:syncfusion="https://schemas.syncfusion.com/wpf"
(or)
xmlns:syncfusion="clr-
namespace:Syncfusion.UI.Xaml.Charts;assembly=Syncfusion.SfChart.WPF"
```

### Initialization

One of the biggest differences between Chart and SfChart is, Chart is constructed using one or more ChartArea. So, logically ChartArea is equivalent to SfChart.

**Note:** In SfChart, you can split the area into multiple plotting areas using RowDefinitions and ColumnDefinitions API.

Following code example illustrates the initialization of Chart with ChartArea,

##### XML

```
<syncfusion:Chart>
<syncfusion:ChartArea BorderBrush="Green" BorderThickness="5"
Background="Gray" SideBySideSeriesPlacement="True" />
</syncfusion:Chart>
```

##### C#

```
Chart chart = new Chart();
ChartArea area = new ChartArea();
area.Background = Brushes.Gray;
```

```
area.BorderBrush = Brushes.Red;
area.BorderThickness = new Thickness(5);
area.SideBySideSeriesPlacement = true;
chart.Areas.Add(area);
```

Following code example illustrates the initialization of SfChart that is equivalent to the above code example.

#### XML

```
<syncfusion:SfChart Header="ChartArea" AreaBackground="Red"
AreaBorderBrush="Gray" AreaBorderThickness="5"
SideBySideSeriesPlacement="True" />
```

#### C#

```
SfChart chart = new SfChart();
chart.AreaBackground = Brushes.Gray;
chart.AreaBorderBrush = Brushes.Red;
chart.AreaBorderThickness = new Thickness(5);
chart.SideBySideSeriesPlacement = true;
```

The following table illustrates the API comparison between ChartArea and SfChart.

ChartArea	SfChart	Description
Header	Header	Gets or sets the header name for Chart
Background	AreaBackground	Gets or sets the Chart area background (rectangle bounds excluding Axis and legend)
BorderBrush	AreaBorderBrush	Gets or sets the Chart area border brush.
BorderThickness	AreaBorderThickness	Gets or sets the Chart area border thickness.
SideBySideSeriesPlacement	SideBySideSeriesPlacement	Gets or sets Boolean value whether series is placed side by side or not.
Series	Series	Gets or sets ChartSeriesCollection to Chart.
ShowGridLines	-	Gets or sets Boolean value for showing the gridlines for both axes.
EnableRangeSelection	-	Gets or sets Boolean value that enables you to select a particular range of a primary axis by using two cursors.
StartValue	-	Gets or sets start value for range selection
EndValue	-	Gets or sets end value for range selection
ShowLegend	-	Gets or sets Boolean value for showing the chart legends.

SplitterColor	-	Gets or sets color for SyncChartAreas splitter
SplitterPosition	-	Gets or sets position for SyncChartAreas splitter
SplitterStroke	-	Gets or sets stroke for SyncChartAreas splitter
SplitterVisibility	-	Gets or sets visibility for SyncChartAreas splitter
SplitterWidth	-	Gets or sets width for SyncChartAreas splitter

### Legend

The Legends are same for both Chart and SfChart except that you can add multiple Legends for the SfChart.

The Legend property in ChartArea accepts single Legend for whole Chart. In case of SfChart, the Legend property of

the SfChart is an object that accepts one or more Legends based on your requirement.

### [For more details](#)

The following code example illustrates the usage of Legend in Chart and SfChart.

#### Chart

##### XML

```
<syncfusion:ChartArea>
<syncfusion:ChartArea.Legend>
<syncfusion:ChartLegend syncfusion:Chart.Dock="Floating" Orientation="Vertical"
OffsetX="200" OffsetY="200" ItemTemplate="{StaticResourcelegend}" Check
BoxVisibility="Collapsed"/>
</syncfusion:ChartArea.Legend>
</syncfusion:ChartArea>
```

#### C#

```
ChartArea area = new ChartArea();
ChartLegend legend = new ChartLegend();
Chart.SetDock(legend, ChartDock.Floating);
legend.Orientation = Orientation.Vertical;
legend.OffsetX = 200d;
legend.OffsetY = 200d;
legend.CheckBoxVisibility = Visibility.Visible;
area.Legend = legend;
chart.Areas.Add(area)
```

#### SfChart

##### XML

```
<syncfusion:SfChart>
<syncfusion:SfChart.Legend>
```

```
<syncfusion:ChartLegend ItemTemplate="{StaticResource legend}"CheckBoxVisib
ility="Visible" LegendPosition="Outside" Orientation="Vertical"OffsetY="200
" OffsetX="200" DockPosition="Floating"/>
</syncfusion:SfChart.Legend>
</syncfusion:SfChart>
```

**C#**

```
SfChart chart = new SfChart();
ChartLegend legend = new ChartLegend();
legend.ItemTemplate = grid.Resources["legend"] as DataTemplate;
legend.CheckBoxVisibility = Visibility.Visible;
legend.OffsetX = 200d;
legend.OffsetY = 200d;
legend.Orientation = ChartOrientation.Vertical;
legend.DockPosition = ChartDock.Top;
legend.LegendPosition = LegendPosition.Outside;
chart.Legend = legend;
```

The following table illustrates the API comparison between ChartArea and SfChart.

ChartLegend (Chart)	ChartLegend (SfChart)	Description
CheckBoxVisibility	CheckBoxVisibility	Gets or sets visibility for Legend check box
Orientation	Orientation	Gets or sets orientation for Legends
chart:Chart.Dock	DockPosition	Gets or sets docking position for Legends that decides the positioning for Legends.
OffsetX	OffsetX	Gets or sets X-axis offset value for Legends horizontal placement.
OffsetY	OffsetY	Gets or sets Y-axis offset value for Legends vertical placement. OffsetX and OffsetY are applicable when DockPosition is set as 'Floating'.
ItemTemplate	ItemTemplate	Gets or sets template for the Legend items.
IconVisibility	IconVisibility	Gets or sets the visibility of the Legend icon(s).
IconHeight	IconHeight	Gets or sets the double value that represents the Legend icon(s) width.
IconHeight	IconHeight	Gets or sets the double value that represents that Legend icon(s) height.
ItemMargin	ItemMargin	Gets or sets the margin of Legend items.
ElementMargin	Margin	Gets or sets the margin for Legend element.
ShowSymbol	IconVisibility	Gets or sets the visibility for Legend icon symbol.



## Axis

The class design for Axis is different for both ChartArea and SfChart. In ChartArea, Axis is the instance of ChartAxis type and its type is mentioned in ValueType property as in the following code example.

### XML

```
<syncfusion:ChartArea.PrimaryAxis>
<syncfusion:ChartAxis ValueType="DateTime" />
</syncfusion:ChartArea.PrimaryAxis>
```

In SfChart, Axis can be an instance of NumericalAxis, DateTimeAxis, CategoryAxis, LogarithmicAxis, TimeSpanAxis or DateTimeCategoryAxis. Class name represents the type of data it can plot. For example, NumericalAxis can plot numeric values and DateTimeAxis can plot DateTime values. Following code example illustrates this,

### XML

```
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:DateTimeAxis Header="X-Axis" />
</syncfusion:SfChart.PrimaryAxis>
```

The following table compares the APIs,

ChartAxis	NumericalAxis, DateTimeAxis, CategoryAxis, LogarithmicAxis, TimeSpanAxis and DateTimeCategoryAxis	Description
Header	Header	Gets or sets the header for the chart axis.
IntersectAction	LabelsIntersectAction	Gets or sets a value that determines how to avoid overlapping of labels.
RangePadding	RangePadding	Gets or sets the range padding for chart axis.
IsAutoSetRange, Range	Minimum, Maximum	Gets or sets range for chart axis
Interval	Interval	Gets or sets value for axis label
LabelPosition	LabelPosition	Gets or sets placement position for label.
AxisVisibility	Visibility	Gets or sets the visibility for Chart Axis.
EnableAutoIntervalOnZooming	EnableAutoIntervalOnZooming	Gets or sets a Boolean value to enable auto interval calculation while zooming

IsInversed	IsInversed	Gets or sets a value that indicates whether data is plotted reversely or not.
TickLinesPosition	TickLinesPosition	Gets or sets tick line position
TickLineSize	TickLineSize	Gets or sets tick line size
SmallTickLineSize	SmallTickLineSize	Gets or sets small tick line size
	SmallTickLinesPosition	Gets or sets small tick line position
SmallTicksPerInterval	SmallTicksPerInterval	Gets or sets value to enable small ticks.
LabelRotateAngle	LabelRotateAngle	Gets or sets label rotation angle.
EdgeLabelsDrawingMode	EdgeLabelsDrawingMode	Gets or sets position for edge labels.
DesiredIntervalsCount	DesiredIntervalsCount	Gets or sets desired interval count.
OpposedPosition	OpposedPosition	Gets or sets a value that indicates axis is positioned opposite to its default position.
LabelsSource	LabelsSource	Gets or sets label source for custom labels.
ContentPath	ContentPath	Gets or sets content path property value.
PositionPath	PositionPath	Gets or sets position path property value.
LabelsPostfix	PostfixLabelTemplate	Gets or sets post fix label template.
LabelsPrefix	PrefixLabelTemplate	Gets or sets prefix label template
LabelDateTimeFormat, LabelLogarithmicFormat LabelTimeSpanFormat	LabelFormat	Gets or sets axis label format.
LabelTemplate	LabelTemplate	Gets or sets label template.
chart:ChartArea.ShowGridLines	ShowGridline	Gets or sets a value to enable grid lines.
chart:ChartArea.SmallGridLineStroke	MajorGridLineStyle	Gets or sets major grid line style.
chart:ChartArea.SmallGridLineStroke	MajorTickLineStyle	Gets or sets tick line style.

ZoomFactor	ZoomFactor	Gets or sets the zoom factor value.
ZoomPosition	ZoomPosition	Gets or sets the zoom position value.

*Chart***XML**

```

<chart:ChartArea>
<chart:Chart.PrimaryAxis>
<chart:ChartAxis Value="DateTime" IntersectAction="Hide"
IsAutoSetRange="False" DateTimeRange="2010/01/01,2011/01/01"
SmallTickSize="6" SmallTicksPerInterval="5"
TickLinesPosition="Outside" TickSize="10"
LabelsSource="{Binding power}" ContentPath="Content"
PositionPath="Position"
LabelsPostfix="{StaticResource postLabel}"
LabelsPrefix="{StaticResource preLabel}"
DesiredIntervalsCount="5"
LabelDateTimeFormat="MM:dd:yy" RangePadding="Normal"
LabelPosition="Outside" Header="X-Axis"
EdgeLabelsDrawingMode="Center" IsInversed="False"
LabelRotateAngle="0" AxisVisibility="Visible"
EnableAutoIntervalOnZooming="False" ZoomFactor="1"
ZoomPosition="1" chart:ChartArea.ShowGridLines="True" />
</chart:Chart.PrimaryAxis>
</chart:ChartArea>

```

**C#**

```

ChartArea area = new ChartArea();
ChartAxis primaryAxis = new ChartAxis();
primaryAxis.ValueType = ChartValueType.DateTime;
primaryAxis.LabelDateTimeFormat = "MM:dd:yy";
primaryAxis.LabelPosition = LabelPositions.Outside;
primaryAxis.AxisVisibility = Visibility.Visible;
primaryAxis.Header = "X-Axis";
primaryAxis.RangePadding = ChartRangePaddingType.Normal;
primaryAxis.TickLinesPosition = AxisPositions.Outside;
primaryAxis.EdgeLabelsDrawingMode = EdgeLabelsDrawingMode.Center;
primaryAxis.IsInversed = false;
primaryAxis.IsAutoSetRange = false;
primaryAxis.DateTimeRange = new DateTimeRange(new DateTime(2010, 01, 01),
new DateTime(2011, 01, 01));
primaryAxis.DateTimeInterval = new TimeSpan(24, 0, 0);
primaryAxis.DesiredIntervalsCount = 5;
primaryAxis.TickLinesPosition = AxisPositions.Outside;
primaryAxis.TickSize = 10d;
primaryAxis.SmallTickSize = 6d;
primaryAxis.SmallTicksPerInterval = 5;
primaryAxis.IntersectAction = ChartLabelIntersectAction.Hide;
ChartArea.SetShowGridLines(primaryAxis, true);
primaryAxis.LabelsSource = viewmodel.power;

```

```
primaryAxis.ContentPath = "Content";
primaryAxis.PositionPath = "Position";
primaryAxis.ShowGridline= true;
primaryAxis.OpposedPosition = true;
primaryAxis.ZoomFactor = 1d;
primaryAxis.ZoomPosition = 0d;
area.PrimaryAxis = primaryAxis;
```

## SfChart

### XML

```
<sfchart:SfChart>
  <sfchart:SfChart.PrimaryAxis>
    <sfchart:DateTimeAxis Header="X-Axis" OpposedPosition="True"
      RangePadding="Auto"
      DesiredIntervalsCount="5" IsInversed="False"
      LabelFormat="MM/dd/yy" RangePadding="Auto"
      Interval="1" Minimum="2010/01/01" Maximum="2011/01/10"
      IntervalType="Years" LabelsPosition="Outside"
      LabelsIntersectAction="Hide" TickLinesPosition="Outside"
      Visibility="Visible" EnableScrollBar="False"
      EdgeLabelsDrawingMode="Center"
      EnableAutoIntervalOnZooming="True"
      ShowTrackBallInfo="True" PlotOffset="20"
      LabelsSource="{Binding power}" ShowGridline="True"
      ContentPath="Content" PositionPath="Postion"
      PostfixLabelTemplate="{StaticResource postLabel}"
      PrefixLabelTemplate="{StaticResource preLabel}"
      SmallTickLinesPosition="Outside" SmallTicksPerInterval="5"
      TickLineSize="6" ZoomFactor="1" ZoomPosition="0"/>
    </sfchart:SfChart.PrimaryAxis/>
  </sfchart:SfChart>
```

### C#

```
SfChart chart = new SfChart();
DateTimeAxis primaryAxis = new DateTimeAxis();
primaryAxis.Interval = 1;
primaryAxis.LabelFormat = "MM/dd/yy";
primaryAxis.IntervalType = DateTimeIntervalType.Years;
primaryAxis.IsInversed = false;
primaryAxis.LabelsPosition = AxisElementPosition.Outside;
primaryAxis.LabelsIntersectAction = AxisLabelsIntersectAction.Hide;
primaryAxis.TickLinesPosition = AxisElementPosition.Outside;
primaryAxis.Visibility = Visibility.Visible;
primaryAxis.EnableScrollBar = false;
primaryAxis.EdgeLabelsDrawingMode = EdgeLabelsDrawingMode.Center;
primaryAxis.RangePadding = DateTimeRangePadding.Auto;
primaryAxis.PlotOffset = 10d;
primaryAxis.Header = "X-Axis";
primaryAxis.ShowTrackBallInfo = true;
primaryAxis.DesiredIntervalsCount = 5;
primaryAxis.OpposedPosition = true;
primaryAxis.SmallTickLineSize = 6d;
primaryAxis.SmallTickLinesPosition = AxisElementPosition.Outside;
```

```

primaryAxis.SmallTicksPerInterval = 5;
primaryAxis.TickLineSize = 10d;
primaryAxis.TickLinesPosition = AxisElementPosition.Outside;
primaryAxis.LabelsSource = viewmodel.power;
primaryAxis.ContentPath = "Content";
primaryAxis.PositionPath = "Position";
primaryAxis.ShowGridline= true;
primaryAxis.OpposedPosition = true;
primaryAxis.Minimum = new DateTime(2011, 1, 1);
primaryAxis.Maximum = new DateTime(2012, 1, 1);
primaryAxis.ZoomFactor = 1d;
primaryAxis.ZoomPosition = 0d;
chart.PrimaryAxis = primaryAxis;

```

Following code example illustrates how to customize gridlines.

*Chart*

**XML**

```

<chart:ChartArea.GridLineStyle>
<Pen Brush="#c6c6c6" >
<Pen.DashStyle>
<DashStyle Dashes="1,3"/>
</Pen.DashStyle>
</Pen>
</chart:ChartArea.GridLineStyle>

```

**C#**

```

Pen pen = new Pen();
DashStyle style = new DashStyle() { Dashes = new DoubleCollection(3) };
pen.DashStyle = style;
pen.Brush = Brushes.Red;
ChartArea.SetGridLineStyle(primaryAxis, pen);

```

*SfChart*

**XML**

```

<sfchart:NumericalAxis.MajorGridLineStyle>
<Style TargetType="Line">
<Setter Property="StrokeDashArray" Value="1,3"></Setter>
<Setter Property="Stroke" Value="#c6c6c6" />
</Style>
</sfchart:NumericalAxis.MajorGridLineStyle>

```

**C#**

```

Style lineStyle = new Style(typeof(Line));
lineStyle.Setters.Add(new Setter(Line.StrokeDashArrayProperty, new
DoubleCollection(3)));
lineStyle.Setters.Add(new Setter(Line.StrokeProperty, Brushes.Red));
primaryAxis.MajorGridLineStyle = lineStyle;

```

## Series

Like Axis, ChartSeries architecture is also different for Chart and SfChart. Instead of specifying the respective ChartType, you can create the instance of required series (Class name itself represents the type of series) in SfChart as in the following code example.

### Chart

#### XML

```
<syncfusion:ChartArea>
<syncfusion:ChartSeries Type="Line" ShowToolTip="True"
Interior="Blue" ShowSmartLabels="True"
LegendIcon="Diamond"
AdornmentIntersectAction="AdjustAroundPoints"
ColorEach="True" Palette="Metro"
StrokePalette="Metro" IsRotated="False"
IsIndexed="True" IsVisibleOnLegend="True"
IsSortData="False" SortBy="X"
SortDirection="Ascending"
ShowDataLabels="True" EnableEffects="True"
IsVisible="True" StrokeThickness="3"
EnableAnimation="True" AnimateOption="Fade"
AnimateOneByOne="True" ShowEmptyPoints="True"
EmptyPointInterior="Purple" EmptyPointValue="Average"
EmptyPointStyle="Interior" DataSource="{Binding power}"
BindingPathX="Year" BindingPathsY="Sports"
Label="LineSeries" >
<syncfusion:ChartSeries.YAxis>
<syncfusion:ChartAxis IsAutoSetRange="False" Range="20,50"/>
</syncfusion:ChartSeries.YAxis>
</syncfusion:ChartSeries>
</syncfusion:ChartArea>
```

### C#

```
ChartArea area = new ChartArea();
ChartSeries lineseries = new ChartSeries();
lineseries.BindingPathX = "Year";
lineseries.Label = "ColumnSeries";
lineseries.BindingPathsY = new string[] { "Sports" };
lineseries.DataSource = viewmodel.power;
lineseries.ShowToolTip = true;
lineseries.ShowEmptyPoints = true;
lineseries.EmptyPointValue = EmptyPointValue.Average;
lineseries.EmptyPointStyle = EmptyPointStyle.Interior;
lineseries.EmptyPointInterior = Brushes.Purple;
lineseries.EnableAnimation = true;
lineseries.IsSortData = true;
lineseries.IsRotated = true;
lineseries.IsVisible = true;
lineseries.SortBy = SortingAxis.X;
lineseries.SortDirection = Direction.Ascending;
lineseries.YAxis = new ChartSeries() { IsAutosetRange = false, Range = new
DoubleRange(20, 50) };
lineseries.ShowSmartLabels = true;
area.Series.Add(lineseries);
```

*SfChart***XML**

```

<syncfusion:SfChart>
<syncfusion:LineSeries ItemsSource="{Binding power}"
XBindingPath="Year" YBindingPath="Sports"
Palette="Metro" AnimationDuration="00:00:03"
SortBy="X" SortDirection="Ascending"
EnableSegmentDragging="True" LegendIcon="Diamond"
VisibilityOnLegend="Visible" Label="LineSeries"
IsSeriesVisible="True" IsSortData="True"
IsTransposed="True" EnableAnimation="True"
ShowTooltip="True" StrokeThickness="3"
ShowEmptyPoints="True" EmptyPointValue="Average"
EmptyPointInterior="Purple" EmptyPointStyle="Interior"
<syncfusion:LineSeries.YAxis>
<syncfusion:NumericalAxis Minimum="20" Maximum="50"/>
</syncfusion:LineSeries.YAxis>
</syncfusion:LineSeries>
</syncfusion:SfChart>

```

**C#**

```

SfChart chart = new Sfchart();
LineSeries lineseries = new LineSeries();
lineseries.ItemsSource = viewmodel.power;
lineseries.XBindingPath = "Year";
lineseries.YBindingPath = "Sports";
lineseries.Label = "LineSeries";
lineseries.Interior = Brushes.Brown;
lineseries.ShowTooltip = true;
lineseries.ShowEmptyPoints = true;
lineseries.EmptyPointInterior = Brushes.Purple;
lineseries.EmptyPointStyle = EmptyPointStyle.Interior;
lineseries.EmptyPointValue = EmptyPointValue.Average;
lineseries.StrokeThickness = 3;
lineseries.EnableAnimation = true;
lineseries.AnimationDuration = new TimeSpan(00, 00, 03);
lineseries.IsSeriesVisible = true;
lineseries.IsSortData = true;
lineseries.IsTransposed = true;
lineseries.SortBy = SortingAxis.X;
lineseries.YAxis = new NumericalAxis() { Minimum = 20, Maximum = 50 };
lineseries.SortDirection = Direction.Ascending;
chart.Series.Add(lineseries);

```

The following table illustrates the API comparison for series,

ChartSeries	LineSeries, ColumnSeries, SplineSeries etc.,	Description
Label	Label	Gets or sets series label name.

ShowTooltip	ShowTooltip	Gets or sets a value to enable tooltip.
EnableAnimation	EnableAnimation	Gets or sets a value to enable animation.
AnimationDuration	AnimationDuration	Gets or sets timespan for animation duration.
AnimateOneByOne	-	Gets or sets a value whether to animate the series one by one or not.
AnimateOption	-	Gets or sets enum for animation direction.
ColorEach	Palette	Gets or sets a value that indicates whether to color each segment differently or not.
BindingPathX	XBindingPath	Gets or sets x axis binding property.
BindingPathsY	YBindingPath,High,Low,Open,Close,Size	Gets or sets y axis binding property. In Chart, it is a collection of y value paths whereas in SfChart, each y path is represented by individual properties. For example, Candle, HiLoOpenClose series contains four y values, each value is represented by High, Low, Open and Close properties. HiLo contains two y values represented by High and Low properties. Bubble series contains two y values represented by YBindingPath and Size properties. Other



		series that contains one y value is represented by YBindingPath property.
DataSource	ItemsSource	Gets or sets items source for chart series.
Interior	Interior	Gets or sets interior color for chart series.
ShowEmptyPoints	ShowEmptyPoints	Gets or sets a value that indicates whether to show empty points in different style or color based on EmptyPointValue and EmptyPointValue.
EmptyPointValue	EmptyPointValue	Gets or sets a value that indicates whether to represent empty point as average of previous and next value or just zero.
EmptyPointInterior	EmptyPointInterior	Gets or sets empty point value interior.
IsSortData	IsSortData	Gets or sets a value to enable sorting in chart series.
SortBy	SortBy	Sorting the series based on X or Y axis value.
SortDirection	SortDirection	Sorting the chart series based on direction like ascending or descending.
ShowSmartLabels	EnableSmartLabels	Gets or sets a value whether to avoid the adornment intersection. In

		SfChart EnableSmartLabels is used for Pie and Doughnut series.
EnableEffects	-	Gets or sets effect for Chart series.
Unit	-	Gets or sets legend unit.
UnitVisibility	-	Gets or sets legend unit visibility
chart:ChartPieType.ExplodedAll chart:ChartDoughnutType.ExplodedAll	ExplodedAll	Gets or sets a value whether to explode all the segments of pie and doughnut chart
chart:ChartPieType.ExplodedIndex chart:ChartDoughnutType.ExplodedIndex	ExplodedIndex	Gets or sets the index of the segment to be exploded.
chart:ChartPieType.ExplodeRadius chart:ChartDoughnutType.ExplodeRadius	ExplodeRadius	Gets or sets explode radius for all the segment of pie, doughnut chart.
ChartPyramidType.GapRatio	GapRatio	Gets or sets gap ratio for pyramid series.
ChartPyramidType.PyramidMode	PyramidMode	Gets or sets mode for pyramid series.

## Adornments

The Adornments are same as Chart and you can define the Adornments inside the Series like Chart.

## Chart

### XML

```
<syncfusion:ChartSeries>
  <syncfusion:ChartSeries.ChartAdornmentInfo >
    <syncfusion:ChartAdornmentInfo SegmentLabelContent="YValue"
    SegmentShowLine="True" Visible="True"
    SymbolInterior="Red"
    AdornmentsPosition="Top" Symbol="Ellipse"
    SegmentLabelRotation="20"
    SymbolStroke="Red" SymbolHeight="15"
    SymbolWidth="15" />
  </syncfusion:ChartSeries.ChartAdornmentInfo />
</syncfusion:ChartSeries>
```

**C#**

```

ChartSeries lineseries = new ChartSeries();
ChartAdornmentInfo adornment = new ChartAdornmentInfo();
adornment.Visible = true;
adornment.Symbol = Symbol.Ellipse;
adornment.SymbolInterior = Brushes.Red;
adornment.SymbolWidth = 15d;
adornment.SymbolHeight = 15d;
adornment.SegmentLabelContent = LabelContent.YValue;
adornment.SegmentShowLine = true;
adornment.SegmentLabelRotation = 30d;
adornment.AdornmentsPosition = AdornmentsPosition.Top;
lineseries.AdornmentsInfo = adornment;

```

*SfChart***XML**

```

<syncfusion:LineSeries >
<syncfusion:LineSeries.ChartAdornmentInfo>
<syncfusion:ChartAdornmentInfo Symbol="Cross"
UseSeriesPalette="True"
AdornmentsPosition="Top"
ShowConnectorLine="True"
ConnectorHeight="10"
SegmentLabelContent="YValue"
ShowLabel="True" SymbolStroke="Red"
SymbolHeight="20" SymbolWidth="20"
SymbolInterior="Red"/>
</syncfusion:LineSeries.ChartAdornmentInfo>
</syncfusion:LineSeries >

```

**C#**

```

LineSeries lineseries = new LineSeries();
ChartAdornmentInfo adornment = new ChartAdornmentInfo();
adornment.ShowLabel = true;
adornment.Symbol = ChartSymbol.Ellipse;
adornment.SymbolHeight = 20d;
adornment.SymbolWidth = 20d;
adornment.SymbolInterior = Brushes.Red;
adornment.SegmentLabelContent = LabelContent.YValue;
adornment.SymbolStroke = Brushes.Red;
adornment.AdornmentsPosition = AdornmentsPosition.Top;
adornment.ConnectorHeight = 10d;
adornment.UseSeriesPalette = true;
adornment.ShowConnectorLine = true;
lineseries.AdornmentsInfo = adornment;

```

The following table illustrates the API comparison for Adornments.

ChartAdornmentInfo (Chart)	ChartAdornmentInfo (SfChart)	Description
----------------------------	------------------------------	-------------

Visible	ShowLabel	Gets or sets a value whether to show Adornment label or not.
Symbol	Symbol	Gets or sets symbol for Adornment.
SymbolInterior	SymbolInterior	Gets or sets color for Adornment.
SymbolStroke	SymbolStroke	Gets or sets stroke color for Adornment symbol.
AdornmentsPosition	AdornmentsPosition	Gets or sets position for Adornment.
LabelTemplate	LabelTemplate	Gets or sets template for label.
SymbolTemplate	SymbolTemplate	Gets or sets template for symbol.
SegmentLabelContent	SegmentLabelContent	Gets or sets Adornment segment label content.
ConnectorTemplate	ConnectorHeight,ConnectorLineStyle	Customizing the Adornment connector line.
	UseSeriesPalette	Gets or sets color for each connector line
SegmentShowLine	ShowConnectorLine	Gets or sets a value to enable connector line.

### Interactive Cursor

The InteractiveCursor is represented as ChartCrosshairBehavior and ChartTrackballBehavior in SfChart. Following code example illustrates this,

#### Chart

##### XML

```
<syncfusion:ChartArea>
<syncfusion:ChartArea.InteractiveCursors>
<syncfusion:InteractiveCursor VerticalLabelVisibility="Visible"HorizontalLabelVisibility="Visible" IsBindWithMouseMove="True"IsBindWithSegment="False" OffsetX="100" OffsetY="100"/>
</syncfusion:ChartArea.InteractiveCursors>
</syncfusion:ChartArea>
```

##### C#

```
ChartArea area= new ChartArea();
InteractiveCursor incCursor = new InteractiveCursor();
incCursor.VerticalLabelVisibility = Visibility.Visible;
incCursor.HorizontalLabelVisibility = Visibility.Visible;
incCursor.IsBindWithMouseMove = true;
incCursor.IsBindWithSegment = false;
incCursor.OffsetX = 100d;
incCursor.OffsetY = 100d;
```

```
area.InteractiveCursors.Add(incCursor);
```

## SfChart

### XML

```
<syncfusion:SfChart>
<syncfusion:SfChart.Behaviors>
<syncfusion:ChartCrossHairBehavior HorizontalAxisLabelAlignment="Center"VerticalAxisLabelAlignment="Center" />
</syncfusion:SfChart.Behaviors/>
</syncfusion:SfChart>
```

### C#

```
SfChart chart = new SfChart();
ChartCrossHairBehavior crosshair = new ChartCrossHairBehavior();
crosshair.VerticalAxisLabelAlignment = ChartAlignment.Center;
crosshair.HorizontalAxisLabelAlignment = ChartAlignment.Center;
chart.Behaviors.Add(crosshair);
```

## Zooming and Panning

The Zooming and Panning are achieved using the ChartZoomPanBehavior in SfChart as in the following code example.

## Chart

### XML

```
<syncfusion:ChartArea EnableZoomOnScroll="True" ZoomAllAxes="True"chart:ChartZoomingToolkit.ZoomingToolkitVisibility="Visible"EnableMouseDragZooming="True" />
```

### C#

```
ChartArea area= new ChartArea();
area.ZoomAllAxes = true;
area.EnableZoomOnScroll = true;
ChartZoomingToolkit.SetZoomingToolkitVisibility(area, Visibility.Visible);
```

## SfChart

### XML

```
<syncfusion:SfChart>
<syncfusion:SfChart.Behaviors>
<syncfusion:ChartZoomPanBehavior ZoomMode="XY" EnablePinchZooming="False"ZoomRelativeToCursor="False" EnableMouseWheelZooming="True"EnableSelectionZooming="False" EnablePanning="True" />
</syncfusion:SfChart.Behaviors/>
</syncfusion:SfChart>
```

### C#

```
SfChart chart = new SfChart();
ChartZoomPanBehavior zoom = new ChartZoomPanBehavior();
```

```
zoom.EnableMouseWheelZooming = true;
zoom.EnablePanning = true;
zoom.EnableSelectionZooming = false;
zoom.ZoomMode = ZoomMode.XY;
zoom.ZoomRelativeToCursor = false;
zoom.EnablePinchZooming = false;
chart.Behaviors.Add(zoom);
```

## StripLines

The following code example demonstrates the usage of StripLines in Chart and SfChart.

### Chart

#### XML

```
<syncfusion:ChartAxis.StripLines>
<syncfusion:ChartStripLine x:Name="strip" Start="40" IsSegmented="False" SegmentStartValue="20" SegmentEndValue="40" RepeatEvery="10" RepeatUntil="50" TextRotationAngle="30" StartFromAxis="False" Width="5" IsPixelWidth="False" Interior="#b4e8f3" >
</syncfusion:ChartAxis.StripLines/>
```

### C#

```
ChartArea area= new ChartArea();
ChartStripLine stripline = new ChartStripLine();
stripline.Offset = 20d;
stripline.Width = 5d;
stripline.StartFromAxis = false;
stripline.IsSegmented = false;
stripline.SegmentStartValue = 20d;
stripline.SegmentEndValue = 40d;
stripline.RepeatEvery = 10d;
stripline.RepeatUntil = 50d;
stripline.TextRotationAngle = 30;
stripline.IsPixelWidth = false;
stripline.Interior = new SolidColorBrush(Colors.Red);
stripline.Text = new FormattedText("StripLine", CultureInfo.CurrentCulture, FlowDirection.LeftToRight, new Typeface("Times New Roman"), 20, Brushes.Black);
area.SecondaryAxis.StripLines.Add(stripline);
```

### SfChart

#### XML

```
<syncfusion:NumericalAxis.StripLines>
<syncfusion:ChartStripLine Start="40" LabelAngle="30" LabelHorizontalAlignment="Center" LabelVerticalAlignment="Center" RepeatEvery="10" RepeatUntil="50" IsSegmented="False" SegmentStartValue="20" SegmentEndValue="50" IsPixelWidth="True" Width="5" Label="StripLine" Background="#b4e8f3" />
</syncfusion:NumericalAxis.StripLines/>
```

### C#

```
SfChart chart = new SfChart();
```

```

ChartStripLine stripline = new ChartStripLine();
stripline.Start = 40d;
stripline.LabelAngle = 30d;
stripline.LabelHorizontalAlignment = HorizontalAlignment.Center;
stripline.LabelVerticalAlignment = VerticalAlignment.Center;
stripline.RepeatEvery = 10d;
stripline.RepeatUntil = 50d;
stripline.IsSegmented = false;
stripline.SegmentStartValue = 20d;
stripline.SegmentEndValue = 50d;
stripline.IsPixelWidth = true;
stripline.Background = new SolidColorBrush(Colors.Red);
stripline.Width = 5d;
stripline.Background = Brushes.LightGray;
stripline.Label = "StripLine";
chart.SecondaryAxis.StripLines.Add(stripline);

```

The following table illustrates the API comparison between Chart and SfChart.

ChartStripLine (Chart)	ChartStripLine (SfChart)	Description
Start	Start	Gets or sets start value for axis.
Width	Width	Gets or sets the width. When IsPixelWidth is <i>true</i> , then it considers unit of width as pixel otherwise it is axis value.
Interior	Background	Gets or sets color for strip line background.
Text	Label	Gets or sets text for strip line. In Chart Text as a 'Formatted Text' and in SfChart label as a 'String'.
TextRotationAngle	LabelAngle	Gets or sets label angle.
TextAlignment	LabelHorizontalAlignmentLabelVerticalAlignment	Gets or sets alignment for strip line labels.
RepeatEvery	RepeatEvery	Gets or sets value for strip line occurrence.
RepeatUntil	RepeatUntil	Gets or sets value where the strip line occurrence ends.
IsSegmented	IsSegmented	Gets or sets value that indicates whether strip line is segmented.
SegmentStartValue	SegmentStartValue	Gets or sets segment start value.
SegmentEndValue	SegmentEndValue	Gets or sets segment end value.

IsPixelWidth	IsPixelWidth	Gets or sets a value that indicates the unit of the value of Width is pixel.
--------------	--------------	--

## Watermark

Following code example illustrates the WatermarkAPI comparison,

### Chart

#### XML

```
<syncfusion:ChartArea>
<syncfusion:ChartArea.Watermark>
<VisualBrush Stretch="None" Opacity="0.5" AlignmentX="Center" AlignmentY="Center">
<VisualBrush.Visual>
<TextBlock Text="Chart" FontSize="60" Foreground="Gray">
</TextBlock>
</VisualBrush.Visual>
</VisualBrush>
</syncfusion:ChartArea.Watermark>
</syncfusion:ChartArea>
```

### C#

```
ChartArea area = new ChartArea();
area.Watermark = new VisualBrush()
{
    Opacity = 0.5,
    AlignmentX = AlignmentX.Center,
    AlignmentY = AlignmentY.Center,
    Stretch = Stretch.None,
    Visual = new TextBlock()
    {
        Text = "Chart", Foreground = Brushes.Gray, FontSize = 60d
    }
};
```

### SfChart

#### XML

```
<syncfusion:SfChart>
<syncfusion:SfChart.Watermark>
<syncfusion:Watermark Canvas.ZIndex="-1" HorizontalAlignment="Center" VerticalAlignment="Center">
<syncfusion:Watermark.Content>
<TextBlock Text="SfChart" FontSize="60" Foreground="Gray" Opacity="0.5"></TextBlock>
</syncfusion:Watermark.Content>
</syncfusion:Watermark>
</syncfusion:SfChart.Watermark>
</syncfusion:SfChart>
```

### C#



```
SfChart chart = new SfChart();
Watermark waterMark = new Watermark();
waterMark.HorizontalAlignment = HorizontalAlignment.Center;
waterMark.VerticalAlignment = VerticalAlignment.Center;
Canvas.SetZIndex(waterMark, -1);
waterMark.Content = new TextBlock() { Text = "SfChart", Foreground =
Brushes.Gray, Opacity = 0.5, FontSize = 60d };
chart.Watermark = waterMark;
```

## Annotation

In Chart, you can add Annotations to chart and series. Annotations added to Chart are positioned based on OffsetX and OffsetY whose values are in the unit of pixel. Annotations added to series are positioned relative to axis. Annotation shape is specified in AnnotationShape property.

Following code example illustrates the Annotation types and Annotation API's for both Charts:

### Chart

Chart having following Annotations

- Chart Annotation

### XML

```
<syncfusion:Chart.AnnotationLabels>
< syncfusion:ChartAnnotationLabel AnnotationShape="None" Template="{StaticRe
sourcetemplateImage}" Content="Text
Annotation" OffsetX="500" OffsetY="100"IsAnnotationDragDrop="True" />
</syncfusion:Chart.AnnotationLabels>
```

### C#

```
Chart chart = new Chart();
ChartAnnotationLabel chartannotation=new ChartAnnotationLabel();
chartannotation.AnnotationShape = AnnotationShapes.None;
chartannotation.Content = "Text Annotation";
chartannotation.OffsetX = 500d;
chartannotation.OffsetY = 100d;
chartannotation.IsAnnotationDragDrop = true;
chart.AnnotationLabels.Add(chartannotation);
```

- Series Annotation

### XML

```
<syncfusion:ChartSeries.Annotations>
<syncfusion:AnnotationsCollection >
<syncfusion:ChartSeriesAnnotation X="40702" Y="468" AnnotationShape="Diamon
d"Stroke="Black" Fill="Orange"/>
</syncfusion:AnnotationsCollection>
</syncfusion:ChartSeries.Annotations>
```

### C#

```

ChartSeries columnSeries = new ChartSeries();
AnnotationsCollection annoCollection = new AnnotationsCollection();
ChartSeriesAnnotation seriesAnnotation=new ChartSeriesAnnotation();
seriesAnnotation.X = 40702;
seriesAnnotation.Y = 468;
seriesAnnotation.AnnotationShape = AnnotationShapes.Diamond;
seriesAnnotation.Stroke = Brushes.Black;
seriesAnnotation.Fill = Brushes.Orange;
annoCollection.m_annotationsCollection.Add(seriesAnnotation);
columnSeries.Annotations = annoCollection;

```

### SfChart

In SfChart, you can position Annotations in pixel unit or axis. Units are specified using `CoordinateUnit` property. Unlike Chart, shape is represented by class name itself in SfChart.

#### *SfChart having the following Annotations*

- Image Annotation

### XML

```

<syncfusion:SfChart.Annotations>
< syncfusion:ImageAnnotation ImageSource="Annotation.png"VerticalTextAlignme
nt="Center" Foreground="White" CoordinateUnit="Axis" X1="30"X2="75" Y1="520"
Y2="560">
</syncfusion:ImageAnnotation>
</syncfusion:SfChart.Annotations/>

```

### C#

```

SfChart chart = new SfChart();
ImageAnnotation imgAnnotation = new ImageAnnotation();
imgAnnotation.ImageSource = new BitmapImage(new
Uri(@"Images/Annotation.png", UriKind.Relative));
imgAnnotation.VerticalTextAlignment = VerticalAlignment.Center;
imgAnnotation.Foreground = new SolidColorBrush(Colors.White);
imgAnnotation.CoordinateUnit = CoordinateUnit.Axis;
imgAnnotation.X1 = 30d;
imgAnnotation.X2 = 75d;
imgAnnotation.Y1 = 520d;
imgAnnotation.Y2 = 560d;
chart.Annotations.Add(imgAnnotation);

```

- Line Annotation

### XML

```

<syncfusion:SfChart.Annotations>
< syncfusion:LineAnnotation X1="3" Y1="34" X2="5" Y2="38" CanResize="True"Ca
nDrag="True"></ syncfusion:LineAnnotation>
</syncfusion:SfChart.Annotations/>

```

**C#**

```
SfChart chart = new SfChart();
LineAnnotation lineAnnotation = new LineAnnotation();
lineAnnotation.CoordinateUnit = CoordinateUnit.Axis;
lineAnnotation.CanResize = true;
lineAnnotation.CanDrag = true;
lineAnnotation.X1 = 3d;
lineAnnotation.X2 = 34d;
lineAnnotation.Y1 = 5d;
lineAnnotation.Y2 = 38d;
chart.Annotations.Add(lineAnnotation);
```

- Ellipse Annotation

**XML**

```
<syncfusion:SfChart.Annotations>
<syncfusion:EllipseAnnotation X1="1" Y1="36" X2="2" Y2="42" CanResize="True"
CanDrag="True"></ syncfusion:EllipseAnnotation>
</syncfusion:SfChart.Annotations/>
```

**C#**

```
SfChart chart = new SfChart();
EllipseAnnotation ellipseAnnotation = new EllipseAnnotation();
ellipseAnnotation.CoordinateUnit = CoordinateUnit.Axis;
ellipseAnnotation.CanResize = true;
ellipseAnnotation.CanDrag = true;
ellipseAnnotation.X1 = 36d;
ellipseAnnotation.X2 = 2d;
ellipseAnnotation.Y1 = 36d;
ellipseAnnotation.Y2 = 42d;
chart.Annotations.Add(ellipseAnnotation);
```

- Rectangle Annotation

**XML**

```
<syncfusion:SfChart.Annotations>
<syncfusion:RectangleAnnotation X1="4" Y1="40" X2="6" Y2="42" CanResize="True"
CanDrag="True"></syncfusion:RectangleAnnotation>
</syncfusion:SfChart.Annotations/>
```

**C#**

```
SfChart chart = new SfChart();
RectangleAnnotation rectAnnotation = new RectangleAnnotation();
rectAnnotation.CoordinateUnit = CoordinateUnit.Axis;
rectAnnotation.CanResize = true;
rectAnnotation.CanDrag = true;
rectAnnotation.X1 = 36d;
rectAnnotation.X2 = 2d;
```

```
rectAnnotation.Y1 = 36d;  
rectAnnotation.Y2 = 42d;  
chart.Annotations.Add(rectAnnotation);
```

- Vertical Line Annotation

#### XML

```
<syncfusion:SfChart.Annotations>  
<syncfusion:VerticalLineAnnotation CanDrag="True" ShowAxisLabel="True"CanRes  
ize="True" CoordinateUnit="Axis" X1="3">  
</syncfusion:VerticalLineAnnotation>  
</syncfusion:SfChart.Annotations/>
```

#### C#

```
SfChart chart = new SfChart();  
VerticalLineAnnotation verAnnotation = new VerticalLineAnnotation();  
verAnnotation.CoordinateUnit = CoordinateUnit.Axis;  
verAnnotation.ShowAxisLabel = true;  
verAnnotation.CanDrag = true;  
verAnnotation.X1 = 3d;  
chart.Annotations.Add(verAnnotation);
```

- Horizontal Line Annotation

#### XML

```
<syncfusion:SfChart.Annotations>  
<syncfusion:HorizontalLineAnnotation CanDrag="True" CanResize="True"Coordina  
teUnit="Axis" ShowAxisLabel="True" Y1="40" >  
</syncfusion:HorizontalLineAnnotation>  
</syncfusion:SfChart.Annotations/>
```

#### C#

```
SfChart chart = new SfChart();  
HorizontalLineAnnotation horAnnotation = new HorizontalLineAnnotation();  
horAnnotation.CoordinateUnit = CoordinateUnit.Axis;  
horAnnotation.ShowAxisLabel = true;  
horAnnotation.CanDrag = true;  
horAnnotation.Y1 = 40d;  
chart.Annotations.Add(horAnnotation);
```

- Text Annotation

#### XML

```
<syncfusion:SfChart.Annotations>  
<syncfusion:TextAnnotation X1="0" Y1="150" Text="Release  
1" ContentTemplate="{StaticResource textTemplate}" ToolTipContent="Volume 1
```

```
Released!!!"ToolTipTemplate="{StaticResource
tooltip}" VerticalAlignment="Top" HorizontalAlignment="Left" ToolTipPlaceme
nt="Top"ShowToolTip="True">
</syncfusion:TextAnnotation>
<syncfusion:SfChart.Annotations/>
```

**C#**

```
SfChart chart = new SfChart();
TextAnnotation textAnnotation = new TextAnnotation();
textAnnotation.CoordinateUnit = CoordinateUnit.Axis;
textAnnotation.X1 = 0d;
textAnnotation.Y1 = 150d;
textAnnotation.Text = "Release 1";
textAnnotation.ToolTipContent = "Volume 1 Released!!!";
textAnnotation.HorizontalAlignment = HorizontalAlignment.Left;
textAnnotation.ToolTipPlacement = ToolTipLabelPlacement.Top;
textAnnotation.ShowToolTip = true;
chart.Annotations.Add(textAnnotation);
```

*Technical Indicators*

The following table illustrates the API comparison for TechnicalIndicators,

ChartTechnicalIndicator (Chart)	AccumulationDistributionIndicator, BollingerBandIndicator etc.,(SfChart)	Description
SignalLineColor	SignalLineColor	Gets or sets the Brush value that represents the indicator signal line color.
UpperLineColor	UpperLineColor	Gets or sets the Brush value that represents the color for the Upper Bollinger Band.
ConvergenceLineColor	ConvergenceLineColor	Gets or sets the Brush value that represents the line color for the convergence.
DivergenceLineColor	DivergenceLineColor	Gets or sets the Brush value that represents the Divergence Line color.

chart:ChartBollingerBand.BollingerMovingAverage	Period	Gets or sets the value that represents the indicator moving average period
chart:ChartExponentialAverage.ExponentialAverage	Period	Gets or sets the value that represents the indicator moving average period
chart:ChartSimpleAverage.MovingAverage	Period	Gets or sets the value that represents the indicator moving average period
chart:ChartTriangularAverage.TriangularAverage	Period	Gets or sets the value that represents the indicator moving average period
chart:ChartMomentum.MomentumTimeSpan	Period	Measures the amount that a security's price has changed over a given time span.
IndicatorType	-	In Chart, you can set the type of the indicator where as in SfChart, indicator type is indicated by the class name itself.

[Chart](#)

**XML**

```
<syncfusion:ChartSeries.Indicators>
```

```

<syncfusion:IndicatorCollection>
<syncfusion:IndicatorCollection.Items>
<syncfusion:ChartTechnicalIndicator
IndicatorType="BollingerBands"
chart:ChartBollingerBand.LowerLineColor ="Blue"
chart:ChartBollingerBand.UpperLineColor="Red"
chart:ChartBollingerBand.SignalLineColor ="Green"
chart:ChartBollingerBand.BollingerMovingAverage="50" >
</syncfusion:ChartTechnicalIndicator>
</syncfusion:IndicatorCollection.Items>
</syncfusion:IndicatorCollection>
</syncfusion:ChartSeries.Indicators>

```

**C#**

```

ChartSeries columnSeries = new ChartSeries();
IndicatorCollection indiCollection = new IndicatorCollection();
ChartTechnicalIndicator indicator = new ChartTechnicalIndicator();
indicator.IndicatorType = IndicatorTypes.BollingerBands;
ChartBollingerBand.SetLowerLineColor(chart, Brushes.Blue);
ChartBollingerBand.SetSignalLineColor(chart, Brushes.Red);
ChartBollingerBand.SetSignalLineColor(chart, Brushes.Green);
ChartBollingerBand.SetBollingerMovingAverage(chart, 50);
indiCollection.Items.Add(indicator);
columnSeries.Indicators = indiCollection;

```

## SfChart

**XML**

```

<syncfusion:SfChart.TechnicalIndicators>
<syncfusion:BollingerBandIndicator UpperLineColor="Brown"LowerLineColor="DarkBlue" Period="1" XBindingPath="Year" ItemsSource="{BindingCompanyDetails}" High="High" Open="Open" Close="Close" Low="Low"/>
</syncfusion:SfChart.TechnicalIndicators>

```

**C#**

```

SfChart chart = new SfChart();
BollingerBandIndicator indicator = new BollingerBandIndicator();
indicator.UpperLineColor=new SolidColorBrush(Colors.Brown);
indicator.LowerLineColor=new SolidColorBrush(Colors.DarkBlue);
indicator.XBindingPath = "Year";
indicator.High = "High";
indicator.Open = "Open";
indicator.Close = "Close";
indicator.Low = "Low";
indicator.ItemsSource = viewmodel.CompanyDetails;
chart.TechnicalIndicators.Add(indicator);

```

## SyncChartAreas

In Chart, you can split chart area into multiple plotting areas that share common axis, crosshair/interactive cursor etc., using SyncChartAreas class.

In SfChart, you can split chart into multiple plotting areas that share common axis, crosshair/interactive cursor and zooming functionalities by defining multiple rows and columns using RowDefinitions and ColumnDefinition properties. It is similar to defining rows and columns in Grid panel.

The following code example illustrates the API's for both charts,

#### Chart

#### SyncChartAreas

#### XML

```
<syncfusion:Chart>
<syncfusion:SyncChartAreas >
<syncfusion:SyncChartAreas.PrimaryAxis>
<syncfusion:ChartAxis ValueType="DateTime" LabelDateTimeFormat="MM/dd/yy" Header="Date">
</syncfusion:ChartAxis>
</syncfusion:SyncChartAreas.PrimaryAxis>
<syncfusion:SyncChartAreas.Areas>
<syncfusion:ChartArea x:Name="Area1" >
<syncfusion:ChartArea.SecondaryAxis >
<syncfusion:ChartAxis EdgeLabelsDrawingMode="Shift" Interval="50" Header="Prices">
</syncfusion:ChartAxis>
</syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartSeries Type="Line" DataSource="{Binding datalist}" BindingPathX="Time" BindingPathsY="Low" >
</syncfusion:ChartSeries>
</syncfusion:ChartArea>
<syncfusion:ChartArea x:Name="Area2" >
<syncfusion:ChartArea.SecondaryAxis >
<syncfusion:ChartAxis EdgeLabelsDrawingMode="Shift" Interval="50" Header="Prices">
</syncfusion:ChartAxis>
</syncfusion:ChartArea.SecondaryAxis>
<syncfusion:ChartSeries Type="Line" DataSource="{Binding datalist}" BindingPathX="Time" BindingPathsY="High" >
</syncfusion:ChartSeries>
</syncfusion:ChartArea>
</syncfusion:SyncChartAreas.Areas>
</syncfusion:SyncChartAreas>
</syncfusion:Chart>
```

#### C#

```
SyncChartAreas areas=new SyncChartAreas();
areas.PrimaryAxis = new ChartAxis();
ChartArea area1 = new ChartArea();
area1.SecondaryAxis = new ChartAxis();
ChartSeries lineSeries1 = new ChartSeries();
lineSeries1.BindingPathX = "Time";
lineSeries1.BindingPathsY = new string[] { "Low" };
lineSeries1.DataSource = viewmodel.datalist;
lineSeries1.Type = ChartTypes.Line;
area1.Series.Add(lineSeries1);
ChartArea area2 = new ChartArea();
area2.SecondaryAxis = new ChartAxis();
```



```

ChartSeries lineSeries2 = new ChartSeries();
lineSeries2.BindingPathX = "Time";
lineSeries2.BindingPathsY = new string[] { "High" };
lineSeries2.DataSource = viewmodel.datalist;
lineSeries2.Type = ChartTypes.Line;
area2.Series.Add(lineSeries1);
areas.Areas.Add(area1);
areas.Areas.Add(area2);

```

## SfChart

### Column Row Definition

#### XML

```

<syncfusion:SfChart>
  <syncfusion:SfChart.RowDefinitions>
    <syncfusion:ChartRowDefinition/>
    <syncfusion:ChartRowDefinition/>
  </syncfusion:SfChart.RowDefinitions>
  <syncfusion:SfChart.PrimaryAxis>
    <syncfusion:CategoryAxis ShowGridLines="False" FontSize="20"Header="Company
Name" />
  </syncfusion:SfChart.PrimaryAxis>
  <syncfusion:SfChart.SecondaryAxis>
    <syncfusion:NumericalAxis syncfusion:ChartBase.Row="0" Interval="40"FontSiz
e="20" Header="Gross Revenue (cr.)"/>
  </syncfusion:SfChart.SecondaryAxis>
  <syncfusion:LineSeries Stroke="Red" StrokeThickness="3"XBindingPath="Compan
yName" YBindingPath="CompanyTurnOver1" ItemsSource="{BindingCompanyDetails}"
/>
  <syncfusion:LineSeries Stroke="Green" StrokeThickness="3"XBindingPath="Comp
anyName" YBindingPath="CompanyTurnOver2" ItemsSource="{BindingCompanyDetails
}">
  <syncfusion:LineSeries.YAxis>
    <syncfusion:NumericalAxis syncfusion:ChartBase.Row="1"Header="Additional
Axis" PlotOffset="30"/>
  </syncfusion:LineSeries.YAxis>
  </syncfusion:LineSeries>
</syncfusion:SfChart>

```

#### C#

```

SfChart chart = new SfChart();
ChartRowDefinition row1=new ChartRowDefinition();
ChartRowDefinition row2 = new ChartRowDefinition();
chart.RowDefinitions.Add(row1);
chart.RowDefinitions.Add(row2);
chart.PrimaryAxis = new CategoryAxis();
NumericalAxis numAxis1 = new NumericalAxis();
ChartBase.SetRow(numAxis1,0);
LineSeries lineseries1 = new LineSeries();
lineseries1.ItemsSource = viewmodel.CompanyDetails;
lineseries1.XBindingPath = "CompanyName";
lineseries1.YBindingPath = "CompanyTurnOver1";
LineSeries lineseries2 = new LineSeries();
lineseries2.ItemsSource = viewmodel.CompanyDetails;

```

```

lineseries2.XBindingPath = "CompanyName";
lineseries2.YBindingPath = "CompanyTurnOver2";
NumericalAxis numAxis2 = new NumericalAxis();
ChartBase.SetRow(numAxis2, 1);
lineseries2.YAxis = numAxis2;
chart.Series.Add(lineseries1);
chart.Series.Add(lineseries2);

```

## Exporting and Printing

### Export to Image

#### Chart

##### C#

```

Chart chart = new Chart();
SaveFileDialog saveFileDialog = new SaveFileDialog();
string C_imageFilesFilter =
"Bitmap (*.bmp) | *.bmp | JPEG (*.jpg, *.jpeg) | *.jpg; *.jpeg | Gif (*.gif) | *.gif | TIFF (*.tiff) | *.tiff | PNG (*.png) | *.png | WDP (*.wdp) | *.wdp | Xps file (*.xps) | *.xps | All files (*.*) | *.*";
saveFileDialog.Filter = C_imageFilesFilter;
if (saveFileDialog.ShowDialog() == true)
{
    chart.Save(saveFileDialog.FileName);
}

```

#### SfChart

##### C#

```

SfChart sfchart = new SfChart();
SaveFileDialog sfd = new SaveFileDialog();
sfd.Filter = "Bitmap (*.bmp) | *.bmp | JPEG (*.jpg, *.jpeg) | *.jpg; *.jpeg | Gif (*.gif) | *.gif | PNG (*.png) | *.png | All files (*.*) | *.*";
if (sfd.ShowDialog() == true)
{
    using (Stream fs = sfd.OpenFile())
    {
        sfchart.Save(fs, new PngBitmapEncoder());
    }
}

```

## Printing Chart

### Chart

#### XML

```

<syncfusion:Chart x:Name="chart"/>
<Button Content="Print" Command="{x:Static ApplicationCommands.Print}"
CommandTarget="{Binding ElementName=chart}"/>
<Button Content="Printing
Mode" Command="{x:Static syncfusion:ChartCommands.SwitchPrinting}"
CommandTarget="{Binding ElementName=chart}"/>

```

## SfChart

### C#

```
SfChart chart = new SfChart();  
chart.Print();
```

**Note:** You can refer to our [WPF Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [WPF Charts example](#) to know various chart types and how to easily configured with built-in support for creating stunning visual effects.

## How to

### Add custom labels to track ball behavior

In the ChartTrackBallBehavior, each data point will have a label aligned vertically and horizontally using the LabelVerticalAlignment and LabelHorizontalAlignment properties by default. However, you can also add custom labels to the ChartTrackBallBehavior.

In order to add a custom label, you need to write a class derived from ChartTrackBallBehavior. You need to override GenerateLabels method, which will be called whenever new labels are going to be generated, and add the labels using AddLabel method. The following code sample demonstrates this:

### C#

```
public class CustomTrackBallBehavior: ChartTrackBallBehavior  
{  
    public DataTemplate CustomLabelTemplate  
    {  
        get { return (DataTemplate)GetValue(CustomLabelTemplateProperty); }  
        set { SetValue(CustomLabelTemplateProperty, value); }  
    }  
    // Using a DependencyProperty as the backing store for CustomLabelTemplate.  
    This enables animation, styling, binding, etc.  
    public static readonly DependencyProperty CustomLabelTemplateProperty =  
        DependencyProperty.Register("CustomLabelTemplate", typeof(DataTemplate),  
            typeof(CustomTrackBallBehavior), new PropertyMetadata(null));  
    protected override void GenerateLabels()  
    {  
        AddLabel(PointInfos[0], LabelVerticalAlignment, LabelHorizontalAlignment,  
            PointInfos[0].Series.TrackBallLabelTemplate);  
        AddLabel(PointInfos[1], LabelVerticalAlignment, LabelHorizontalAlignment,  
            PointInfos[1].Series.TrackBallLabelTemplate);  
        ChartPointInfo pointInfo1 = PointInfos[2];  
        ChartPointInfo pointInfo2 = PointInfos[3];  
        CustomLabel label = new CustomLabel();  
        label.Value1 = pointInfo2.ValueY;  
        label.Value2 = pointInfo1.ValueY;  
        Rect rect = pointInfo1.Series.YAxis.ArrangeRect;  
        //Custom label.  
        AddLabel(label, LabelVerticalAlignment, LabelHorizontalAlignment,  
            CustomLabelTemplate, pointInfo1.X, rect.Bottom);  
    }  
}  
  
public class CustomLabel  
{  
    public string Value1 { get; set; }  
    public string Value2 { get; set; }  
}
```

### Transform axis value to pixel value and vice versa

SfChart offers two utility methods for converting your data points into pixel values (device coordinates).

- ValueToPoint(ChartAxis axis, double value)
- PointToValue(ChartAxis axis, Point point)

#### C#

```
private void LineChart_MouseMove(object sender, MouseEventArgs e)
{
    Point mousePoint = new Point
    {
        X = e.GetPosition(LineChart).X - LineChart.SeriesClipRect.Left -
        LineChart.Margin.Left,
        Y = e.GetPosition(LineChart).Y - LineChart.SeriesClipRect.Top -
        LineChart.Margin.Top
    };
    // Converts mouse co-ordinate points into a value related to ChartAxis.
    double xValue = this.LineChart.PointToValue(this.LineChart.PrimaryAxis,
        mousePoint);
    double yValue = this.LineChart.PointToValue(this.LineChart.SecondaryAxis,
        mousePoint);
    // Converts the data point value of the chart to Chart coordinate.
    double chartPointX = this.LineChart.ValueToPoint(this.LineChart.PrimaryAxis,
        xValue);
    double chartPointY
    = this.LineChart.ValueToPoint(this.LineChart.SecondaryAxis, yValue);
}
```

### Add range of points dynamically

Whenever you add a data point to ItemsSource dynamically, corresponding data will be updated inside chart series synchronously. This operation will be happening for each and every data point that we add subsequently. You can avoid this by calling SuspendSeriesNotification method of Chart before adding range of data points and then call ResumeSeriesNotification to update all the data points that have been added between these two method calls.

#### C#

```
Chart.SuspendSeriesNotification();
// ...
// Add multiple data points.
// ...
Chart.ResumeSeriesNotification();
```

### Print the SfChart in WPF

To print the SfChart, call the Print method.

The following code sample can be used to print the SfChart:

#### C#

```
chart.Print();
```

### Export Chart to Image (Windows 8.1)

The export chart to image feature in the SfChart control enables the user to export the image of the chart in different image file formats.

#### Supported Formats

- JPG or JPEG
- JPG-XR
- GIF
- TIFF
- PNG
- BMP

#### Method Table

Method	Prototype	Description
Save	Save(string fileName, StorageFolder folderLocation)	Export the chart image with the given file name to the mentioned location.If folderLocation is null then it exports the image to the app installed location.
Save	Save(IRandomAccessStream stream, Guid bitmapEncoderID)	Export the chart image using the stream and its corresponding encoder.

#### Method 1

##### C#

```
chart.Save("sfchart.jpg", KnownFolders.PicturesLibrary);
```

#### Method 2

##### C#

```
var memoryStream = new InMemoryRandomAccessStream();
chart.Save(memoryStream, BitmapEncoder.BmpEncoderId);
StorageFolder storageFolder =
Windows.ApplicationModel.Package.Current.InstalledLocation;
var file = await storageFolder.CreateFileAsync("chartwithstream.jpg",
CreationCollisionOption.GenerateUniqueName);
var stream = await file.OpenAsync(Windows.Storage.FileAccessMode.ReadWrite);
{
chart.Save(stream, BitmapEncoder.BmpEncoderId);
}
```

#### Serialization and Deserialization:

SfChart provides the support for serializing and deserializing control. This section explains on how to serialize and deserialize SfChart.

*Methods:*

Method Name	Description
Serialize()	Serialize the control and saves as the XML file in the parent folder.
Serialize(string filePath)	Serialize the control and saves as the XML file in the given file path.
Serialize(Stream stream)	Serialize the control and saves as the XML file in the given stream.
Deserialize()	Deserialize the XML file from the parent folder and returns the SfChart control object.
Deserialize(string filePath)	Deserialize the XML file from the given file path and returns the SfChart control object.
Deserialize(Stream stream)	Deserialize the XML file from the given stream and returns the SfChart control object.

## XAML

**XML**

```
<chart:SfChart x:Name=chart>
<chart:ColumnSeries ItemsSource="{Binding CategoricalDatas}"
XBindingPath="Category" YBindingPath="Value" Palette="RedChrome"/>
</chart:SfChart>
<StackPanel>
<Button x:Name="Serialize" Content="Serialize" Height="50" Width="100"
Margin="10" Click="Serialize_Click"/>
<Button x:Name="Deserialize" Content="Deserialize" Height="50" Width="100"
Margin="10" Click="Deserialize_Click"/>
</StackPanel>
```

## C#

**C#**

```
private void Serialize_Click(object sender, RoutedEventArgs e)
{
    string filePath = @"E:/Documents/chart.xml";
    chart.Serialize(filePath);
}
private void Deserialize_Click(object sender, RoutedEventArgs e)
{
    string filePath = @"E:/Documents/chart.xml";
    var deserializedChart = (SfChart)chart.Deserialize();
}
```

## SfChart3D

## WPF SfChart3D Overview

3D charts are used to view two-dimensional data in a three-dimensional view, and can be rotated in all 3 dimensions to get the best possible view of the data.



### Key features

- SfChart3D supports different type of [Series](#) which can be used for different data visualizations. Each type of chart represents a unique style of representing data with more user friendly and greater UI visualization.
- Supports several axes types which can be used with the charts. There are axes specialized for Numerical, Category, DateTime, TimeSpan, Logarithmic scenarios.
- SfChart3D supports DepthAxis (Z Axis) helps us to plot chart data based on X, Y and Z Co – ordinates.
- Supports multiple axes that can be stacked and spanned for multiple panes.
- Allows you to map the data from the specified path, by achieving, Data binding concept.
- Animations allows to animate the chart series on loading, and whenever the ItemsSource changes.
- Dynamic rotation allows to view the best possible view of data dynamically using a mouse or touch device.
- Selection support allows to select Segment and series programmatically or user interaction.

### Getting Started with WPF SfChart3D

This section explains you the steps required to populate the Chart with data, header, add data labels, legend and tooltips to the Chart. This section covers only the minimal features that you need to learn to get started with the Chart.

#### Adding chart reference

Refer to this [article](#) to learn how to add Syncfusion controls to Visual Studio projects in various ways. You can also refer to [this](#) link to learn about the assemblies required for adding Chart to your project.

#### Initialize chart

Import the SfChart3D namespace in your XAML page.

#### XML

```
xmlns:chart="clr-namespace:Syncfusion.UI.Xaml.Charts;assembly=Syncfusion.SfChart.WPF"
```

#### C#

```
using Syncfusion.UI.Xaml.Charts;
```

Then initialize an empty chart with [PrimaryAxis](#) and [SecondaryAxis](#) as shown below,

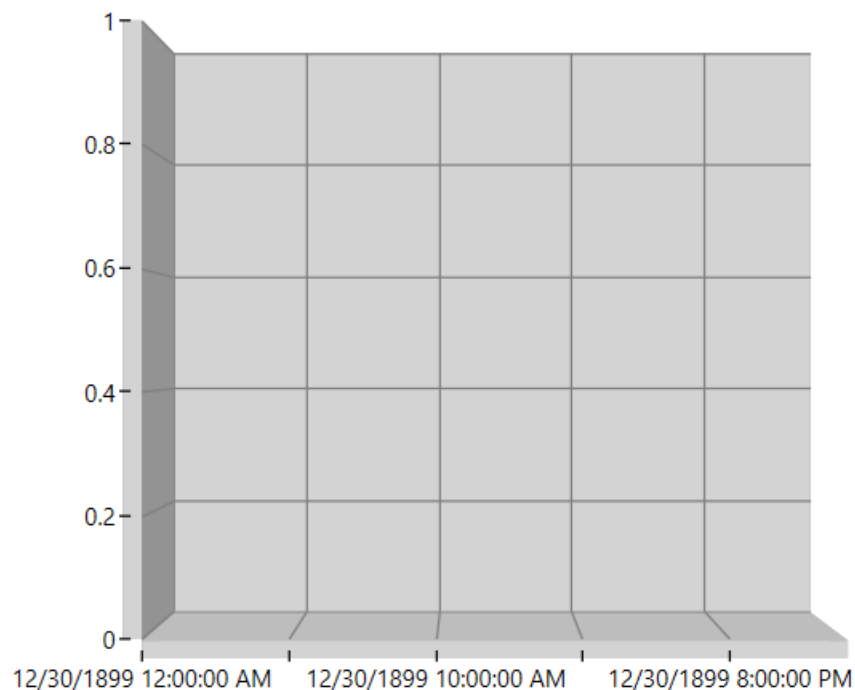
#### XML

```
<chart:SfChart3D >  
  <!--PrimaryAxis-->  
  <chart:SfChart3D.PrimaryAxis>  
  <chart:DateTimeAxis3D/>  
  </chart:SfChart3D.PrimaryAxis>  
  <!--SecondaryAxis-->  
  <chart:SfChart3D.SecondaryAxis>
```

```
<chart:NumericalAxis3D/>
</chart:SfChart3D.SecondaryAxis>
</chart:SfChart3D>
```

**C#**

```
SfChart3D Chart3D = new SfChart3D()
{
    PrimaryAxis = new DateTimeAxis3D(),
    SecondaryAxis = new NumericalAxis3D(),
};
```

**Initialize view model**

Since, the above step will produce only an empty column 3D chart, plotting data must be added to the chart. This step illustrates how to create a sample data source. The data source must implement the `IEnumerable` interface.

**C#**

```
public class UserProfile
{
    public DateTime TimeStamp { get; set; }
    public double NoOfUsers { get; set; }
}
```

Next, create a view model class and initialize a list of `UserProfile` objects as follows.

**C#**

```
public class UsersViewModel
{
    public UsersViewModel()
```



```

{
    this.UsersList = new ObservableCollection<UserProfile>();
    DateTime date = DateTime.Today;
    UsersList.Add(new UserProfile { Timestamp = date.AddHours(0.5), NoOfUsers = 1000 });
    UsersList.Add(new UserProfile { Timestamp = date.AddHours(1), NoOfUsers = 5000 });
    UsersList.Add(new UserProfile { Timestamp = date.AddHours(1.5), NoOfUsers = 3000 });
    UsersList.Add(new UserProfile { Timestamp = date.AddHours(2), NoOfUsers = 4000 });
    UsersList.Add(new UserProfile { Timestamp = date.AddHours(2.5), NoOfUsers = 2000 });
    UsersList.Add(new UserProfile { Timestamp = date.AddHours(3), NoOfUsers = 1000 });
}
public ObservableCollection<UserProfile> UsersList { get; set; }
}

```

Set the ViewModel instance as the DataContext of your window; this is done to bind properties of ViewModel.

**Note:** Add namespace of `ViewModel` class to your XAML window if you prefer to set `DataContext` in XAML.

#### XML

```

<Window x:Class="GettingStarted_3DCharts.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:GettingStarted_3DCharts"
    xmlns:chart="clr-namespace:Syncfusion.UI.Xaml.Charts;assembly=Syncfusion.SfChart.WPF"
    mc:Ignorable="d"
    Title="MainWindow" Height="350" Width="525">
    <Window.DataContext>
    <local:UsersViewModel/>
    </Window.DataContext>
</Window>

```

#### C#

```

this.DataContext = new UsersViewModel();

```

#### Populate chart with data

As we are going to visualize the comparison of heights in the data model, add [ColumnSeries3D](#) to [SfChart3D.Series](#) property, and then bind the Data property of the above ViewModel to the [ColumnSeries3D.ItemsSource](#) property as follows.

N You need to set [XBindingPath](#) and [YBindingPath](#) properties, so that [SfChart3D](#) would fetch values from the respective properties in the data model to plot the series.

**XML**

```
<chart:SfChart3D x:Name="Chart3D" Width="500" Height="500">
  <!--PrimaryAxis-->
  <chart:SfChart3D.PrimaryAxis>
  <chart:DateTimeAxis3D/>
  </chart:SfChart3D.PrimaryAxis>
  <!--SecondaryAxis-->
  <chart:SfChart3D.SecondaryAxis>
  <chart:NumericalAxis3D/>
  </chart:SfChart3D.SecondaryAxis>
  <chart:ColumnSeries3D ItemsSource="{Binding UsersList}"
  XBindingPath="TimeStamp"
  YBindingPath="NoOfUsers"></chart:ColumnSeries3D>
</chart:SfChart3D>
```

**C#**

```
SfChart3D chart3D = new SfChart3D();
chart3D.PrimaryAxis = new CategoryAxis3D();
chart3D.SecondaryAxis = new NumericalAxis3D();
ColumnSeries3D series = new ColumnSeries3D()
{
    ItemsSource = new UsersViewModel().UsersList,
    XBindingPath = "TimeStamp",
    YBindingPath = "NoOfUsers"
};
Chart3D.Series.Add(series);
```

**Add Title**

The header of the chart acts as the title to provide quick information to the user about the data being plotted in the chart. You can set title using the [Header](#) property of chart as follows.

**XML**

```
<Grid>
  <chart:SfChart3D Header="Chart">
  </chart:SfChart3D>
</Grid>
```

**C#**

```
SfChart3D chart3D = new SfChart3D();
chart3D.Header = "Chart";
```

**Enable data labels**

You can add data labels to improve the readability of the chart and it can be enabled using [AdornmentsInfo](#) property of [ChartSeries3D](#). By default, there is no label displayed, you have to set [ShowLabel](#) property of [ChartAdornmentInfo3D](#) to True.

**XML**

```
<Grid>
  <chart:SfChart3D >
```

```

...
<chart:ColumnSeries3D ItemsSource="{Binding UsersList}"
XBindingPath="TimeStamp"
YBindingPath="NoOfUsers">
<chart:ColumnSeries3D.AdornmentsInfo>
<chart:ChartAdornmentInfo3D></chart:ChartAdornmentInfo3D>
</chart:ColumnSeries3D.AdornmentsInfo>
</chart:ColumnSeries3D>
...
</chart:SfChart3D>
</Grid>

```

**C#**

```
series.AdornmentsInfo = new ChartAdornmentInfo3D () { ShowLabel = true };
```

Refer to [Adornments](#) to learn more about the options to customize chart adornments.

**Enable legend**

You can enable legend using the [Legend](#) property as follows.

**XML**

```

<Grid>
<chart:SfChart3D >
...
<!--Legend-->
<chart:SfChart3D.Legend>
<chart:ChartLegend></chart:ChartLegend>
</chart:SfChart3D.Legend>
...
</chart:SfChart3D>
</Grid>

```

**C#**

```
chart.Legend = new ChartLegend();
```

Additionally, you need to set label for each series using the [Label](#) property of ChartSeries, which will be displayed in corresponding legend.

**XML**

```

<chart:SfChart3D >
...
<chart:ColumnSeries3D Label="UserProfile" ItemsSource="{Binding UsersList}"
XBindingPath="TimeStamp"
YBindingPath="NoOfUsers" >
</chart:ColumnSeries3D>
...
</chart:SfChart3D>

```

**C#**

```
ColumnSeries3D series = new ColumnSeries3D()
{
    ItemsSource = new UsersViewModel().UsersList,
    XBindingPath = "TimeStamp",
    YBindingPath = "NoOfUsers",
    Label = "UserProfile"
};
```

### Enable tooltip

Tooltips are used to show information about the segment, when you click the segment. You can enable tooltip by setting series [ShowTooltip](#) property to true.

### XML

```
<chart:SfChart3D >
...
<chart:ColumnSeries3D Label="UserProfile" ItemsSource="{Binding UsersList}"
XBindingPath="TimeStamp"
YBindingPath="NoOfUsers" ShowTooltip="True" >
</chart:ColumnSeries3D>
...
</chart:SfChart3D>
```

### C#

```
ColumnSeries3D series = new ColumnSeries3D()
{
    ItemsSource = new UsersViewModel().UsersList,
    XBindingPath = "TimeStamp",
    YBindingPath = "NoOfUsers",
    Label = "UserProfile",
    ShowTooltip = true,
};
```

The following code example gives you the complete code of above configurations.

### XML

```
<Window x:Class="GettingStarted_3DCharts.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:chart="clr-namespace:Syncfusion.UI.Xaml.Charts;assembly=Syncfusion.SfChart.WPF"
xmlns:local="clr-namespace:GettingStarted_3DCharts"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Window.DataContext>
<local:UsersViewModel/>
</Window.DataContext>
<Grid>
<chart:SfChart3D x:Name="Chart3D" Width="500" Height="500" Header="Chart">
<!--PrimaryAxis-->
<chart:SfChart3D.PrimaryAxis>
```

```

<chart:DateTimeAxis3D/>
</chart:SfChart3D.PrimaryAxis>
<!--SecondaryAxis-->
<chart:SfChart3D.SecondaryAxis>
<chart:NumericalAxis3D/>
</chart:SfChart3D.SecondaryAxis>
<!--Legend-->
<chart:SfChart3D.Legend>
<chart:ChartLegend></chart:ChartLegend>
</chart:SfChart3D.Legend>
<chart:ColumnSeries3D Label="UserProfile" ItemsSource="{Binding UsersList}"
XBindingPath="TimeStamp"
YBindingPath="NoOfUsers" ShowTooltip="True">
<!--Adornments-->
<chart:ColumnSeries3D.AdornmentsInfo>
<chart:ChartAdornmentInfo3D ShowLabel="True"></chart:ChartAdornmentInfo3D>
</chart:ColumnSeries3D.AdornmentsInfo>
</chart:ColumnSeries3D>
</chart:SfChart3D>
</Grid>
</Window>

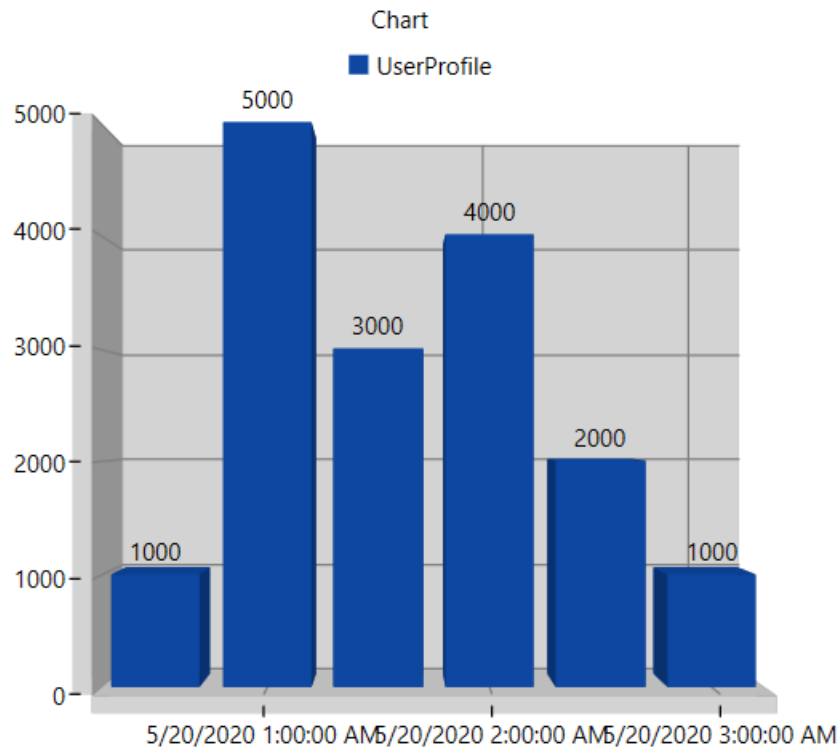
```

## C#

```

SfChart3D chart = new SfChart3D() { Header = "Chart", Height = 500, Width =
500 };
//Adding horizontal axis to the chart
CategoryAxis3D primaryAxis = new CategoryAxis3D();
primaryAxis.Header = "Time";
primaryAxis.FontSize = 14;
chart.PrimaryAxis = primaryAxis;
//Adding vertical axis to the chart
NumericalAxis3D secondaryAxis = new NumericalAxis3D();
secondaryAxis.Header = "Users";
secondaryAxis.FontSize = 14;
chart.SecondaryAxis = secondaryAxis;
//Adding Legends for the chart
ChartLegend legend = new ChartLegend();
chart.Legend = legend;
//Initializing column series
ColumnSeries3D series = new ColumnSeries3D();
series.ItemsSource = new UsersViewModel().UsersList;
series.XBindingPath = "TimeStamp";
series.YBindingPath = "NoOfUsers";
series.Label = "UserProfile";
//Enable Tooltip
series.ShowTooltip = true;
//Setting adornment to the chart series
series.AdornmentsInfo = new ChartAdornmentInfo3D() { ShowLabel = true };
//Adding Series to the Chart Series Collection
chart.Series.Add(series);
this.Content = chart;

```

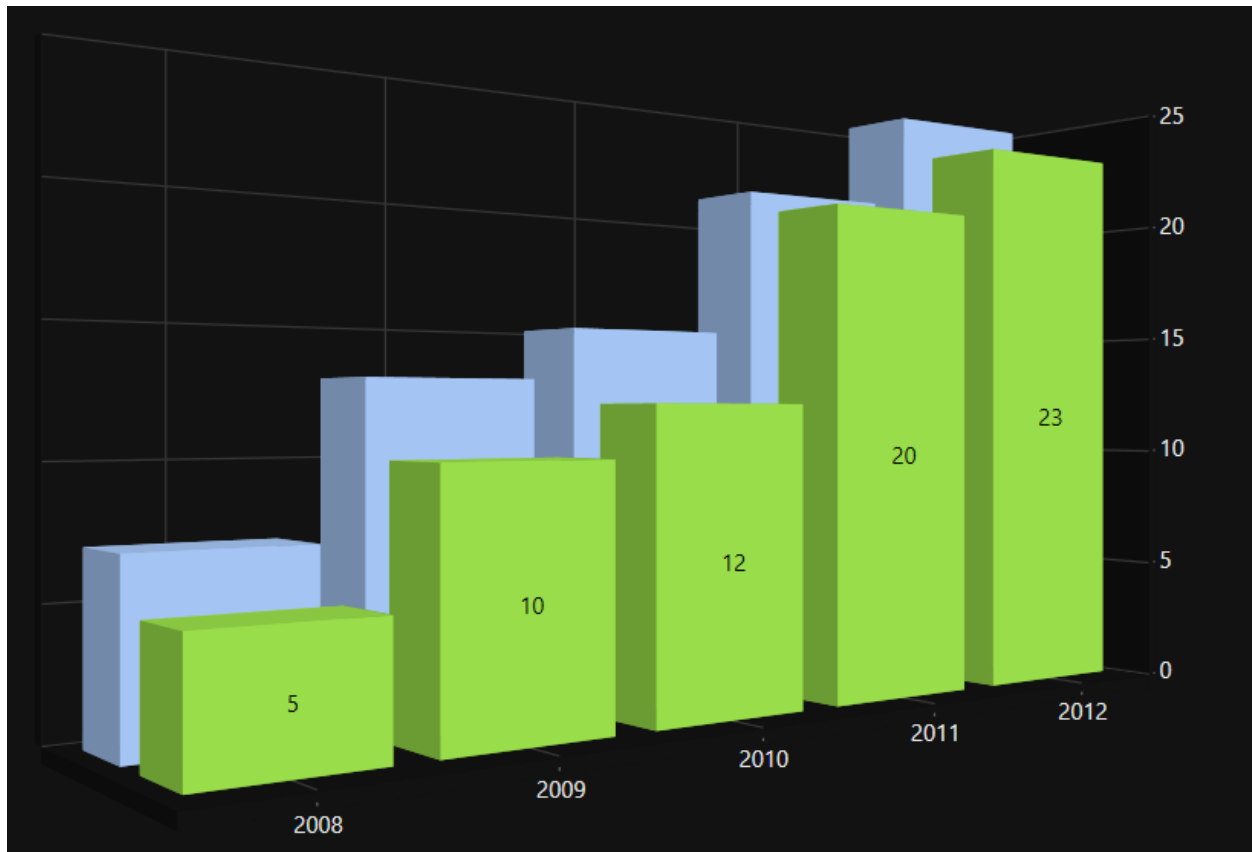


You can find the complete getting started sample : [GettingStarted](#)

#### Theme

SfChart3D supports various built-in themes. Refer to the below links to apply themes for the SfChart3D,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Axis in WPF SfChart3D

[ChartAxis](#) is used to locate a data point inside the chart area. Charts typically have two axes that are used to measure and categorize data: a vertical (Y) axis and a horizontal (X) axis.

- [PrimaryAxis](#) – Gets or sets the horizontal x axis for the chart.
- [SecondaryAxis](#) – Gets or sets the vertical y axis for the chart.

Additionally, SfChart3D have horizontal (z) Axis called Depth Axis.

- [DepthAxis](#) - Gets or sets the horizontal z axis for the chart.

### Types of Axis

ChartAxis supports the following types.

- Numerical Axis
- Category Axis
- DateTime Axis
- TimeSpan Axis
- Logarithmic Axis

### Numerical Axis

[NumericalAxis3D](#) is used to plot numerical values to the chart and can be defined for both [PrimaryAxis](#) and [SecondaryAxis](#).

**XML**

```

<chart:SfChart3D>
  <!--PrimaryAxis-->
  <chart:SfChart3D.PrimaryAxis>
    <chart:NumericalAxis3D/>
  </chart:SfChart3D.PrimaryAxis>
  <!--SecondaryAxis-->
  <chart:SfChart3D.SecondaryAxis>
    <chart:NumericalAxis3D />
  </chart:SfChart3D.SecondaryAxis>
</chart:SfChart3D>

```

**C#**

```

SfChart3D Chart3D = new SfChart3D()
{
    PrimaryAxis = new NumericalAxis3D(),
    SecondaryAxis = new NumericalAxis3D()
};

```

*Category Axis*

[CategoryAxis3D](#) is an indexed based axis that plots values based on the index of the data point collection. The points are equally spaced here.

**XML**

```

<chart:SfChart3D>
  <!--PrimaryAxis-->
  <chart:SfChart3D.PrimaryAxis>
    <chart:CategoryAxis3D/>
  </chart:SfChart3D.PrimaryAxis>
</chart:SfChart3D>

```

**C#**

```

SfChart3D Chart3D = new SfChart3D()
{
    PrimaryAxis = new CategoryAxis3D()
};

```

*DateTime Axis*

[DateTimeAxis3D](#) is used to plot DateTime values and it is widely used to make financial charts in places like the Stock Market, where index plotting is done every day.

**XML**

```

<chart:SfChart3D>
  <!--PrimaryAxis-->
  <chart:SfChart3D.PrimaryAxis>
    <chart:DateTimeAxis3D/>
  </chart:SfChart3D.PrimaryAxis>
</chart:SfChart3D>

```



**C#**

```
SfChart3D Chart3D = new SfChart3D()
{
    PrimaryAxis = new DateTimeAxis3D()
};
```

*Logarithmic Axis*

[LogarithmicAxis3D](#) is used to plot the logarithmic scale for the chart. The Logarithmic values will be plotted based on the logarithmic base value as 10.

**XML**

```
<chart:SfChart3D>
  <!--SecondaryAxis-->
  <chart:SfChart3D.SecondaryAxis>
    <chart:LogarithmicAxis3D />
  </chart:SfChart3D.SecondaryAxis>
</chart:SfChart3D>
```

**C#**

```
SfChart3D Chart3D = new SfChart3D()
{
    SecondaryAxis = new LogarithmicAxis3D()
};
```

*TimeSpan Axis*

[TimeSpanAxis3D](#) is used to plot the time span values in the PrimaryAxis. TimeSpanAxis has the advantage of plotting data with milliseconds difference. The limitation of TimeSpanAxis is that it can only accept timespan values (hh:mm:ss) and date time values are not accepted.

**XML**

```
<chart:SfChart3D>
  <!--PrimaryAxis-->
  <chart:SfChart3D.PrimaryAxis>
    <chart:TimeSpanAxis3D/>
  </chart:SfChart3D.PrimaryAxis>
</chart:SfChart3D>
```

**C#**

```
SfChart3D Chart3D = new SfChart3D()
{
    PrimaryAxis = new TimeSpanAxis3D()
};
```

*Depth Axis*

[DepthAxis](#) helps us to plot chart data based on X, Y and Z Co – ordinates. This feature is supported in [Line](#), [Column](#), [Bar](#), [StackingColumnSeries](#), [StackingBarSeries](#) and [Scatter](#) series.

The depth axis is implemented by defining the required axis type to the [DepthAxis](#) property of the [SfChart3D](#) and by mapping the Z data points to the series using the [ZBindingPath](#) of the series. When [DepthAxis](#) is not defined, by default it is created based on the [ZBindingPath](#) data type.

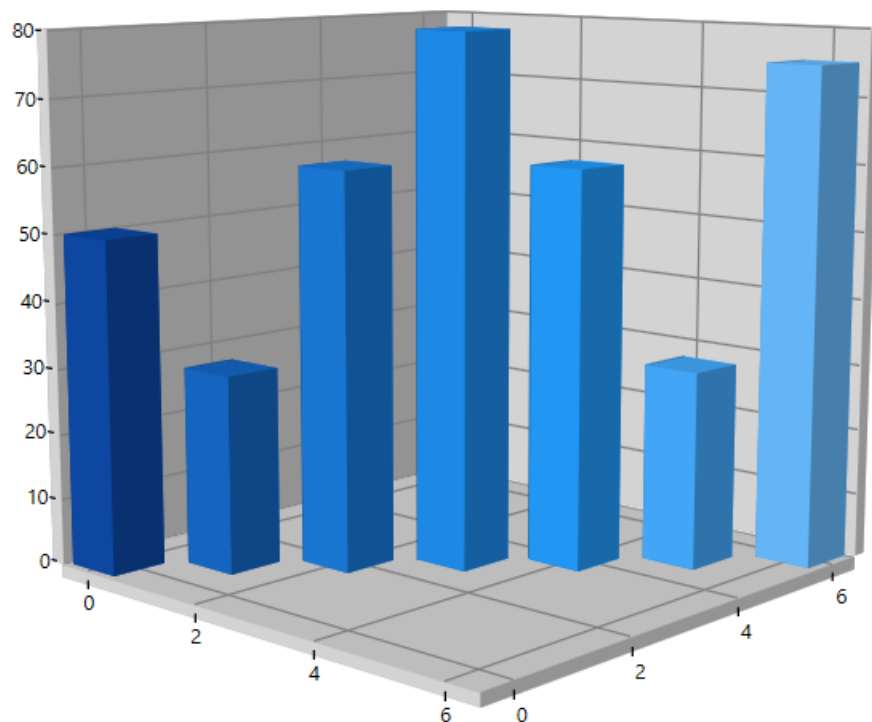
The following code example illustrates how to create Depth Axis.

#### XML

```
<chart:SfChart3D Margin="120,20,120,30" x:Name="chart" Rotation="43"
Tilt="10"
EnableRotation="True" PerspectiveAngle="100">
. . .
<chart:SfChart3D.DepthAxis>
<chart:NumericalAxis3D Interval="1"/>
</chart:SfChart3D.DepthAxis>
<chart:ColumnSeries3D XBindingPath="XValue"
YBindingPath="YValue"
ZBindingPath="ZValue"
ItemsSource="{Binding Data}"/>
</chart:SfChart3D>
```

#### C#

```
SfChart3D chart = new SfChart3D();
chart.Rotation = 43;
chart.Tilt = 10;
chart.Margin = new Thickness(120, 20, 120, 30);
chart.PerspectiveAngle = 100;
chart.EnableRotation = true;
NumericalAxis3D depthAxis = new NumericalAxis3D();
depthAxis.Interval = 1;
chart.DepthAxis = depthAxis;
ColumnSeries3D series1 = new ColumnSeries3D();
series1.ItemsSource = (new ViewModel()).Data;
series1.XBindingPath = "XValue";
series1.YBindingPath = "YValue";
series1.ZBindingPath = "ZValue";
chart.Series.Add(series1);
this.Content = chart;
```



### 3D Manhattan Chart

In this type of chart, multiple series can be plotted in [DepthAxis](#). To enable Manhattan chart add the required number of series and define the [DepthAxis](#). The Manhattan axis is of type category with the axis labels mapped to the [Label](#) property of the series. If the [Label](#) property of the series is not defined, the labels are displayed as Series1, Series2 and so on.

#### XML

```
<chart:SfChart3D x:Name="chart" Rotation="43" >
    . . .
    <chart:SfChart3D.DepthAxis>
    <chart:NumericalAxis3D Interval="1"/>
    </chart:SfChart3D.DepthAxis>
    <chart:LineSeries3D XBindingPath="XValue"
    YBindingPath="YValue"
    ItemsSource="{Binding Data1}"
    Label="First"/>
    <chart:LineSeries3D XBindingPath="XValue"
    YBindingPath="YValue"
    ItemsSource="{Binding Data2}"
    Label="Second"/>
</chart:SfChart3D>
```

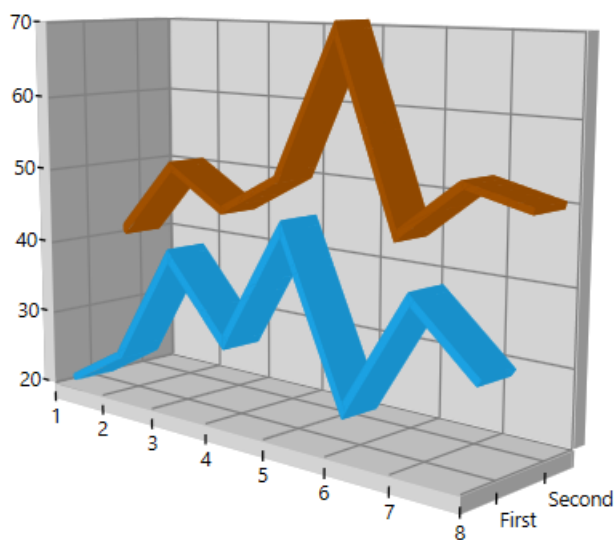
#### C#

```
SfChart3D chart = new SfChart3D() { Header = "Chart", Height = 300, Width =
500 };
chart.Rotation = 43;
. . .
NumericalAxis3D depthAxis = new NumericalAxis3D();
depthAxis.Interval = 1;
```

```

chart.DepthAxis = depthAxis;
LineSeries3D series1 = new LineSeries3D();
series1.ItemsSource = (new ViewModel()).Data1;
series1.XBindingPath = "XValue";
series1.YBindingPath = "YValue";
series1.Label = "First";
LineSeries3D series2 = new LineSeries3D();
series2.ItemsSource = (new ViewModel()).Data2;
series2.XBindingPath = "XValue";
series2.YBindingPath = "YValue";
series2.Label = "Second";
chart.Series.Add(series1);
chart.Series.Add(series2);
this.Content = chart;

```



The sample with Manhattan chart can be downloaded from the [link](#).

### Series in WPF SfChart3D

ChartSeries is the visual representation of the data. SfChart3D offers eight types of series. Based on your requirements and specifications, any type of Series can be added for data visualization.

- Column
- Bar
- Line
- Scatter
- Area
- Stacking column
- Stacking column 100
- Stacking bar
- Stacking bar
- Pie
- Doughnut

The following APIs are common for the most of the series types:

- [XBindingPath](#) – A string property that represents the X values for the series.
- [YBindingPath](#) – A string property that represents the Y values for the series.

Eight types of chart [Series](#).

- [Interior](#) – Represents the brush to fill the series.

### Column Charts

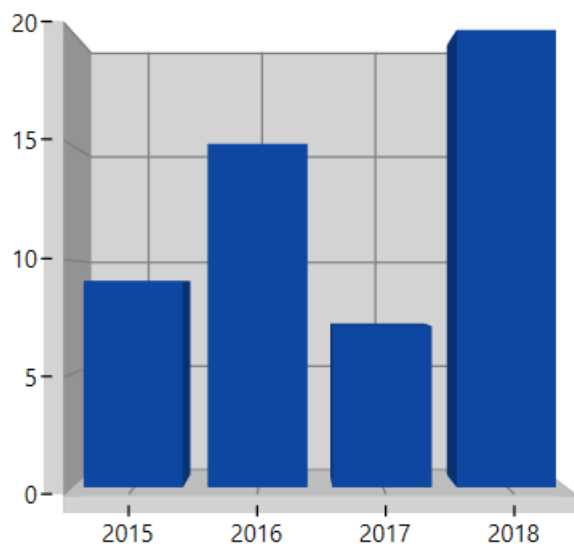
[ColumnSeries3D](#) plots discrete rectangles for the given values.

#### XML

```
<chart:ColumnSeries3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year" YBindingPath="Metal"></chart:ColumnSeries3D>
```

#### C#

```
ColumnSeries3D series = new ColumnSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Metal"
};
chart3D.Series.Add(series);
```



### Bar Charts

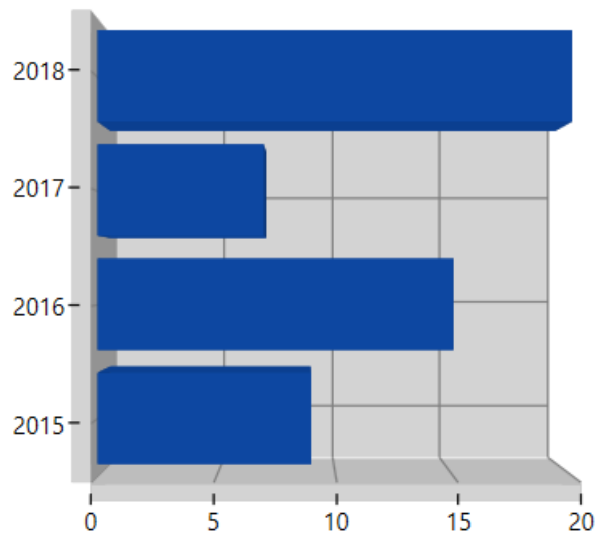
[BarSeries3D](#) are similar to column series, excepts its orientation.

#### XML

```
<chart:BarSeries3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year"
YBindingPath="Plastic"></chart:BarSeries3D>
```

#### C#

```
BarSeries3D series = new BarSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Plastic"
};
chart3D.Series.Add(series);
```



### Spacing

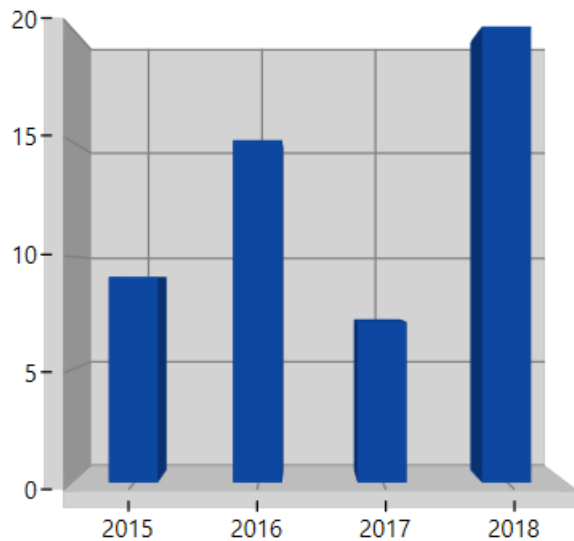
[Spacing](#) property of series is used to decide the width of a segment. [Spacing](#) value ranges from 0 to 1. The following code illustrates how to set [Spacing](#) property of the series,

### XML

```
<chart:ColumnSeries3D chart:ChartSeriesBase.Spacing="0.6"
    ItemsSource="{Binding CategoricalData}"
    XBindingPath="Year" YBindingPath="Plastic"></chart:ColumnSeries3D>
```

### C#

```
ColumnSeries3D series = new ColumnSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Plastic"
};
ChartSeriesBase.SetSpacing(series, 0.6);
chart3D.Series.Add(series);
```



### Segment Spacing

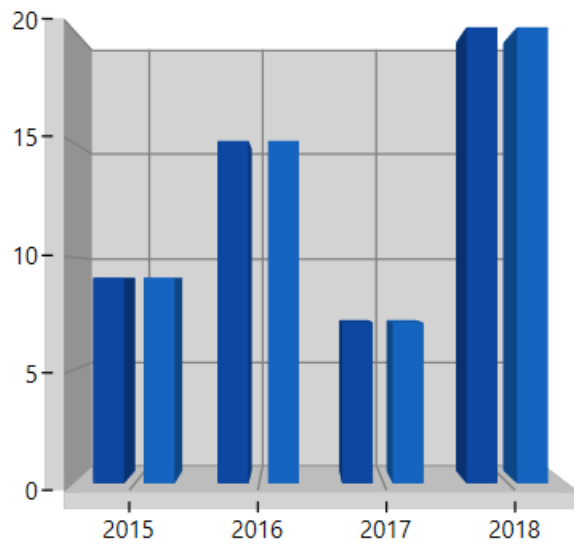
[SegmentSpacing](#) property is used to set the spacing among the segments, when multiple series are added in chart. Its value ranges from 0 to 1. The following code illustrates how to use the [SegmentSpacing](#) property in series,

#### XML

```
<chart:ColumnSeries3D SegmentSpacing="0.6" ItemsSource="{Binding
CategoricalData}"
XBindingPath="Year" YBindingPath="Plastic"></chart:ColumnSeries3D>
<chart:ColumnSeries3D SegmentSpacing="0.6" ItemsSource="{Binding
CategoricalData}"
XBindingPath="Year" YBindingPath="Iron"></chart:ColumnSeries3D>
```

#### C#

```
ColumnSeries3D series = new ColumnSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Plastic",
    SegmentSpacing = 0.6,
};
ColumnSeries3D series1 = new ColumnSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Iron",
    SegmentSpacing = 0.6,
};
chart3D.Series.Add(series);
chart3D.Series.Add(series1);
```



### Line Charts

[LineSeries3D](#) join points on a plot by straight lines, showing data trends at equal intervals.

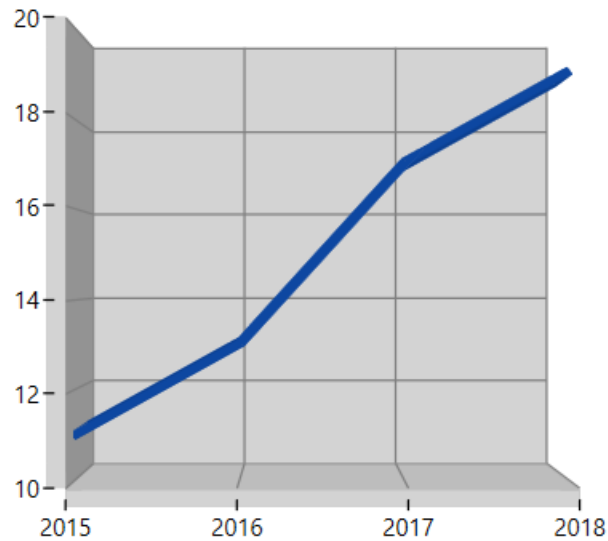
### XML

```
<chart:LineSeries3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year"
YBindingPath="Metal"></chart:LineSeries3D>
```

### C#

```
LineSeries3D line = new LineSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Metal"
};
chart3D.Series.Add(series);
```





### Scatter Chart

[ScatterSeries3D](#) will represent each point by a Rectangle with equal size.

This size can be defined by using below properties.

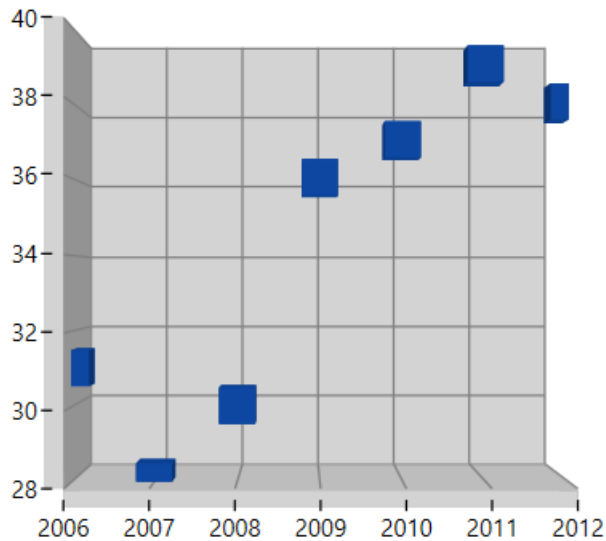
- [ScatterHeight](#)
- [ScatterWidth](#)

### XML

```
<chart:ScatterSeries3D ItemsSource="{Binding DataPoints}"
XBindingPath="Year" YBindingPath="Germany"></chart:ScatterSeries3D>
```

### C#

```
ScatterSeries3D series = new ScatterSeries3D()
{
    ItemsSource = new ViewModel().DataPoints,
    XBindingPath = "Year",
    YBindingPath = "Germany"
};
chart3D.Series.Add(series);
```



### Area Chart

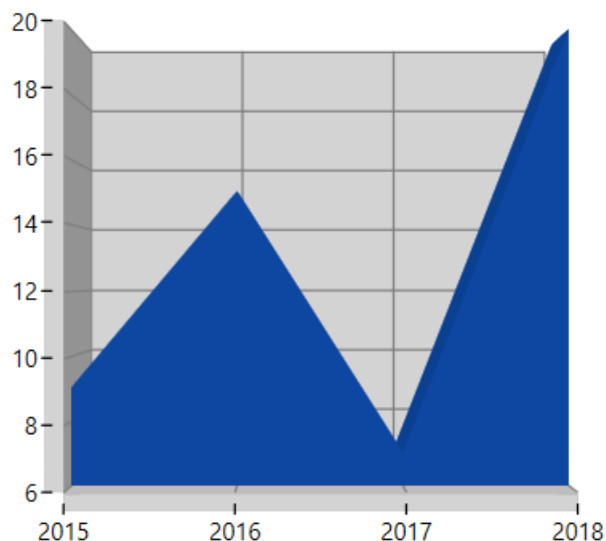
[AreaSeries3D](#) is rendered using a collection of line segments connected to form a closed loop area, filled with the specified color.

### XML

```
<chart:AreaSeries3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year"
YBindingPath="Plastic"></chart:AreaSeries3D>
```

### C#

```
AreaSeries3D series = new AreaSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Plastic"
};
chart3D.Series.Add(series);
```



## Stacking Charts

### Stacking Column

[StackingColumnSeries3D](#) resembles multiple types of [ColumnSeries3D](#). Each series is vertically stacked one above the other. When there is only one series, then it is [ColumnSeries](#).

The following code example illustrates how to use [StackingColumnSeries3D](#):

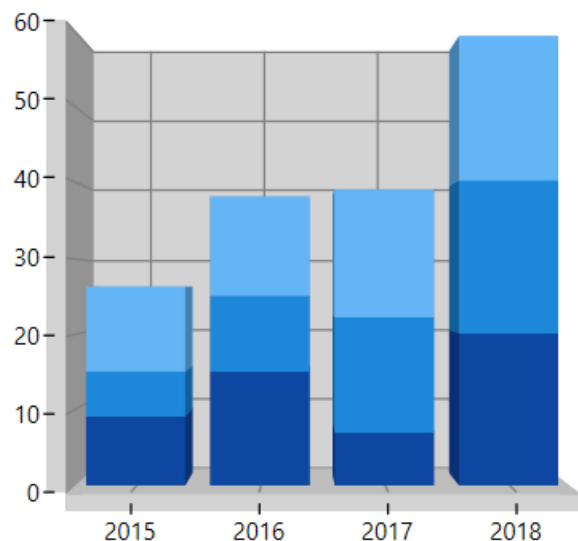
### XML

```
<chart:StackingColumnSeries3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year"
YBindingPath="Plastic"></chart:StackingColumnSeries3D>
<chart:StackingColumnSeries3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year"
YBindingPath="Iron"></chart:StackingColumnSeries3D>
<chart:StackingColumnSeries3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year"
YBindingPath="Metal"></chart:StackingColumnSeries3D>
```

### C#

```
StackingColumnSeries3D stack1 = new StackingColumnSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Plastic"
};
StackingColumnSeries3D stack2 = new StackingColumnSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Iron"
};
StackingColumnSeries3D stack3 = new StackingColumnSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Metal"
};
```

```
};
chart3D.Series.Add(stack1);
chart3D.Series.Add(stack2);
chart3D.Series.Add(stack3);
```



#### Stacking Column 100

[StackingColumn100Series3D](#) resembles [StackingColumnSeries3D](#) but the cumulative portion of each stacked element always comes to a total of 100%.

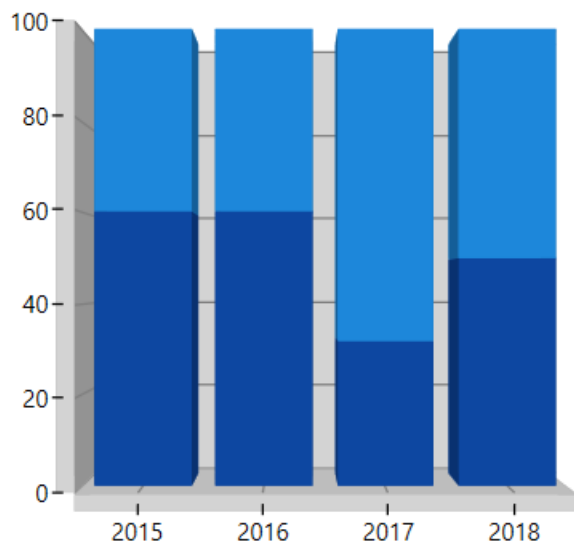
The following code example illustrates how to use [StackingColumn100Series3D](#):

#### XML

```
<chart:StackingColumn100Series3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year"
YBindingPath="Plastic"></chart:StackingColumn100Series3D>
<chart:StackingColumn100Series3D Interior="Brown" ItemsSource="{Binding
CategoricalData}" XBindingPath="Year"
YBindingPath="Iron"></chart:StackingColumn100Series3D>
```

#### C#

```
StackingColumn100Series3D stack1 = new StackingColumn100Series3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Plastic"
};
StackingColumn100Series3D stack2 = new StackingColumn100Series3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Iron",
    Interior=new SolidColorBrush(Colors.Brown)
};
chart3D.Series.Add(stack1);
chart3D.Series.Add(stack2);
```



### Stacking Bar

[StackingBarSeries3D](#) is a multiple series type of [BarSeries3D](#). Each [BarSeries3D](#) is then stacked horizontally, side by side to each other. When there exists only one series, it resembles a simple [BarSeries3D](#).

The following code example illustrates how to use [StackingBarSeries3D](#):

### XML

```
<chart:StackingBarSeries3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year"
YBindingPath="Plastic"></chart:StackingBarSeries3D>
<chart:StackingBarSeries3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year"
YBindingPath="Iron"></chart:StackingBarSeries3D>
<chart:StackingBarSeries3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year"
YBindingPath="Metal"></chart:StackingBarSeries3D>
```

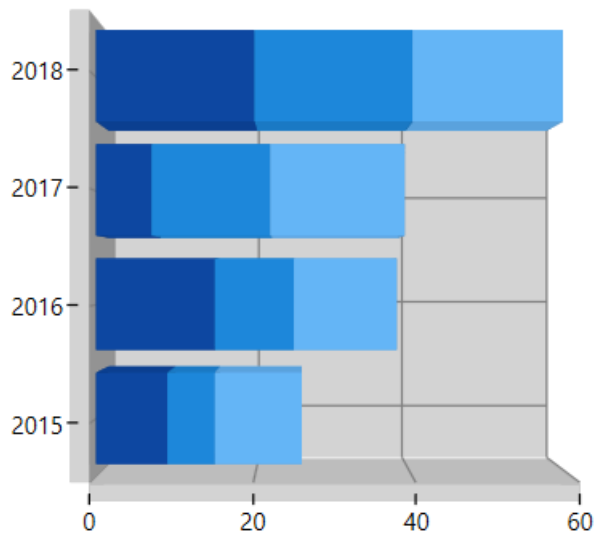
### C#

```
StackingBarSeries3D stack1 = new StackingBarSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Plastic"
};
StackingBarSeries3D stack2 = new StackingBarSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Iron"
};
StackingBarSeries3D stack3 = new StackingBarSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
```

```

XBindingPath = "Year",
YBindingPath = "Metal"
};
chart3D.Series.Add(stack1);
chart3D.Series.Add(stack2);
chart3D.Series.Add(stack3);

```



#### Stacking Bar 100

[StackingBar100Series3D](#) resembles a [StackingBarSeries3D](#). [StackingBar100Series3D](#) displays multiple series as stacked bars and the cumulative portion of each stacked element is always 100%.

The following code example illustrates how to use [StackingBar100Series3D](#):

#### XML

```

<chart:StackingBar100Series3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year"
YBindingPath="Plastic"></chart:StackingBar100Series3D>
<chart:StackingBar100Series3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year"
YBindingPath="Iron"></chart:StackingBar100Series3D>

```

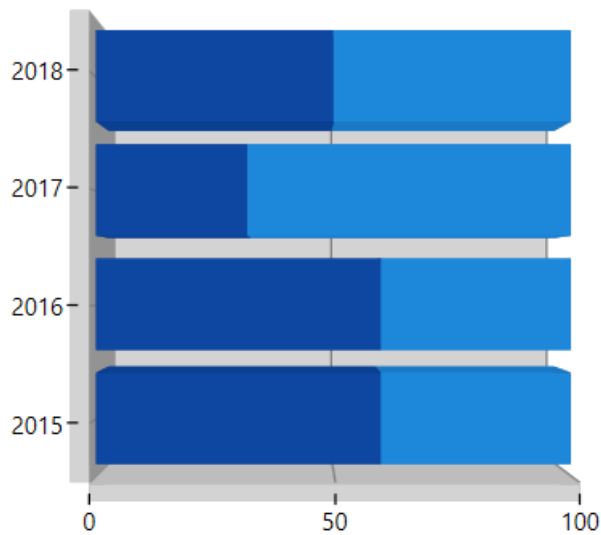
#### C#

```

StackingBar100Series3D stack1 = new StackingBar100Series3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Plastic"
};
StackingBar100Series3D stack2 = new StackingBar100Series3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Iron"
};
chart3D.Series.Add(stack1);

```

```
chart3D.Series.Add(stack2);
```



### Pie Chart

[PieSeries3D](#) is divided into sectors, illustrating numerical proportion.

The following code example illustrates the [PieSeries3D](#).

### XML

```
<chart:PieSeries3D ItemsSource="{Binding CategoricalData}"
  XBindingPath="Year"
  YBindingPath="Iron"></chart:PieSeries3D>
```

### C#

```
PieSeries3D series = new PieSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Iron"
};
chart3D.Series.Add(series);
```



## Doughnut Chart

[DoughnutSeries3D](#) is similar to [PieSeries](#). It is used to show the relationship between parts of data and whole data.

The [DoughnutSeries3D](#) can be added to chart as in below code example:

### XML

```
<chart:DoughnutSeries3D ItemsSource="{Binding CategoricalData}"
XBindingPath="Year" YBindingPath="Iron">
</chart:DoughnutSeries3D>
```

### C#

```
DoughnutSeries3D series = new DoughnutSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Iron"
};
chart.Series.Add(series);
```



### Coefficient

The [DoughnutCoefficient](#) and [CircleCoefficient](#) property to define the inner circle of Doughnut and Pie Charts.

### XML

```
<chart:DoughnutSeries3D ItemsSource="{Binding CategoricalData}"
DoughnutCoefficient="0.5"
XBindingPath="Year" YBindingPath="Iron"> </chart:DoughnutSeries3D>
```

### C#

```
DoughnutSeries3D series = new DoughnutSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Iron",
    DoughnutCoefficient=0.5
};
```



```
chart.Series.Add(series);
```



### *Semi Pie and Doughnut*

By using custom [StartAngle](#) and [EndAngle](#) properties, you can draw pie series in different shapes such as semi or quarter circular series.

### XML

```
<chart:DoughnutSeries3D StartAngle="180" EndAngle="360"
ItemsSource="{Binding CategoricalData}"
DoughnutCoefficient="0.7" XBindingPath="Year" YBindingPath="Iron">
</chart:DoughnutSeries3D>
```

### C#

```
DoughnutSeries3D series = new DoughnutSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Iron",
    DoughnutCoefficient=0.7,
    StartAngle=180,
    EndAngle=360
};
chart.Series.Add(series);
```



### **Pie**

### XML

```
<chart:PieSeries3D StartAngle="180" EndAngle="360" ItemsSource="{Binding
CategoricalData}"
CircleCoefficient="0.7" XBindingPath="Year" YBindingPath="Iron">
</chart:DoughnutSeries3D>
```

## C#

```
PieSeries3D series = new PieSeries3D()
{
    ItemsSource = new CategoryDataViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Iron",
    CircleCoefficient=0.7,
    StartAngle=180,
    EndAngle=360
};
chart.Series.Add(series);
```



## Dynamic explode

This feature allows users to explode a particular segment in a circular series using [ExplodeOnClick](#). This can also be achieved by setting the [ExplodeIndex](#) or [ExplodeAll](#) property.

The following code example illustrates how to enable dynamic explode for circular series, for data please refer series category in 3D charts.

## XML

```
<chart:SfChart3D EnableRotation="True" Tilt="-30" Rotation="45"
Depth="30" PerspectiveAngle="90" Width="500" Height="500">
<!--PrimaryAxis-->
<chart:SfChart3D.PrimaryAxis>
<chart:CategoryAxis3D Header="Year"/>
</chart:SfChart3D.PrimaryAxis>
<!--SecondaryAxis-->
<chart:SfChart3D.SecondaryAxis>
<chart:NumericalAxis3D Header="Metal"/>
</chart:SfChart3D.SecondaryAxis>
<!--PieSeries3D - Dynamic explode-->
<chart:PieSeries3D ExplodeOnClick="True" ItemsSource="{Binding
CategoricalData}" XBindingPath="Year"
YBindingPath="Iron"></chart:PieSeries3D>
</chart:SfChart3D>
```



### Data Markers in WPF SfChart3D

Adornments (Data Markers) are used to provide information about the data points to the user. Values from data point(x, y) or other custom properties from a data source can be displayed. You can add a shape and label to adorn each data point.

Each adornment can be represented by the following:

- Marker- Displays the desired symbol at the (X, Y) point.
- Label - Displays the segment label content at the (X, Y) point.
- ConnectorLine - Line used to connect the (X, Y) point and the label element.

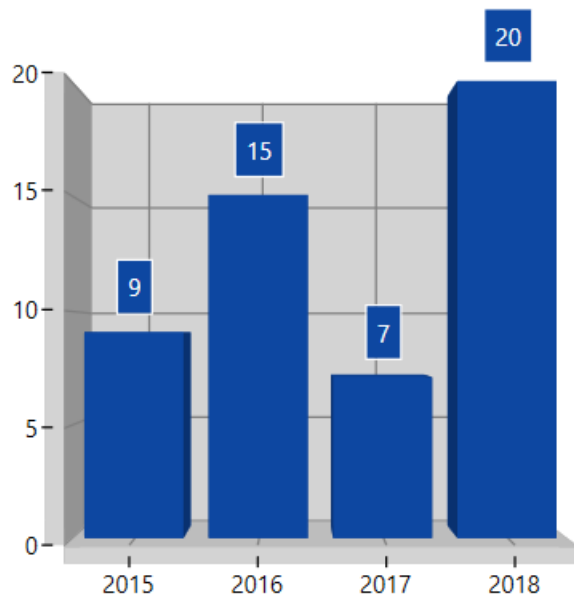
The following code example illustrates how to initialize the adornment.

#### XML

```
<chart:ColumnSeries3D ItemsSource="{Binding CategoricalDatas}"
XBindingPath="Year" YBindingPath="Plastic">
<!--AdornmentsInfo-->
<chart:ColumnSeries3D.AdornmentsInfo>
<chart:ChartAdornmentInfo3D UseSeriesPalette="True" BorderBrush="White"
BorderThickness="1" ShowLabel="True" ></chart:ChartAdornmentInfo3D>
</chart:ColumnSeries3D.AdornmentsInfo>
</chart:ColumnSeries3D>
```

#### C#

```
ColumnSeries3D series = new ColumnSeries3D()
{
    ItemsSource = new ViewModel().CategoricalData,
    XBindingPath = "Year",
    YBindingPath = "Plastic"
};
ChartAdornmentInfo3D adornmentInfo = new ChartAdornmentInfo3D() { ShowLabel
= true };
series.AdornmentsInfo = adornmentInfo;
chart.Series.Add(series);
```



### Interactive Features in WPF SfChart3D

3D charts provide interactive features such as dynamic rotation, segment selection, and dynamic segment explode for circular series.

#### Dynamic rotation

3D charts allow us to view the best possible view of data dynamically using a mouse or touch device. To enable dynamic rotation, set the [EnableRotation](#) property to true.

The following code example illustrates how to enable the dynamic rotation:

#### XML

```
<Syncfusion:SfChart3D EnableRotation="True" x:Name="Chart" Height="500"
Width="600"/>
```

#### Segment Selection

To enable segment selection in a 3D chart, set the SegmentSelectionBrush property in chart series.

The following code example illustrates how to set the selection brush for individual series. For data refer to the Series category in 3D charts.

#### XML

```
<chart:SfChart3D EnableRotation="True" PerspectiveAngle="50"
Rotation="29" Depth="100" Palette="BlueChrome" Width="300" Height="280">
<chart:ColumnSeries3D SegmentSelectionBrush="SkyBlue" ItemsSource="{Binding
DataPoints}"
XBindingPath="Year" YBindingPath="India">
</chart:ColumnSeries3D>
</chart:SfChart3D>
```

#### C#

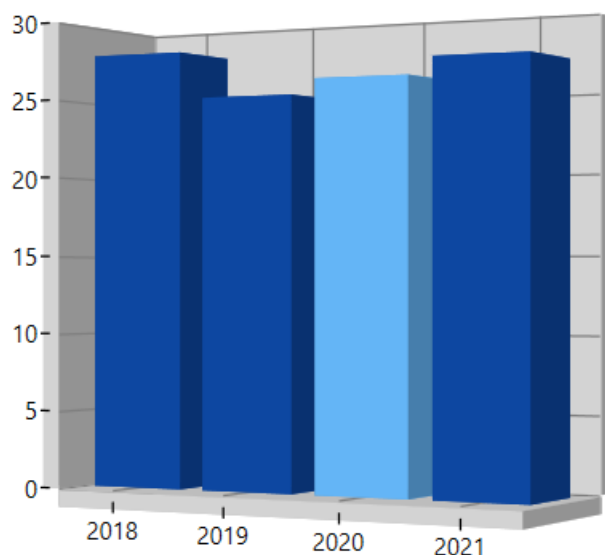
```
SfChart3D chart3D = new SfChart3D()
{
```

```

EnableSeriesSelection=true,
EnableRotation=true,
PerspectiveAngle=50,
SeriesSelectedIndex=0,
Rotation=29,
Depth=100,
Palette=ChartColorPalette.BlueChrome,
Width=300,
Height=280
};
ColumnSeries3D series = new ColumnSeries3D()
{
    ItemsSource = new StockViewModel().DataPoints,
    XBindingPath = "Year",
    YBindingPath = "India",
    SegmentSelectionBrush = new SolidColorBrush(Colors.SkyBlue),
    SegmentSpacing = 0.5
};
chart3D.Series.Add(series);

```

The following screenshot illustrates the result of the above code example.



### Series Selection

Series selection support is used to highlight the series programmatically or by user interaction. Also you can get a series SelectedIndex, PreviousSelectedIndex value in SelectionChanged event arguments.

The following code example can be used to set series selection in a SfChart3D.

### XML

```

<chart:SfChart3D EnableRotation="True" PerspectiveAngle="50"
EnableSeriesSelection="True" SeriesSelectedIndex="0"
Rotation="29" Depth="100" Palette="BlueChrome" Width="300" Height="280">
  <chart:ColumnSeries3D SeriesSelectionBrush="LightGreen" SegmentSpacing="0.5"
ItemsSource="{Binding Demands}" XBindingPath="Category"
YBindingPath="Value">
  </chart:ColumnSeries3D>

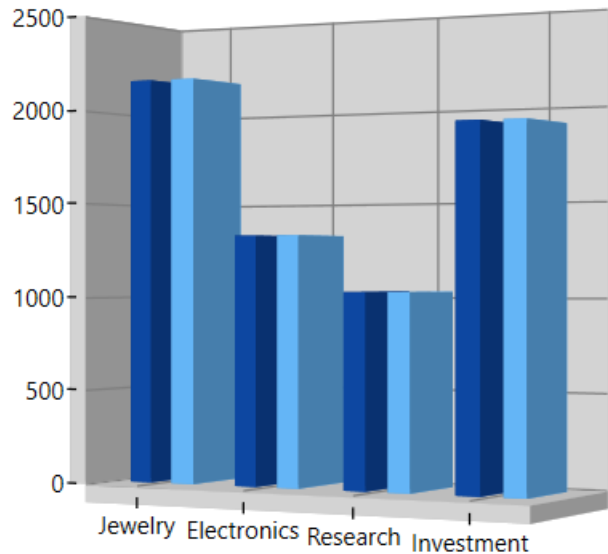
```

```
<chart:ColumnSeries3D SegmentSpacing="0.5" SeriesSelectionBrush="SkyBlue"
ItemsSource="{Binding Demands}" XBindingPath="Category"
YBindingPath="Value">
</chart:ColumnSeries3D>
</chart:SfChart3D>
```

## C#

```
SfChart3D chart3D = new SfChart3D()
{
    EnableSeriesSelection=true,
    EnableRotation=true,
    PerspectiveAngle=50,
    SeriesSelectedIndex=0,
    Rotation=29,
    Depth=100,
    Palette=ChartColorPalette.BlueChrome,
    Width=300,
    Height=280
};
ColumnSeries3D series = new ColumnSeries3D()
{
    ItemsSource = new StockViewModel().Demands,
    XBindingPath = "Category",
    YBindingPath = "Value",
    SeriesSelectionBrush = new SolidColorBrush(Colors.LightGreen),
    SegmentSpacing = 0.5
};
ColumnSeries3D series1 = new ColumnSeries3D()
{
    ItemsSource = new StockViewModel().Demands,
    XBindingPath = "Category",
    YBindingPath = "Value",
    SeriesSelectionBrush = new SolidColorBrush(Colors.SkyBlue),
    SegmentSpacing = 0.5
};
chart3D.Series.Add(series);
chart3D.Series.Add(series1);
```

The following screenshot is an example of a SfChart3D with series selection.



### Appearance in WPF SfChart3D

SfChart3D supports various customizing and styling options that allow you to enrich the application.

#### Palettes

SfChart3D provides the options to apply the different kinds of themes or palettes to your chart. You can define the [Palette](#) either for the entire chart or for an individual series.

We have some predefined palette such as,

- Metro
- AutumnBrights
- FloraHues
- Pineapple
- TomotoSpectrum
- RedChrome
- PurpleChrome
- BlueChrome
- GreenChrome
- Elite
- LightCandy
- SandyBeach

#### Applying Palette to Series

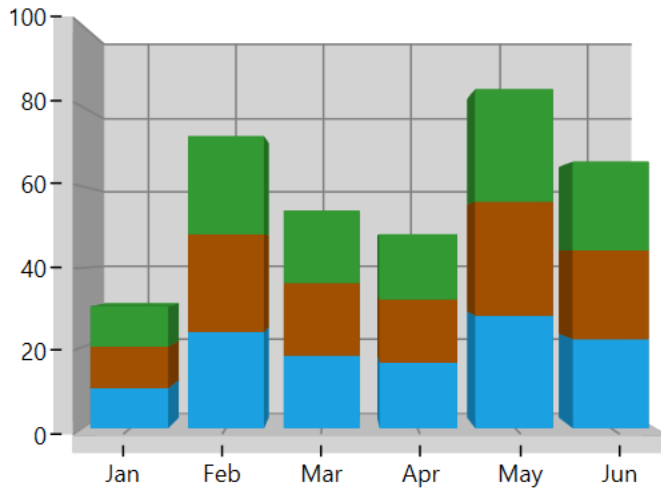
Each palette applies a set of predefined brushes to the series in a predefined order. The following code example shows you how to set the Metro Palette for the chart series.

#### XML

```
<chart:SfChart3D Height="250" Width="350" Palette="Metro" >
```

#### C#

```
chart.Palette = ChartColorPalette.Metro;
```



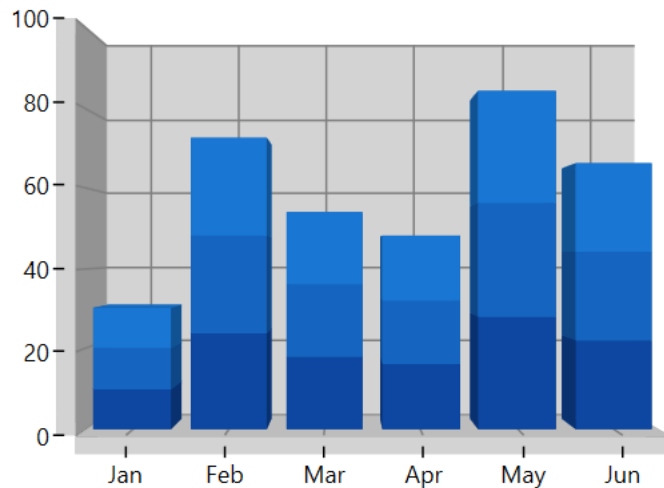
The following code example defined the palette as a [BlueChrome](#).

#### XML

```
<chart:SfChart3D Height="250" Width="350" Palette="BlueChrome">
```

#### C#

```
chart.Palette = ChartColorPalette.BlueChrome;
```



#### Applying Palette to Segment

Each palette applies a set of predefined brushes to the segment in a predefined order. The following code example shows you how to set the Metro Palette for the chart series.

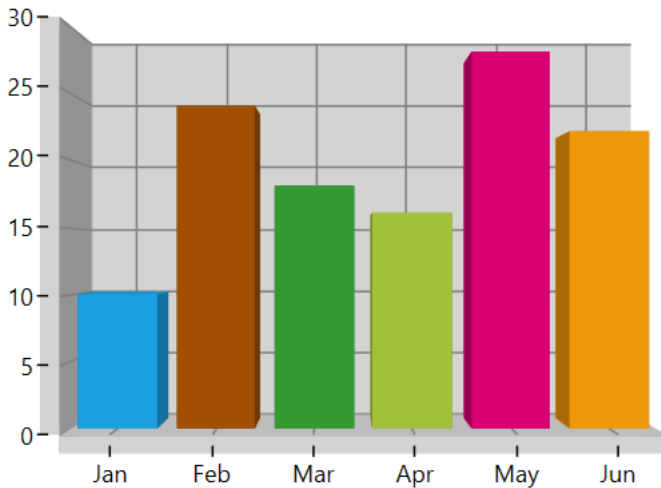
#### XML

```
<syncfusion:ColumnSeries3D ItemsSource="{Binding Data}"
XBindingPath="XValue"
YBindingPath="YValue"
Palette="Metro"/>
```



**C#**

```
ColumnSeries3D columnSeries = new ColumnSeries3D()  
{  
    ItemsSource = viewModel.Data,  
    XBindingPath = "XValue",  
    YBindingPath = "YValue",  
    Palette = ChartColorPalette.Metro  
};  
chart.Series.Add(columnSeries);
```



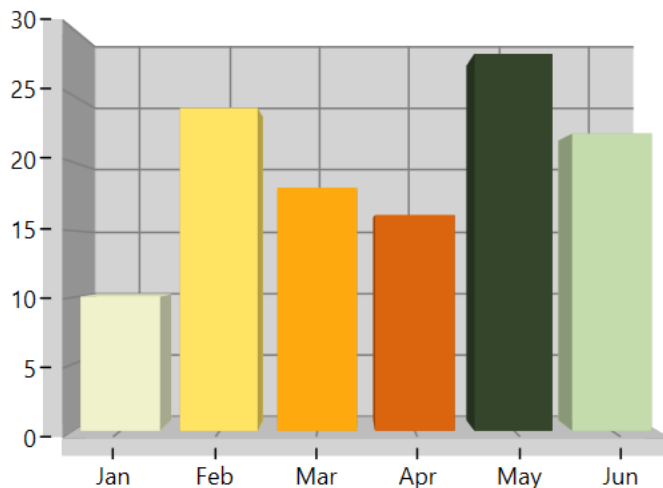
The following code example defined the palette as an **AutumnBrights**.

**XML**

```
<syncfusion:ColumnSeries3D ItemsSource="{Binding Data}"  
    XBindingPath="XValue"  
    YBindingPath="YValue"  
    Palette="AutumnBrights"/>
```

**C#**

```
ColumnSeries3D columnSeries = new ColumnSeries3D()  
{  
    ItemsSource = viewModel.Data,  
    XBindingPath = "XValue",  
    YBindingPath = "YValue",  
    Palette = ChartColorPalette.AutumnBrights  
};  
chart.Series.Add(series);
```



**Note:** Metro palette is the default palette for both Series and Segment.

### Custom Palette

SfChart3D provides an option that enables you to define your own color brushes with your preferred order for the Palette using the [ColorModel](#) as shown in the following code example.

### XML

```
<chart:DoughnutSeries3D YBindingPath="Percentage" Palette="Custom"
XBindingPath="Category" ItemsSource="{Binding Tax}" >
  <chart:DoughnutSeries3D.ColorModel>
    <chart:ChartColorModel>
      <chart:ChartColorModel.CustomBrushes>
        <SolidColorBrush Color="Cyan"/>
        <SolidColorBrush Color="DarkCyan"/>
      </chart:ChartColorModel.CustomBrushes>
    </chart:ChartColorModel>
  </chart:DoughnutSeries3D.ColorModel>
</chart:DoughnutSeries3D>
```

### C#

```
ChartColorModel colorModel = new ChartColorModel();
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.Cyan));
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.DarkCyan));
DoughnutSeries3D series = new DoughnutSeries3D()
{
    ItemsSource = new ViewModel().Tax,
    XBindingPath = "Category",
    YBindingPath = "Percentage",
    Palette = ChartColorPalette.Custom,
    ColorModel = colorModel
};
chart.Series.Add(series);
```



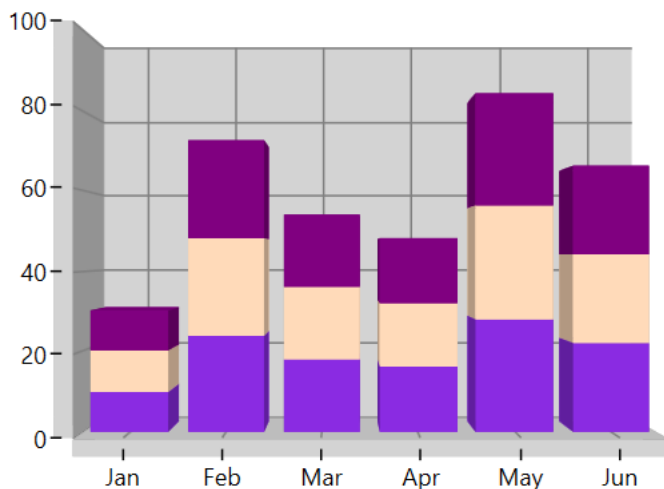
You can define the custom palette for the series as shown in the following code example:

#### XML

```
<chart:SfChart3D Height="250" Width="350" Palette="Custom">
  <chart:SfChart3D.ColorModel>
    <chart:ChartColorModel>
      <chart:ChartColorModel.CustomBrushes>
        <SolidColorBrush Color="BlueViolet"/>
        <SolidColorBrush Color="PeachPuff"/>
        <SolidColorBrush Color="Purple"/>
      </chart:ChartColorModel.CustomBrushes>
    </chart:ChartColorModel>
  </chart:SfChart3D.ColorModel>
</chart:SfChart3D>
```

#### C#

```
chart.Palette = ChartColorPalette.Custom;
ChartColorModel colorModel = new ChartColorModel();
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.BlueViolet));
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.PeachPuff));
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.Purple));
chart.ColorModel = colorModel;
```



### SegmentColorPath

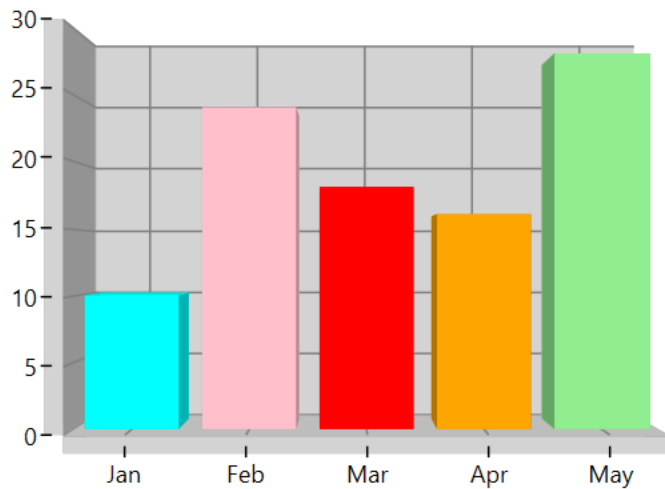
The color for the chart segments can be bound from its items source collection by using the [SegmentColorPath](#) property of series. The following code shows how to bind the color to the series with the [SegmentColorPath](#) property.

### XML

```
<chart:ColumnSeries3D ItemsSource="{Binding Data}" XBindingPath="XValue"
YBindingPath="YValue" SegmentColorPath="ColorPath">
</chart:ColumnSeries3D>
```

### C#

```
ColumnSeries3D series = new ColumnSeries3D()
{
    ItemsSource = viewModel.Data,
    XBindingPath = "XValue",
    YBindingPath = "YValue",
    SegmentColorPath = "ColorPath"
};
Data = new ObservableCollection<Model>();
Data.Add(new Model() { XValue = "Jan", YValue = 10, ColorPath = new
SolidColorBrush(Colors.Cyan) });
Data.Add(new Model() { XValue = "Feb", YValue = 24, ColorPath = new
SolidColorBrush(Colors.Pink) });
Data.Add(new Model() { XValue = "Mar", YValue = 18, ColorPath = new
SolidColorBrush(Colors.Red) });
Data.Add(new Model() { XValue = "Apr", YValue = 16, ColorPath = new
SolidColorBrush(Colors.Orange) });
Data.Add(new Model() { XValue = "May", YValue = 28, ColorPath = new
SolidColorBrush(Colors.LightGreen) });
```



---

**Note:** The SegmentColorPath property is not applicable to the Area and CircularSeries.

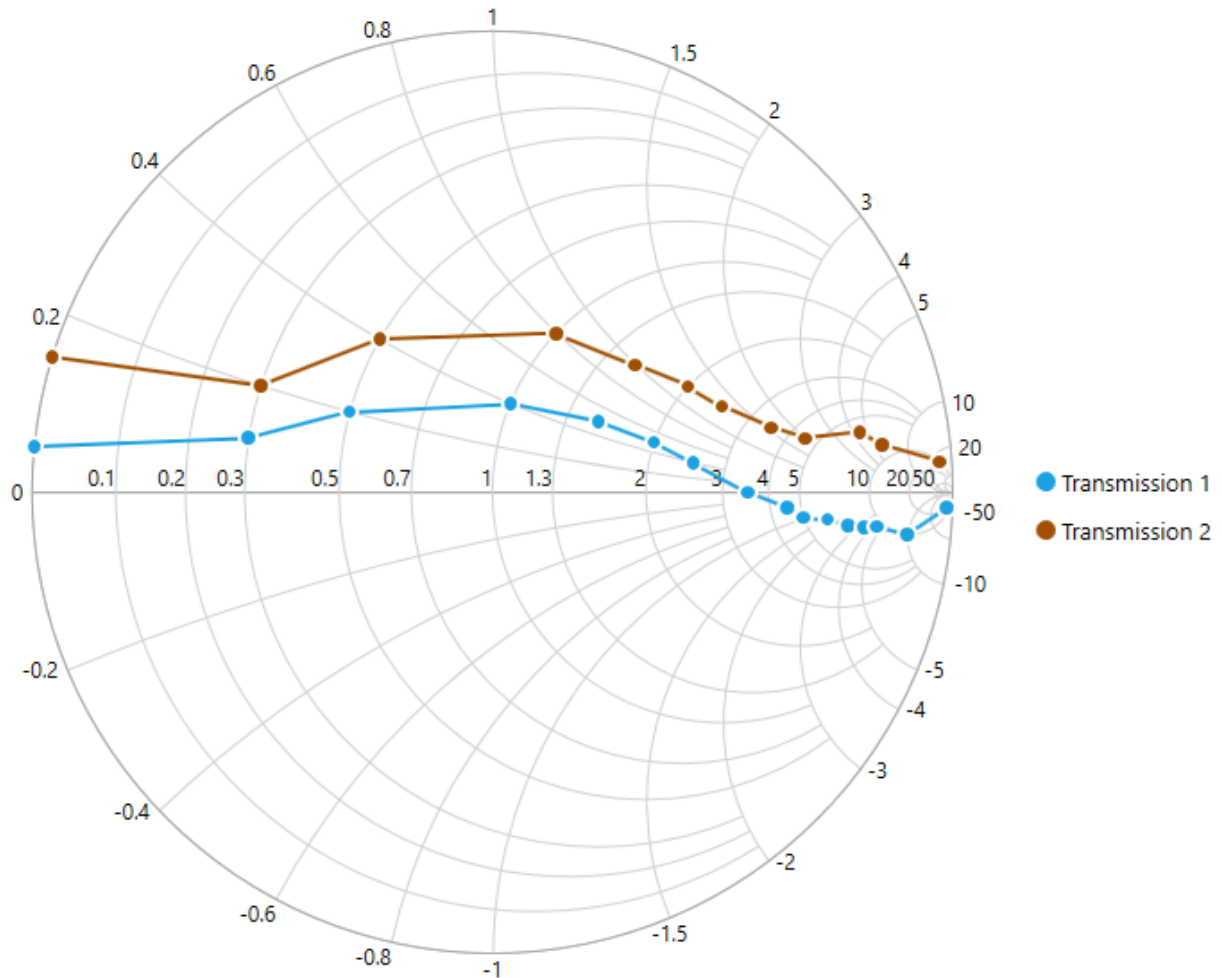
---

## SfSmithChart

### [WPF Smith Chart \(SfSmithChart\) Overview](#)

Smith chart is one of the most useful data visualization tools for high frequency circuit applications. It contains two sets of circles to plot the parameters of transmission lines.

SfSmithChart provides a perfect way to visualize data with high level of user interactivity that focuses on development, productivity, and simplicity of use.



### Key features

- Visualization of the impedance and admittance of a transmission line.
- Representation of data with line series and various types of markers.
- Data label support for better readability.
- Interactive tooltip support.
- Interactive legend.
- Customizable colors.
- Allows to map the data from the specified path by achieving [data binding]() concept.

### Getting Started with WPF Smith Chart (SfSmithChart)

This section explains the steps required to build the application with SfSmithChart.

#### Steps

1. Create new WPF project using Visual Studio. For more [details](#).
2. Add the SfSmithChart assembly to your application.
3. Initialize smith chart control.
4. Add header to the smith chart control.

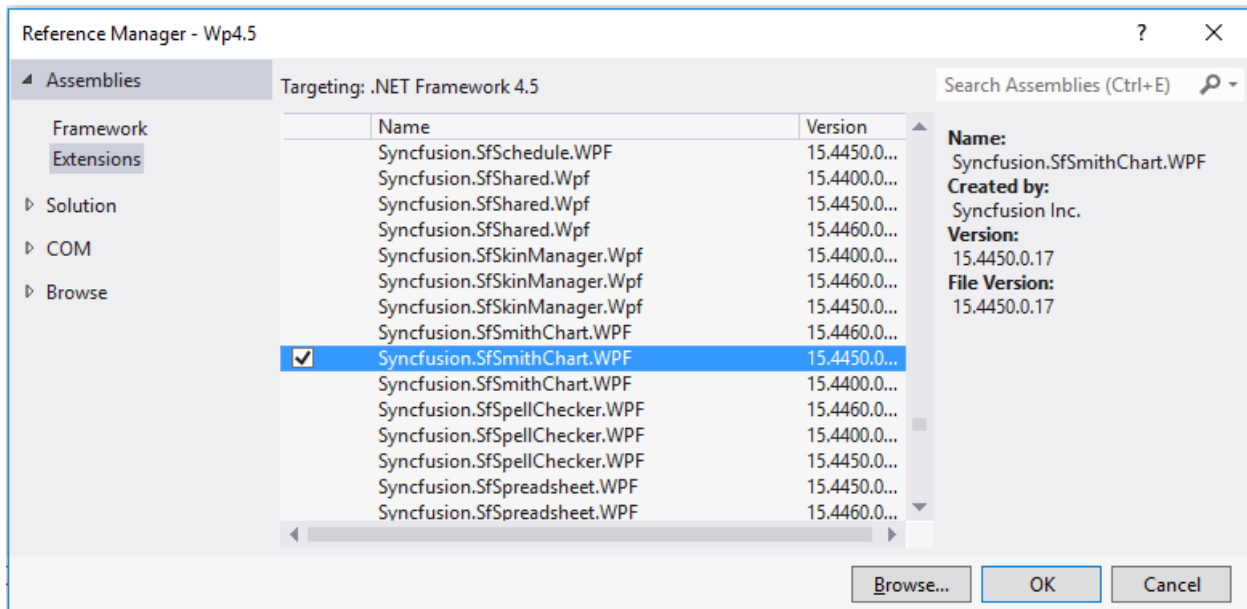
5. Add series to the smith chart control.
6. Add legends for the series.

These steps are explained below in both XAML and code behind.

Create a simple smith chart from XAML

*Adding assembly reference*

1. Open the Add Reference window in your project.
2. Choose Windows > Extensions > Syncfusion.SfSmithChart.WPF.



3. Select the .NET Framework version with respect to your application. The version can be identified as below:

XX.X450.0.X	4.5 Framework
XX.X451.0.X	4.5.1 Framework
XX.X460.0.X	4.6 Framework

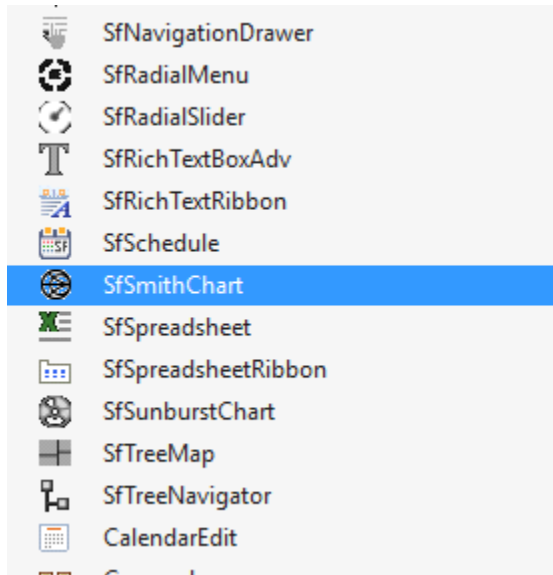
4. Add the following namespace in your XAML window.

#### **XML**

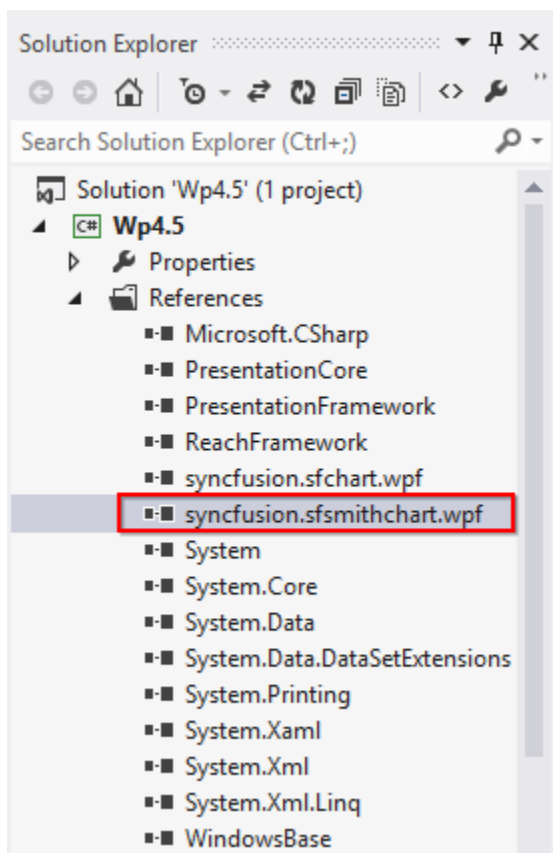
```
xmlns:syncfusion="clr-namespace:Syncfusion.UI.Xaml.SmithChart;assembly=Syncfusion.SfSmithChart.WPF"
"
```

### *Add SfSmithChart from Toolbox*

Drag and drop the SfSmithChart control from the Toolbox into your application.



Now, the Syncfusion.SfSmithChart.WPF reference has been added to the application references and the xmlns namespace code has been generated in MainWindow.xaml as below.





```
<Window xmlns:chart="http://schemas.syncfusion.com/wpf"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:syncfusion="clr-namespace:Syncfusion.UI.Xaml.SmithChart;assembly=Syncfusion.SfSmithChart.WPF"
        x:Class="Wp4._5.MainWindow"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <syncfusion:SfSmithChart >

        </syncfusion:SfSmithChart>
    </Grid>
</Window>
```

In this section, the data in the following table is used for demonstration.

### Impedance Transmission of SmithChart

Resistance	Reactance
0	0.05
0.3	0.1
0.5	0.2
1.0	0.4
1.5	0.5
2.0	0.5
2.5	0.4
3.5	0.0
4.5	-0.5
5.0	-1.0
6.0	-1.5
7.0	-2.5
8.0	-3.5
9.0	-4.5
10	-10
20	-50

Before proceeding with the smith chart, create data model with the above details as follows.

### C#

```
public class TransmissionData
{
    public double Resistance { get; set; }
    public double Reactance { get; set; }
}
```

```
}

```

Create a collection property in MainWindow class as below:

### C#

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }
    public ObservableCollection<TransmissionData> Data { get; set; }
}
```

Add the values to this TraceData property with the values illustrated in the above table.

### C#

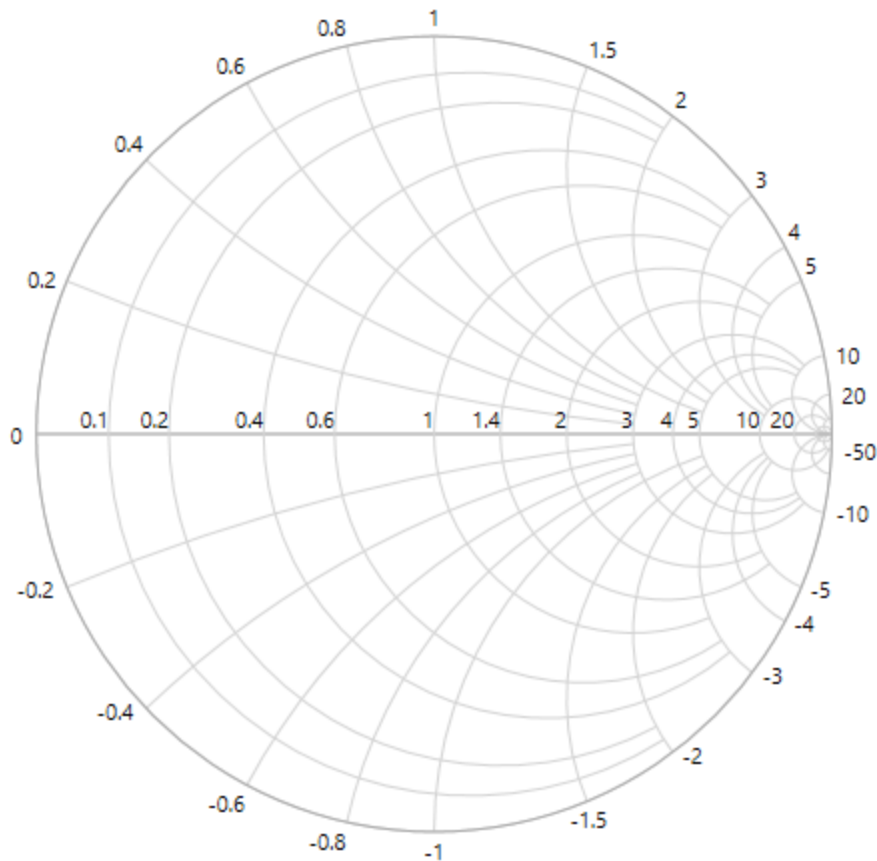
```
public MainWindow()
{
    InitializeComponent();
    Data = new ObservableCollection<TransmissionData>();
    Data.Add(new TransmissionData() { Resistance = 0, Reactance = 0.05 });
    Data.Add(new TransmissionData() { Resistance = 0.3, Reactance = 0.1 });
    Data.Add(new TransmissionData() { Resistance = 0.5, Reactance = 0.2 });
    Data.Add(new TransmissionData() { Resistance = 1.0, Reactance = 0.4 });
    Data.Add(new TransmissionData() { Resistance = 1.5, Reactance = 0.5 });
    Data.Add(new TransmissionData() { Resistance = 2.0, Reactance = 0.5 });
    Data.Add(new TransmissionData() { Resistance = 2.5, Reactance = 0.4 });
    Data.Add(new TransmissionData() { Resistance = 3.5, Reactance = 0.0 });
    Data.Add(new TransmissionData() { Resistance = 4.5, Reactance = -0.5 });
    Data.Add(new TransmissionData() { Resistance = 5, Reactance = -1.0 });
    Data.Add(new TransmissionData() { Resistance = 6, Reactance = -1.5 });
    Data.Add(new TransmissionData() { Resistance = 7, Reactance = -2.5 });
    Data.Add(new TransmissionData() { Resistance = 8, Reactance = -3.5 });
    Data.Add(new TransmissionData() { Resistance = 9, Reactance = -4.5 });
    Data.Add(new TransmissionData() { Resistance = 10, Reactance = -10 });
    Data.Add(new TransmissionData() { Resistance = 20, Reactance = -50 });
}
```

### Initialize the smith chart

To to initialize the smith chart, use the following class Syncfusion.UI.Xaml.SfSmithChart.

### XML

```
<syncfusion:SfSmithChart>
</syncfusion:SfSmithChart>
```



#### Add header to smith chart

The header of the smith chart acts as the title and it is used to identify the purpose of the smith chart.

Specify **Impedance Transmission** as header in the below code example.

#### XML

```
<Grid>
<syncfusion:SfSmithChart Header="Impedance Transmission" Height="400"
Width="500">
</syncfusion:SfSmithChart>
</Grid>
```

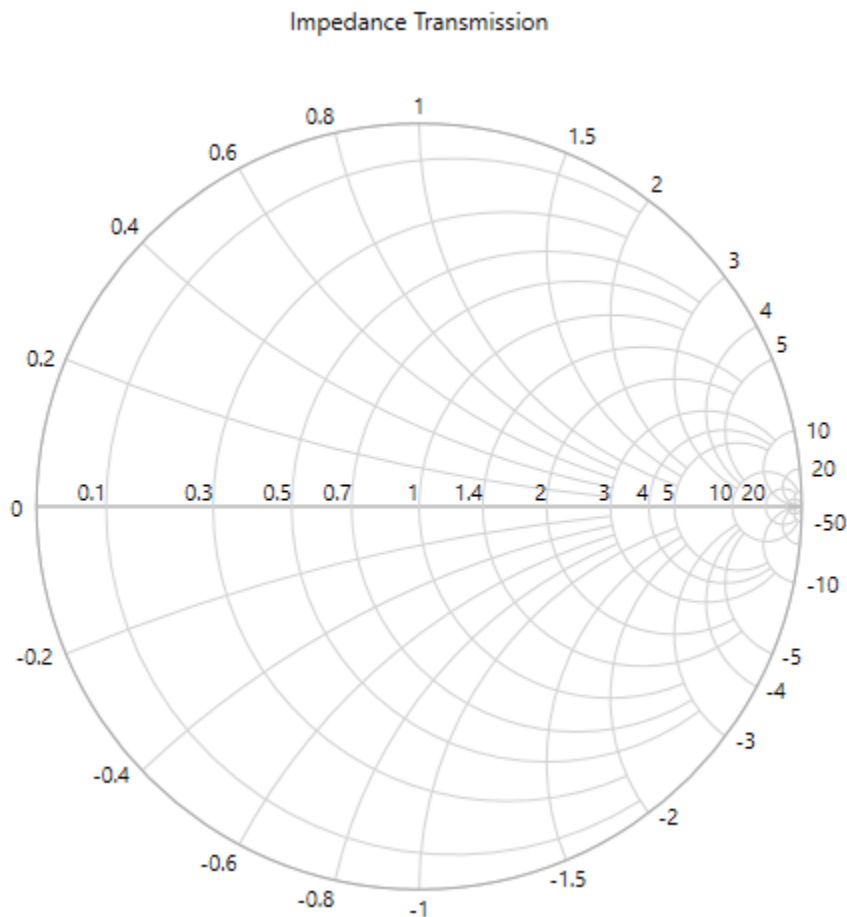
#### Adding Axes

The following code example illustrates how to add and customize the resistance (Horizontal) and reactance (Radial) axes in the SfSmithChart.

#### XML

```
<syncfusion:SfSmithChart.HorizontalAxis>
<syncfusion:HorizontalAxis FontSize="11" FontFamily="Segoe
UI"></syncfusion:HorizontalAxis>
</syncfusion:SfSmithChart.HorizontalAxis>
<syncfusion:SfSmithChart.RadialAxis>
<syncfusion:RadialAxis FontSize="11" FontFamily="Segoe
UI"></syncfusion:RadialAxis>
```

&lt;/syncfusion:SfSmithChart.RadialAxis&gt;



## Adding series

You can plot the line on smith chart map by adding line series.

You should initialize the series for representing the **Transmission Data**.

## XML

```
<syncfusion:LineSeries>
</syncfusion:LineSeries>
```

After the series has been added, you should add `ItemSource`, `ResistancePath`, and `ReactancePath` APIs to populate the data in smith chart.

- **ItemsSource** - It is a property to hold the data source, the data source or data collection can be bound with ItemsSource.
- **ResistancePath** - It is a string property, used to map properties. It needs to be bound with Resistance Axis (or HorizontalAxis). It is like a value member path in ListBox.
- **ReactancePath** - It is a string property, used to map properties. It needs to be bound with the Reactance Axis (Or RadialAxis). It is like a value member path in ListBox.
- **Label** - This property gives names for the series, which in turn mapped to the Legend.

**XML**

```
<syncfusion:LineSeries ResistancePath="Resistance" ReactancePath="Reactance"
ItemsSource="{Binding Data}" Label="TransmissionLine">
</syncfusion:LineSeries>
```

*Add legends to the smith chart*

The following code example illustrates how to add the syntax `[legends]()` in smith chart.

**XML**

```
<syncfusion:SfSmithChart.Legend>
<syncfusion:SmithChartLegend>
</syncfusion:SmithChartLegend>
</syncfusion:SfSmithChart.Legend>
```

Now, the SmithChart has been prepared to demonstrate the studies related to Transmission Line of Impedance.

The following code example illustrates the complete code for creating a smith chart.

**XML**

```
<syncfusion:SfSmithChart Header="Impedance Transmission" Height="400"
Width="500">
<!--Initialize the series for SfSmithChart-->
<syncfusion:LineSeries ResistancePath="Resistance" ReactancePath="Reactance"
ItemsSource="{Binding Data}" Label="TransmissionLine">
</syncfusion:LineSeries>
<!--Initialize the resistance axis for SfSmithChart-->
<syncfusion:SfSmithChart.HorizontalAxis>
<syncfusion:HorizontalAxis FontSize="11"></syncfusion:HorizontalAxis>
</syncfusion:SfSmithChart.HorizontalAxis>
<!--Initialize the reactance axis for SfSmithChart-->
<syncfusion:SfSmithChart.RadialAxis>
<syncfusion:RadialAxis FontSize="11"></syncfusion:RadialAxis>
</syncfusion:SfSmithChart.RadialAxis>
<!--Adding Legend to the SfSmithChart-->
<syncfusion:SfSmithChart.Legend>
<syncfusion:SmithChartLegend></syncfusion:SmithChartLegend>
</syncfusion:SfSmithChart.Legend>
</syncfusion:SfSmithChart>
```

**C#**

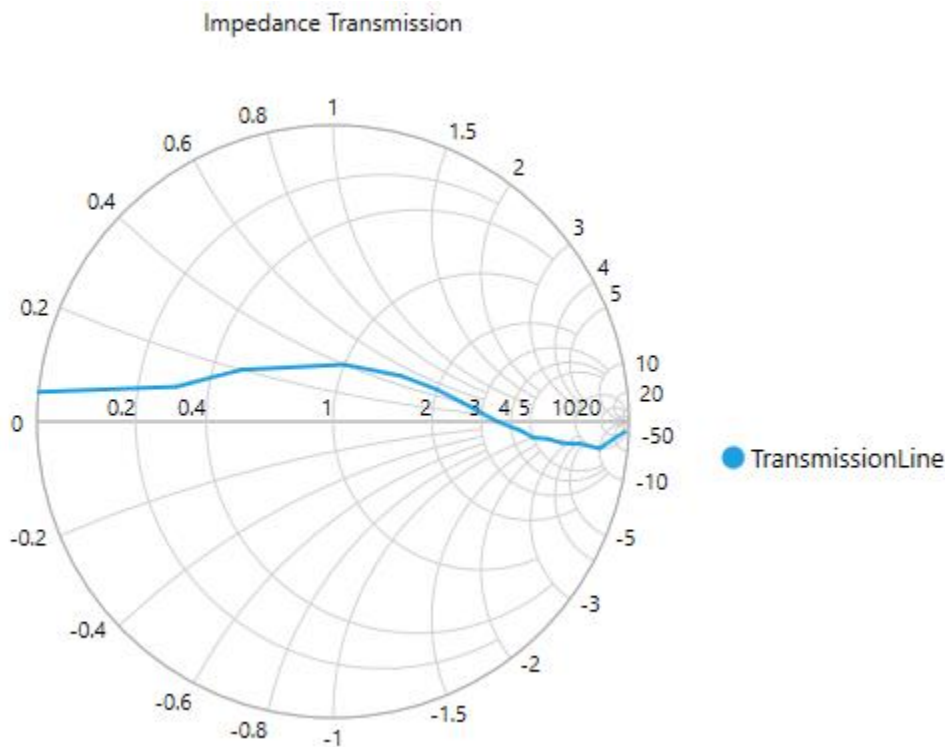
```
public partial class MainWindow : Window
{
    public ObservableCollection<TransmissionData> Data { get; set; }
    public MainWindow()
    {
        this.DataContext = this;
        InitializeComponent();
        Data = new ObservableCollection<TransmissionData>();
        Data.Add(new TransmissionData() { Resistance = 0, Reactance = 0.05 });
        Data.Add(new TransmissionData() { Resistance = 0.3, Reactance = 0.1 });
        Data.Add(new TransmissionData() { Resistance = 0.5, Reactance = 0.2 });
    }
}
```

```

Data.Add(new TransmissionData() { Resistance = 1.0, Reactance = 0.4 });
Data.Add(new TransmissionData() { Resistance = 1.5, Reactance = 0.5 });
Data.Add(new TransmissionData() { Resistance = 2.0, Reactance = 0.5 });
Data.Add(new TransmissionData() { Resistance = 2.5, Reactance = 0.4 });
Data.Add(new TransmissionData() { Resistance = 3.5, Reactance = 0.0 });
Data.Add(new TransmissionData() { Resistance = 4.5, Reactance = -0.5 });
Data.Add(new TransmissionData() { Resistance = 5, Reactance = -1.0 });
Data.Add(new TransmissionData() { Resistance = 6, Reactance = -1.5 });
Data.Add(new TransmissionData() { Resistance = 7, Reactance = -2.5 });
Data.Add(new TransmissionData() { Resistance = 8, Reactance = -3.5 });
Data.Add(new TransmissionData() { Resistance = 9, Reactance = -4.5 });
Data.Add(new TransmissionData() { Resistance = 10, Reactance = -10 });
Data.Add(new TransmissionData() { Resistance = 20, Reactance = -50 });
}
}
public class TransmissionData
{
    public double Resistance { get; set; }
    public double Reactance { get; set; }
}

```

The following smith chart is created as the result of above codes.



#### Create a simple smith chart from code behind (C#)

Some developers prefer code behind as the first approach for development to create things dynamically. This section explains the steps required to create SfSmithChart from code behind.

### *Adding assembly reference*

1. Open the Add Reference window in your project.
2. Choose Windows > Extensions > Syncfusion.SfSmithChart.WPF
3. Add the following namespace in your C# file, MainWindow.xaml.cs.

### **C#**

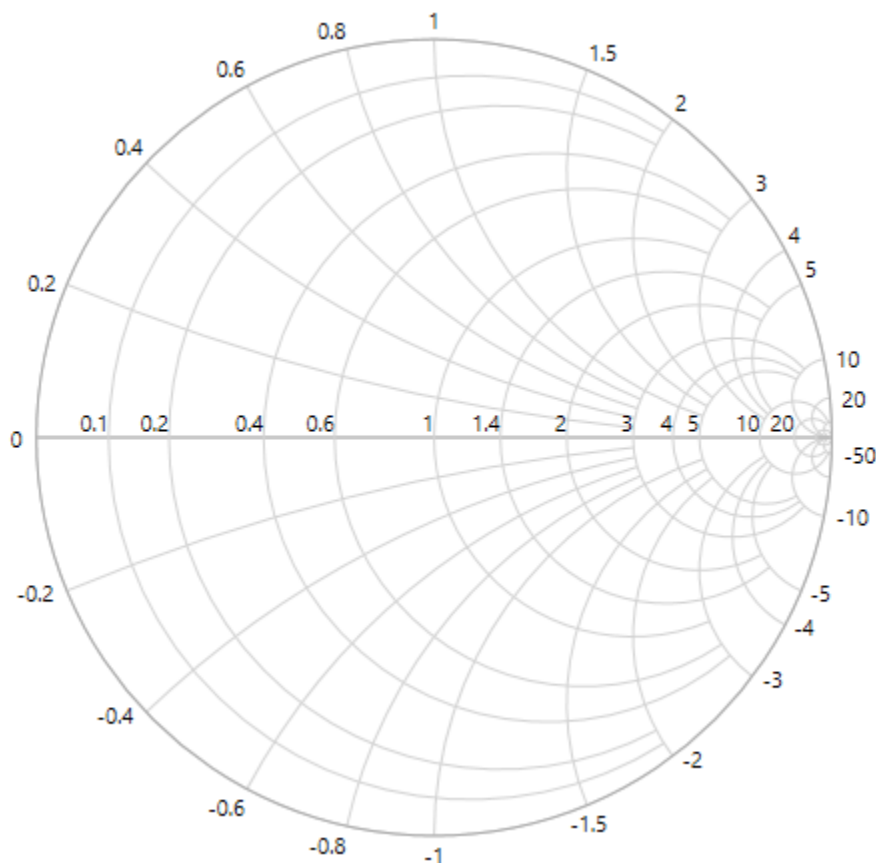
```
using Syncfusion.UI.Xaml.SmithChart;
```

### *Initialize the chart*

To initialize the chart, create an instance for the SfSmithChart as below.

### **C#**

```
SfSmithChart chart = new SfSmithChart();
```



### *Adding header to the smith chart*

The header of the SmithChart acts as the title and it is used to identify the purpose of the smith chart.

### **C#**

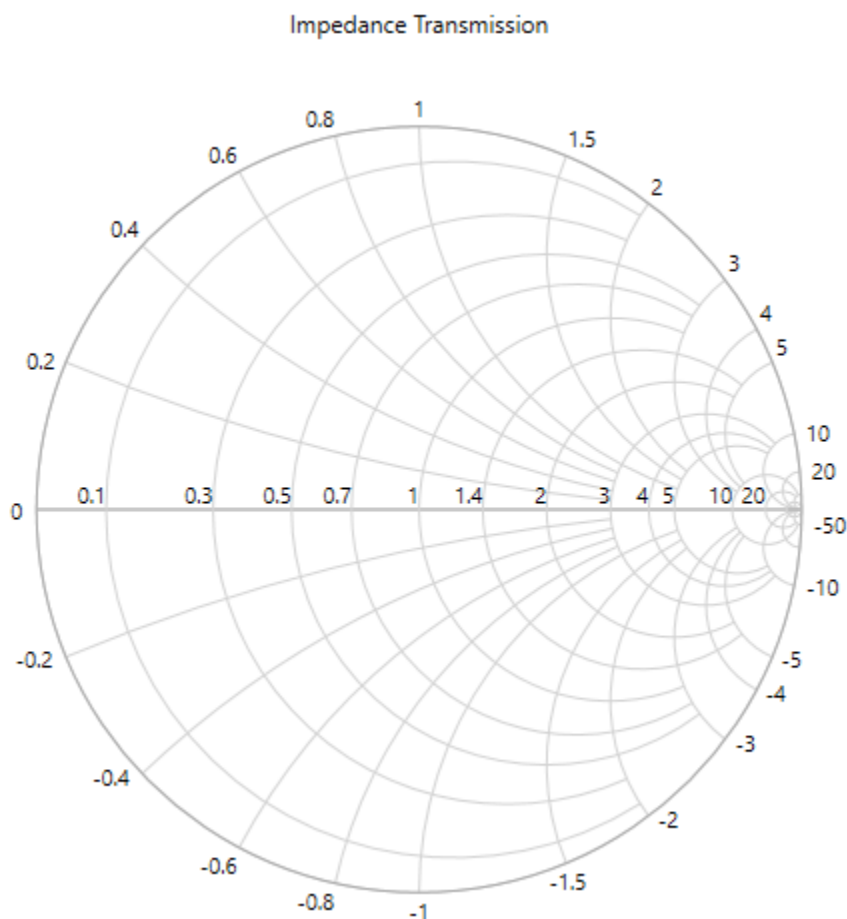
```
chart.Header = "Impedance Transmission";
```

### Adding Axes

The following code example illustrates how to add and customize the resistance (Horizontal) and reactance (Radial) axes to the SfSmithChart.

#### C#

```
//Customizing horizontal(Resistance) axis to the smith chart
chart.HorizontalAxis = new HorizontalAxis();
chart.HorizontalAxis.FontSize = 11;
chart.HorizontalAxis.FontFamily = new FontFamily("Segoe UI");
//Customizing radial(Reactance) axis to the smith chart
chart.RadialAxis = new RadialAxis();
chart.RadialAxis.FontSize = 11;
chart.RadialAxis.FontFamily = new FontFamily("Segoe UI");
```



### Adding series

You can plot the line on smith chart map by adding line series.

You should initialize the series for representing the **Transmission Data**.

#### C#

```
LineSeries series = new LineSeries();
```



After the series has been added, you should add ItemSource, ResistancePath and, ReactancePath APIs to populate the data in smith chart.

- **ItemsSource** - It is a property to hold the data source, the data source or data collection can be bound with ItemsSource.
- **ResistancePath** - It is a string property, used to map properties. It needs to be bound with Resistance Axis (or HorizontalAxis). It is like a value member path in ListBox.
- **ReactancePath** - It is a string property, used to map properties. It needs to be bound with the Reactance Axis (Or RadialAxis). It is like a value member path in ListBox.
- **Label** - This property gives names for the series, which in turn mapped to the Legend.

### C#

```
LineSeries series = new LineSeries();
series.ItemsSource = Data;
series.ResistancePath = "Resistance";
series.ReactancePath = "Reactance";
series.Label = "TransmissionLine";
chart.Series.Add(series);
```

### Adding legends to the chart

The following code examples illustrates how to add **legends**() to your smith chart.

### C#

```
SmithChartLegend legend = new SmithChartLegend();
chart.Legend = legend;
```

Now, the SmithChart has been prepared to demonstrate the studies related to Transmission Line of Impedance.

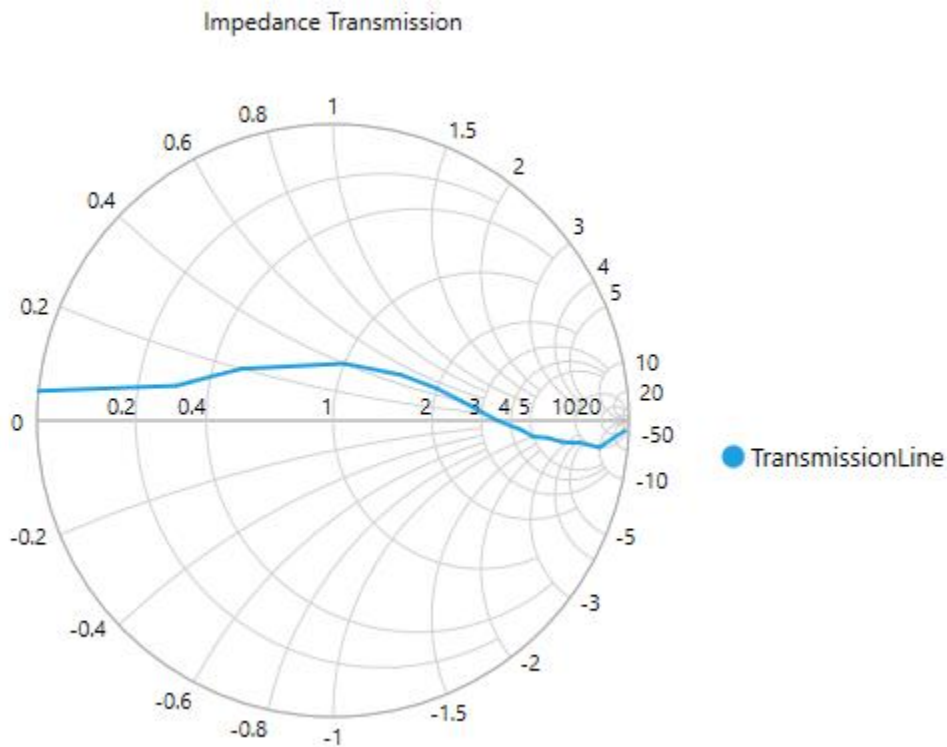
The following code example illustrates the complete code for creating a smith chart.

### C#

```
SfSmithChart chart = new SfSmithChart();
chart.Header = "Impedance Transmission";
//Customizing horizontal(Resistance) axis to the smith chart
chart.HorizontalAxis = new HorizontalAxis();
chart.HorizontalAxis.FontSize = 11;
chart.HorizontalAxis.FontFamily = new FontFamily("Segoe UI");
//Customizing radial(Reactance) axis to the smith chart
chart.RadialAxis = new RadialAxis();
chart.RadialAxis.FontSize = 11;
chart.RadialAxis.FontFamily = new FontFamily("Segoe UI");
//Adding series to SmithChart
LineSeries series = new LineSeries();
series.ItemsSource = Data;
series.ResistancePath = "Resistance";
series.ReactancePath = "Reactance";
series.Label = "TransmissionLine";
chart.Series.Add(series);
//Adding legend to the SmithChart
SmithChartLegend legend = new SmithChartLegend();
```

```
chart.Legend = legend;  
this.DataContext = this;  
//Setting SmithChart as a Content for the Grid in Page  
this.Grid1.Children.Add(chart);
```

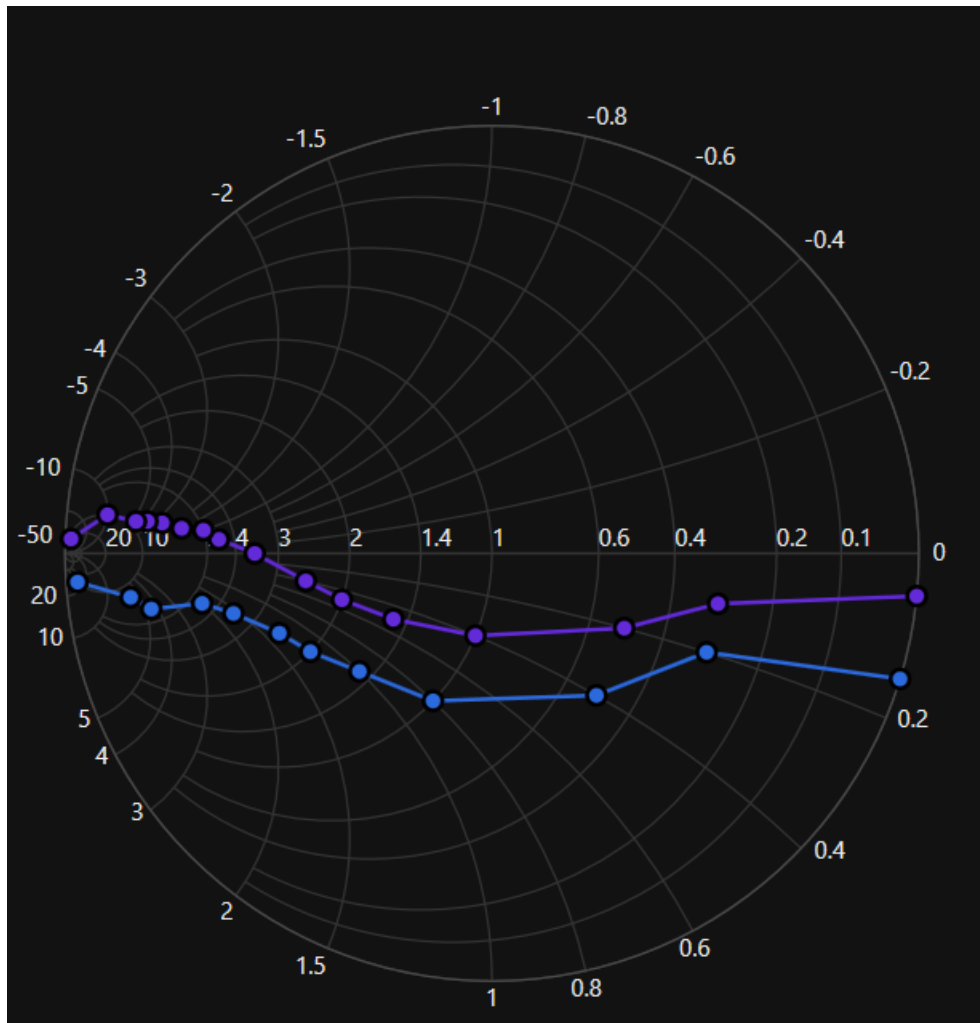
The following output is displayed as the result of above code example.



### Theme

Smith chart supports various built-in themes. Refer to the below links to apply themes for the Smith chart,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Rendering Type in WPF Smith Chart (SfSmithChart)

SfSmithChart plots the transmission line in two different ways by using `RenderingType` property. The two ways are given below.

#### Impedance

In impedance smith chart, normalized resistance circles and normalized reactance curves are drawn from right to left. Axis label ranges are start from left to right.

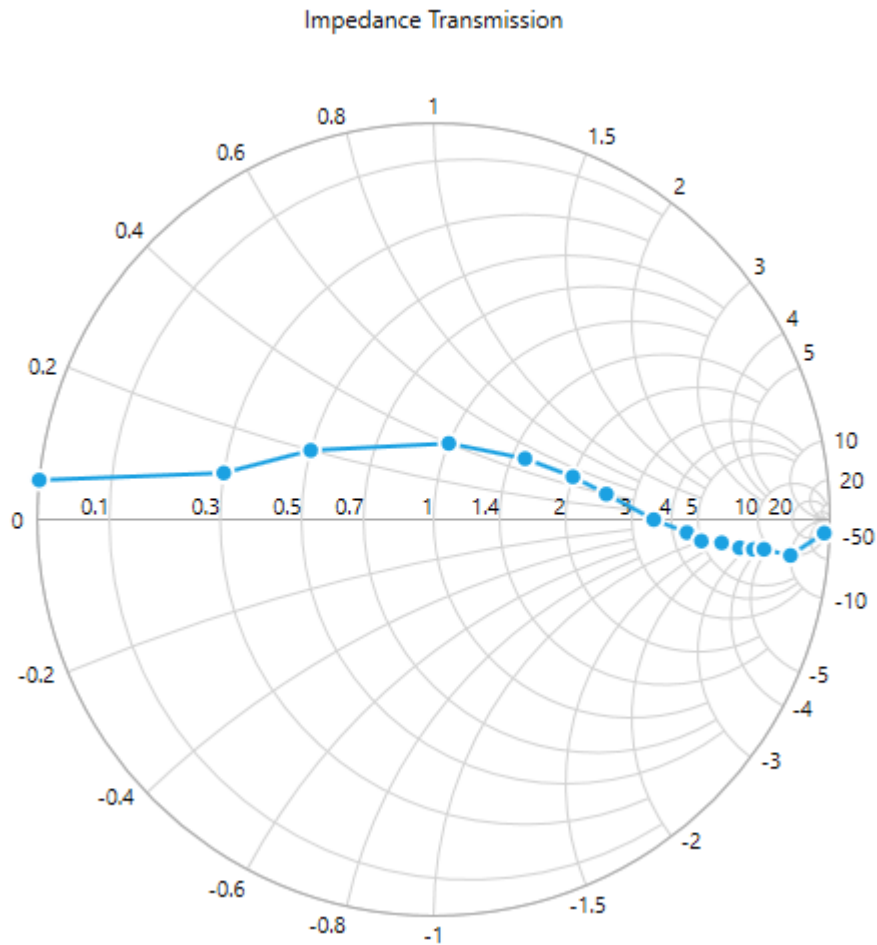
Impedance is the default rendering type of SmithChart.

#### XML

```
<syncfusion:SfSmithChart RenderingType="Impedance">  
</syncfusion:SfSmithChart>
```

#### C#

```
SfSmithChart chart = new SfSmithChart();  
chart.RenderingType = RenderingType.Impedance;
```



### Admittance

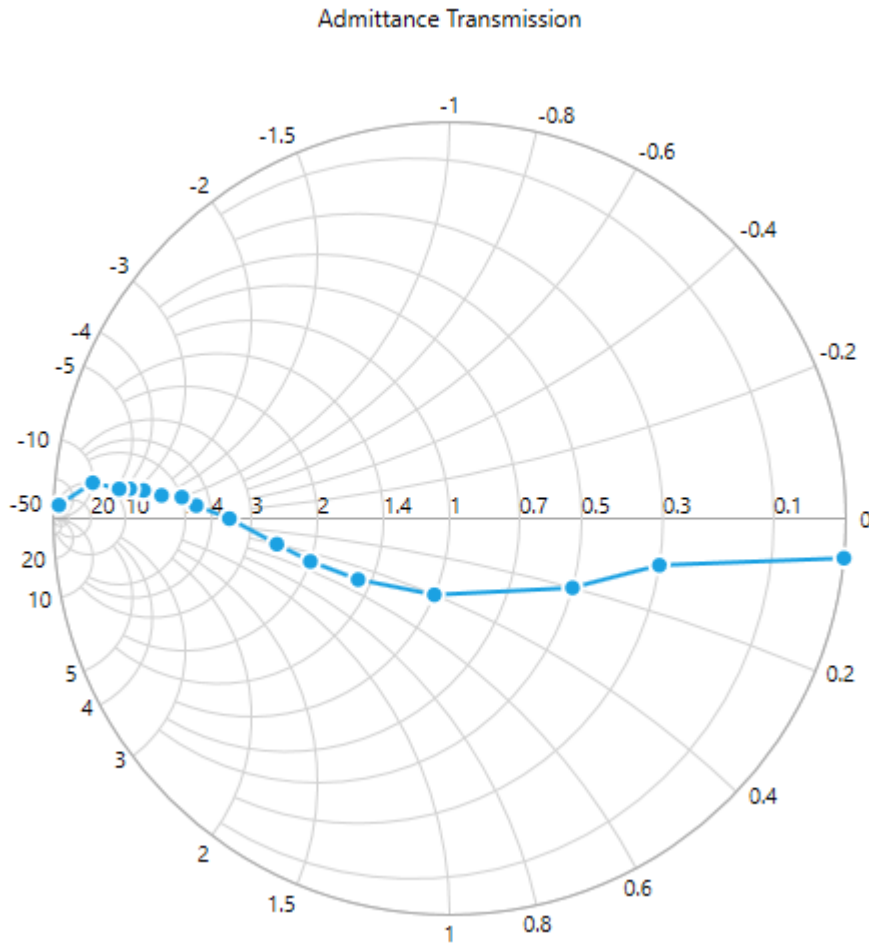
In Admittance smith chart, normalized resistance circles and normalized reactance curves are drawn from left to right. Axis label ranges are start from right to left.

### XML

```
<syncfusion:SfSmithChart RenderingType="Admittance">  
</syncfusion:SfSmithChart>
```

### C#

```
SfSmithChart chart = new SfSmithChart();  
chart.RenderingType = RenderingType.Admittance;
```



### Axes in WPF Smith Chart (SfSmithChart)

Typically, SmithChart has been used two axes that are used to measure and categorize the data.

1. Horizontal Axis (Resistance)
2. Radial Axis (Reactance)

#### Horizontal Axis

Horizontal axis scale is used to measure normalized resistance value.

The following topics explain in detail about the axis and its parts.

#### MajorGridlines

By default, major gridlines are automatically added to the Axis. SfSmithChart supports the customization of major gridline. The visibility of the major gridlines can be controlled by using the **ShowMajorGridlines** property.

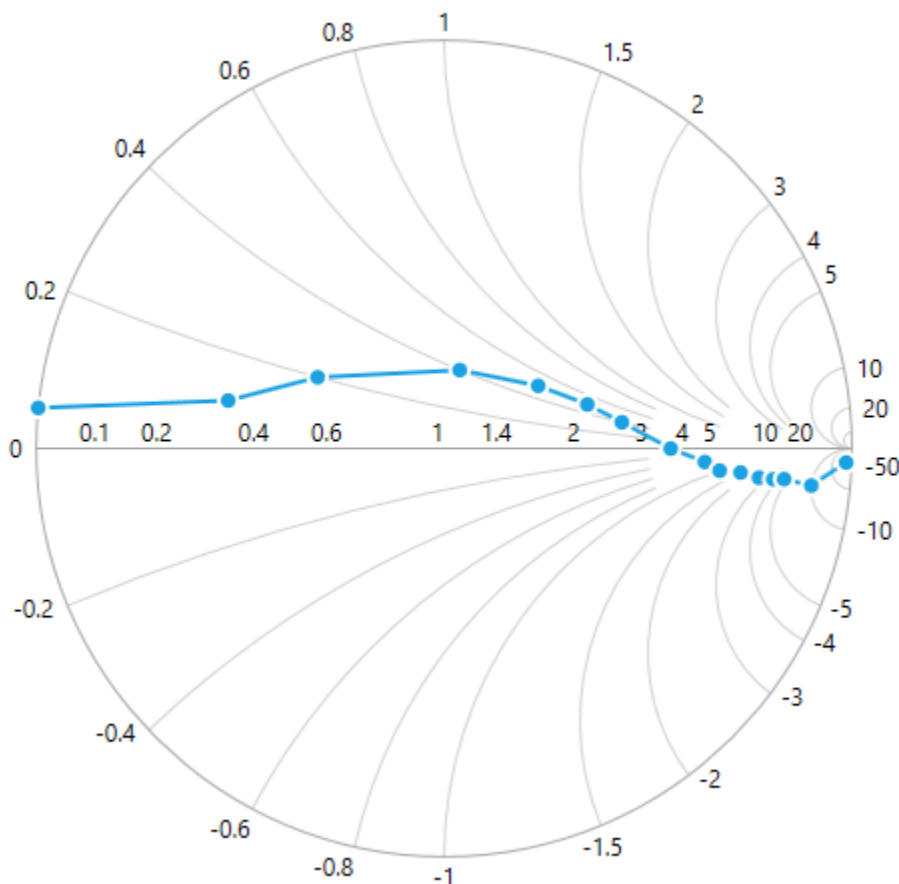
#### XML

```
<syncfusion:SfSmithChart>
  <syncfusion:SfSmithChart.HorizontalAxis>
    <syncfusion:HorizontalAxis ShowMajorGridlines="False">
    </syncfusion:HorizontalAxis>
  </syncfusion:SfSmithChart.HorizontalAxis>
</syncfusion:SfSmithChart>
```

```
</syncfusion:SfSmithChart.HorizontalAxis>
</syncfusion:SfSmithChart>
```

**C#**

```
//Customizing horizontal(Resistance) axis to the chart
chart.HorizontalAxis = new HorizontalAxis();
chart.HorizontalAxis.ShowMajorGridlines = false;
```

**MajorGridlineStyle**

SfSmithChart provides support to customize the style of the major gridlines by defining the *MajorGridlineStyle* property as shown in the below code snippet.

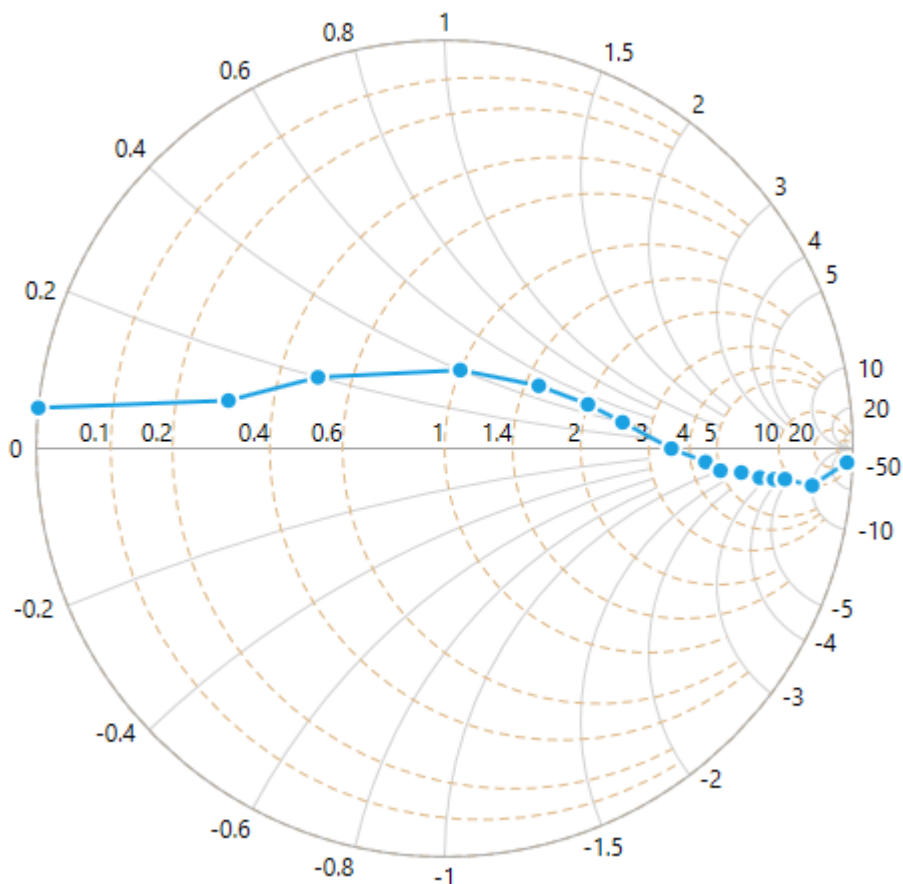
**XML**

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.Resources>
    <Style TargetType="Path" x:Key="lineStyle">
      <Setter Property="Stroke" Value="BurlyWood"></Setter>
      <Setter Property="StrokeDashArray" Value="5,3"></Setter>
    </Style>
  </syncfusion:SfSmithChart.Resources>
  <syncfusion:SfSmithChart.HorizontalAxis>
```

```
<syncfusion:HorizontalAxis MajorGridLineStyle="{StaticResource lineStyle}">
</syncfusion:HorizontalAxis>
</syncfusion:SfSmithChart.HorizontalAxis>
</syncfusion:SfSmithChart>
```

**C#**

```
chart.HorizontalAxis.MajorGridLineStyle = this.Grid1.Resources["lineStyle"]
as Style;
```

*MinorGridlines*

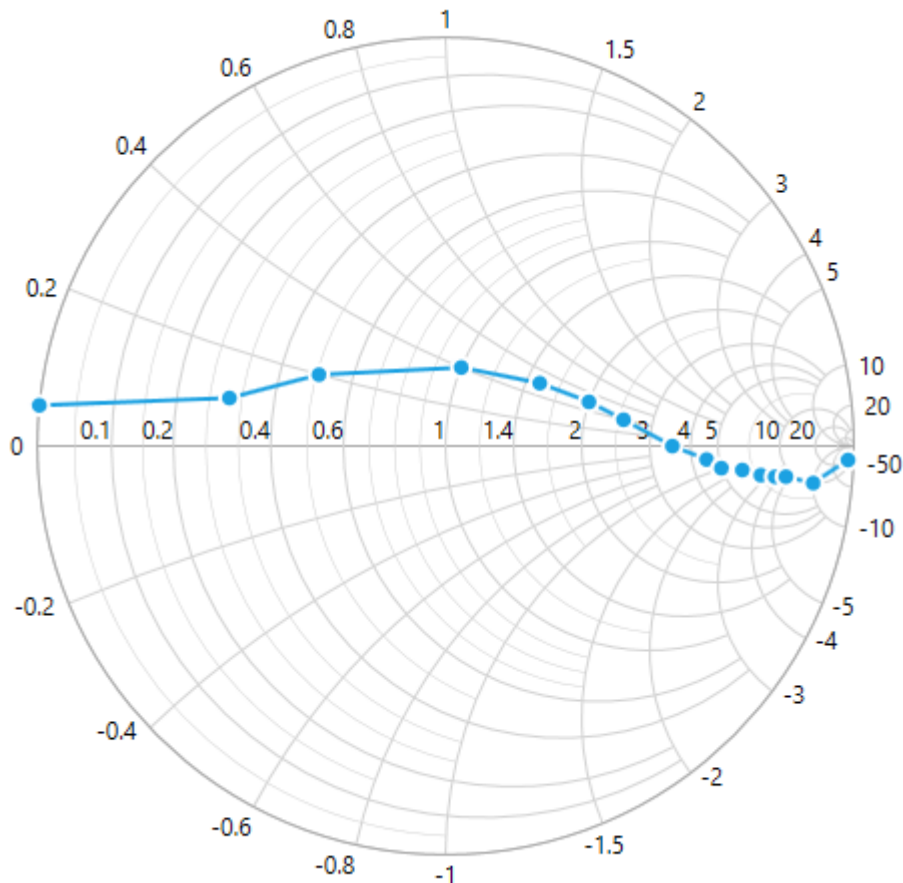
By default, minor gridlines are not added to the Axis. The visibility of the minor gridlines can be controlled by using the `ShowMinorGridlines` property.

**XML**

```
<syncfusion:SfSmithChart>
<syncfusion:SfSmithChart.HorizontalAxis>
<syncfusion:HorizontalAxis ShowMinorGridlines="True">
</syncfusion:HorizontalAxis>
</syncfusion:SfSmithChart.HorizontalAxis>
</syncfusion:SfSmithChart>
```

**C#**

```
//Customizing horizontal(Resistance) axis to the chart
chart.HorizontalAxis = new HorizontalAxis();
chart.HorizontalAxis.ShowMinorGridlines = true;
```

**MinorGridlinesCount**

Minor gridlines can be added by defining *MinorGridlinesCount* property. By default, this value is eight. It means, every 100 pixels it renders maximum eight minor gridlines.

**XML**

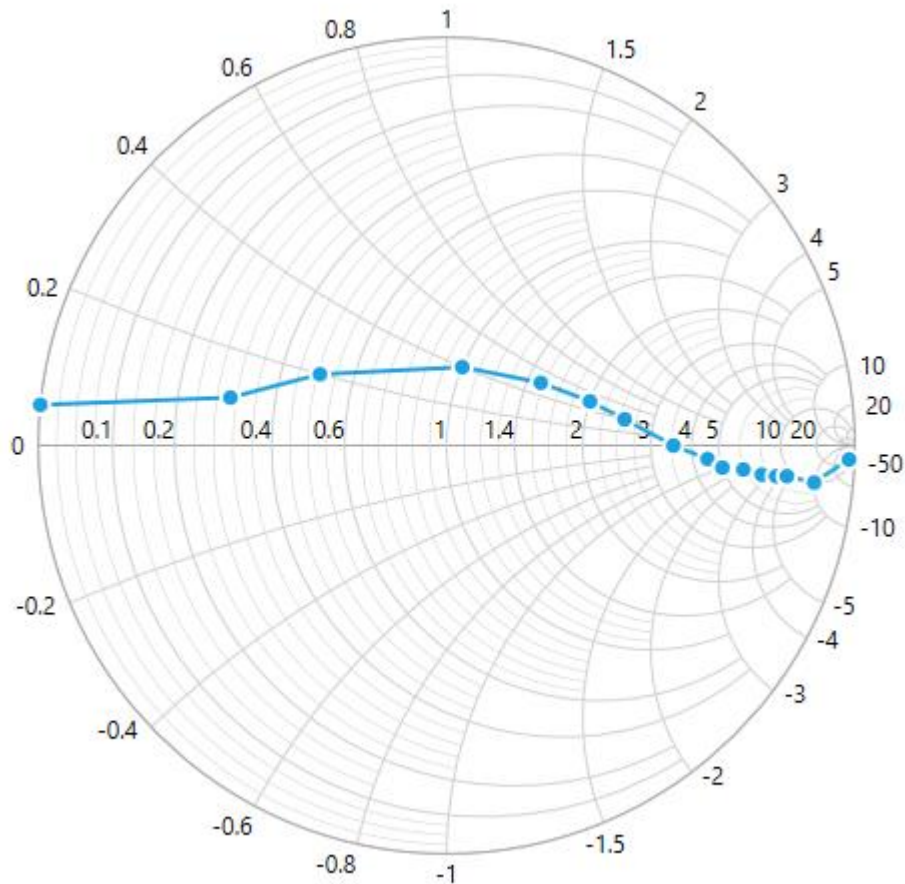
```
<syncfusion:SfSmithChart>
  <syncfusion:SfSmithChart.HorizontalAxis>
    <syncfusion:HorizontalAxis ShowMinorGridlines="True"
      MinorGridlinesCount="12">
    </syncfusion:HorizontalAxis>
  </syncfusion:SfSmithChart.HorizontalAxis>
</syncfusion:SfSmithChart>
```

**C#**

```
//Customizing horizontal(Resistance) axis to the chart
```



```
chart.HorizontalAxis = new HorizontalAxis();
chart.HorizontalAxis.ShowMinorGridlines = true;
chart.HorizontalAxis.MinorGridlinesCount = 12;
```



## MinorGridLineStyle

SfSmithChart provides support to customize the style of the minor gridlines by defining the *MinorGridLineStyle* property as shown in the below code snippet.

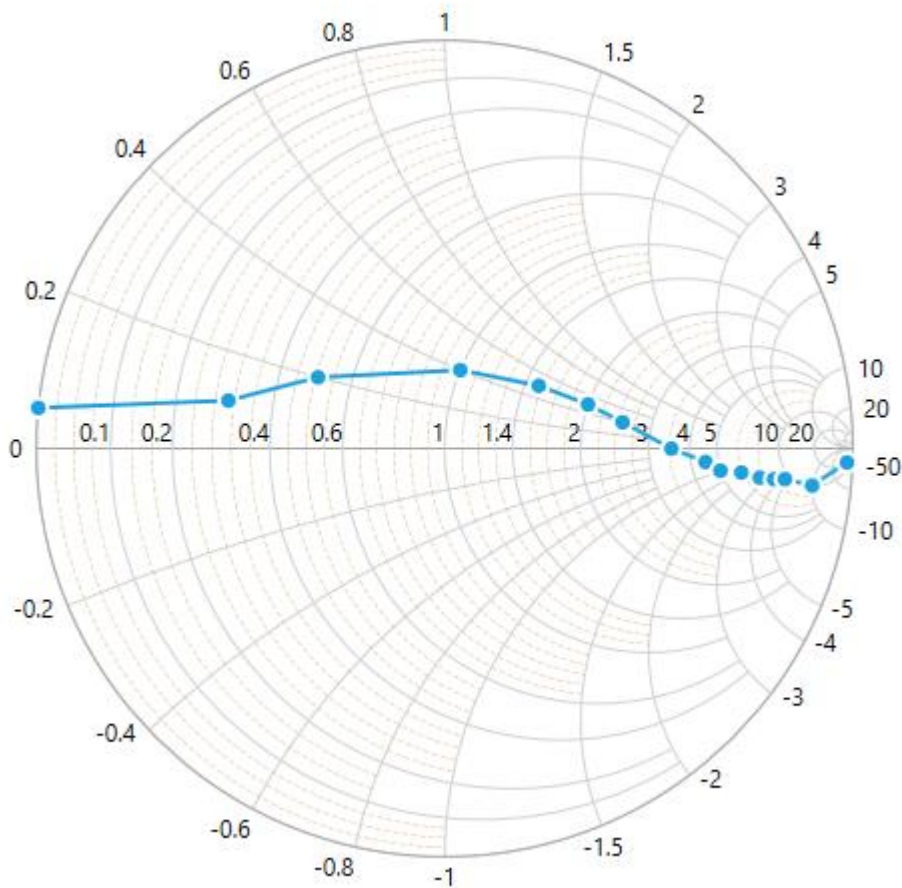
XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
<syncfusion:SfSmithChart.Resources>
<Style TargetType="Path" x:Key="lineStyle">
<Setter Property="Stroke" Value="BurlyWood"></Setter>
<Setter Property="StrokeThickness" Value="0.45"></Setter>
<Setter Property="StrokeDashArray" Value="7,3"></Setter>
</Style>
</syncfusion:SfSmithChart.Resources>
<syncfusion:SfSmithChart.HorizontalAxis>
<syncfusion:HorizontalAxis ShowMinorGridlines="True"
MinorGridlinesCount="12" MinorGridlineStyle="{StaticResource lineStyle}">
</syncfusion:HorizontalAxis>
</syncfusion:SfSmithChart.HorizontalAxis>
```

```
</syncfusion:SfSmithChart>
```

**C#**

```
chart.HorizontalAxis.MinorGridLineStyle = this.Grid1.Resources["lineStyle"]
as Style;
```

**AxisLine**

SfSmithChart provides support to customize the style of the axis line by defining the *AxisLineStyle* property and change the visibility by using *ShowAxisLine* property as shown in the below code snippet.

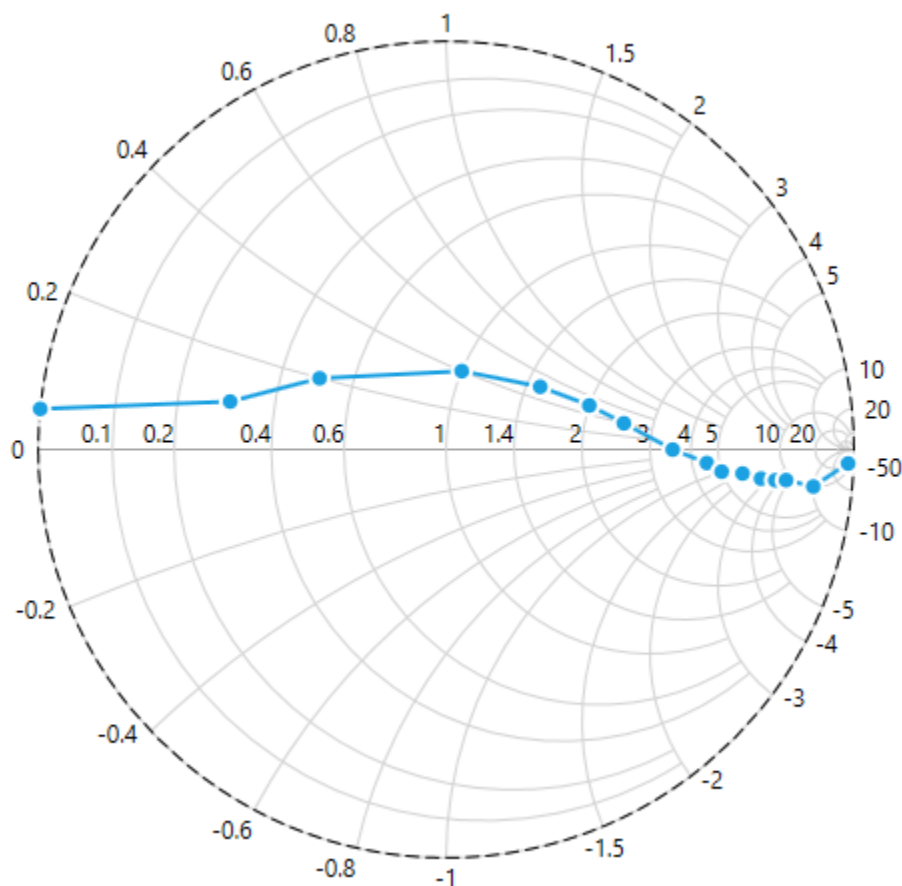
**XML**

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.Resources>
    <Style TargetType="Path" x:Key="lineStyle">
      <Setter Property="Stroke" Value="Black"></Setter>
      <Setter Property="StrokeThickness" Value="1"></Setter>
      <Setter Property="StrokeDashArray" Value="7,3"></Setter>
    </Style>
  </syncfusion:SfSmithChart.Resources>
  <syncfusion:SfSmithChart.HorizontalAxis>
```

```
<syncfusion:HorizontalAxis ShowAxisLine="True"
AxisLineStyle="{StaticResource lineStyle}">
</syncfusion:HorizontalAxis>
</syncfusion:SfSmithChart.HorizontalAxis>
</syncfusion:SfSmithChart>
```

**C#**

```
chart.HorizontalAxis.AxisLineStyle = this.Grid1.Resources["lineStyle"] as
Style;
```

*LabelPlacement*

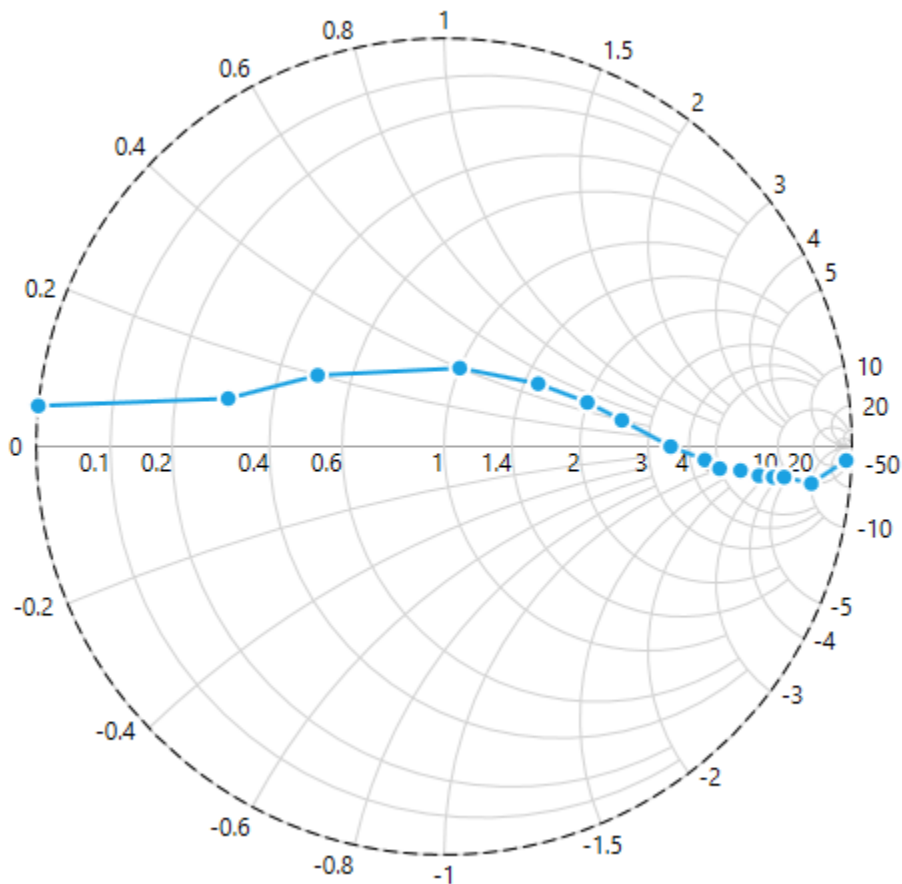
The *LabelPlacement* property is used to position the axis label either inside or outside of the chart plotting area. By default, LabelsPlacement is **Outside**.

**XML**

```
<syncfusion:SfSmithChart x:Name="SmithChart">
<syncfusion:SfSmithChart.HorizontalAxis>
<syncfusion:HorizontalAxis LabelPlacement="Inside">
</syncfusion:HorizontalAxis>
</syncfusion:SfSmithChart.HorizontalAxis>
</syncfusion:SfSmithChart>
```

**C#**

```
//Positioning the horizontal axis labels to bottom of AxisLine
chart.HorizontalAxis.LabelPlacement = LabelPlacement.Inside;
```

*LabelIntersectAction*

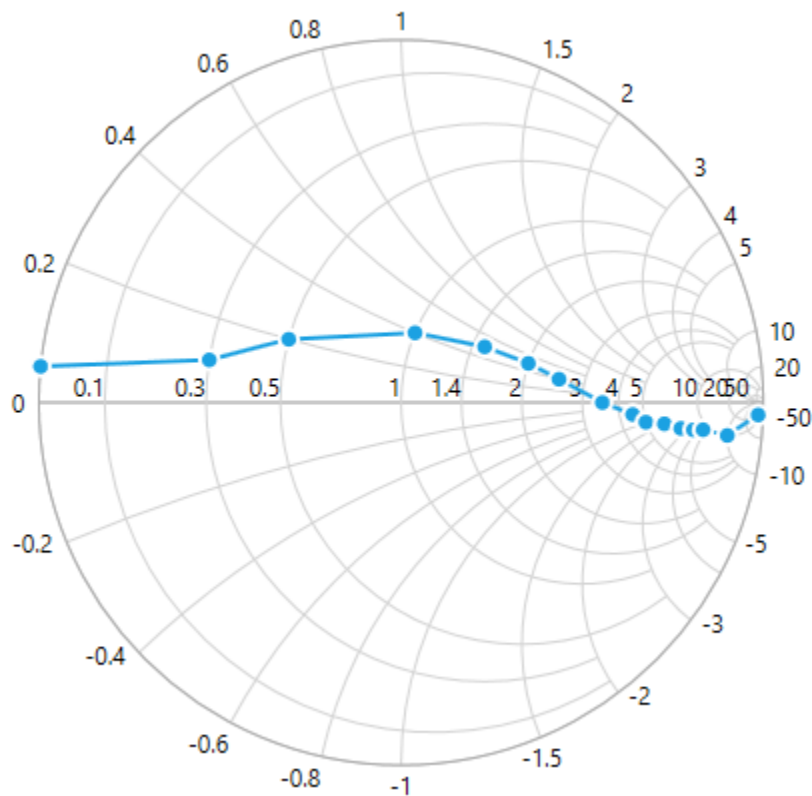
When the Axis labels overlap with each other based on the chart dimensions and label size, use the *LabelIntersectAction* property of the axis to avoid overlapping. The default value of the *LabelIntersectAction* is **Hide**.

**XML**

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.HorizontalAxis>
    <syncfusion:HorizontalAxis LabelIntersectAction="None">
    </syncfusion:HorizontalAxis>
  </syncfusion:SfSmithChart.HorizontalAxis>
</syncfusion:SfSmithChart>
```

**C#**

```
//Set axis label intersect action
chart.HorizontalAxis.LabelIntersectAction = LabelIntersectActions.None;
```



### Radial Axis

Radial axis scale is used to measure the normalized reactance values.

### MajorGridlines

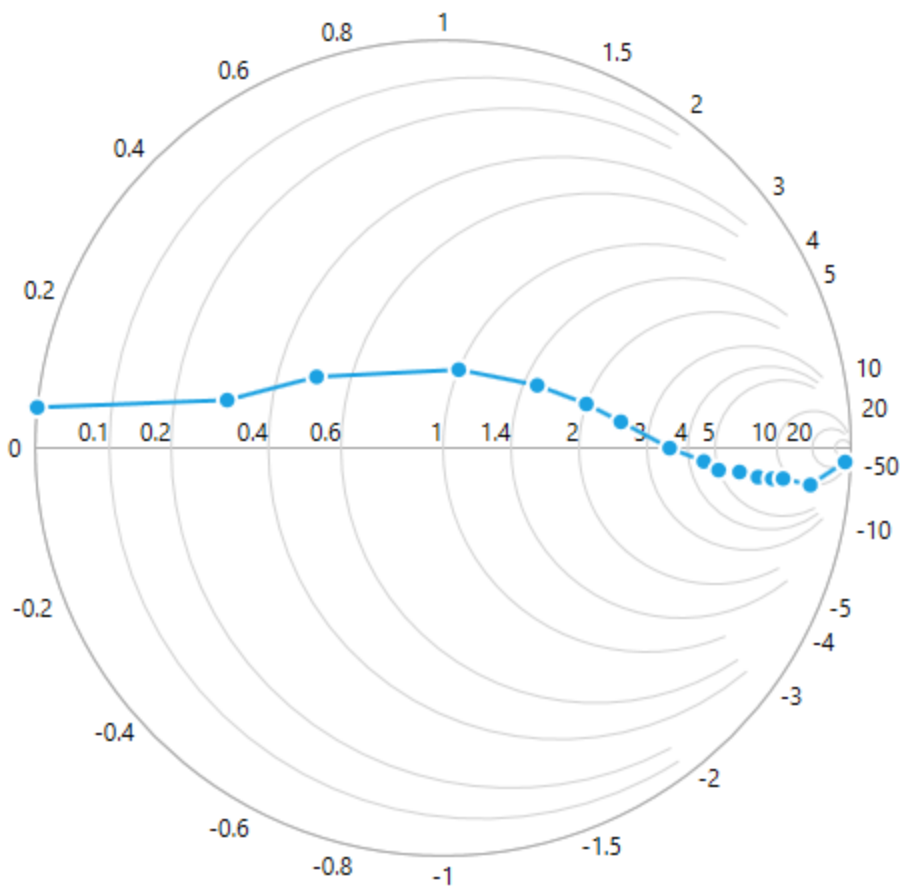
By default, major gridlines are automatically added to the Axis. SfSmithChart supports customization of major gridline. The visibility of the major gridlines can be controlled by using the *ShowMajorGridlines* property.

### XML

```
<syncfusion:SfSmithChart>
  <syncfusion:SfSmithChart.RadialAxis>
    <syncfusion:RadialAxis ShowMajorGridlines="False">
    </syncfusion:RadialAxis >
  </syncfusion:SfSmithChart.RadialAxis >
</syncfusion:SfSmithChart>
```

### C#

```
//Customizing radial(Reactance) axis to the chart
chart.RadialAxis = new RadialAxis();
chart.RadialAxis.ShowMajorGridlines = false;
```



### MajorGridlineStyle

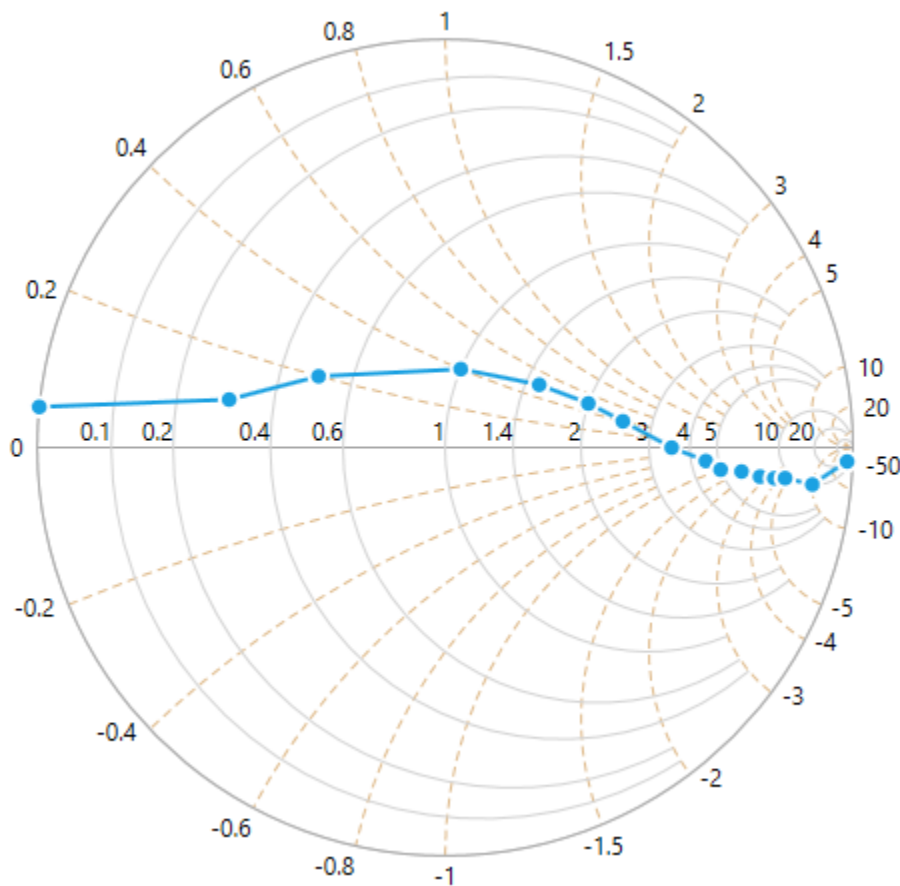
SfSmithChart provides support to customize the style of the major gridlines by defining the *MajorGridlineStyle* property as shown in the below code snippet.

#### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.Resources>
    <Style TargetType="Path" x:Key="lineStyle">
      <Setter Property="Stroke" Value="BurlyWood"></Setter>
      <Setter Property="StrokeDashArray" Value="5,3"></Setter>
    </Style>
  </syncfusion:SfSmithChart.Resources>
  <syncfusion:SfSmithChart.RadialAxis>
    <syncfusion:RadialAxis MajorGridlineStyle="{StaticResource lineStyle}">
    </syncfusion:RadialAxis >
  </syncfusion:SfSmithChart.RadialAxis>
</syncfusion:SfSmithChart>
```

#### C#

```
chart.RadialAxis.MajorGridlineStyle = this.Grid1.Resources["lineStyle"] as
Style;
```



### MinorGridlines

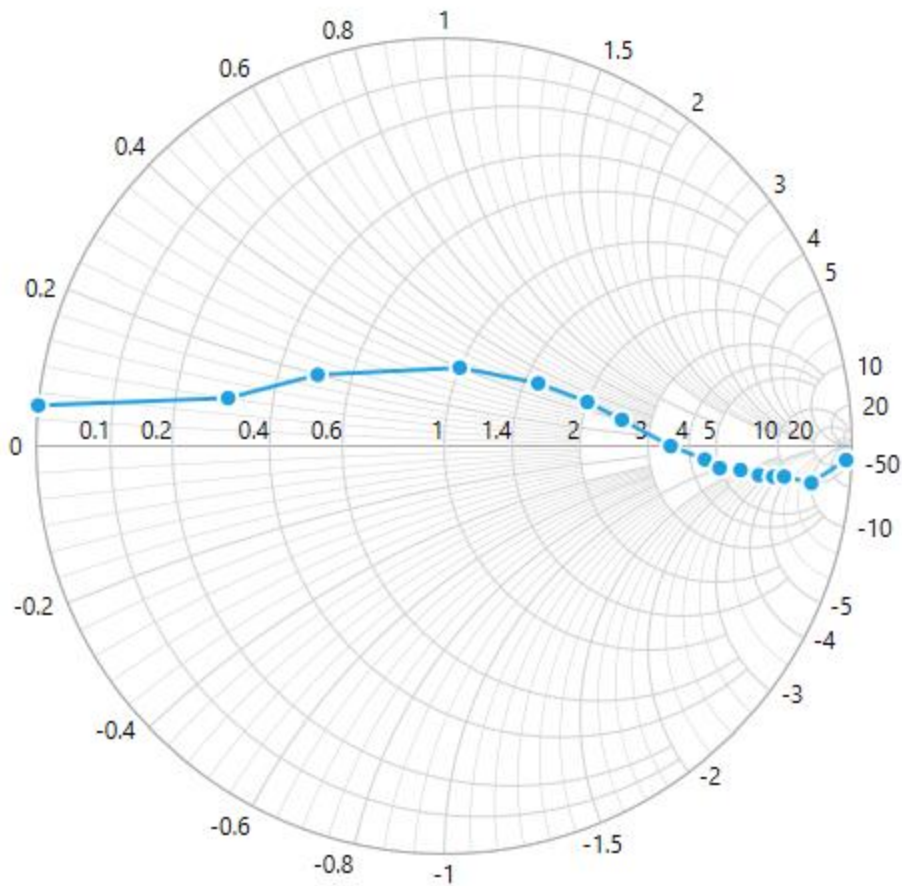
By default, minor gridlines are not added to the Axis. The visibility of the minor gridlines can be controlled by using the *ShowMinorGridlines* property.

### XML

```
<syncfusion:SfSmithChart>
  <syncfusion:SfSmithChart.RadialAxis>
    <syncfusion:RadialAxis ShowMinorGridlines="True">
    </syncfusion:RadialAxis>
  </syncfusion:SfSmithChart.RadialAxis>
</syncfusion:SfSmithChart>
```

### C#

```
//Customizing radial(Reactance) axis to the chart
chart.RadialAxis = new RadialAxis();
chart.RadialAxis.ShowMinorGridlines = true;
```



### MinorGridlinesCount

Minor gridlines can be added by defining *MinorGridlinesCount* property. By default, this value is eight. It means, every 100 pixels it renders maximum eight minor gridlines.

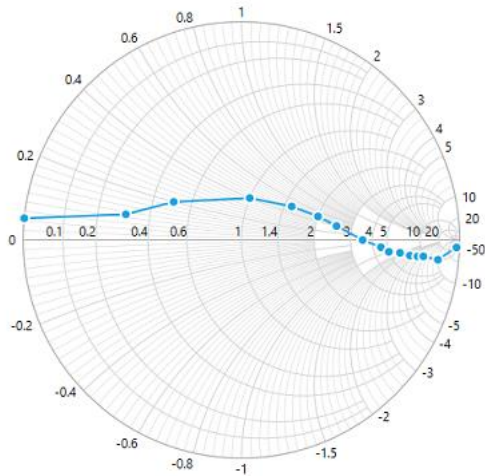
### XML

```
<syncfusion:SfSmithChart>
  <syncfusion:SfSmithChart.RadialAxis>
    <syncfusion:RadialAxis ShowMinorGridlines="True" MinorGridlinesCount="12">
    </syncfusion:RadialAxis>
  </syncfusion:SfSmithChart.RadialAxis>
</syncfusion:SfSmithChart>
```

### C#

```
//Customizing radial(Reactance) axis to the chart
chart.RadialAxis = new RadialAxis();
chart.RadialAxis.ShowMinorGridlines = true;
chart.RadialAxis.MinorGridlinesCount = 12;
```





### MinorGridlineStyle

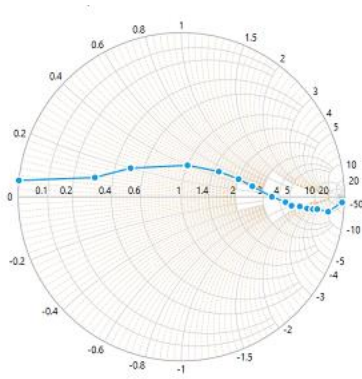
SfSmithChart provides support to customize the style of the minor gridlines by defining the *MinorGridlineStyle* property as shown in the below code snippet.

#### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.Resources>
    <Style TargetType="Path" x:Key="lineStyle">
      <Setter Property="Stroke" Value="BurlyWood"></Setter>
      <Setter Property="StrokeThickness" Value="0.45"></Setter>
      <Setter Property="StrokeDashArray" Value="7,3"></Setter>
    </Style>
  </syncfusion:SfSmithChart.Resources>
  <syncfusion:SfSmithChart.RadialAxis>
    <syncfusion:RadialAxis ShowMinorGridlines="True" MinorGridlinesCount="12"
      MinorGridlineStyle="{StaticResource lineStyle}">
    </syncfusion:RadialAxis>
  </syncfusion:SfSmithChart.RadialAxis>
</syncfusion:SfSmithChart>
```

#### C#

```
chart.RadialAxis.MinorGridlineStyle = this.Grid1.Resources["lineStyle"] as
Style;
```



### AxisLine

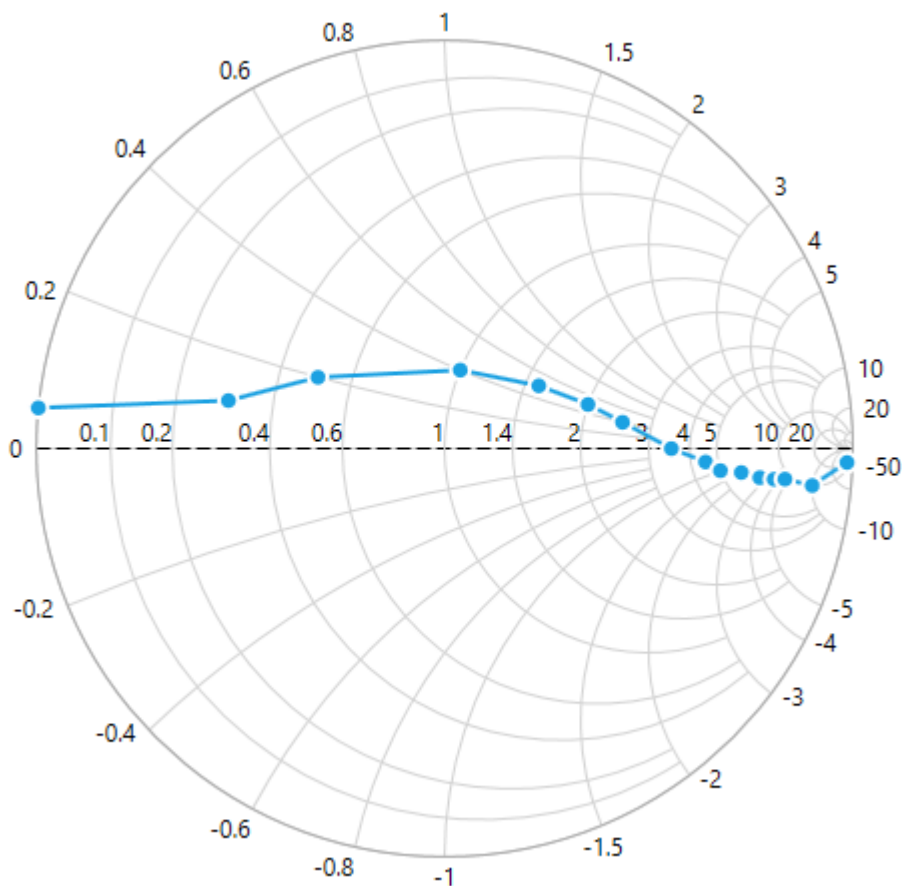
SfSmithChart provides support to customize the style of the axis line by defining the *AxisLineStyle* property and change the visibility by using *ShowAxisLine* property as shown in the below code snippet.

### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.Resources>
    <Style TargetType="Path" x:Key="lineStyle">
      <Setter Property="Stroke" Value="Black"></Setter>
      <Setter Property="StrokeThickness" Value="1"></Setter>
      <Setter Property="StrokeDashArray" Value="7,3"></Setter>
    </Style>
  </syncfusion:SfSmithChart.Resources>
  <syncfusion:SfSmithChart.RadialAxis>
    <syncfusion:RadialAxis ShowAxisLine="True" AxisLineStyle="{StaticResource lineStyle}">
  </syncfusion:RadialAxis>
</syncfusion:SfSmithChart>
```

### C#

```
chart.RadialAxis.AxisLineStyle = this.Grid1.Resources["lineStyle"] as Style;
```



### LabelPlacement

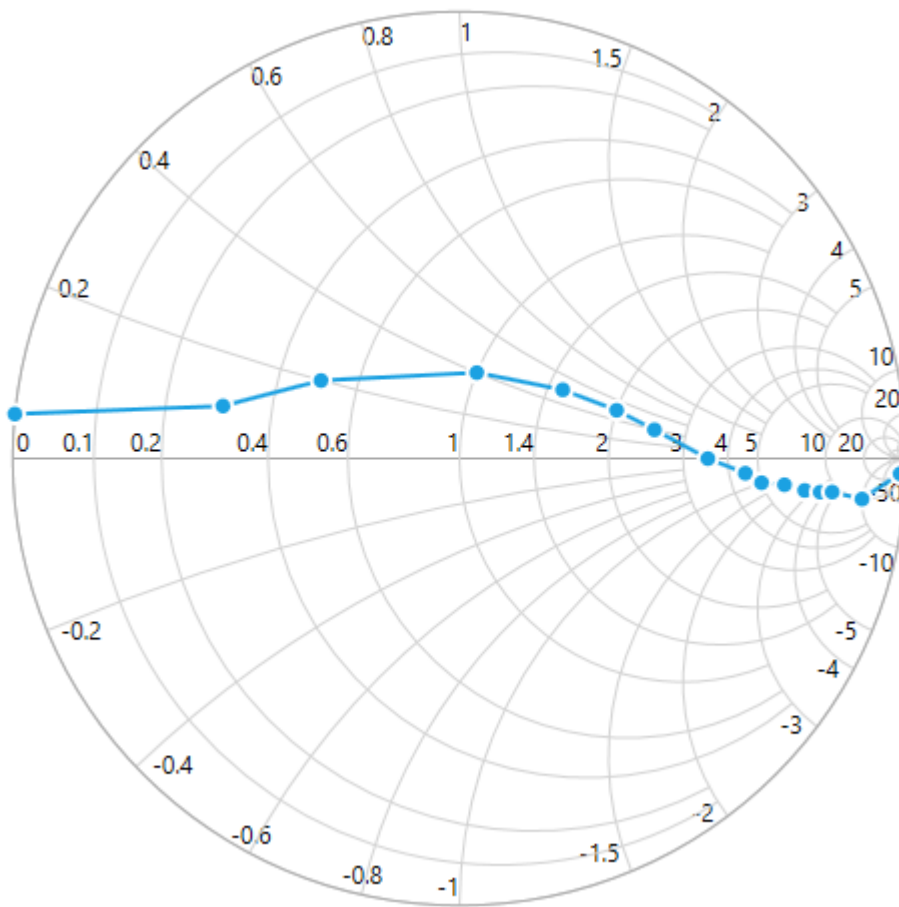
The *LabelPlacement* property is used to position the axis label either inside or outside of the chart plotting area. By default, LabelsPlacement is **Outside**.

### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.RadialAxis>
    <syncfusion:RadialAxis LabelPlacement="Inside">
    </syncfusion:RadialAxis>
  </syncfusion:SfSmithChart.RadialAxis>
</syncfusion:SfSmithChart>
```

### C#

```
//Positioning the radial axis labels to inside of Circle
chart.RadialAxis.LabelPlacement = LabelPlacement.Inside;
```



### LabelIntersectAction

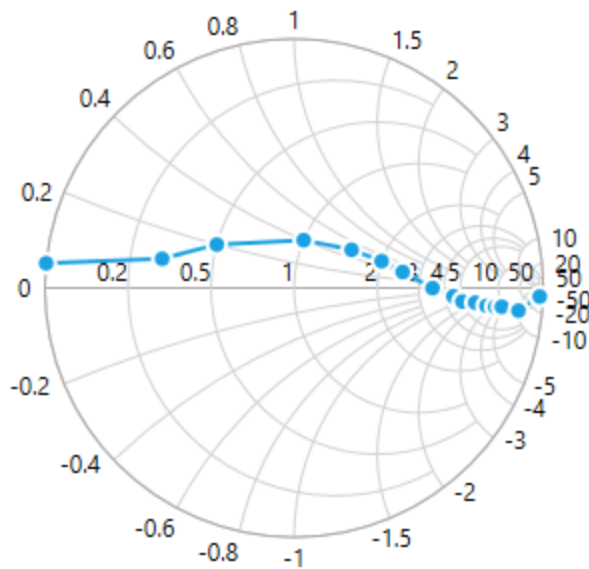
When the Axis labels overlap with each other based on the chart dimensions and label size, use the *LabelIntersectAction* property of the axis to avoid overlapping. The default value of the *LabelIntersectAction* is **Hide**.

### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.RadialAxis>
    <syncfusion:RadialAxis LabelIntersectAction="None">
    </syncfusion:RadialAxis>
  </syncfusion:SfSmithChart.RadialAxis>
</syncfusion:SfSmithChart>
```

### C#

```
//Set axis label intersect action
chart.RadialAxis.LabelIntersectAction = LabelIntersectActions.None;
```

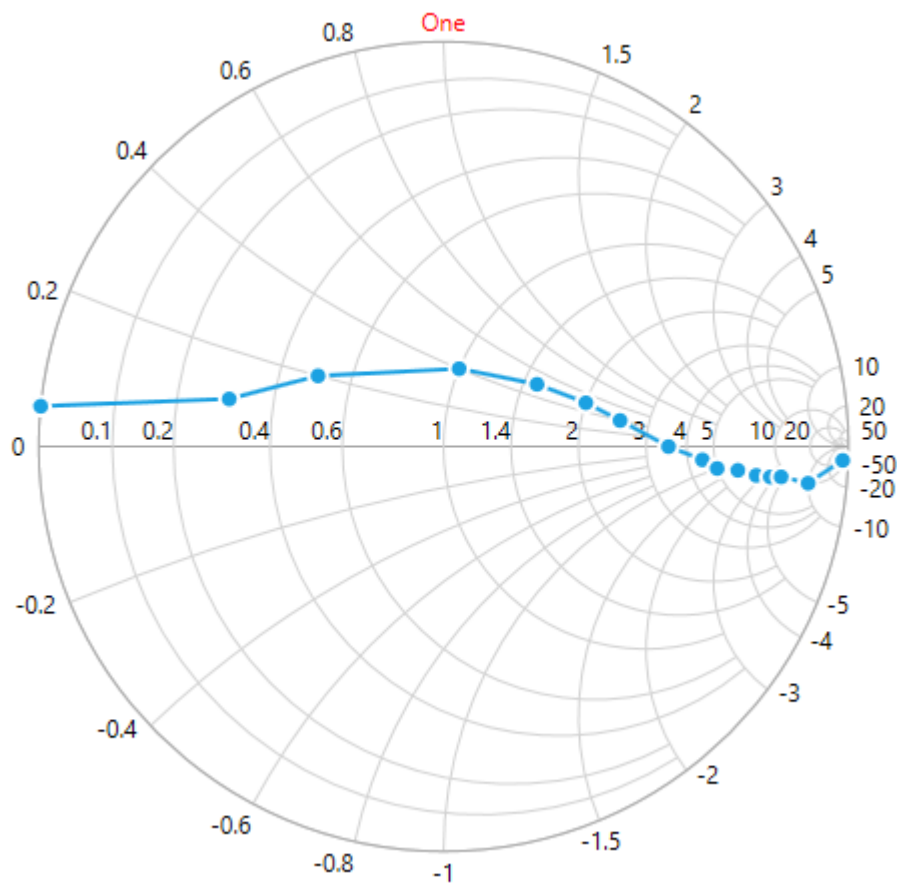


## Events

- **LabelCreated**– Occurs when the label is created.

## C#

```
//Axis label created event hooked for RadialAxis
chart.RadialAxis.LabelCreated += RadialAxis_LabelCreated;
//Event called for every label created in the axis.
void RadialAxis_LabelCreated(object sender, EventArgs e)
{
    var axisLabel = e as ChartAxisLabelEventArgs;
    //Customizing the specific label text and color
    if (axisLabel.Label.Text == "1")
    {
        axisLabel.Label.Text = "One";
        axisLabel.Label.Foreground = new SolidColorBrush(Colors.Red);
    }
}
```



### Series in WPF Smith Chart (SfSmithChart)

Chart series is the visual representation of the given data.

The following APIs are used in line series:

- ResistancePath – A string property that represents the X values for the series.
- ReactancePath – A string property that represents the Y values for the series.
- Interior – Represents the brush for the series color.
- StrokeThickness – Represents the thickness of the series outline.
- Palette – Represents the set of pre-defined or custom colors for the series.

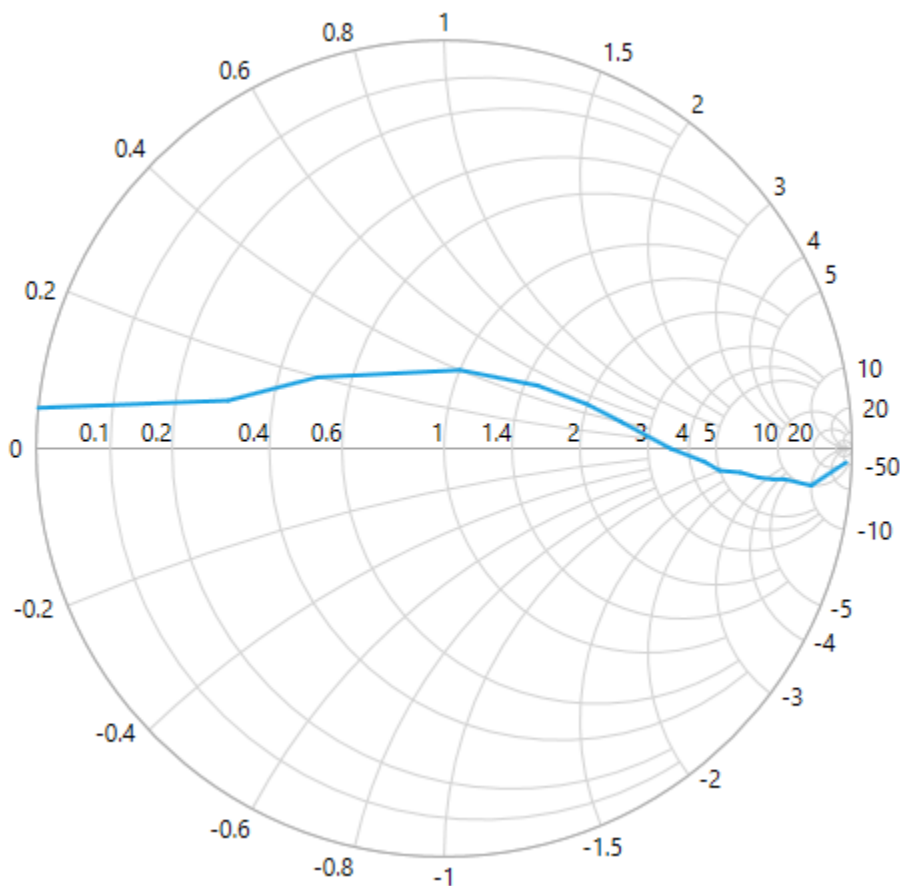
The below code example explains how to create a simple `LineSeries` by using given data

#### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:LineSeries ResistancePath="Resistance" ReactancePath="Reactance"
    ItemsSource="{Binding Data}">
  </syncfusion:LineSeries>
</syncfusion:SfSmithChart>
```

#### C#

```
//Create SfSmithChart instance
SfSmithChart chart = new SfSmithChart();
//Create line series
LineSeries series = new LineSeries();
series.ItemsSource = Data;
series.ResistancePath = "Resistance";
series.ReactancePath = "Reactance";
series.Label = "TransmissionLine";
//Adding series to SmithChart
chart.Series.Add(series);
this.Grid1.Children.Add(chart);
```



### Customizing LineSeries

The line stroke and thickness can be customized by using *Interior* and *StrokeThickness* property of line series.

### XML

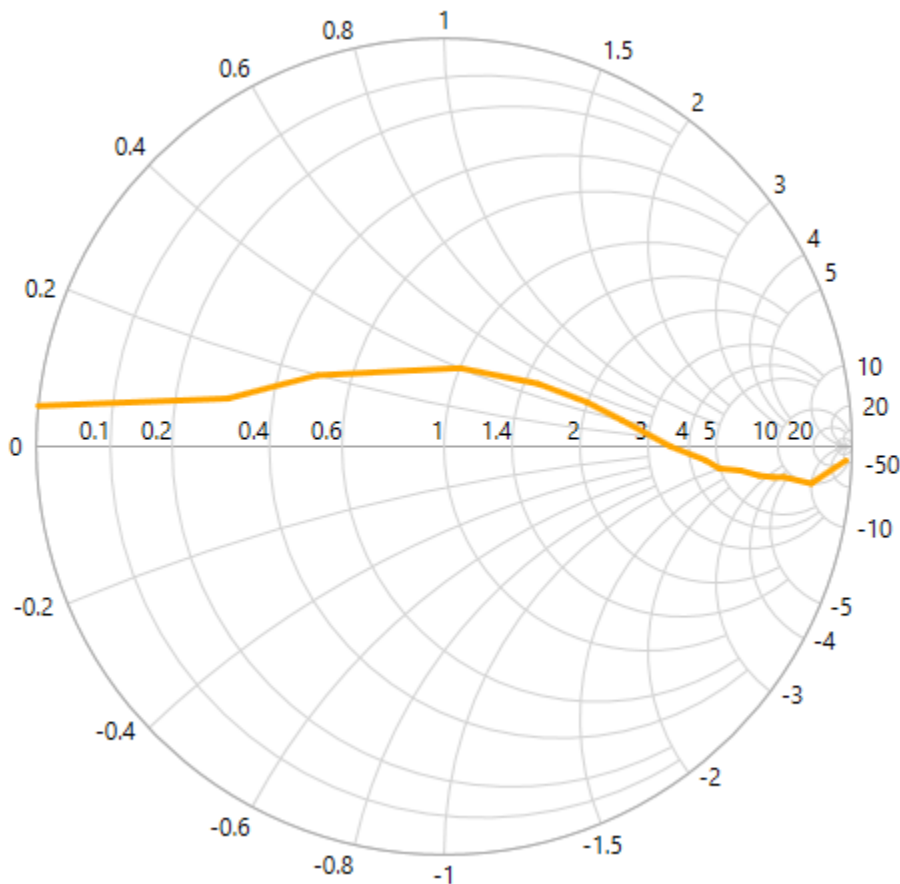
```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:LineSeries Interior="Orange" StrokeThickness="3">
  </syncfusion:LineSeries>
</syncfusion:SfSmithChart>
```

**C#**

```

LineSeries series = new LineSeries();
//Customizing line color and thickness
series.StrokeThickness = 3;
series.Interior = new SolidColorBrush(Colors.Orange);
chart.Series.Add(series);

```

**Animation**

SfSmithChart allows to animate the chart series on loading whenever the *ItemsSource* changes. Animation in the chart can be enabled by setting the *EnableAnimation* property as True and defining the corresponding animation speed with *AnimationDuration* property.

**XML**

```

<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:LineSeries EnableAnimation="True" AnimationDuration="0:0:3">
  </syncfusion:LineSeries>
</syncfusion:SfSmithChart>

```

**C#**

```

LineSeries series = new LineSeries();
series.EnableAnimation = true;

```



```
series.AnimationDuration = TimeSpan.FromSeconds(2);  
chart.Series.Add(series);
```

### Series Visibility

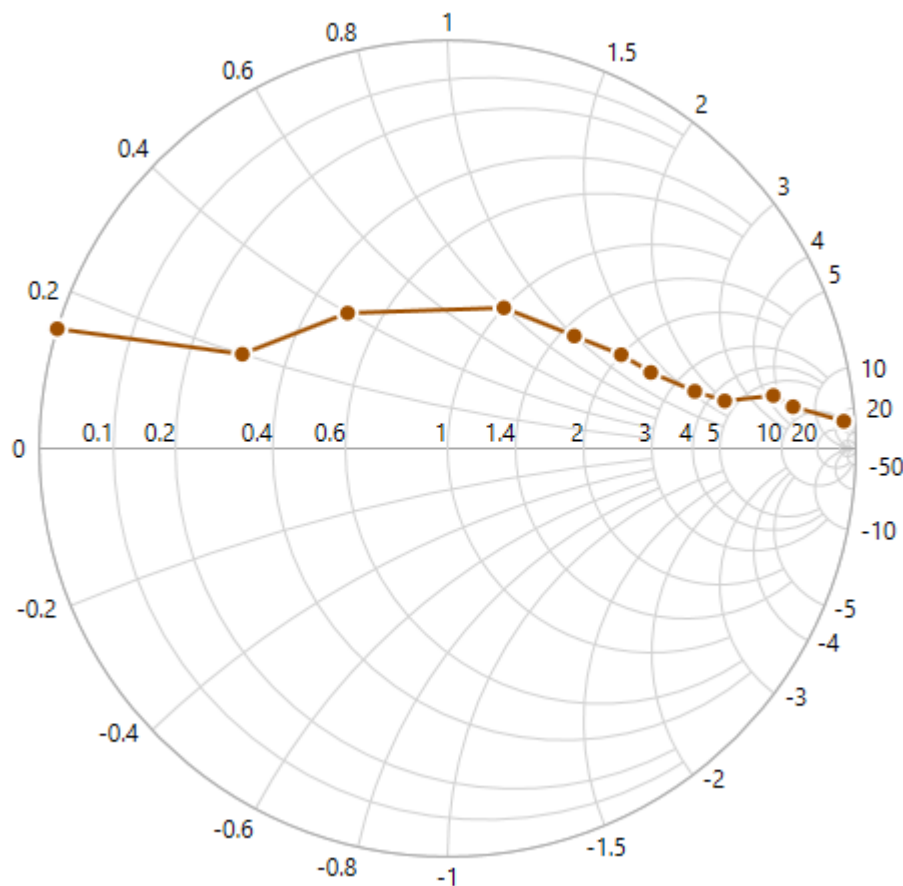
To hide the series segment programmatically, set *IsSeriesVisible* property as False for the specific series.

### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">  
  <syncfusion:LineSeries IsSeriesVisible="False" Label="Transmission-1"  
    ShowMarker="True" ResistancePath="Resistance" ReactancePath="Reactance"  
    ItemsSource="{Binding Data1}">  
  </syncfusion:LineSeries>  
  <syncfusion:LineSeries Label="Transmission-2" ShowMarker="True"  
    ResistancePath="Resistance" ReactancePath="Reactance" ItemsSource="{Binding  
    Data2}">  
  </syncfusion:LineSeries>  
</syncfusion:SfSmithChart>
```

### C#

```
//Create line series1  
LineSeries series1 = new LineSeries();  
series1.Label = "Transmission-1";  
//Hide the series visibility in Chart.  
series1.IsSeriesVisible = false;  
chart.Series.Add(series1);  
//Create line series2  
LineSeries series2 = new LineSeries();  
series2.Label = "Transmission-2";  
chart.Series.Add(series2);
```



### Data Markers in WPF Smith Chart (SfSmithChart)

Data markers are used to provide information about data point to the user. You can add a shape and label to adorn each data point.

#### Add Shapes

Shapes can be added to line series to indicate each data point and it also can be added to the chart by enabling the **ShowMarker** option of the *Series* property. There are different shapes can be added to the chart by using the **MarkerType** option such as rectangle, circle, diamond, etc.

The following code example explains how to enable series marker and add shapes,

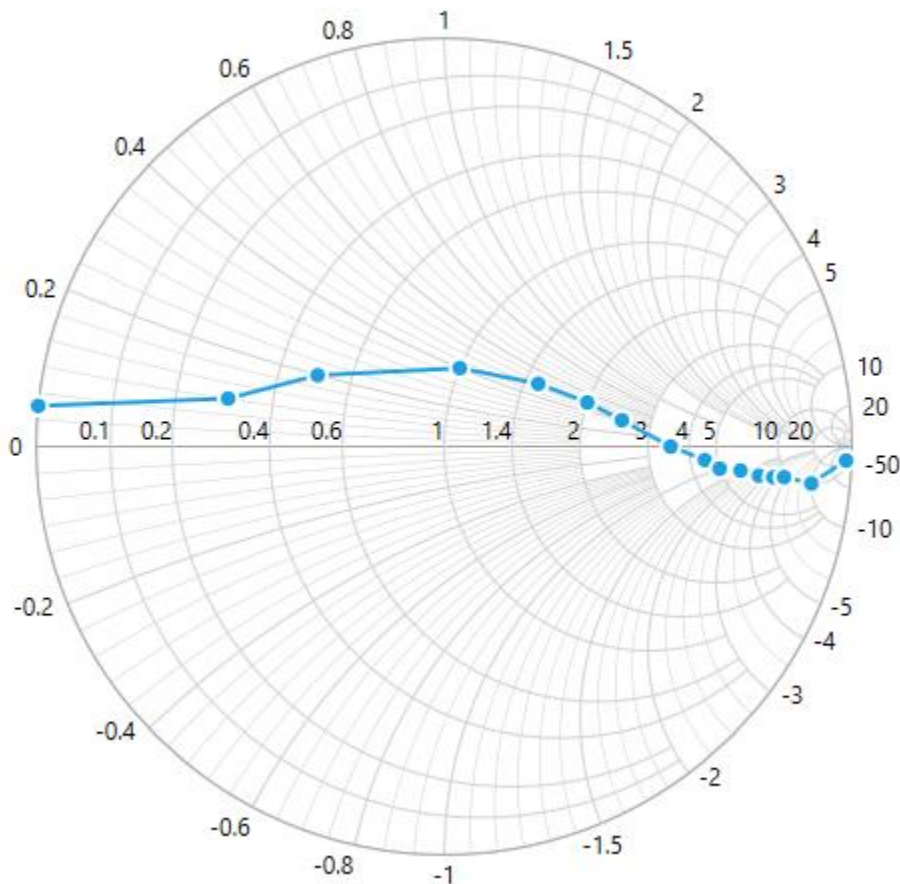
#### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:LineSeries ShowMarker="True" MarkerType="Circle">
  </syncfusion:LineSeries>
</syncfusion:SfSmithChart>
```

#### C#

```
LineSeries series = new LineSeries();
series.ShowMarker = true;
series.MarkerType = MarkerType.Circle;
```

```
chart.Series.Add(series);
```



### Customizing Marker

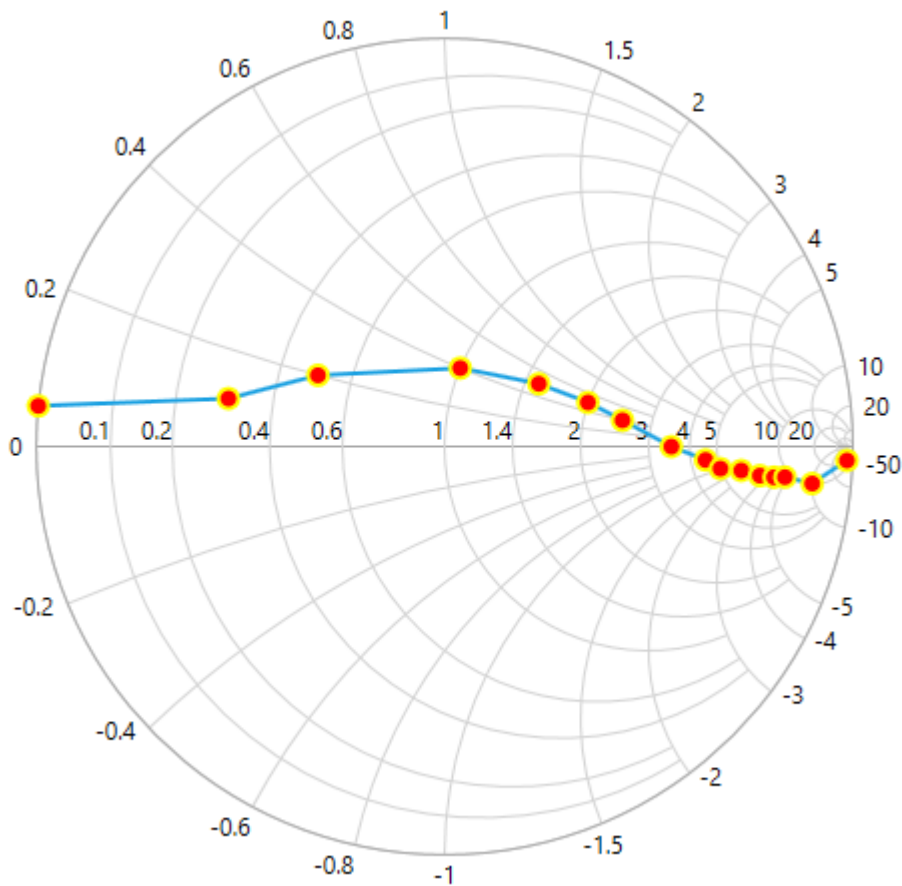
The markers interior, stroke, and size can be customized as demonstrated in the below code snippet.

#### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:LineSeries ShowMarker="True" MarkerType="Circle"
    MarkerHeight="12" MarkerWidth="12" MarkerInterior="Red"
    MarkerStroke="Yellow">
  </syncfusion:LineSeries>
</syncfusion:SfSmithChart>
```

#### C#

```
LineSeries series = new LineSeries();
series.ShowMarker = true;
series.MarkerType = MarkerType.Circle;
series.MarkerHeight = 12;
series.MarkerWidth = 12;
series.MarkerInterior = new SolidColorBrush(Colors.Red);
series.MarkerStroke = new SolidColorBrush(Colors.Yellow);
chart.Series.Add(series);
```



## MarkerTemplate

Apart from the shapes, custom shapes also can be added to mark the data point by using the `MarkerTemplate` property. To add custom shapes, define the `MarkerType` as **Custom**.

The following code example illustrates how to add custom shapes,

### XML

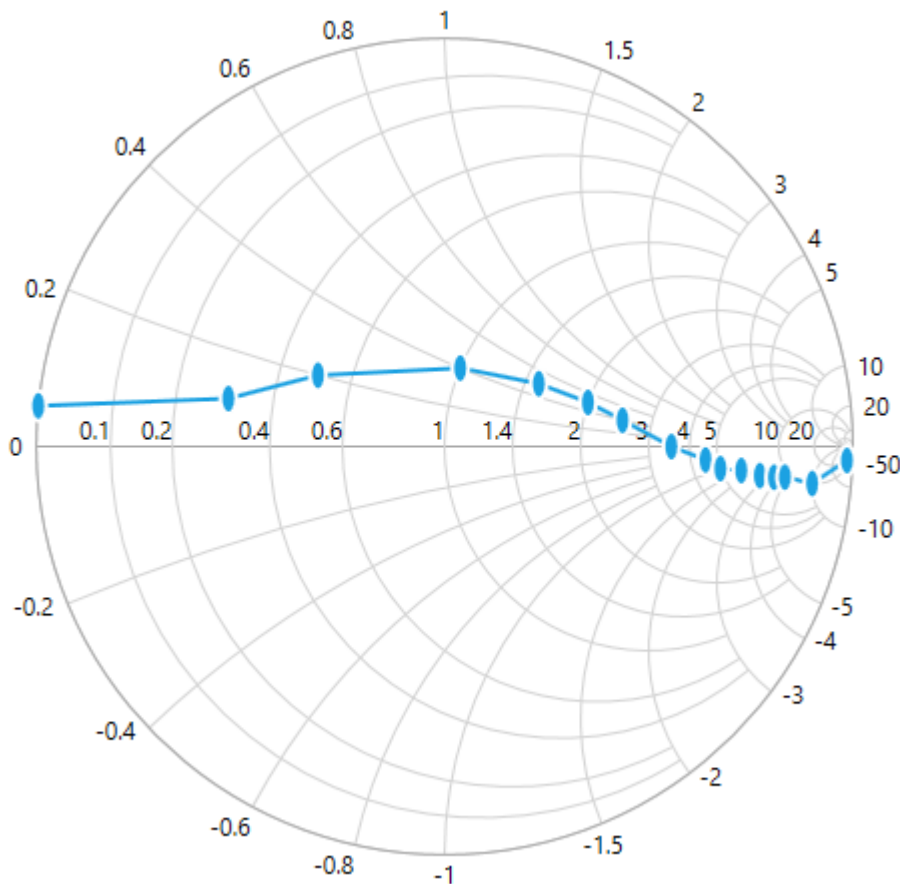
```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.Resources>
    <DataTemplate x:Key="Ellipse">
      <Ellipse Stretch="Fill" Fill="{Binding Interior}" Stroke="{Binding Stroke}"
        StrokeThickness="2" Width="10" Height="17" />
    </DataTemplate>
  </syncfusion:SfSmithChart.Resources>
  <syncfusion:LineSeries ShowMarker="True" MarkerType="Custom"
    MarkerTemplate="{StaticResource Ellipse}">
  </syncfusion:LineSeries>
</syncfusion:SfSmithChart>
```

### C#

```

LineSeries series = new LineSeries();
series.ShowMarker = true;
series.MarkerType = MarkerType.Custom;
series.MarkerTemplate = this.Grid1.Resources["Ellipse"] as DataTemplate;
chart.Series.Add(series);

```



### Add Labels

Data label can be added to a chart series by setting the `ShowLabel` property as `True` in the series **DataLabel** option. By default, the data labels are displayed on top of the data point and it can be automatically adjusted its position when collide with another label.

### XML

```

<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:LineSeries ItemsSource="{Binding Data}"
    ResistancePath="Resistance" ReactancePath="Reactance">
    <syncfusion:LineSeries.DataLabel>
      <syncfusion:DataLabel ShowLabel="True"></syncfusion:DataLabel>
    </syncfusion:LineSeries.DataLabel>
  </syncfusion:LineSeries>
</syncfusion:SfSmithChart>

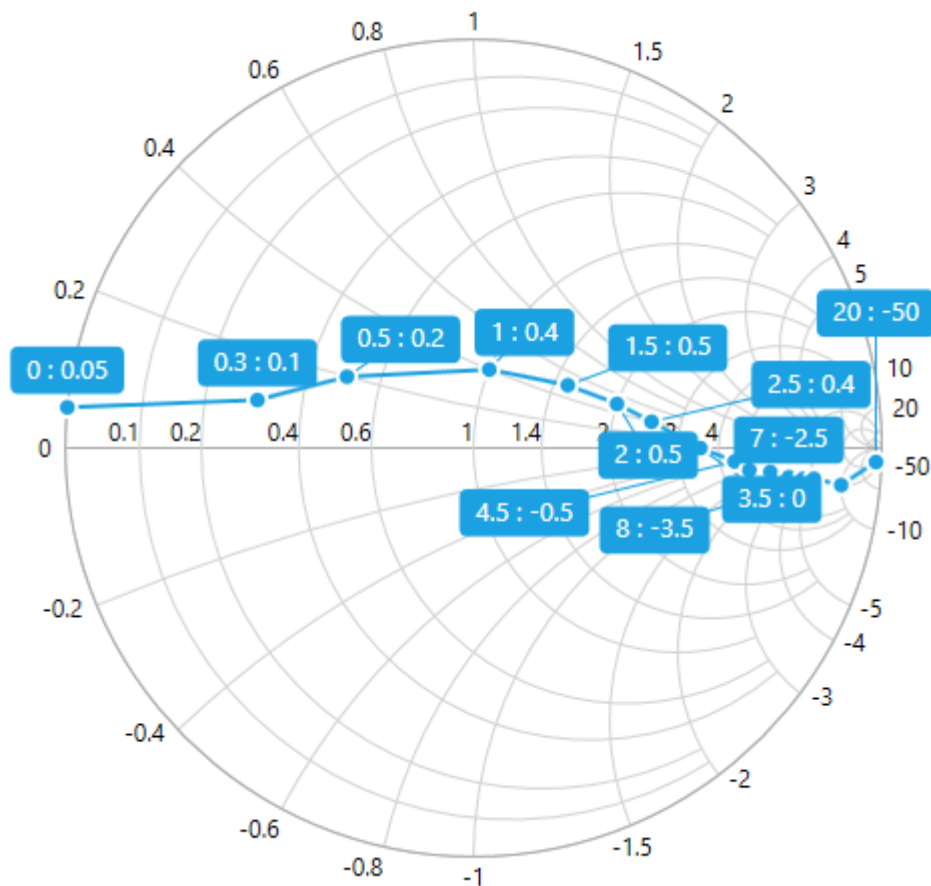
```

**C#**

```
LineSeries series = new LineSeries();
series.DataLabel.ShowLabel = true;
chart.Series.Add(series);
```

**Note:** The position changed data label will be connected by using connector line and the label will be hidden if there is no place to position it around the data point.

The following screenshot illustrates how the data labels can be positioned.

**LabelStyle**

The style for the data label can be defined by using *LabelStyle* property of **DataLabel**.

**XML**

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.Resources>
    <Style TargetType="TextBlock" x:Key="labelStyle">
      <Setter Property="Foreground" Value="Yellow"/>
      <Setter Property="FontSize" Value="12"/>
      <Setter Property="FontFamily" Value="Calibri"/>
      <Setter Property="FontWeight" Value="Bold"/>
    </Style>
```

```

</syncfusion:SfSmithChart.Resources>
<syncfusion:LineSeries>
<syncfusion:LineSeries.DataLabel>
<syncfusion:DataLabel ShowLabel="True" LabelStyle="{StaticResource
labelStyle}">
</syncfusion:DataLabel>
</syncfusion:LineSeries.DataLabel>
</syncfusion:LineSeries>
</syncfusion:SfSmithChart>

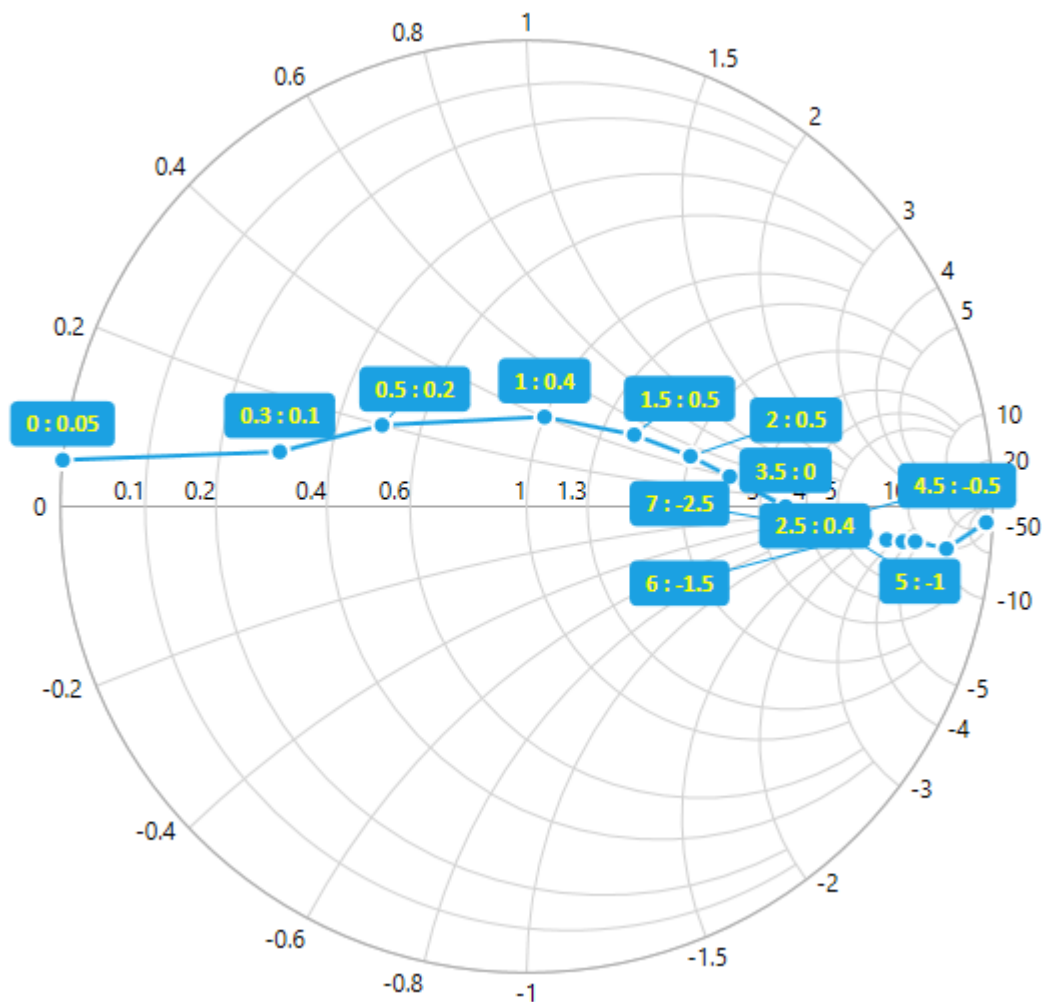
```

## C#

```

LineSeries series = new LineSeries();
series.DataLabel.ShowLabel = true;
series.DataLabel.LabelStyle = this.Grid1.Resources["labelStyle"] as Style;
chart.Series.Add(series);

```



## LabelTemplate

The label content can be formatted and customized by using *LabelTemplate* property of **DataLabel**. The following code example illustrates how the label content can be formatted and customized.

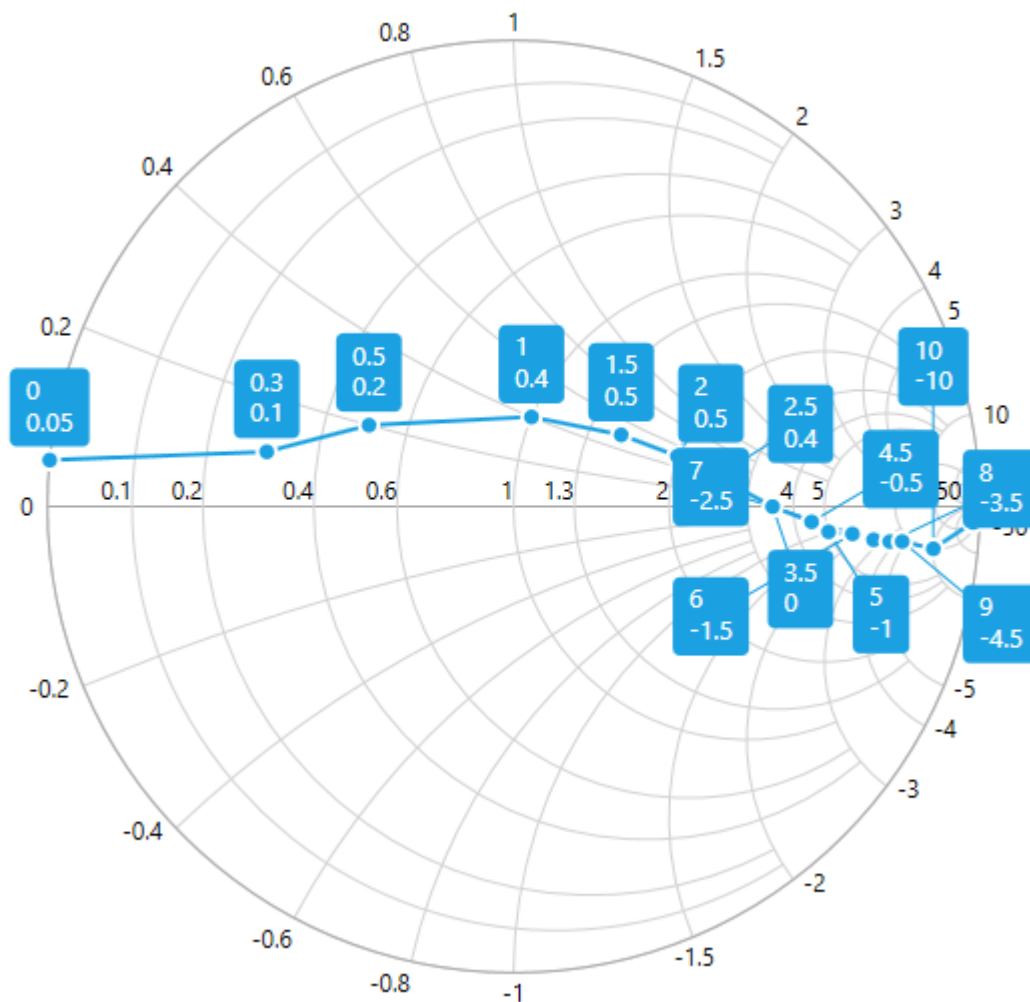
#### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.Resources>
    <DataTemplate x:Key="labelTemplate">
      <Border CornerRadius="4" Background="{Binding Background}"
        BorderThickness="1" Padding="8,4,8,4" BorderBrush="{Binding BorderBrush}">
        <Grid>
          <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
          </Grid.RowDefinitions>
          <TextBlock Grid.Row="0" Text="{Binding Resistance}" Style="{Binding
            LabelStyle}"/>
          <TextBlock Grid.Row="1" Text="{Binding Reactance}" Style="{Binding
            LabelStyle}"/>
        </Grid>
      </Border>
    </DataTemplate>
  </syncfusion:SfSmithChart.Resources>
  <syncfusion:LineSeries ShowMarker="True" ItemsSource="{Binding Data}"
    ResistancePath="Resistance" ReactancePath="Reactance">
    <syncfusion:LineSeries.DataLabel>
      <syncfusion:DataLabel ShowLabel="True" LabelTemplate="{StaticResource
        labelTemplate}"></syncfusion:DataLabel>
    </syncfusion:LineSeries.DataLabel>
  </syncfusion:LineSeries>
</syncfusion:SfSmithChart>
```

#### C#

```
LineSeries series = new LineSeries();
series.DataLabel.ShowLabel = true;
series.DataLabel.LabelTemplate = this.Grid1.Resources["labelTemplate"] as
DataTemplate;
chart.Series.Add(series);
```





### Legend in WPF Smith Chart (SfSmithChart)

The legend contains the list of chart series that appear in a SmithChart. It can be defined by using the following code example.

**Note:** Add name to **Label** property of **Series**, which in turn mapped to the Legend.

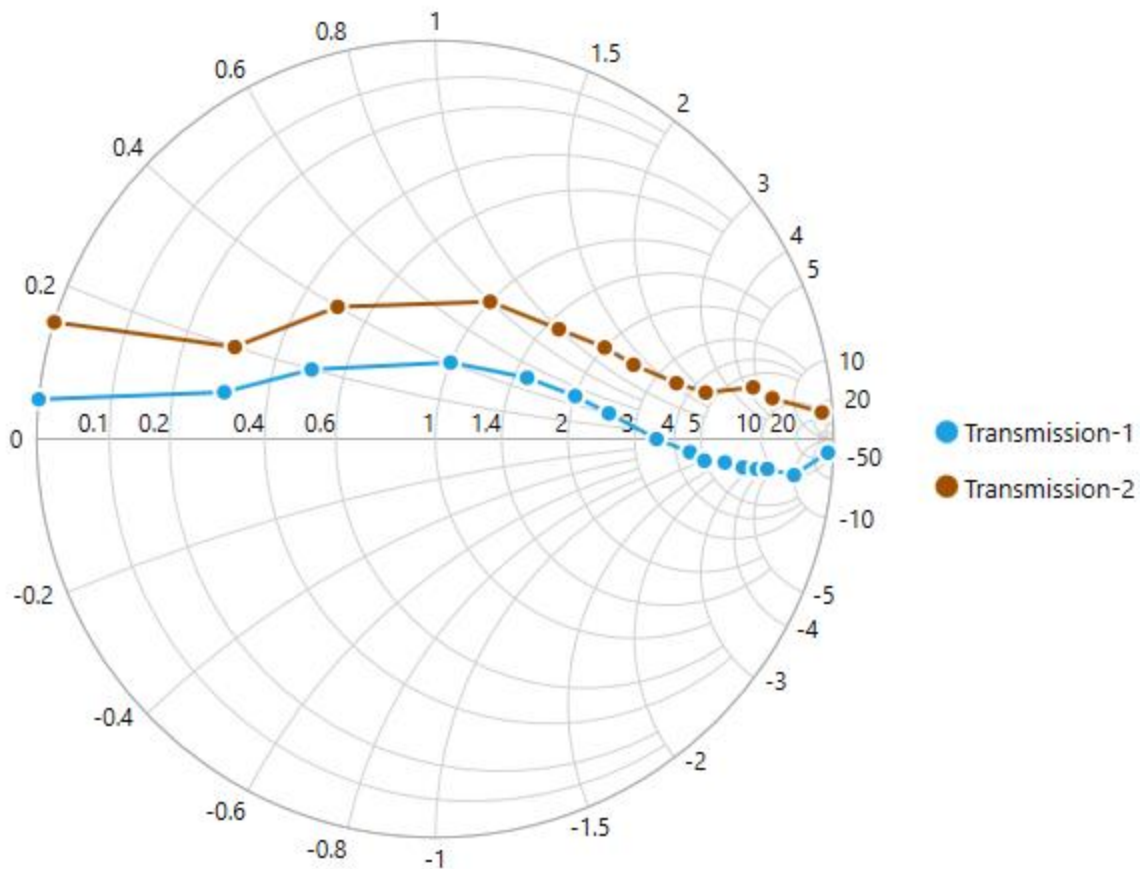
#### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:LineSeries Label="Transmission-1" ShowMarker="True"
    ResistancePath="Resistance" ReactancePath="Reactance" ItemsSource="{Binding
    Data1}">
  </syncfusion:LineSeries>
  <syncfusion:LineSeries Label="Transmission-2" ShowMarker="True"
    ResistancePath="Resistance" ReactancePath="Reactance" ItemsSource="{Binding
    Data2}">
  </syncfusion:LineSeries>
  <!--Adding Legend to SmithChart-->
  <syncfusion:SfSmithChart.Legend>
    <syncfusion:SmithChartLegend></syncfusion:SmithChartLegend>
  </syncfusion:SfSmithChart.Legend>
```

```
</syncfusion:SfSmithChart>
```

**C#**

```
SfSmithChart chart = new SfSmithChart();  
//Create line series1  
LineSeries series1 = new LineSeries();  
//Display the legend text for the series.  
series1.Label = "Transmission-1";  
series1.ItemsSource = Data1;  
series1.ResistancePath = "Resistance";  
series1.ReactancePath = "Reactance";  
series1.ShowMarker = true;  
chart.Series.Add(series1);  
//Create line series2  
LineSeries series2 = new LineSeries();  
//Display the legend text for the series.  
series2.Label = "Transmission-2";  
series2.ItemsSource = Data2;  
series2.ResistancePath = "Resistance";  
series2.ReactancePath = "Reactance";  
series2.ShowMarker = true;  
chart.Series.Add(series2);  
//Adding legend to the SmithChart  
SmithChartLegend legend = new SmithChartLegend();  
chart.Legend = legend;  
this.Grid1.Children.Add(chart);
```



### Positioning the Legend

#### Docking

Legends can be docked at left, right, and top or bottom around the chart area by using *DockPosition* property.

By default, the Smith chart Legend is docked at the Right of the chart.

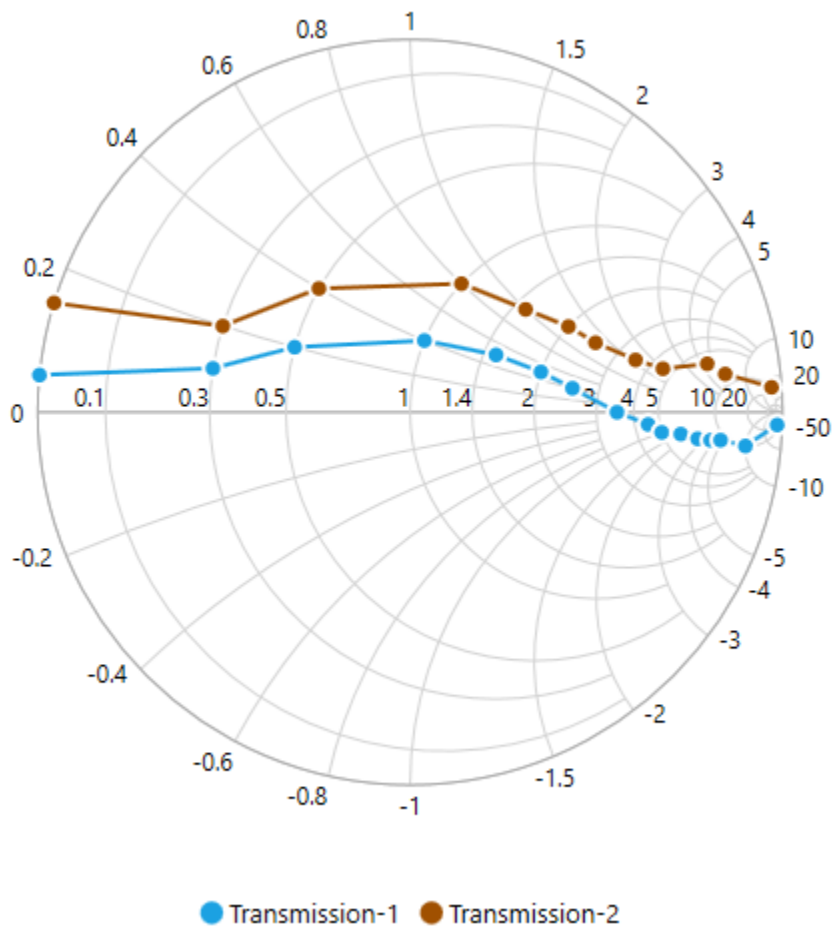
To display the legend at the bottom, set the *DockPosition* as Bottom as in below code snippet.

#### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <!--Adding Legend to SmithChart-->
  <syncfusion:SfSmithChart.Legend>
    <syncfusion:SmithChartLegend DockPosition="Bottom">
    </syncfusion:SmithChartLegend>
  </syncfusion:SfSmithChart.Legend>
</syncfusion:SfSmithChart>
```

#### C#

```
//Adding legend to the SmithChart
SmithChartLegend legend = new SmithChartLegend();
legend.DockPosition = ChartDock.Bottom;
chart.Legend = legend;
```



### Floating Legends

To position the legend at any arbitrary location inside the chart, set *DockPosition* as **Floating** and provide its relative position by using *OffsetX* and *OffsetY* properties.

Offset specifies x or y distance from origin.

### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <!--Adding legend to SmithChart-->
  <syncfusion:SfSmithChart.Legend>
    <syncfusion:SmithChartLegend DockPosition="Floating" OffsetX="200"
    OffsetY="120">
    </syncfusion:SmithChartLegend>
  </syncfusion:SfSmithChart.Legend>
</syncfusion:SfSmithChart>
```

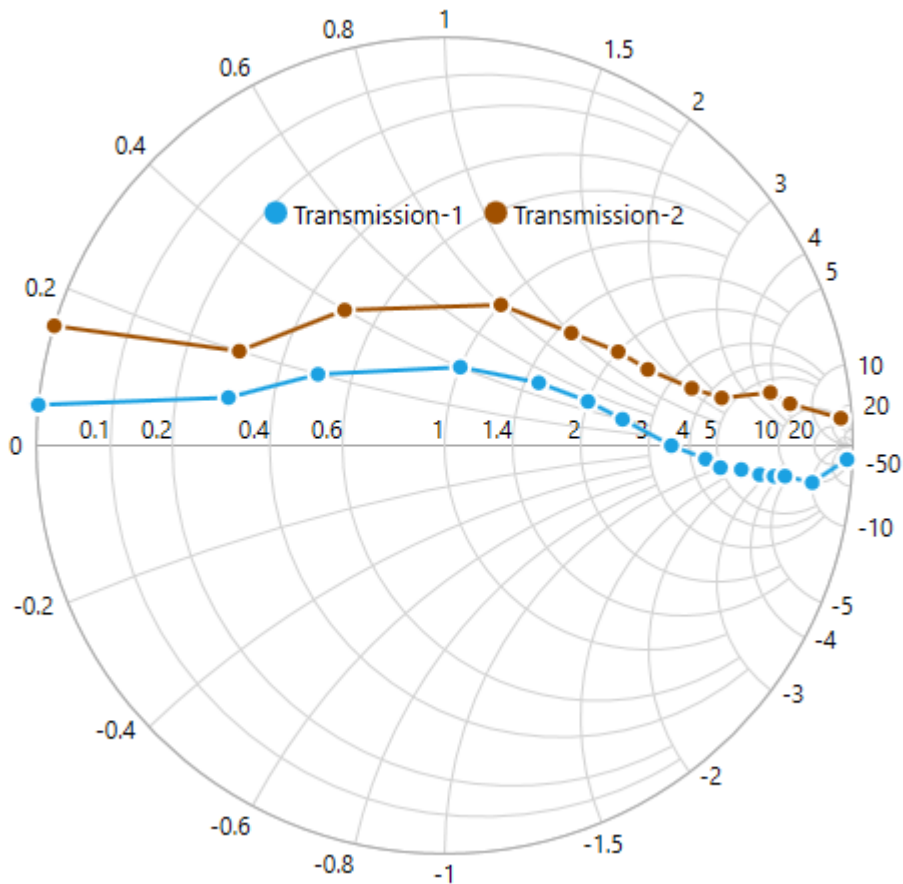
### C#

```
//Adding legend to the SmithChart
SmithChartLegend legend = new SmithChartLegend();
legend.DockPosition = ChartDock.Floating;
```

```

legend.OffsetX = 200;
legend.OffsetY = 120;
chart.Legend = legend;

```



### Legend Icon

Represents the symbol associated with each legend item. By default, the legend icon is Circle.

Legend Icon can be customized by using the `LegendIcon` property in Smith chart legend as in the below code snippet:

### XML

```

<syncfusion:SfSmithChart x:Name="SmithChart">
  <!--Adding legend to SmithChart-->
  <syncfusion:SfSmithChart.Legend>
    <syncfusion:SmithChartLegend LegendIcon="HorizontalLine">
    </syncfusion:SmithChartLegend>
  </syncfusion:SfSmithChart.Legend>
</syncfusion:SfSmithChart>

```

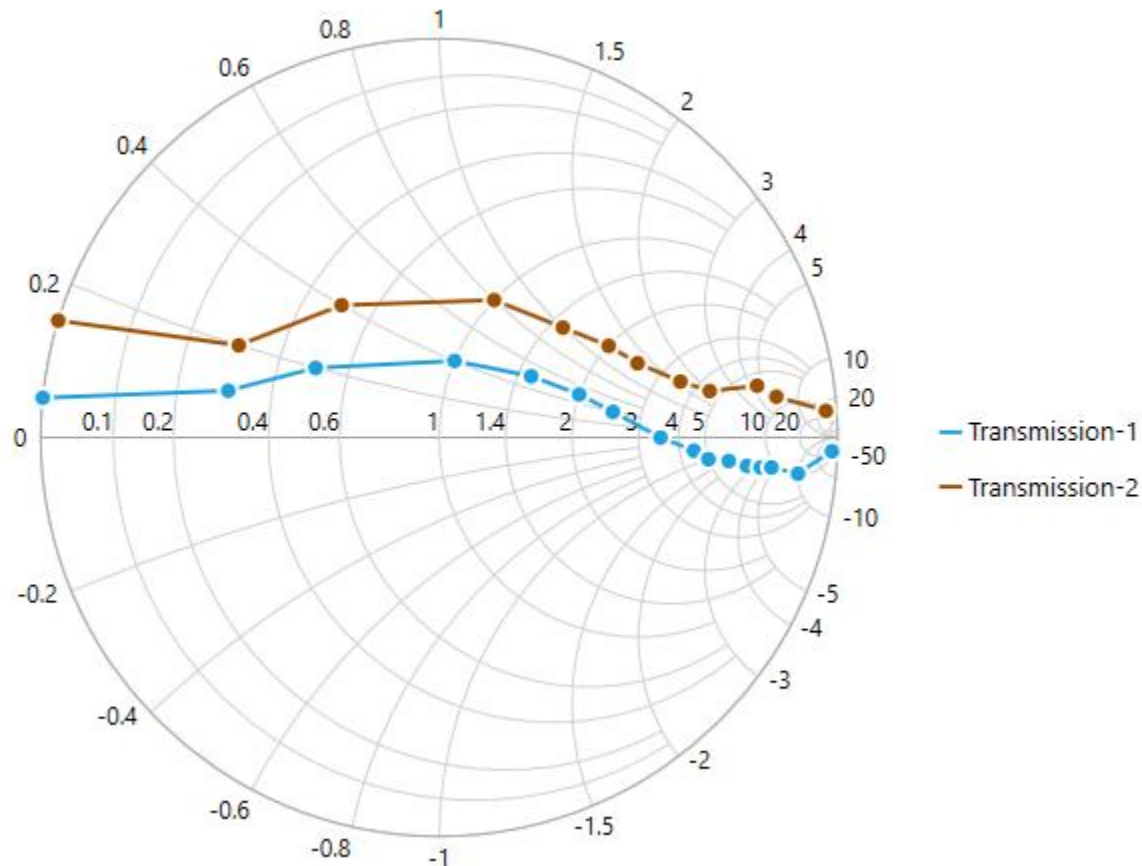
### C#

```

//Adding legend to the SmithChart

```

```
SmithChartLegend legend = new SmithChartLegend();
legend.LegendIcon = SmithChartLegendIcon.HorizontalLine;
chart.Legend = legend;
```



### Custom Legend Icon

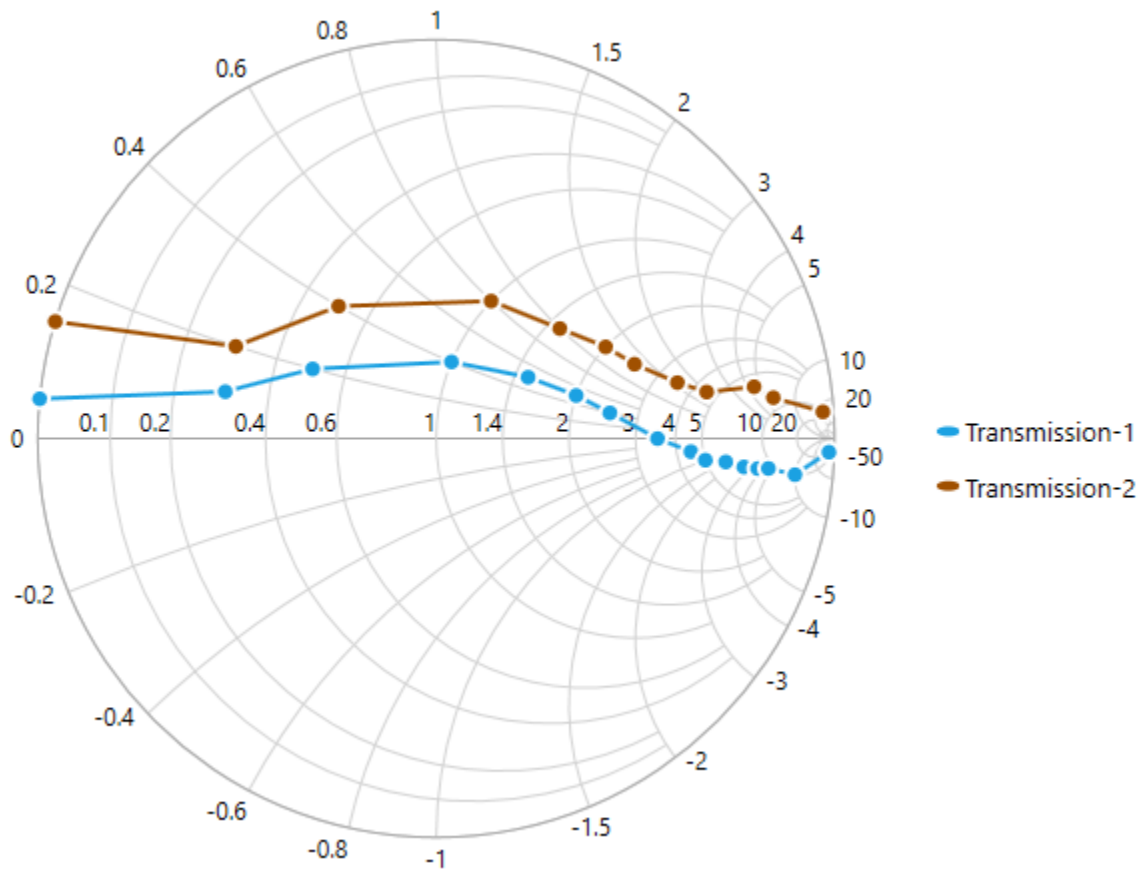
Custom icon for the legend can be added by using **LegendIconTemplate** property in Smith chart Legend as in below code example.

### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.Resources>
    <DataTemplate x:Key="Ellipse">
      <Ellipse Stretch="Fill" Fill="{Binding Interior}" Width="12" Height="5" />
    </DataTemplate>
  </syncfusion:SfSmithChart.Resources>
  <!--Adding custom legend icon to SmithChart-->
  <syncfusion:SfSmithChart.Legend>
    <syncfusion:SmithChartLegend LegendIcon="Custom"
      LegendIconTemplate="{StaticResource Ellipse}">
    </syncfusion:SmithChartLegend>
  </syncfusion:SfSmithChart.Legend>
</syncfusion:SfSmithChart>
```

**C#**

```
SmithChartLegend legend = new SmithChartLegend();
//Adding custom legend icon to SmithChart
legend.LegendIcon = SmithChartLegendIcon.Custom;
legend.LegendIconTemplate = this.Grid1.Resources["Ellipse"] as DataTemplate;
chart.Legend = legend;
```



## Customizing Legend

The following code example illustrates the customization of legend icon and text.

**XML**

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.Legend>
    <syncfusion:SmithChartLegend IconHeight="15" IconWidth="15" ItemMargin="10"
      FontWeight="Bold" Foreground="Red" FontSize="13"
      Background="LightGray" BorderBrush="CadetBlue"
      BorderThickness="3">
    </syncfusion:SmithChartLegend>
  </syncfusion:SfSmithChart.Legend>
</syncfusion:SfSmithChart>
```

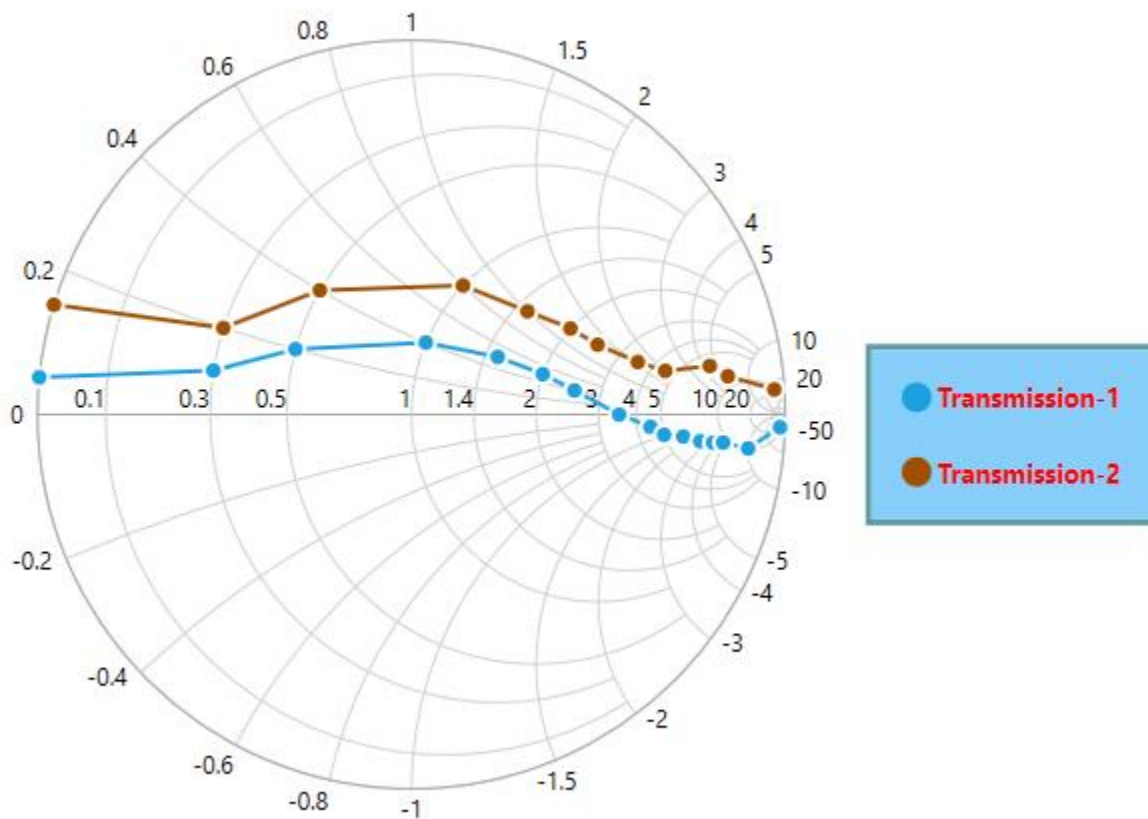
**C#**

```
//Adding legend to the Smith chart
```

```

SmithChartLegend legend = new SmithChartLegend();
//Customizing the Smith chart legend icon
legend.IconHeight = 15;
legend.IconWidth = 15;
legend.ItemMargin = new Thickness(10);
//Customizing legend text
legend.Foreground = new SolidColorBrush(Colors.Red);
legend.FontSize = 13;
legend.FontWeight = FontWeights.Bold;
legend.Background = new SolidColorBrush(Colors.LightSkyBlue);
legend.BorderBrush = new SolidColorBrush(Colors.CadetBlue);
legend.BorderThickness = new Thickness(3);
chart.Legend = legend;

```



### VisibilityOnLegend

To limit the number of series to be displayed in chart, use *VisibilityOnLegend* property as shown in the below code example.

### XML

```

<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:LineSeries VisibilityOnLegend="False" Label="Transmission-1"
    ShowMarker="True" ResistancePath="Resistance" ReactancePath="Reactance"
    ItemsSource="{Binding Data1}">
  </syncfusion:LineSeries>

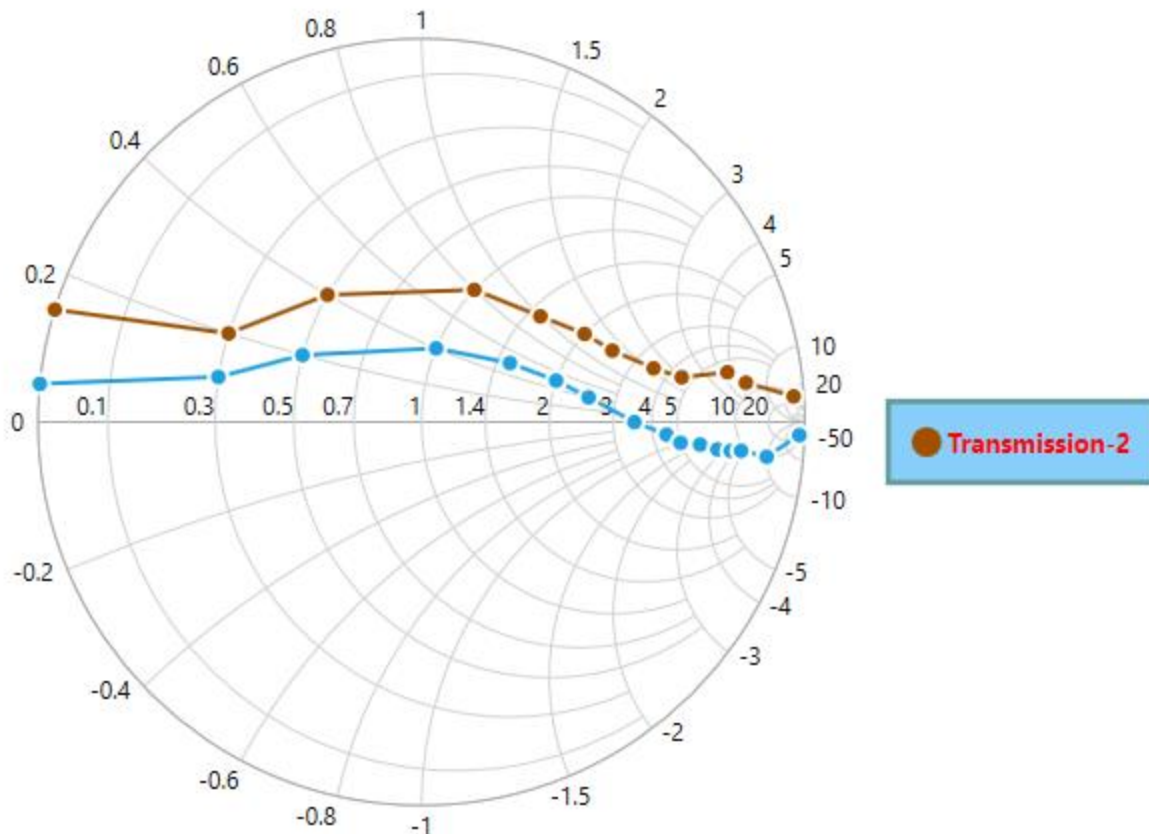
```



```
<syncfusion:LineSeries Label="Transmission-2" ShowMarker="True"
ResistancePath="Resistance" ReactancePath="Reactance" ItemsSource="{Binding
Data2}">
</syncfusion:LineSeries>
<syncfusion:SfSmithChart.Legend>
<syncfusion:SmithChartLegend IconHeight="15" IconWidth="15"
FontWeight="Bold" Foreground="Red" FontSize="13"
Background="LightGray" BorderBrush="CadetBlue"
BorderThickness="3">
</syncfusion:SmithChartLegend>
</syncfusion:SfSmithChart.Legend>
</syncfusion:SfSmithChart>
```

## C#

```
//Create line series1
LineSeries series1 = new LineSeries();
series1.Label = "Transmission-1";
//Remove the LegendItem from legend collections.
series1.VisibilityOnLegend = false;
chart.Series.Add(series1);
//Create line series2
LineSeries series2 = new LineSeries();
series2.Label = "Transmission-2";
chart.Series.Add(series2);
//Adding legend to the SmithChart
SmithChartLegend legend = new SmithChartLegend();
chart.Legend = legend;
```



### Series Visibility

To hide the series segment programmatically, set *IsSeriesVisible* property as False for the specific series. After the property has been set, the legend item for the specific series will be displayed with shade.

Now, you can show/hide the series segment on chart by clicking on the particular legend item.

### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:LineSeries IsSeriesVisible="False" Label="Transmission-1"
    ShowMarker="True" ResistancePath="Resistance" ReactancePath="Reactance"
    ItemsSource="{Binding Data1}">
  </syncfusion:LineSeries>
  <syncfusion:LineSeries Label="Transmission-2" ShowMarker="True"
    ResistancePath="Resistance" ReactancePath="Reactance" ItemsSource="{Binding
    Data2}">
  </syncfusion:LineSeries>
  <syncfusion:SfSmithChart.Legend>
  <syncfusion:SmithChartLegend>
  </syncfusion:SmithChartLegend>
  </syncfusion:SfSmithChart.Legend>
</syncfusion:SfSmithChart>
```

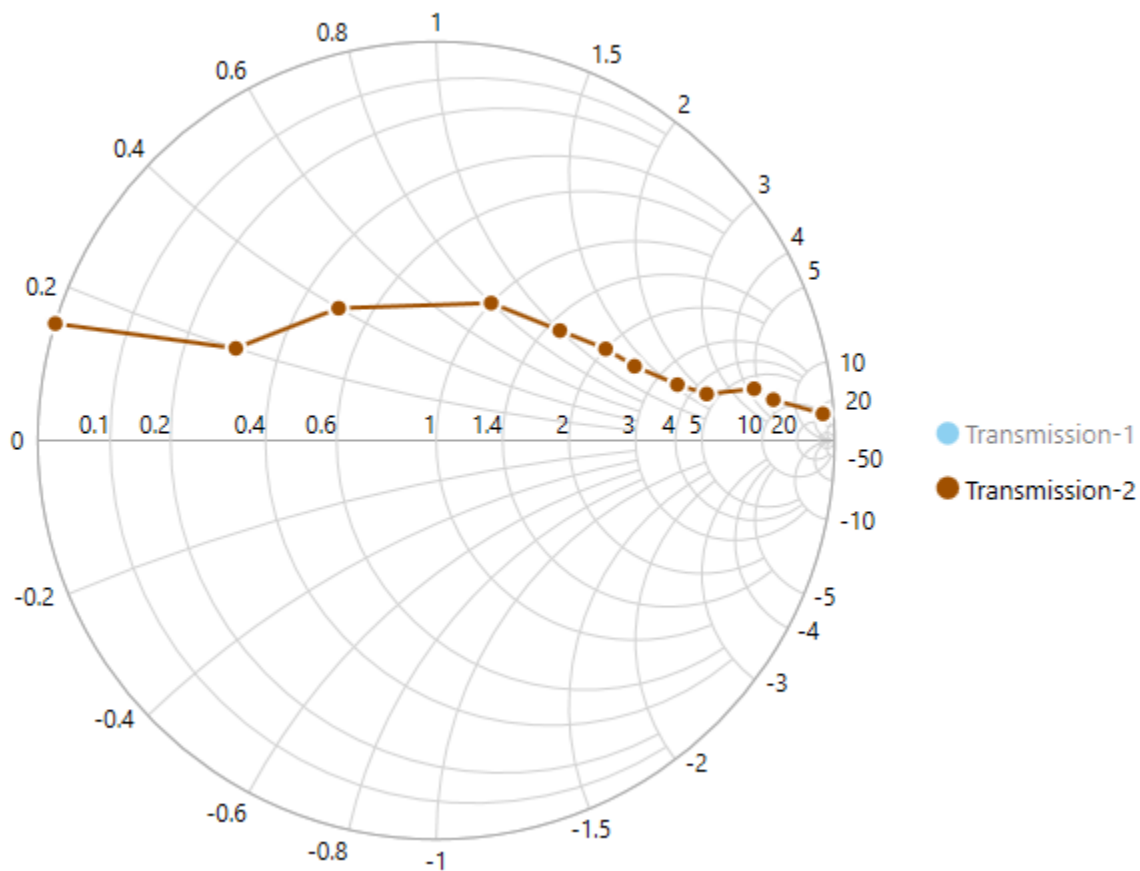
### C#

```
//Create line series1
```

```

LineSeries series1 = new LineSeries();
series1.Label = "Transmission-1";
//Hide the series visibility in Chart.
series1.IsSeriesVisible = false;
chart.Series.Add(series1);
//Create line series2
LineSeries series2 = new LineSeries();
series2.Label = "Transmission-2";
chart.Series.Add(series2);
//Adding legend to the SmithChart
SmithChartLegend legend = new SmithChartLegend();
chart.Legend = legend;

```



## Appearance in WPF Smith Chart (SfSmithChart)

### SmithChart Palette

The Smith chart displays different series in different color by using [Palette](#) property of [ColorModel](#). By default, Metro palette color has been applied.

### XML

```

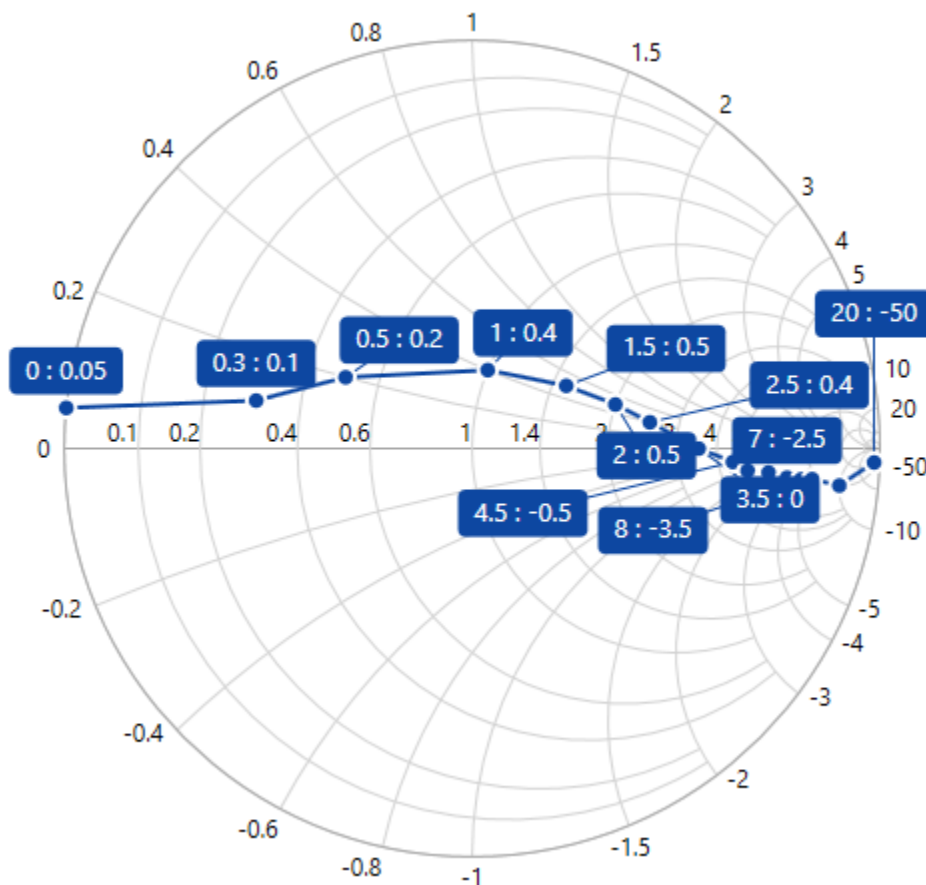
<syncfusion:SfSmithChart x:Name="SmithChart">
<syncfusion:SfSmithChart.ColorModel>

```

```
<syncfusion:SmithChartColorModel
  Palette="BlueChrome"></syncfusion:SmithChartColorModel>
</syncfusion:SfSmithChart.ColorModel>
</syncfusion:SfSmithChart>
```

**C#**

```
SfSmithChart chart = new SfSmithChart();
chart.ColorModel = new SmithChartColorModel();
chart.ColorModel.Palette = ColorPalette.BlueChrome;
```

**Series Palette**

The palette color to each data points of specific series can be defined by using [Palette](#) property of [ColorModel](#) in the Series.

**XML**

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:LineSeries>
    <syncfusion:LineSeries.ColorModel>
      <syncfusion:SmithChartColorModel Palette="Metro">
      </syncfusion:SmithChartColorModel>
    </syncfusion:LineSeries.ColorModel>
  </syncfusion:LineSeries>
</syncfusion:SfSmithChart>
```

```
</syncfusion:SfSmithChart>
```

**C#**

```
LineSeries series = new LineSeries();
series.ColorModel = new SmithChartColorModel();
series.ColorModel.Palette = ColorPalette.Metro;
chart.Series.Add(series);
```

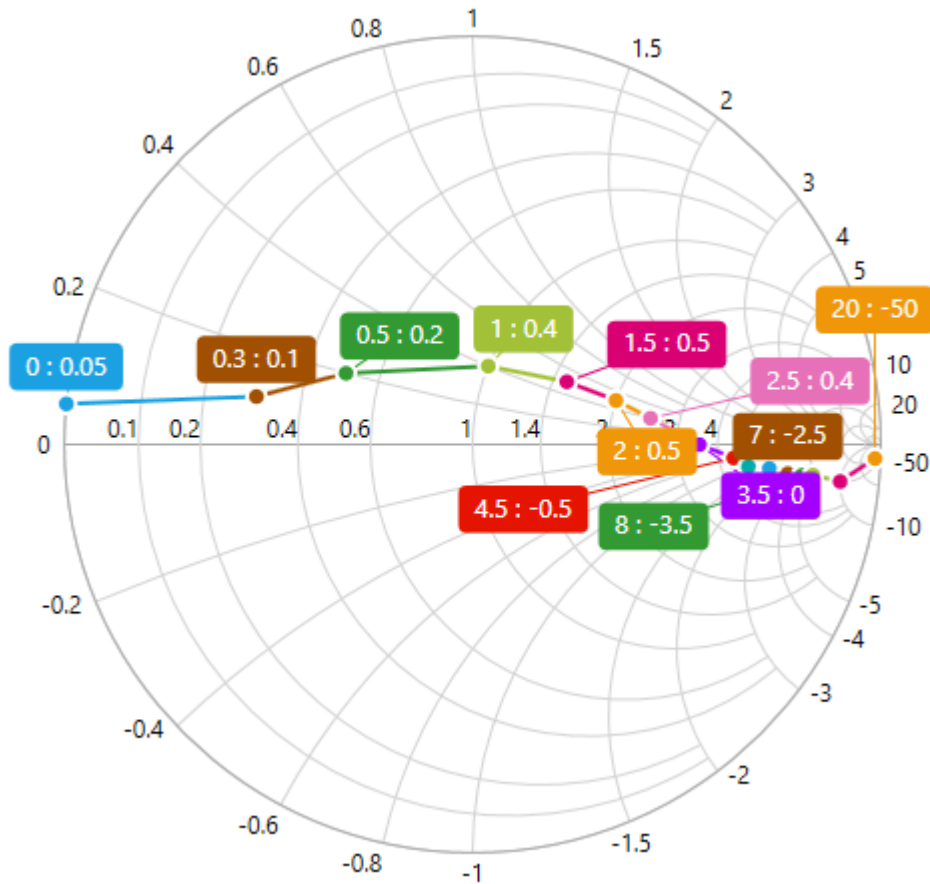
**Chart Area Customization**

Chart and chart area (circle plotting area) can be customized by using the below properties in [SfSmithChart](#).

**XML**

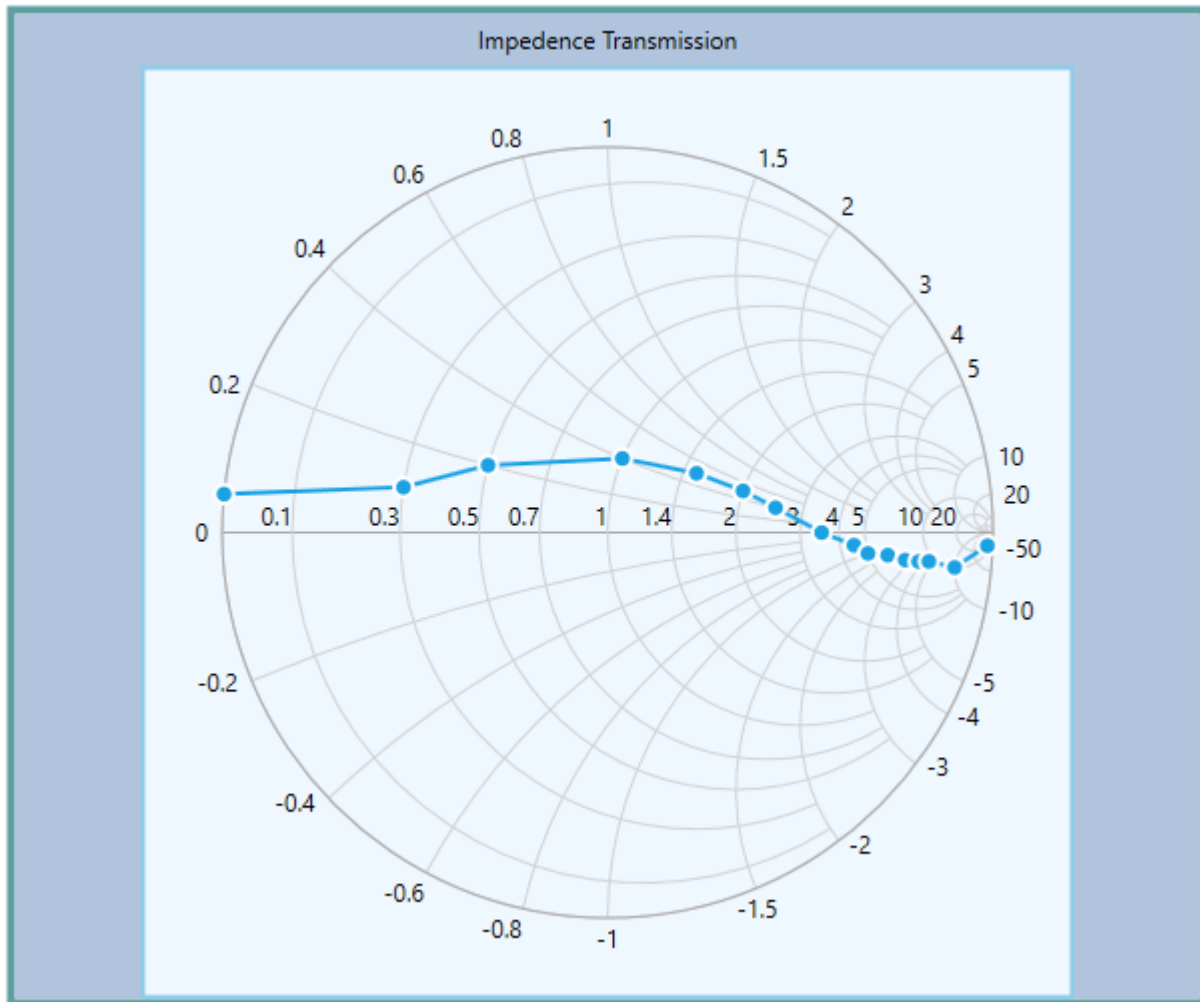
```
<syncfusion:SfSmithChart x:Name="SmithChart" Background="LightSteelBlue"
BorderBrush="CadetBlue" BorderThickness="4"
ChartAreaBackground="AliceBlue" ChartAreaBorderBrush="SkyBlue"
ChartAreaBorderThickness="2">
</syncfusion:SfSmithChart>
```

**C#**

```

chart.Background = new SolidColorBrush(Colors.LightSteelBlue);
chart.BorderBrush = new SolidColorBrush(Colors.CadetBlue);
chart.BorderThickness = new Thickness(4);
chart.ChartAreaBackground = new SolidColorBrush(Colors.AliceBlue);
chart.ChartAreaBorderBrush = new SolidColorBrush(Colors.SkyBlue);
chart.ChartAreaBorderThickness = new Thickness(2);

```



### Circle Radius

To change the diameter of the Smith chart circle with respect to the plot area, use the [Radius](#) property. It ranges from 0.1 to 1 and the default value is 0.95.

### XML

```

<syncfusion:SfSmithChart x:Name="SmithChart" Radius="0.5"
ChartAreaBorderBrush="CadetBlue">
</syncfusion:SfSmithChart>

```

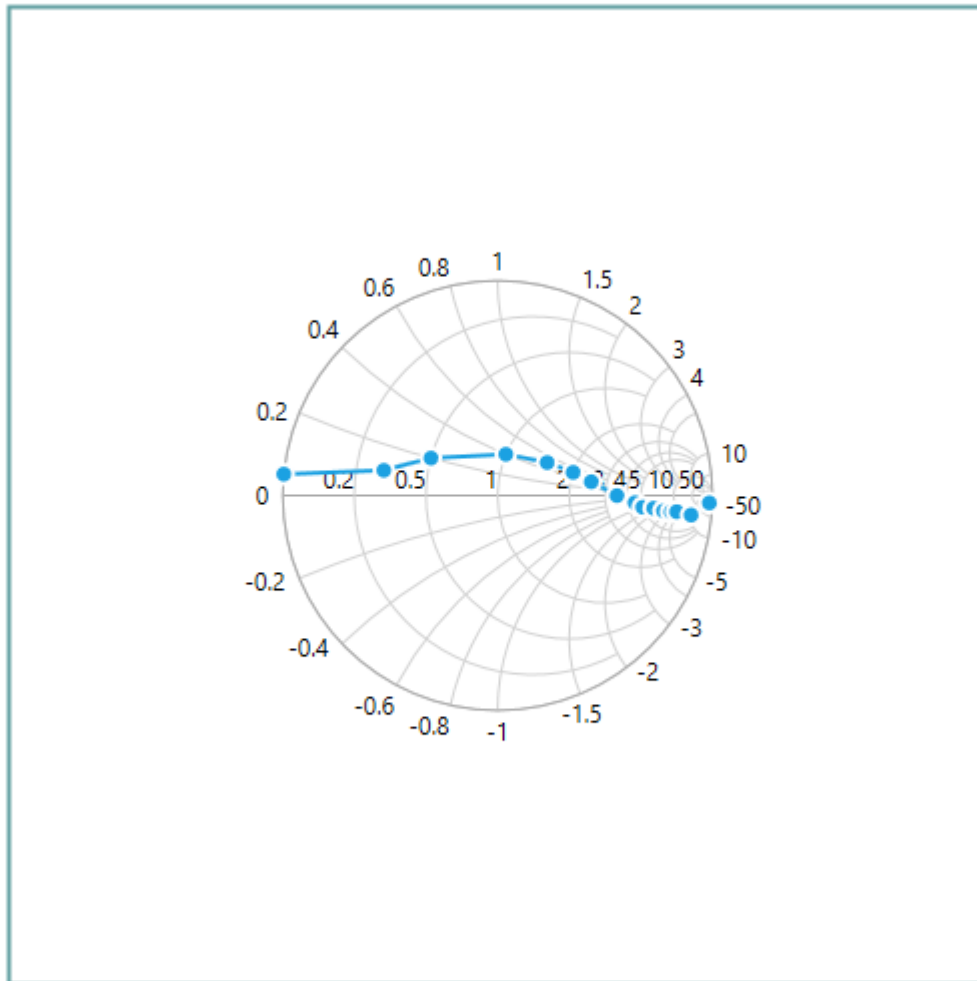
### C#

```

SfSmithChart chart = new SfSmithChart();
//Change the area circle radius value

```

```
chart.Radius = 0.5;
chart.ChartAreaBorderBrush = new SolidColorBrush(Colors.CadetBlue);
this.Grid1.Children.Add(chart);
```



Get smith chart properties

#### Area bounds

You can get the area bounds of the smith chart by using the [AreaBounds](#) property in [ChartAreaInfo](#).

#### C#

```
Rect areaBounds = smithChart.ChartAreaInfo.AreaBounds;
```

#### Center point

You can get the center point (X and Y) of the smith chart by using the [CenterPoint](#) property in the [ChartAreaInfo](#).

#### C#

```
Point centerPoint = smithChart.ChartAreaInfo.CenterPoint;
```

### Radius

You can get the radius of the smith chart by using the [Radius](#) property in the [ChartAreaInfo](#)

Code.

#### C#

```
double radius = smithChart.ChartAreaInfo.Radius;
```

## User Interactions in WPF Smith Chart (SfSmithChart)

### ToolTip

The ToolTip for data points can be enabled by setting the *ShowToolTip* property as True. ToolTip displays the duration that is defined by the *ToolTipDuration* property.

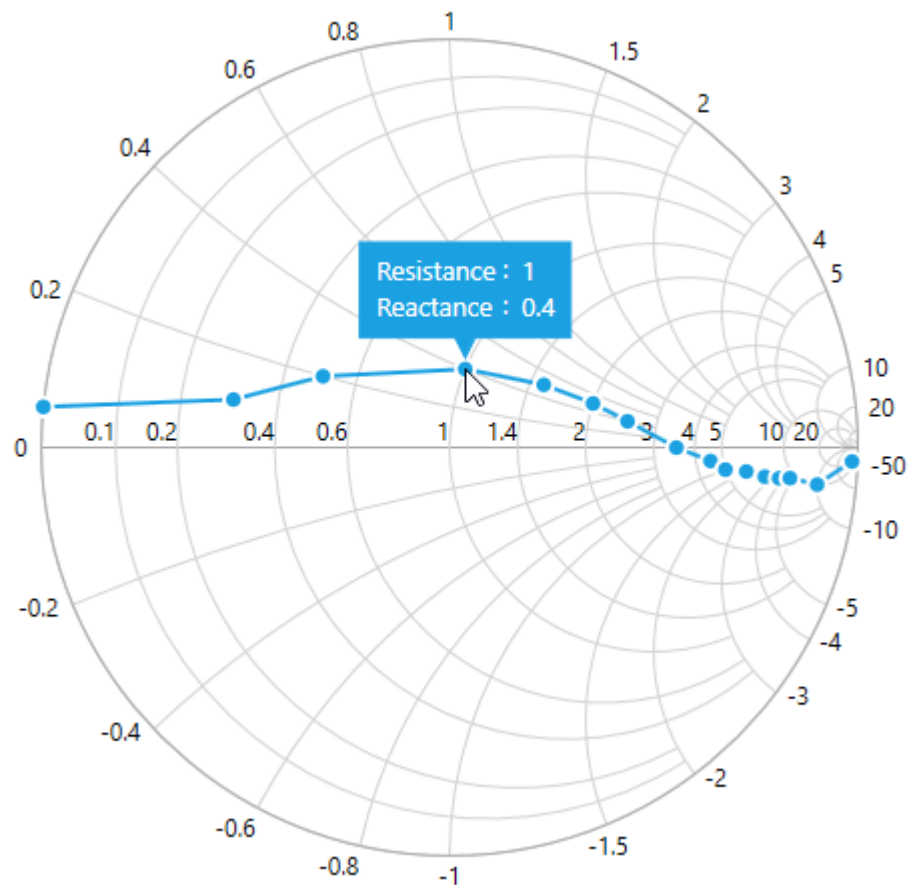
#### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">  
  <syncfusion:LineSeries ShowToolTip="True" ToolTipDuration="0:0:3">  
  </syncfusion:LineSeries>  
</syncfusion:SfSmithChart>
```

#### C#

```
LineSeries series = new LineSeries();  
series.ShowToolTip = true;  
series.ToolTipDuration = TimeSpan.FromSeconds(3);  
chart.Series.Add(series);
```





### ToolTipTemplate

*ToolTipTemplate* property allows to customize the default appearance of the tooltip as illustrated in the below code example.

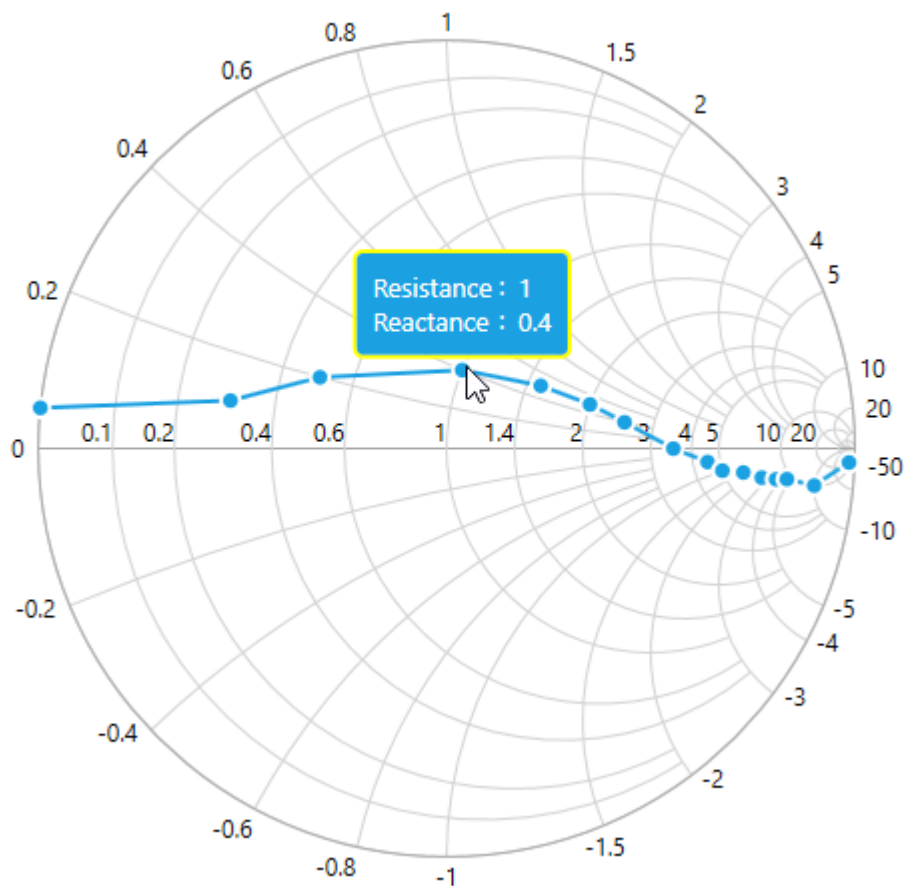
### XML

```
<syncfusion:SfSmithChart x:Name="SmithChart">
  <syncfusion:SfSmithChart.Resources>
    <DataTemplate x:Key="toolTipTemplate">
      <Border CornerRadius="4" Background="{Binding Interior}"
        BorderBrush="Yellow" BorderThickness="2">
        <Grid HorizontalAlignment="Center" Margin="8,8,8,8">
          <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
          </Grid.RowDefinitions>
          <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
            <ColumnDefinition/>
          </Grid.ColumnDefinitions>
          <TextBlock Grid.Row="0" Grid.Column="0" Text="Resistance"
            Foreground="{Binding ForeColor}" FontSize="13" />
```

```
<TextBlock Grid.Row="0" Grid.Column="1" Text=" : " Foreground="{Binding
ForeColor}" FontSize="13" Margin="0,-1,0,0" />
<TextBlock Grid.Row="0" Grid.Column="2" Text="{Binding Resistance}"
Foreground="{Binding ForeColor}" FontSize="13" Margin="3,0,0,0" />
<TextBlock Grid.Row="1" Grid.Column="0" Text="Reactance"
Foreground="{Binding ForeColor}" FontSize="13" />
<TextBlock Grid.Row="1" Grid.Column="1" Text=" : " Foreground="{Binding
ForeColor}" FontSize="13" Margin="0,-1,0,0" />
<TextBlock Grid.Row="1" Grid.Column="2" Text="{Binding Reactance}"
Margin="3,0,0,0" Foreground="{Binding ForeColor}" FontSize="13" />
</Grid>
</Border>
</DataTemplate>
</syncfusion:SfSmithChart.Resources>
<syncfusion:LineSeries ShowToolTip="True" ToolTipTemplate="{StaticResource
toolTipTemplate}" ShowMarker="True" MarkerType="Custom"
MarkerTemplate="{StaticResource Ellipse}" ItemsSource="{Binding Data}"
ResistancePath="Resistance" ReactancePath="Reactance">
</syncfusion:LineSeries>
</syncfusion:SfSmithChart>
```

## C#

```
LineSeries series = new LineSeries();
series.ShowToolTip = true;
series.ToolTipTemplate = this.Grid1.Resources["toolTipTemplate"] as
DataTemplate;
chart.Series.Add(series);
```



## SfSunburstChart

### WPF Sunburst Chart (SfSunburstChart) Overview

Sunburst Chart is useful for visualizing hierarchical data. The center circle represents the root level in the hierarchy, with outer circles representing higher levels of the hierarchy.

#### Key Features


- Visualize hierarchical data
- Data label support for better readability
- Interactively select or highlight segments.
- Interactive legend
- Colors can be customized

### Getting Started with WPF Sunburst Chart (SfSunburstChart)

This section explains you the steps required to populate the sunburst chart with data, add data labels, legends and header. This section covers only the minimal features that you need to know to get started with the Sunburst chart.

## Adding assembly reference



















1. Open the Add Reference window from your project
2. Choose Assemblies -> Extensions -> Syncfusion.SfSunburstChart.WPF

Assemblies		Targeting: .NET Framework 4	
		Name	Version
Framework			
Extensions		Syncfusion.SfRichTextRibbon.WPF	14.4400....
Recent		Syncfusion.SfSchedule.WPF	14.4400....
		Syncfusion.SfShared.Wpf	14.4400....
▸ Solution		Syncfusion.SfSkinManager.Wpf	14.4400....
		Syncfusion.SfSpellChecker.WPF	14.4350....
▸ COM		Syncfusion.SfSpellChecker.WPF	14.4400....
		Syncfusion.SfSpreadsheet.WPF	14.4400....
▸ Browse		Syncfusion.SfSpreadsheetHelper.WPF	14.4400....
		 Syncfusion.SfSunburstChart.WPF	14.4400....
		Syncfusion.SfTreeMap.WPF	14.4400....
		Syncfusion.SfTreeNavigator.WPF	14.4400....
		Syncfusion.Shared.Base	14.4350....
		Syncfusion.Shared.Base	14.4400....

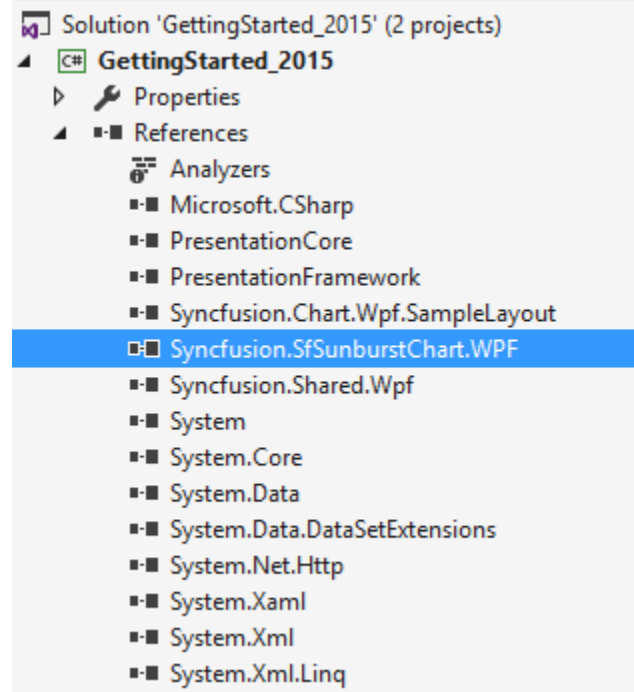
**Note:** This window differs for the Visual Basic project.

## Adding SfSunburstChart from Toolbox

Drag and drop the SfSunburstChart control from the Toolbox to your application,

	SfNavigationDrawer
	SfRadialMenu
	SfRadialSlider
	SfRichTextBoxAdv
	SfRichTextRibbon
	SfSchedule
	SfSpreadsheet
	SfSpreadsheetRibbon
	SfSunburstChart
	SfTreeMap
	SfTreeNavigator
	CalendarEdit
	Carousel
	ColorEdit
	ColorPicker
	CurrencyTextBox
	DateTimeEdit
	DoubleTextBox

Now the Syncfusion.SfSunburstChart.WPF reference is added to the application references and the namespace code is generated in MainWindow.xaml as below.



```
<Window x:Class="SunburstUG.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:syncfusion="clr-namespace:Syncfusion.UI.Xaml.SunburstChart;assembly=Syncfusion.SfSunburstChart"
        xmlns:local="clr-namespace:SunburstUG"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Grid>

        <syncfusion:SfSunburstChart>

        </syncfusion:SfSunburstChart>

    </Grid>
</Window>
```

#### Initialize view model

In this section, the data in the following table is used for demonstration,

Country	Job Description	Job Group	Job Role	Employees Count
America	Sales			70
America	Technical	Testers		35
America	Technical	Developers	Windows	105
America	Technical	Developers	Web	40
America	Management			40

America	Accounts			60
India	Technical	Testers		25
India	Technical	Developers	Windows	155
India	Technical	Developers	Web	60
Germany	Sales	Executive		30
Germany	Sales	Analyst		40
UK	Technical	Developers	Windows	100
UK	Technical	Developers	Web	30
UK	HR Executives			60
UK	Marketing			40

Now, let us define a data model that represents the above data in sunburst chart.

#### C#

```
public class Model
{
    public string Category { get; set; }
    public string Country { get; set; }
    public string JobDescription { get; set; }
    public string JobGroup { get; set; }
    public string JobRole { get; set; }
    public double EmployeesCount { get; set; }
}
```

Next, create a view model class and initialize a list of Model objects as shown below,

#### C#

```
public class ViewModel
{
    public ObservableCollection<Model> Data { get; set; }
    public ViewModel()
    {
        Data = new ObservableCollection<Model>
        {
            new Model
            {
                Country = "America", JobDescription = "Sales",
                EmployeesCount = 70
            },
            new Model
            {
                Country = "America", JobDescription = "Technical",
                JobGroup = "Testers", EmployeesCount = 35
            },
            new Model
            {
                Country = "America", JobDescription = "Technical",
                JobGroup = "Developers", EmployeesCount = 155
            },
            new Model
            {
                Country = "India", JobDescription = "Technical",
                JobGroup = "Testers", EmployeesCount = 25
            },
            new Model
            {
                Country = "Germany", JobDescription = "Sales",
                JobRole = "Executive", EmployeesCount = 30
            },
            new Model
            {
                Country = "Germany", JobDescription = "Sales",
                JobRole = "Analyst", EmployeesCount = 40
            },
            new Model
            {
                Country = "UK", JobDescription = "HR Executives", EmployeesCount = 60
            },
            new Model
            {
                Country = "UK", JobDescription = "Marketing", EmployeesCount = 40
            }
        }
    }
}
```

```
{
Country = "America", JobDescription = "Technical",
JobGroup = "Developers", JobRole = "Windows", EmployeesCount = 105
},
new Model
{
Country = "America", JobDescription = "Technical",
JobGroup = "Developers", JobRole = "Web", EmployeesCount = 40
},
new Model
{
Country = "America", JobDescription = "Management",
EmployeesCount = 40
},
new Model
{
Country = "America", JobDescription = "Accounts",
EmployeesCount = 60
},
new Model
{
Country = "India", JobDescription = "Technical",
JobGroup = "Testers", EmployeesCount = 25
},
new Model
{
Country = "India", JobDescription = "Technical", JobGroup = "Developers",
JobRole = "Windows", EmployeesCount = 155
},
new Model
{
Country = "India", JobDescription = "Technical", JobGroup = "Developers",
JobRole = "Web", EmployeesCount = 60
},
new Model
{
Country = "Germany", JobDescription = "Sales", JobGroup = "Executive",
EmployeesCount = 30
},
new Model
{
Country = "Germany", JobDescription = "Sales", JobGroup = "Analyst",
EmployeesCount = 40
},
new Model
{
Country = "UK", JobDescription = "Technical", JobGroup = "Developers",
JobRole = "Windows", EmployeesCount = 100
},
new Model
{
Country = "UK", JobDescription = "Technical", JobGroup = "Developers",
JobRole = "Web", EmployeesCount = 30
},
new Model
{
Country = "UK", JobDescription = "HR Executives", EmployeesCount = 60
}
```

```

},
new Model
{
    Country = "UK", JobDescription = "Marketing", EmployeesCount = 40
}
};
}
}

```

Set the ViewModel instance as the DataContext of your window; this is done to bind properties of ViewModel to Sunburst chart.

**Note:** Add namespace of ViewModel class in your XAML page if you prefer to set DataContext in XAML.

### XML

```

<Window x:Class="GettingStarted.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:GettingStarted">
    <Window.DataContext>
        <local:ViewModel/>
    </Window.DataContext>

```

### C#

```
this.DataContext = new ViewModel();
```

Populate Sunburst chart with data

Now, bind the **Data** property of the above ViewModel to the [ItemsSource](#) property.

Add **SunburstHierarchicalLevel** to [Levels](#) property. Each hierarchy level is formed based on the property specified in [GroupMemberPath](#) property, and each arc segment size is calculated using [ValueMemberPath](#).

### XML

```

<sunburst:SfSunburstChart ItemsSource="{Binding Data}"
ValueMemberPath="EmployeesCount">
    <sunburst:SfSunburstChart.Levels>
        <sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
        <sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
        <sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
        <sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
    </sunburst:SfSunburstChart.Levels>
</sunburst:SfSunburstChart>

```

### C#

```

SfSunburstChart sunburst = new SfSunburstChart();
sunburst.ValueMemberPath = "EmployeesCount";
sunburst.SetBinding(SfSunburstChart.ItemsSourceProperty, "Data");
sunburst.Levels.Add(new SunburstHierarchicalLevel() {GroupMemberPath =
    "Country"});

```



```
sunburst.Levels.Add(new SunburstHierarchicalLevel() {GroupMemberPath =  
"JobDescription"});  
sunburst.Levels.Add(new SunburstHierarchicalLevel() {GroupMemberPath =  
"JobGroup"});  
sunburst.Levels.Add(new SunburstHierarchicalLevel() {GroupMemberPath =  
"JobRole"});
```

### Add header

You can add header to Sunburst chart to provide quick information to the user about the data being plotted in the chart. You can also set title using the [Header](#) property as follows.

#### XML

```
<sunburst:SfSunburstChart Header="Employees Count" FontSize="22" />
```

#### C#

```
sunburst.Header = "Employees Count";  
sunburst.FontSize = 22d;
```

### Add legend

You can enable legend using the [Legend](#) property as follows.

#### XML

```
<sunburst:SfSunburstChart.Legend>  
<sunburst:SunburstLegend DockPosition="Left" />  
</sunburst:SfSunburstChart.Legend>
```

#### C#

```
SunburstLegend legend = new SunburstLegend();  
legend.DockPosition= ChartDock.Left;
```

### Add data labels

You can add data labels to improve the readability of the Sunburst chart. This can be achieved using the [DataLabelInfo](#) property as follows.

#### XML

```
<sunburst:SfSunburstChart.DataLabelInfo>  
<sunburst:SunburstDataLabelInfo />  
</sunburst:SfSunburstChart.DataLabelInfo>
```

#### C#

```
SunburstDataLabelInfo dataLabel= new SunburstDataLabelInfo();  
sunburst.DataLabelInfo = dataLabel;
```

Below is the complete code to replicate the following output.

#### XML

```

<Grid>
<Grid.DataContext>
<local:ViewModel></local:ViewModel>
</Grid.DataContext>
<sunburst:SfSunburstChart ItemsSource="{Binding Data}"
ValueMemberPath="EmployeesCount" Header="Employees Count" FontSize="22" >
<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstLegend DockPosition="Left"/>
</sunburst:SfSunburstChart.Legend>
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
</sunburst:SfSunburstChart.Levels>
<sunburst:SfSunburstChart.DataLabelInfo>
<sunburst:SunburstDataLabelInfo />
</sunburst:SfSunburstChart.DataLabelInfo>
</sunburst:SfSunburstChart>
</Grid>

```

## C#

```

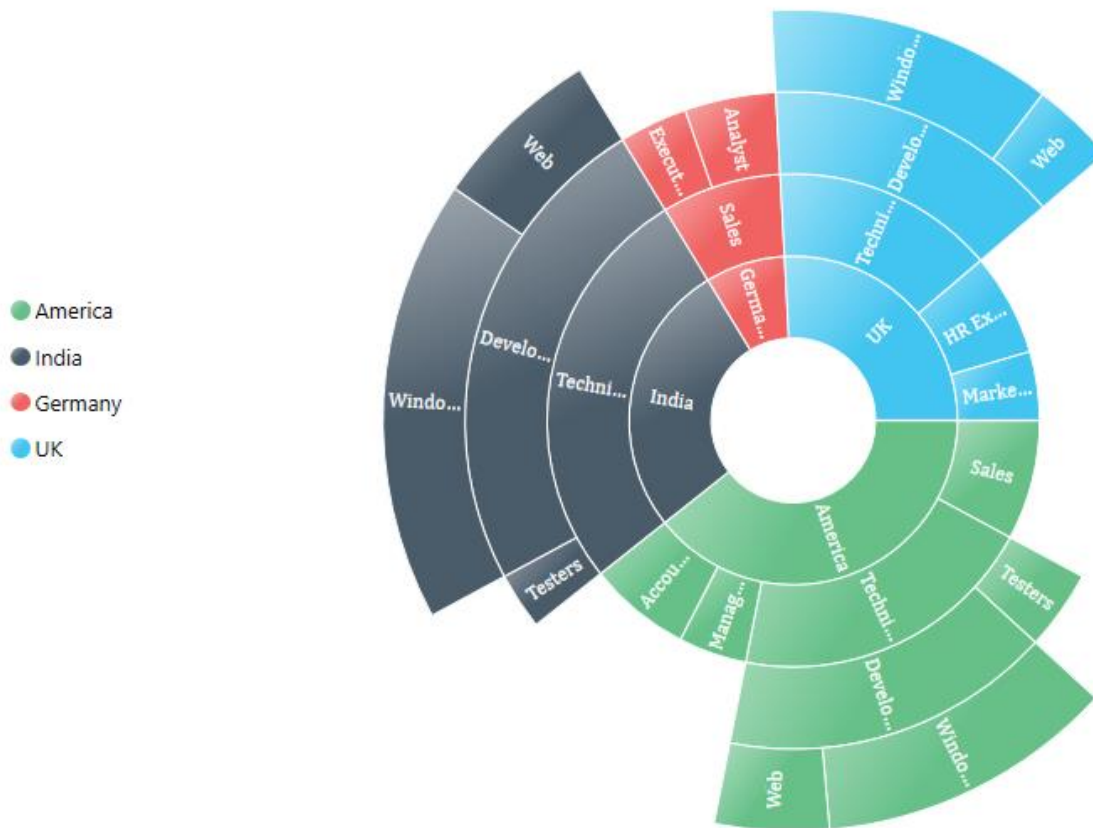
this.DataContext = new ViewModel();
SfSunburstChart sunburst = new SfSunburstChart();
sunburst.ValueMemberPath = "EmployeesCount";
sunburst.SetBinding(SfSunburstChart.ItemsSourceProperty, "Data");
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
sunburst.Header = "Employees Count";
sunburst.FontSize = 22d;
SunburstLegend legend = new SunburstLegend();
legend.DockPosition = ChartDock.Left;
sunburst.Legend = legend;
SunburstDataLabelInfo dataLabel = new SunburstDataLabelInfo();
sunburst.DataLabelInfo = dataLabel;
this.Content = sunburst;

```

You can get the complete getting started sample [here](#).

Following is the final output screenshot,

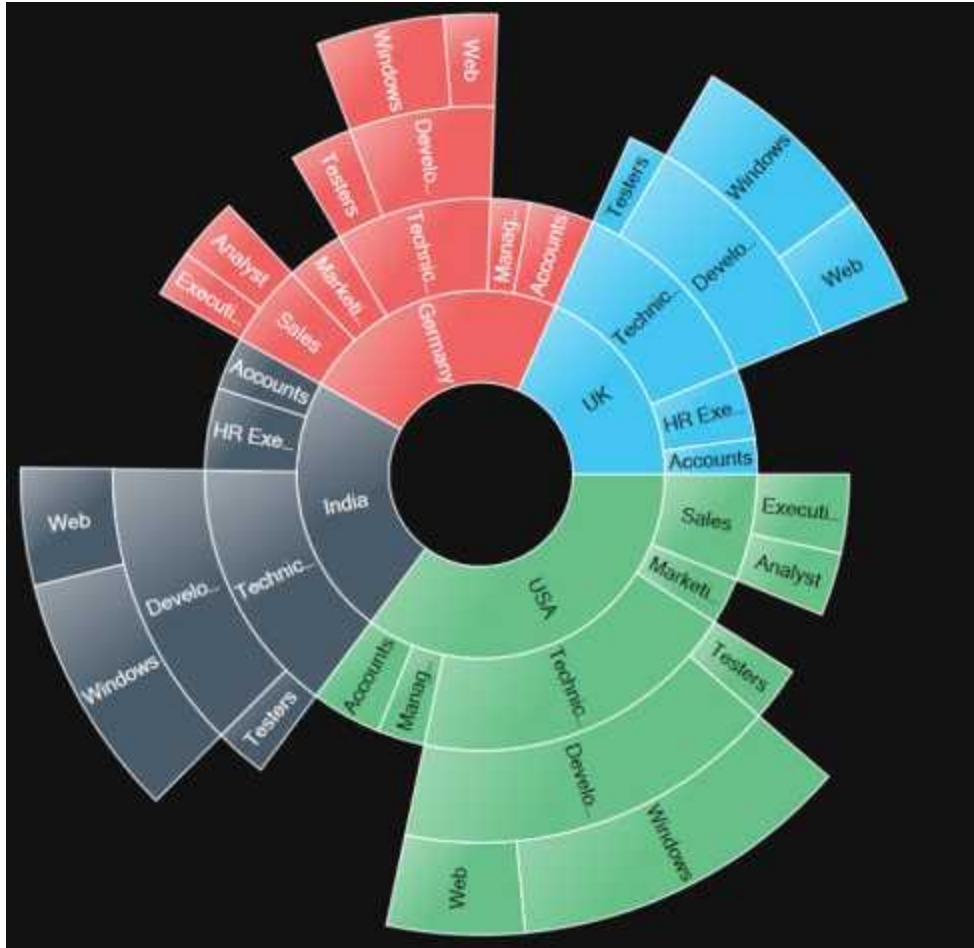
## Employees Count



### Theme

Sunburst chart supports various built-in themes. Refer to the below links to apply themes for the Sunburst chart,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Region in WPF Sunburst Chart (SfSunburstChart)

Sunburst region represents the entire chart and all its elements. It includes all the chart elements like Legend, DataLabel, Levels etc. It has some major properties as

[ItemsSource](#) – Gets or sets the IEnumerable values used to generate the chart.

[ValueMemberPath](#) – Gets or Sets the property path of the value.

[Legend](#) – Gets or Sets the legend for the chart.

[Levels](#) – Gets or Sets the collection of hierarchical level for the chart.

[DataLabels](#) – Gets or Sets the collection of DataLabels for the chart.

[Behaviors](#) – Gets or Sets the collection of behavior for the chart.

### Start and End angle

You can change the start and end angle of Sunburst chart using [StartAngle](#) and [EndAngle](#) property as shown in below code

### XML

```
<sunburst:SfSunburstChart StartAngle="180"
EndAngle="360">
</sunburst:SfSunburstChart>
```

**C#**

```
sunburstChart.StartAngle = 180;  
sunburstChart.EndAngle = 360;
```

**Sunburst radius**

Sunburst chart allows you to customize the sunburst radius by using [Radius](#) property. Default value of this property is 0.9 and the value range between 0 to 1.

**XML**

```
<sunburst:SfSunburstChart Radius="0.6">  
</sunburst:SfSunburstChart>
```

**C#**

```
chart.Radius = 0.6;
```



#### Sunburst inner radius

Sunburst chart allows you to customize the inner radius using [InnerRadius](#) property. The default value of this property is 0.2 and value range between 0 to 1.

#### XML

```
<sunburst:SfSunburstChart InnerRadius="0.5">
</sunburst:SfSunburstChart>
```

#### C#

```
chart.InnerRadius = 0.5;
```



### Levels in WPF Sunburst Chart (SfSunburstChart)

Sunburst chart is used to display hierarchical data. You can add more than one hierarchical data in [Levels](#) collection of Sunburst chart. Each level of the hierarchy is represented by circle.

The following code shows how to add the hierarchical level in Levels collection.

#### XML

```
<sunburst:SfSunburstChart.Levels>  
<sunburst:SunburstHierarchicalLevel/>  
</sunburst:SfSunburstChart.Levels>
```

#### C#

```
SunburstHierarchicalLevel level = new SunburstHierarchicalLevel();  
sunburstChart.Levels.Add(level);
```

### GroupMemberPath

It is the string property used to map the group category value in sunburst ItemsSource.

You can define the Levels as shown in the code sample.

#### XML

```
<sunburst:SfSunburstChart.Levels>  
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Level1"/>  
<sunburst:SunburstHierarchicalLevel GroupMemberPath=" Level2"/>  
<sunburst:SunburstHierarchicalLevel GroupMemberPath=" Level3"/>  
</sunburst:SfSunburstChart.Levels>
```

**C#**

```
SunburstHierarchicalLevel level1 = new SunburstHierarchicalLevel();  
level1.GroupMemberPath = "Level_1";  
SunburstHierarchicalLevel level2 = new SunburstHierarchicalLevel();  
level2.GroupMemberPath = "Level_2";  
SunburstHierarchicalLevel level3 = new SunburstHierarchicalLevel();  
level3.GroupMemberPath = "Level_3";  
sunburstChart.Levels.Add(level1);  
sunburstChart.Levels.Add(level2);  
sunburstChart.Levels.Add(level3);
```

**Legend in WPF Sunburst Chart (SfSunburstChart)**

The Legend is used to represent the first level of categories in sunburst chart.

You can initialize the legend as in the below code snippet:

**XML**

```
<sunburst:SfSunburstChart.Legend>  
<sunburst:SunburstLegend/>  
</sunburst:SfSunburstChart.Legend>
```

**C#**

```
SunburstLegend legend = new SunburstLegend();  
chart.Legend = legend;
```





### Legend Icon

You can specify different shapes of legend icon by using the [LegendIcon](#) property. By default, legend icon is Circle. The Sunburst chart have some predefined shapes such as:

- Circle
- Cross
- Diamond
- Pentagon
- Rectangle
- Triangle

### XML

```
<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstLegend LegendIcon="Pentagon"/>
</sunburst:SfSunburstChart.Legend>
```

### C#

```
SunburstLegend legend = new SunburstLegend()
{
    LegendIcon = SunburstLegendIcon.Pentagon
};
chart.Legend = legend;
```



The following properties are used to customize the legend icons size.

- [IconHeight](#) – Gets or sets the double value to represents icon(s) height.
- [IconWidth](#) – Gets or sets the double value to represents icon(s) width.

You can customize your own legend shape by applying custom template using [LegendIconTemplate](#) property as shown in the below code.

#### XML

```
<Grid.Resources>
<DataTemplate x:Key="legendTemplate">
<Path Stroke="{Binding Stroke}" Stretch="Fill" Fill="{Binding Interior}"
StrokeThickness="{Binding StrokeThickness}"
Data="F1 M 133.133,45.7109L 154.307,24.5363L 175.482,45.7109L
154.307,66.8856L 175.482,88.0603L 154.307,109.235L 133.133,88.0603L
111.958,109.235L 90.7835,88.0603L 111.958,66.8856L 90.7835,45.7109L
111.958,24.5363L 133.133,45.7109 Z " />
</DataTemplate>
</Grid.Resources>
<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstLegend LegendIcon="Custom"
LegendIconTemplate="{StaticResource legendTemplate}"/>
</sunburst:SfSunburstChart.Legend>
```

#### C#

```

SfSunburstChart sunburst = new SfSunburstChart();
sunburst.ValueMemberPath = "EmployeesCount";
sunburst.SetBinding(SfSunburstChart.ItemsSourceProperty, "Data");
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
SunburstLegend legend = new SunburstLegend()
{
    LegendIcon = SunburstLegendIcon.Custom,
    LegendIconTemplate = grid.Resources["legendTemplate"] as DataTemplate
};
sunburst.Legend = legend;
grid.Children.Add(sunburst);

```



Note: You need to set [LegendIcon](#) value as Custom in order to apply custom template.

#### Positioning the Legend

You can customize the position to left, right, top, bottom for the legend using [DockPosition](#) property as like in below code snippet

#### XML

```
<sunburst:SfSunburstChart.Legend>
```

```
<sunburst:SunburstLegend DockPosition="Top" />
</sunburst:SfSunburstChart.Legend>
```

**C#**

```
SunburstLegend legend = new SunburstLegend()
{
    DockPosition = ChartDock.Top
};
chart.Legend = legend;
```

**Customize the Legend**

You can arrange the legend items smartly by using `ItemPanelTemplate` and `ItemTemplate` property. The following code shows how to arrange the legend items smartly.

**XML**

```
<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstLegend DockPosition="Top" Margin="0,25,0,0" >
<sunburst:SunburstLegend.ItemsPanel>
<ItemsPanelTemplate>
<WrapPanel Height="100" Width="250" />
</ItemsPanelTemplate>
</sunburst:SunburstLegend.ItemsPanel>
```

```

<sunburst:SunburstLegend.ItemTemplate>
<ItemContainerTemplate>
<StackPanel Width="75" Height="30" Orientation="Horizontal">
<Ellipse Stroke="{Binding Interior}" StrokeThickness="2" Fill="White"
Height="10" Width="10" Margin="5"/>
<TextBlock Text="{Binding Label}" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</StackPanel>
</ItemContainerTemplate>
</sunburst:SunburstLegend.ItemTemplate>
</sunburst:SunburstLegend>
</sunburst:SfSunburstChart.Legend>

```



### Legend Interactivity

You can select a specific category while clicking on corresponding legend item through [ClickAction](#) property. It has three types of action

- ToggleSegmentSelection
- ToggleSegmentVisibility
- None



*ToggleSegmentSelection*

Used to highlight specific category while clicking on legend item.

**XML**

```
<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstLegend ClickAction="ToggleSegmentSelection"/>
</sunburst:SfSunburstChart.Legend>
```

**C#**

```
SunburstLegend legend = new SunburstLegend()
{
    ClickAction = LegendClickAction.ToggleSegmentSelection
};
chart.Legend = legend;
```

*ToggleSegmentVisibility*

Used to disable the specific category while clicking on legend item.

**XML**

```
<sunburst:SfSunburstChart.Legend>
<sunburst:SunburstLegend ClickAction="ToggleSegmentVisibility"/>
</sunburst:SfSunburstChart.Legend>
```

**C#**

```
SunburstLegend legend = new SunburstLegend()
{
    ClickAction = LegendClickAction.ToggleSegmentVisibility
};
chart.Legend = legend;
```

**Data Label in WPF Sunburst Chart (SfSunburstChart)**

Sunburst data labels are used to display the data related to the segment. It helps to provide the information about the data points to the users.

You can enable or disable the data labels by using [ShowLabel](#) property as shown in the below code

**XML**

```
<sunburst:SfSunburstChart.DataLabelInfo>
<sunburst:SunburstDataLabelInfo ShowLabel="True" />
</sunburst:SfSunburstChart.DataLabelInfo>
```

**C#**

```
SunburstDataLabelInfo dataLabelInfo = new SunburstDataLabelInfo()
{
    ShowLabel = true
};
chart.DataLabelInfo = dataLabelInfo;
```



**Note:** By default, the ShowLabel property value is True.

#### LabelOverflowMode

When you represent huge data with data labels, they may intersect each other. You can avoid this using the [LabelOverflowMode](#) property.

The following properties are used to avoid the overlapping.

- Trim – To trim the large data labels.
- Hide – To hide the overlapped data labels.

The following code shows how to set Hide and Trim mode.

#### Hide

##### XML

```
<sunburst:SfSunburstChart.DataLabelInfo>
<sunburst:SunburstDataLabelInfo ShowLabel="True"
LabelOverflowMode="Hide"/>
</sunburst:SfSunburstChart.DataLabelInfo>
```

#### C#

```
SunburstDataLabelInfo dataLabelInfo = new SunburstDataLabelInfo()
{
    ShowLabel = true,
    LabelOverflowMode = LabelOverflowMode.Hide
}
```



```
};
```



Trim

#### XML

```
<sunburst:SfSunburstChart.DataLabelInfo>  
<sunburst:SunburstDataLabelInfo ShowLabel="True"  
LabelOverflowMode="Trim" />  
</sunburst:SfSunburstChart.DataLabelInfo>
```

#### C#

```
SunburstDataLabelInfo dataLabelInfo = new SunburstDataLabelInfo()  
{  
    ShowLabel = true,  
    LabelOverflowMode = LabelOverflowMode.Trim  
};  
chart.DataLabelInfo = dataLabelInfo;
```



### LabelRotationMode

You can rotate the data label by using [LabelRotationMode](#) property.

The following code shows how to set [LabelRotationMode](#) as normal and angle.

### XML

```
<sunburst:SfSunburstChart.DataLabelInfo>
<sunburst:SunburstDataLabelInfo ShowLabel="True"
LabelRotationMode="Normal"/>
</sunburst:SfSunburstChart.DataLabelInfo>
```

### C#

```
SunburstDataLabelInfo dataLabelInfo = new SunburstDataLabelInfo()
{
    ShowLabel = true,
    LabelRotationMode = LabelRotationMode.Normal
};
chart.DataLabelInfo = dataLabelInfo;
```



**Note:** By default, LabelRotationMode value is Angle.

#### Font customization

Data label fonts can be customized using [FontFamily](#), [FontSize](#), [FontStyle](#), [FontWeight](#) properties.

Font color of the label can be customized using the [Foreground](#) property.

#### XML

```
<sunburst:SfSunburstChart ItemsSource="{Binding Data}"
ValueMemberPath="EmployeesCount" >
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
</sunburst:SfSunburstChart.Levels>
<sunburst:SfSunburstChart.DataLabelInfo>
<sunburst:SunburstDataLabelInfo ShowLabel="True" FontSize="12"
FontFamily="Comic Sans MS"
FontStyle="Italic" FontWeight="SemiBold"
Foreground="White"/>
</sunburst:SfSunburstChart.DataLabelInfo>
</sunburst:SfSunburstChart>
```

#### C#

```
SfSunburstChart sunburst = new SfSunburstChart();
sunburst.ValueMemberPath = "EmployeesCount";
sunburst.SetBinding(SfSunburstChart.ItemsSourceProperty, "Data");
```

```

sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
SunburstDataLabelInfo info = new SunburstDataLabelInfo();
info.FontSize = 12;
info.FontStyle = FontStyles.Italic;
info.FontWeight = FontWeights.SemiBold;
info.Foreground = new SolidColorBrush(Colors.White);
info.FontFamily = new FontFamily("Comic Sans MS");
sunburst.DataLabelInfo = info;
grid.Children.Add(sunburst);

```



### Customizing the data labels

You can customize the default appearance or display information about the data point using the [LabelTemplate](#) property.

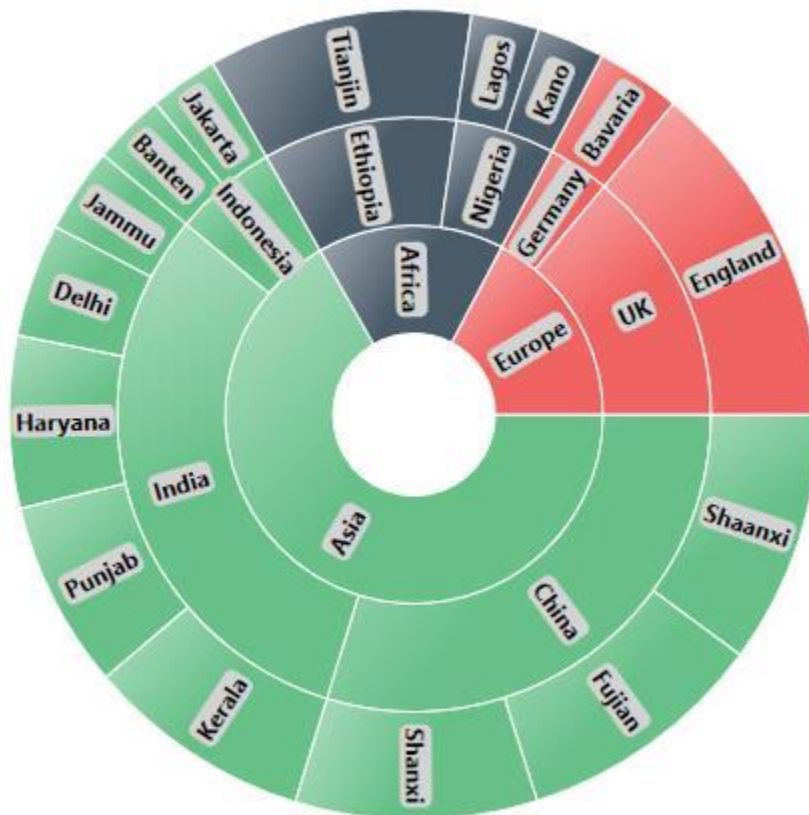
### XML

```
<sunburst:SunburstDataLabelInfo.LabelTemplate>
```

```

<DataTemplate>
<Border Background="LightGray" CornerRadius="4" >
<TextBlock Text="{Binding Category}" Margin="2,0,2,0"
FontWeight="Bold"/>
</Border>
</DataTemplate>
</sunburst:SunburstDataLabelInfo.LabelTemplate>

```



### Tooltip in WPF Sunburst Chart (SfSunburstChart)

ToolTip allows you to display any information over a sunburst segment. It appears when mouse hovered over or touch any chart segment. By default, it displays the corresponding segment category name and its value. Visibility of the tooltip can be controlled using [ShowToolTip](#) property.

#### XML

```

<sunburst:SfSunburstChart.Behaviors>
<sunburst:SunburstToolTipBehavior ShowToolTip="True"/>
</sunburst:SfSunburstChart.Behaviors>

```

#### C#

```

SunburstToolTipBehavior tooltip = new SunburstToolTipBehavior();
tooltip.ShowToolTip = true;
chart.Behaviors.Add(tooltip);

```



### Aligning the ToolTip

The tooltip position can be aligned with respect to the cursor position by using [HorizontalAlignment](#) and [VerticalAlignment](#) properties.

#### *HorizontalAlignment*

The following code shows, how to position the tooltip to right of the cursor.

#### **XML**

```
<sunburst:SfSunburstChart.Behaviors>  
<sunburst:SunburstToolTipBehavior HorizontalAlignment="Right"/>  
</sunburst:SfSunburstChart.Behaviors>
```

#### **C#**

```
SunburstToolTipBehavior tooltip = new SunburstToolTipBehavior();  
tooltip.HorizontalAlignment = HorizontalAlignment.Right;  
sunburstChart.Behaviors.Add(tooltip);
```



#### *VerticalAlignment*

The following code shows, how to position the tooltip to bottom of the cursor.

#### **XML**

```
<sunburst:SfSunburstChart.Behaviors>  
<sunburst:SunburstToolTipBehavior VerticalAlignment="Bottom" />  
</sunburst:SfSunburstChart.Behaviors>
```





#### VerticalOffset and HorizontalOffset

The tooltip position can be customized to a custom position from the cursor using the [HorizontalOffset](#) and [VerticalOffset](#) properties.

#### XML

```
<sunburst:SfSunburstChart.Behaviors>  
<sunburst:SunburstToolTipBehavior HorizontalOffset="50"  
VerticalOffset="50"/>  
</sunburst:SfSunburstChart.Behaviors>
```

#### C#

```
SunburstToolTipBehavior tooltip = new SunburstToolTipBehavior();  
tooltip.HorizontalOffset = 50;  
tooltip.VerticalOffset = 50;  
sunburstChart.Behaviors.Add(tooltip);
```





### TooltipDuration

You can set display duration for tooltip by using the [ShowDuration](#) property.

### XML

```
<sunburst:SfSunburstChart.Behaviors>  
<sunburst:SunburstToolTipBehavior ShowDuration="6000"/>  
</sunburst:SfSunburstChart.Behaviors>
```

### C#

```
SunburstToolTipBehavior tooltip = new SunburstToolTipBehavior();  
tooltip.ShowDuration = 6000;  
chart.Behaviors.Add(tooltip);
```

### Show Delay

We can set the initial display delay for Tooltip by using [InitialShowDelay](#) property as like in below code.

### XML

```
<sunburst:SfSunburstChart.Behaviors>  
<sunburst:SunburstToolTipBehavior InitialShowDelay="500"/>  
</sunburst:SfSunburstChart.Behaviors>
```

### C#

```
SunburstToolTipBehavior tooltip = new SunburstToolTipBehavior();  
tooltip.InitialShowDelay = 500;  
sunburstChart.Behaviors.Add(tooltip);
```

### Animation for Tooltip

You can enable the translate animation for Tooltip by using [EnableAnimation](#) and [AnimationDuration](#) property.

#### XML

```
<sunburst:SfSunburstChart.Behaviors>
<sunburst:SunburstToolTipBehavior EnableAnimation="True"
AnimationDuration="5000"/>
</sunburst:SfSunburstChart.Behaviors>
```

#### C#

```
SunburstToolTipBehavior tooltip = new SunburstToolTipBehavior();
tooltip.EnableAnimation = true;
tooltip.AnimationDuration = 5000;
sunburstChart.Behaviors.Add(tooltip);
```

### Customize the Tooltip

You can customize the default appearance of the tooltip by applying the [ToolTipTemplate](#) property.

#### XML

```
<sunburst:SunburstToolTipBehavior.ToolTipTemplate>
<DataTemplate>
<Border Background="{Binding Interior}"
BorderBrush="{Binding Interior}"
BorderThickness="4">
<StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock Text="Category : "/>
<TextBlock Text="{Binding Category}"/>
</StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock Text="Value : "/>
<TextBlock Text="{Binding Value}"/>
</StackPanel>
</StackPanel>
</Border>
</DataTemplate>
</sunburst:SunburstToolTipBehavior.ToolTipTemplate>
```



### Selection in WPF Sunburst Chart (SfSunburstChart)

Sunburst chart supports selection that enables you to select a segment by using [SunburstSelectionBehavior](#).

The below code shows, how to enable the selection behavior.

#### XML

```
<sunburst:SfSunburstChart.Behaviors>  
<sunburst:SunburstSelectionBehavior/>  
</sunburst:SfSunburstChart.Behaviors>
```

#### C#

```
SunburstSelectionBehavior selection = new SunburstSelectionBehavior();  
chart.Behaviors.Add(selection);
```



### SelectionDisplayMode

You can customize the selected segment appearance by using brush or opacity. You can choose between color or opacity using the [SelectionDisplayMode](#) property in the selection behavior

- HighlightByColor – To display the selected segment appearance using brush.
- HighlightByOpacity – To display the selected segment appearance using opacity.

The following code shows, how to set the display mode using brush.

#### XML

```
<sunburst:SfSunburstChart.Behaviors>
<sunburst:SunburstSelectionBehavior EnableSelection="True"
SelectionBrush="Black"
SelectionDisplayMode="HighlightByColor"/>
</sunburst:SfSunburstChart.Behaviors>
```

#### C#

```
SunburstSelectionBehavior selection = new SunburstSelectionBehavior();
selection.EnableSelection = true;
selection.SelectionBrush = new SolidColorBrush(Colors.Black);
selection.SelectionDisplayMode = SelectionDisplayMode.HighlightByColor;
chart.Behaviors.Add(selection);
```



**Note:** The default value of SelectionDisplayMode is HighlightByOpacity.

### SelectionMode

Sunburst chart provides support to select or highlight the segment by clicking or hovering the mouse over a segment. By default, this property value is MouseClick.

- Both – Select the segment using mouse move and mouse click.
- MouseClick – Select the segment using mouse click.
- MouseMove – Select the segment using mouse move.

### XML

```
<sunburst:SfSunburstChart.Behaviors>
<sunburst:SunburstSelectionBehavior EnableSelection="True"
SelectionMode="MouseClick"/>
</sunburst:SfSunburstChart.Behaviors>
```

### C#

```
SunburstSelectionBehavior selection = new SunburstSelectionBehavior();
selection.EnableSelection = true;
selection.SelectionMode =
Syncfusion.UI.Xaml.SunburstChart.SelectionMode.MouseClick;
chart.Behaviors.Add(selection);
```

### SelectionType

Sunburst chart provides multiple option to represent the selected categories. You can select the segment categories by using the [SelectionType](#) property in selection behavior.

- Child – To select the child of selected parent.
- Group – To select the entire categories in group.
- Parent – To select the parent of selected child.
- Single - To select single item in the category.

### Child

The following code shows, how to set the selection type as child.

### XML

```
<sunburst:SfSunburstChart.Behaviors>
<sunburst:SunburstSelectionBehavior EnableSelection="True"
SelectionType="Child" />
</sunburst:SfSunburstChart.Behaviors>
```

### C#

```
SunburstSelectionBehavior selection = new SunburstSelectionBehavior();
selection.EnableSelection = true;
selection.SelectionType = SelectionType.Child;
chart.Behaviors.Add(selection);
```



### Group

The following code shows, how to set the selection type as group.

#### XML

```
<sunburst:SfSunburstChart.Behaviors>
<sunburst:SunburstSelectionBehavior EnableSelection="True"
SelectionType="Group" />
</sunburst:SfSunburstChart.Behaviors>
```

#### C#

```
SunburstSelectionBehavior selection = new SunburstSelectionBehavior();
selection.EnableSelection = true;
selection.SelectionType = SelectionType.Group;
chart.Behaviors.Add(selection);
```



### Parent

The following code shows, how to set the selection type as parent.

#### XML

```
<sunburst:SfSunburstChart.Behaviors>
<sunburst:SunburstSelectionBehavior EnableSelection="True"
SelectionType="Parent" />
</sunburst:SfSunburstChart.Behaviors>
```

**C#**

```
SunburstSelectionBehavior selection = new SunburstSelectionBehavior();
selection.EnableSelection = true;
selection.SelectionType = SelectionType.Parent;
chart.Behaviors.Add(selection);
```

*Single*

The following code shows, how to set the selection type as single.

**XML**

```
<sunburst:SfSunburstChart.Behaviors>
<sunburst:SunburstSelectionBehavior EnableSelection="True"
SelectionType="Single" />
</sunburst:SfSunburstChart.Behaviors>
```

**C#**

```
SunburstSelectionBehavior selection = new SunburstSelectionBehavior();
selection.EnableSelection = true;
selection.SelectionType = SelectionType.Single;
chart.Behaviors.Add(selection);
```





### Selection Cursor

SelectionCursor property allows you to customize the cursor when mouse is hovered over the segment.

The following code shows, how to set the selection cursor as hand.

### XML

```
<sunburst:SfSunburstChart.Behaviors>
<sunburst:SunburstSelectionBehavior EnableSelection="True"
SelectionCursor="Hand" />
</sunburst:SfSunburstChart.Behaviors>
```

### C#

```
SunburstSelectionBehavior selection = new SunburstSelectionBehavior();
selection.EnableSelection = true;
selection.SelectionCursor = Cursors.Hand;
chart.Behaviors.Add(selection);
```

## Zooming in WPF Sunburst Chart (SfSunburstChart)

Sunburst chart provides zooming (drill down) experience with animation for both mouse and touch enabled devices. It allows you to virtualize large sets of data into minimum data view.

The following code shows how to initialize the zooming behavior.

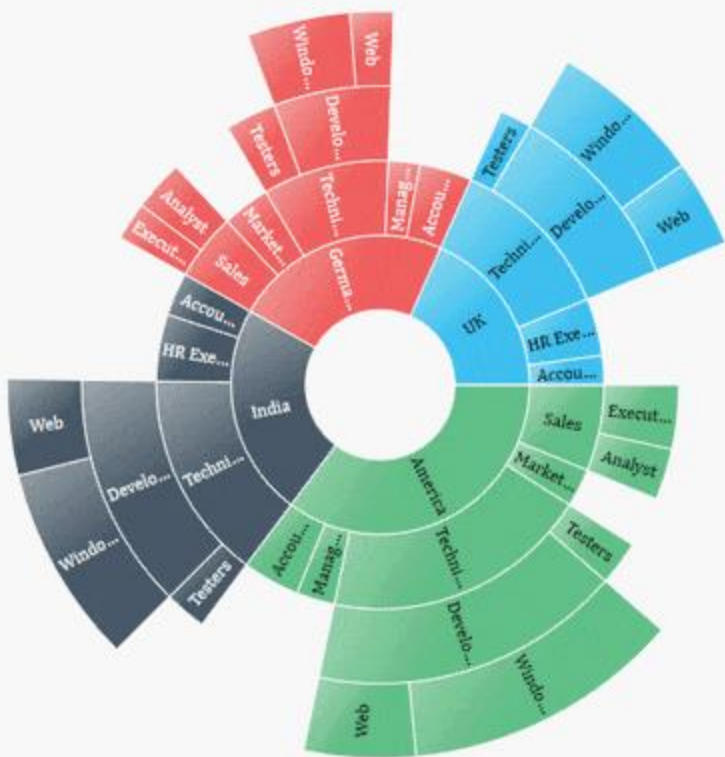
## XML

```
<sunburst:SfSunburstChart.Behaviors>
<sunburst:SunburstZoomingBehavior/>
</sunburst:SfSunburstChart.Behaviors>
```

## C#

```
SunburstZoomingBehavior zoom = new SunburstZoomingBehavior();
chart.Behaviors.Add(zoom);
```

**Note:** You can enable or disable the zooming by using [EnableZooming](#) property. By default, EnableZooming property value is True.



### Zooming Toolbar

By default, zooming toolbar will be enabled while zooming the segment; it contains both back and reset option.

You can align the zooming toolbar position by using [ToolBarHorizontalAlignment](#) and [ToolBarVerticalAlignment](#) property.

### XML

```
<sunburst:SfSunburstChart.Behaviors>
<sunburst:SunburstZoomingBehavior EnableZooming="True"
ToolBarHorizontalAlignment="Center"
ToolBarVerticalAlignment="Center"/>
</sunburst:SfSunburstChart.Behaviors>
```

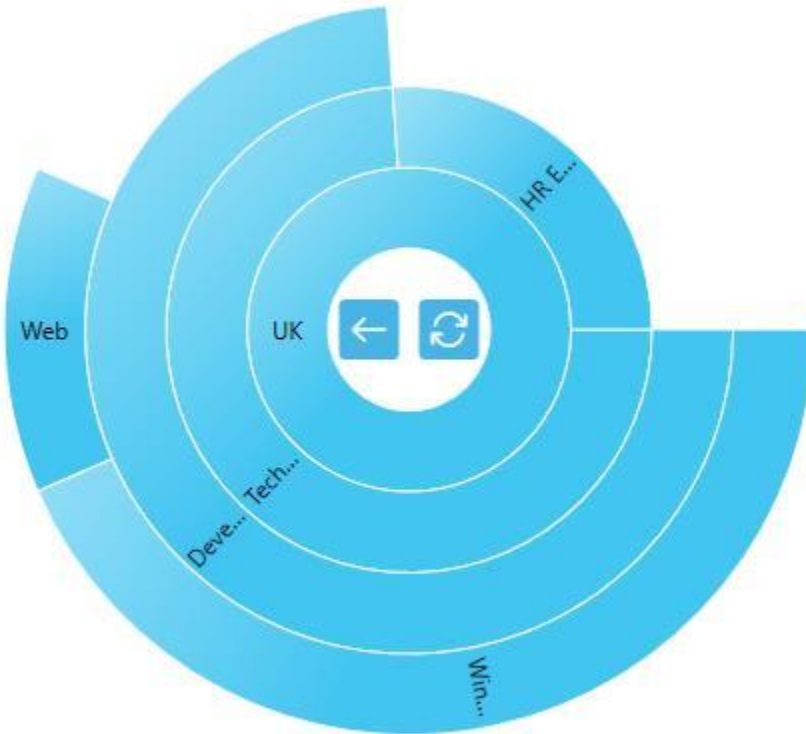
You can customize the zooming toolbar using the following properties.

- [ToolBarItemHeight](#) – Gets or sets height for the toolbar item.
- [ToolBarItemWidth](#) – Gets or sets width for the toolbar item.
- [ToolBarItemMargin](#) – Gets or sets margin of the toolbar item.

### XML

```
<sunburst:SfSunburstChart.Behaviors>
<sunburst:SunburstZoomingBehavior EnableZooming="True"
ToolBarHorizontalAlignment="Center"
ToolBarVerticalAlignment="Center"
ToolBarItemHeight="20"
ToolBarItemWidth="40"
ToolBarItemMargin="5,5,5,5"/>
</sunburst:SfSunburstChart.Behaviors>
```

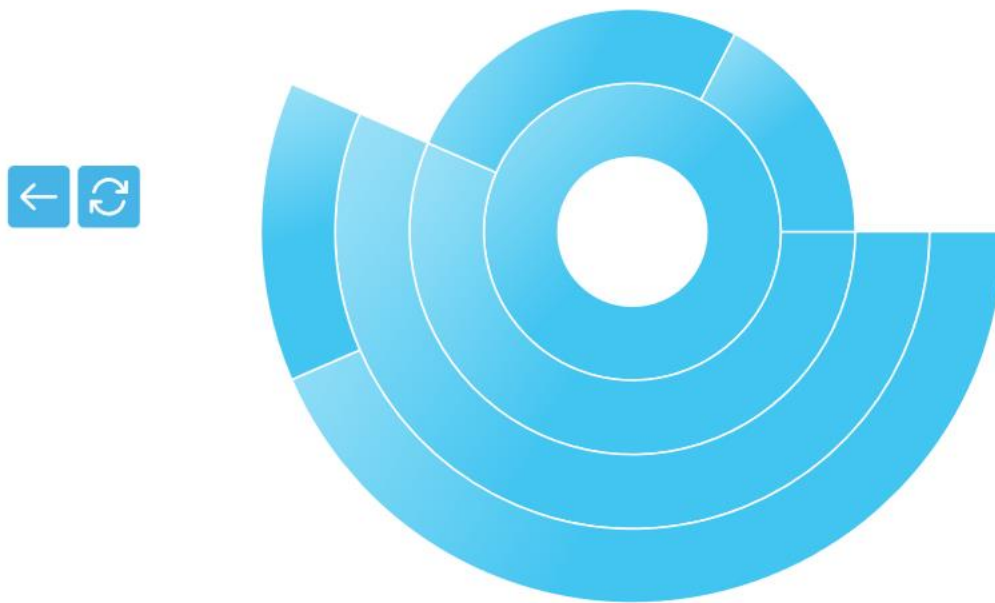
```
ToolBarItemHeight="30"  
ToolBarItemWidth="50"  
ToolBarItemMargin="5"/>  
</sunburst:SfSunburstChart.Behaviors>
```



Toolbar position can also be moved using the [ToolBarOffsetX](#) and [ToolBarOffsetY](#) property. Offset values ranges from 0 to 1 from left to right for x position and top to bottom for y position.

#### XML

```
<sunburst:SfSunburstChart.Behaviors>  
<sunburst:SunburstZoomingBehavior EnableZooming="True"  
ToolBarOffsetX="0.1" ToolBarOffsetY="0.5"/>  
</sunburst:SfSunburstChart.Behaviors>
```



### Animation in WPF Sunburst Chart (SfSunburstChart)

Sunburst chart allows you to animate the chart segments. You can enable animation using [EnableAnimation](#) property. Also you can set the duration for animation by using [AnimationDuration](#) property.

#### XML

```
<sunburst:SfSunburstChart EnableAnimation="True"
AnimationDuration="5000">
</sunburst:SfSunburstChart>
```

#### C#

```
sunburstChart.EnableAnimation = true;
sunburstChart.AnimationDuration = 5000;
```

### Animation types

Sunburst chart provide options to animate the chart segments in different ways using [AnimationType](#) property.

FadeIn – It gradually changes opacity of the chart segment.

Rotation – During an animation, control rotate from 0 to 360 angle.

#### *FadeIn*

The following example shows, how to enable the FadeIn animation.

#### XML

```
<sunburst:SfSunburstChart EnableAnimation="True"
AnimationType="FadeIn">
</sunburst:SfSunburstChart>
```

**C#**

```
sunburstChart.EnableAnimation = true;
sunburstChart.AnimationType = AnimationType.FadeIn;
```

*Rotation*

The following example shows, how to enable the Rotation animation.

**XML**

```
<sunburst:SfSunburstChart EnableAnimation="True"
  AnimationType="Rotation">
</sunburst:SfSunburstChart>
```

**C#**

```
sunburstChart.EnableAnimation = true;
sunburstChart.AnimationType = AnimationType.Rotation;
```



### Palette in WPF Sunburst Chart (SfSunburstChart)

Sunburst chart provides support to apply different types of palettes. You can define the palettes by using [Palette](#) property.

We have some predefined palette such as

- Metro
- AutumnBrights
- FloraHues
- Pineapple
- TomatoSpectrum
- RedChrome
- PurpleChrome
- BlueChrome
- GreenChrome
- Elite
- LightCandy
- SandyBeach

### Applying palette to the chart

Each palette applies a set of predefined brushes to the sunburst chart in a predefined order. The following code shows how to set predefined palette for sunburst chart.



## XML

```
<sunburst:SfSunburstChart ItemsSource="{Binding Data}"
ValueMemberPath="EmployeesCount"
Palette="SandyBeach" >
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
</sunburst:SfSunburstChart.Levels>
</sunburst:SfSunburstChart>
```

## C#

```
SfSunburstChart sunburst = new SfSunburstChart();
sunburst.ValueMemberPath = "EmployeesCount";
sunburst.SetBinding(SfSunburstChart.ItemsSourceProperty, "Data");
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
sunburst.Palette = SunburstColorPalette.SandyBeach;
grid.Children.Add(sunburst);
```





### Custom Palette

Sunburst chart provides option which enables you to define your own color brushes with your preferred order for the Palette, using [ColorModel](#) as shown in the following code example.

#### XML

```
<sunburst:SfSunburstChart ItemsSource="{Binding Data}"
ValueMemberPath="EmployeesCount"
Palette="Custom" >
  <sunburst:SfSunburstChart.Levels>
    <sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
    <sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
    <sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
    <sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
  </sunburst:SfSunburstChart.Levels>
  <sunburst:SfSunburstChart.ColorModel>
    <sunburst:SunburstColorModel>
      <sunburst:SunburstColorModel.CustomBrushes>
        <LinearGradientBrush>
          <GradientStop Color="DeepPink" Offset="0"/>
          <GradientStop Color="Gray" Offset="1"/>
        </LinearGradientBrush>
        <LinearGradientBrush>
          <GradientStop Color="Green" Offset="0"/>
          <GradientStop Color="Yellow" Offset="1"/>
        </LinearGradientBrush>
        <LinearGradientBrush>
          <GradientStop Color="Orange" Offset="0"/>
          <GradientStop Color="Brown" Offset="1"/>
        </LinearGradientBrush>
        <LinearGradientBrush>
          <GradientStop Color="DarkViolet" Offset="0"/>
          <GradientStop Color="White" Offset="1"/>
        </LinearGradientBrush>
      </sunburst:SunburstColorModel.CustomBrushes>
    </sunburst:SunburstColorModel>
  </sunburst:SfSunburstChart.ColorModel>
</sunburst:SfSunburstChart>
```

#### C#

```
SfSunburstChart sunburst = new SfSunburstChart();
sunburst.ValueMemberPath = "EmployeesCount";
sunburst.SetBinding(SfSunburstChart.ItemsSourceProperty, "Data");
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"Country" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobDescription" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobGroup" });
sunburst.Levels.Add(new SunburstHierarchicalLevel() { GroupMemberPath =
"JobRole" });
sunburst.Palette = SunburstColorPalette.Custom;
SunburstColorModel colorModel = new SunburstColorModel();
LinearGradientBrush brush1 = new LinearGradientBrush();
```

```
brush1.GradientStops.Add(new GradientStop() { Color = Colors.DeepPink,
Offset = 0 });
brush1.GradientStops.Add(new GradientStop() { Color = Colors.Gray, Offset =
1 });
LinearGradientBrush brush2 = new LinearGradientBrush();
brush2.GradientStops.Add(new GradientStop() { Color = Colors.Green, Offset =
0 });
brush2.GradientStops.Add(new GradientStop() { Color = Colors.Yellow, Offset
= 1 });
LinearGradientBrush brush3 = new LinearGradientBrush();
brush3.GradientStops.Add(new GradientStop() { Color = Colors.Orange, Offset
= 0 });
brush3.GradientStops.Add(new GradientStop() { Color = Colors.Brown, Offset =
1 });
LinearGradientBrush brush4 = new LinearGradientBrush();
brush4.GradientStops.Add(new GradientStop() { Color = Colors.DarkViolet,
Offset = 0 });
brush4.GradientStops.Add(new GradientStop() { Color = Colors.White, Offset =
1 });
colorModel.CustomBrushes.Add(brush1);
colorModel.CustomBrushes.Add(brush2);
colorModel.CustomBrushes.Add(brush3);
colorModel.CustomBrushes.Add(brush4);
sunburst.ColorModel = colorModel;
grid.Children.Add(sunburst);
```



---

**Note:** Before applying the color model, you need to set the palette value as Custom.

---

## Events in WPF Sunburst Chart (SfSunburstChart)

### SegmentCreated Event

This event occurs when segment is created. You can get the segment details as argument from the [SunburstSegmentCreatedEventArgs](#) handler.

The following example shows how to set the color for each level after creating the segment.

#### XML

```
<sunburst:SfSunburstChart ItemsSource="{Binding Data}"
ValueMemberPath="EmployeesCount"
SegmentCreated="chart_SegmentCreated">
<sunburst:SfSunburstChart.Levels>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="Country"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobDescription"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobGroup"/>
<sunburst:SunburstHierarchicalLevel GroupMemberPath="JobRole"/>
</sunburst:SfSunburstChart.Levels>
</sunburst:SfSunburstChart>
```

#### C#

```
private void chart_SegmentCreated(object sender,
SunburstSegmentCreatedEventArgs e)
{
    byte r = 180, g = 0, b = 96;
    byte r1 = 255, g1 = 153, b1 = 0;
    byte r2 = 102, g2 = 153, b2 = 0;
    byte r3 = 102, g3 = 204, b3 = 255;
    if (e.Segment.CurrentLevel == 0)
        e.Segment.Interior = new SolidColorBrush(Color.FromRgb(r3, g3, b3));
    else if (e.Segment.CurrentLevel == 1)
        e.Segment.Interior = new SolidColorBrush(Color.FromRgb(r1, g1, b1));
    else if (e.Segment.CurrentLevel == 2)
        e.Segment.Interior = new SolidColorBrush(Color.FromRgb(r, g, b));
    else if (e.Segment.CurrentLevel == 3)
        e.Segment.Interior = new SolidColorBrush(Color.FromRgb(r2, g2, b2));
}
```



### SelectionChanged

This event occurs whenever you select the segment by enabling selection behavior. You can get the selected segment details as argument from [SunburstSelectionChangedEventArgs](#) handler.

The following examples shows how to set the selected segment information.

### XML

```
<sunburst:SfSunburstChart ItemsSource="{Binding Data}"
ValueMemberPath="EmployeesCount"
SelectionChanged="chart_SelectionChanged">
  <sunburst:SfSunburstChart.Behaviors>
    <sunburst:SunburstSelectionBehavior EnableSelection="True"
SelectionCursor="Hand"/>
  </sunburst:SfSunburstChart.Behaviors>
</sunburst:SfSunburstChart>
<Popup x:Name="popup" Placement="MousePoint" Height="180" Width="180" >
  <Border Background="White" BorderBrush="Black" BorderThickness="3" >
    <Grid Margin="20,5,5,5">
      <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
      </Grid.ColumnDefinitions>
```

```

<TextBlock Text="Category : " FontWeight="Bold" Grid.Row="0"
Grid.Column="0"/>
<TextBlock x:Name="category"    FontWeight="Bold" Grid.Row="0"
Grid.Column="1"/>
<TextBlock Text="Value : " FontWeight="Bold" Grid.Row="1" Grid.Column="0"/>
<TextBlock x:Name="value"    FontWeight="Bold" Grid.Row="1" Grid.Column="1"/>
<TextBlock Text="Level : " FontWeight="Bold" Grid.Row="2" Grid.Column="0"/>
<TextBlock x:Name="level"    FontWeight="Bold" Grid.Row="2" Grid.Column="1"/>
<TextBlock Text="SliceIndex : " FontWeight="Bold" Grid.Row="3"
Grid.Column="0"/>
<TextBlock x:Name="sliceIndex"    FontWeight="Bold" Grid.Row="3"
Grid.Column="1"/>
</Grid>
</Border>
</Popup>

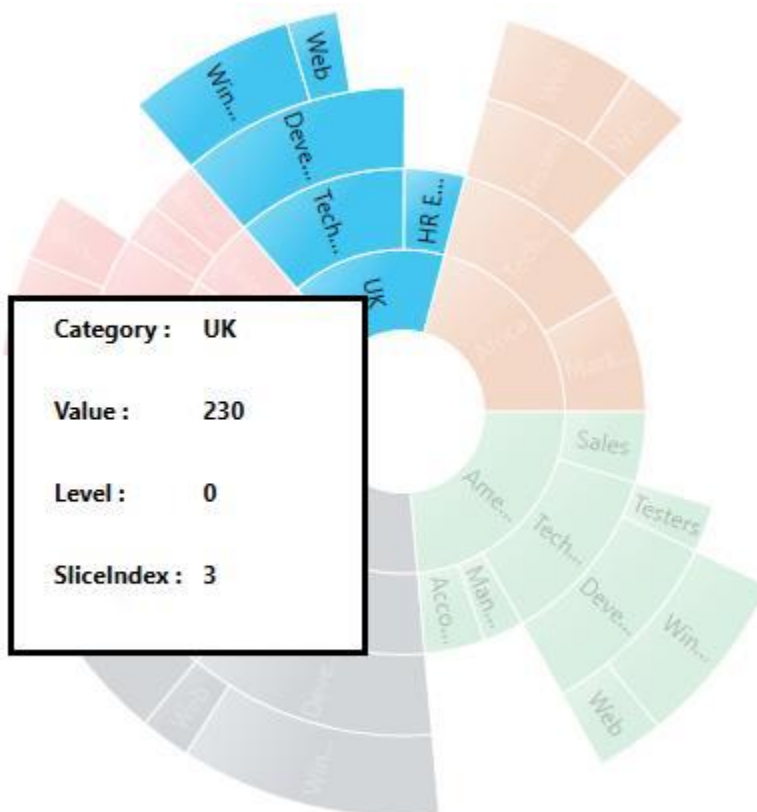
```

**C#**

```

private void chart_SelectionChanged(object sender,
SunburstSelectionChangedEventArgs e)
{
    popup.IsOpen = true;
    category.Text = e.SelectedSegment.Category.ToString();
    value.Text = e.SelectedSegment.Value.ToString();
    level.Text = e.SelectedSegment.CurrentLevel.ToString();
    sliceIndex.Text = e.SelectedSegment.SliceIndex.ToString();
}

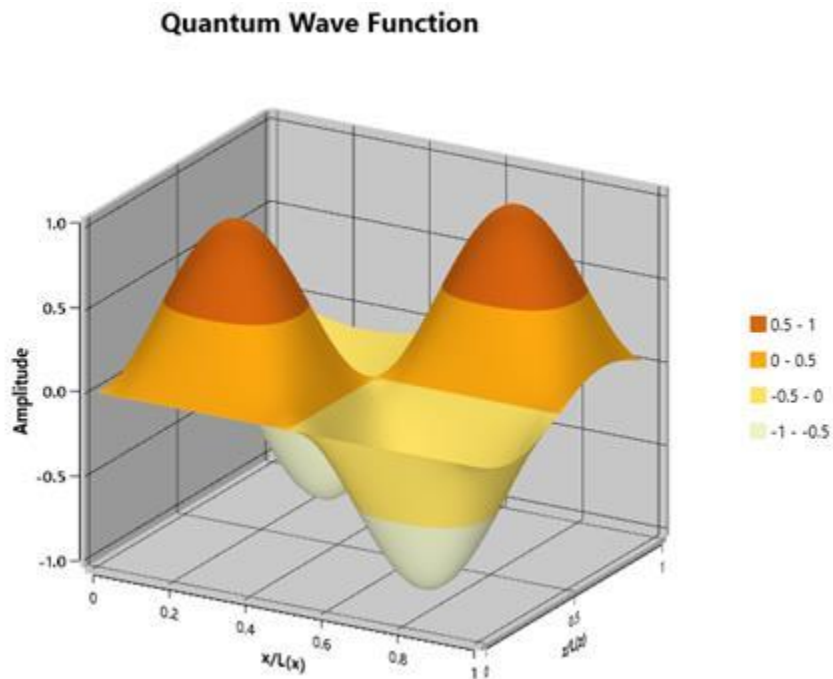
```



## SfSurfaceChart

### WPF Surface Chart (SfSurfaceChart) Overview

Essential Surface Chart shows a three-dimensional surface that connects a set of data points.



### Key Feature of Surface Chart

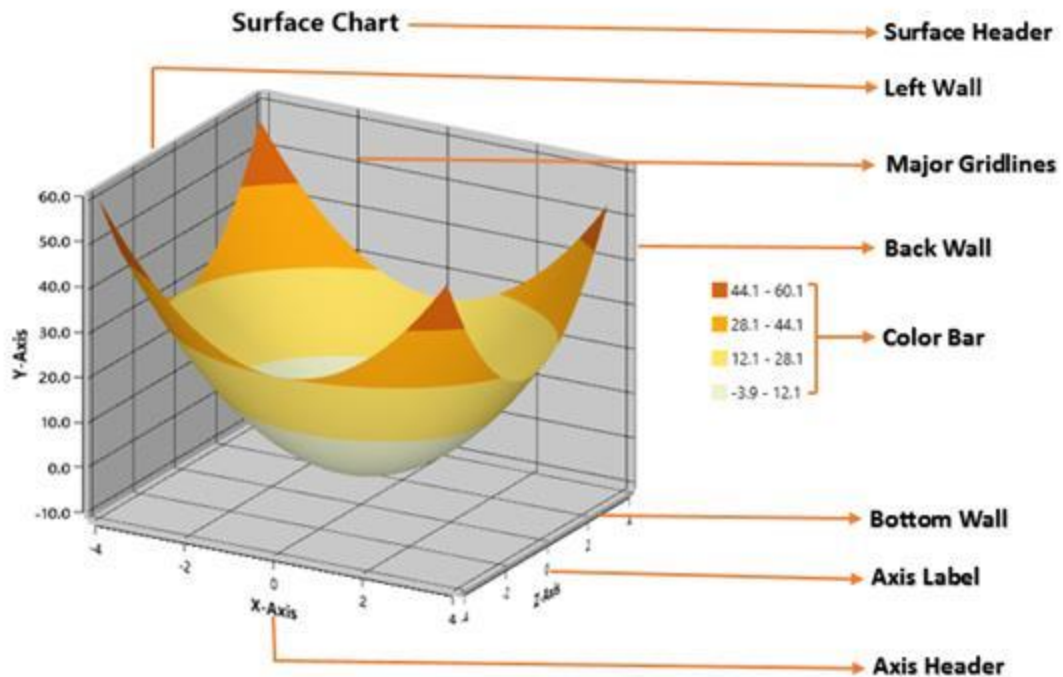
- ColorBar represents a range of values.
- Built-in palettes.
- Supports gradient brushes.
- Perspective and Orthographic view.
- Contour and wireframe support.

### Getting Started with WPF Surface Chart (SfSurfaceChart)

This section helps you get started with [SfSurfaceChart](#) control.

#### Visual Structure

The following image helps you understand various elements in SfSurfaceChart when you use it for the first time.



- Surface Header – Represents the title of surface chart
- Wall – Represents the wall that bounds the surface chart.
- Major Gridlines- Represents the surface axis gridline for surface chart.
- Color Bar- Displays the value range in color for surface chart.
- Axis Label- Displays the label for surface axis.
- Axis Header- Displays the header of Surface axis.

### Create simple surface chart from XAML

This section demonstrates how to create a surface chart using the **SfSurfaceChart** control from XAML. You can add the [SfChart](#) assembly as a reference to your application to create a surface chart using SfSurfaceChart control from XAML.

#### Add Assembly reference

1. Open the Add Reference window from your project.
2. Choose Windows > Extensions > Syncfusion.SfChart.WPF.
3. Add the following namespace in your XAML page.

#### XML

```
xmlns:syncfusion="clr-namespace:Syncfusion.UI.Xaml.Charts;assembly=Syncfusion.SfChart.WPF"
```

**Note:** Adding the extension Syncfusion.SfChart.WPF, adds all the Syncfusion WPF controls. You can also add the SfChart reference alone from the following location. C:\Program Files (x86)\Syncfusion\Essential Studio<version>\Assemblies (4.0/4.5.1/4.6)

### *Initialize the surface chart*

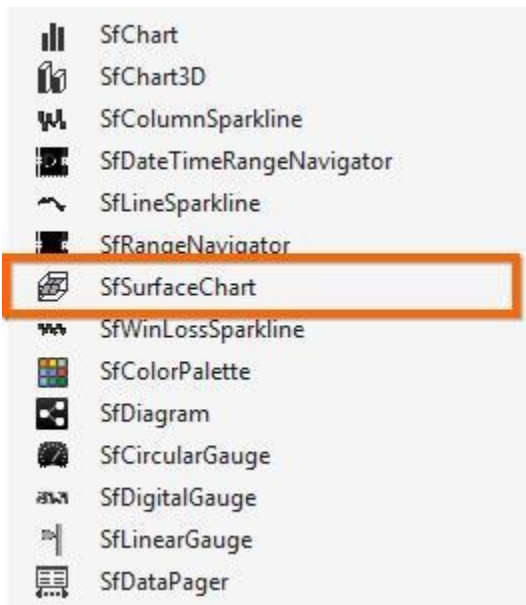
You need to initialize the surface chart from the following namespace Syncfusion.UI.Xaml.Charts.

#### **XML**

```
<syncfusion:SfSurfaceChart>  
</syncfusion:SfSurfaceChart>
```

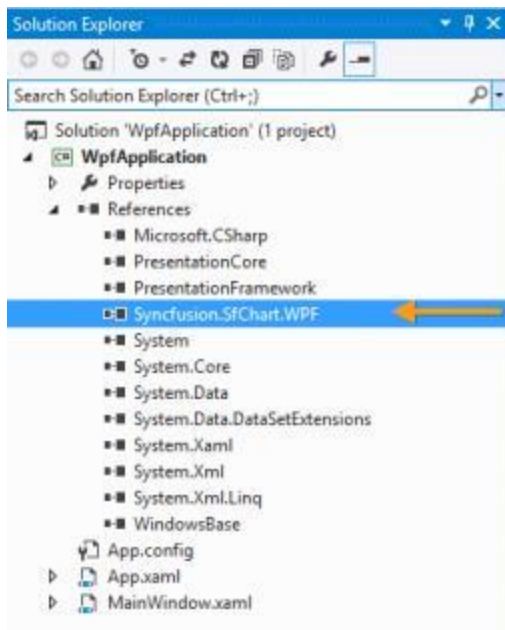
### *Add Surface chart from Toolbox*

Drag the [SfSurfaceChart](#) control from the Toolbox to the required location, where the surface chart should be displayed. You can select Toolbox from the View menu, when the toolbox window is not available in the project. The Toolbox window appears on the left side of the screen.



The Syncfusion WPF reference is added to the application reference and the xmlns namespace is added to MainPage.xaml.





```
<Window xmlns:syncfusion="http://schemas.syncfusion.com/wpf" x:Class="SurfaceChartDemo.
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:SurfaceChartDemo">

    <Grid>
        <syncfusion:SfSurfaceChart />
    </Grid>

</Window>
```

The following dataset is used to plot the surface chart.

X	0	0	0	1	1	1	2	2	2
Y	3	2	1	2	1	2	1	2	3
Z	0	1	2	0	1	2	0	1	2

Before proceeding with the chart, create data model with the above details as follows.

### C#

```
public class Data
{
    public double X { get; set; }
    public double Y { get; set; }
    public double Z { get; set; }
}
```

Now, you have a data model property which hold values of each item in the collection.

### C#

```
public ObservableCollection<Data> DataValues { get; set; }
```

Also you have a property for row and column size in the given data.

### C#

```
public int RowSize { get; set; }
public int ColumnSize { get; set; }
```

Add the values to the above defined collection property called DataValues, with the values illustrated in the above table.

### C#

```
DataValues.Add(new Data() { X = 0, Y = 3, Z = 0 });
DataValues.Add(new Data() { X = 0, Y = 2, Z = 1 });
DataValues.Add(new Data() { X = 0, Y = 1, Z = 2 });
DataValues.Add(new Data() { X = 1, Y = 2, Z = 0 });
DataValues.Add(new Data() { X = 1, Y = 1, Z = 1 });
DataValues.Add(new Data() { X = 1, Y = 2, Z = 2 });
DataValues.Add(new Data() { X = 2, Y = 1, Z = 0 });
DataValues.Add(new Data() { X = 2, Y = 2, Z = 1 });
DataValues.Add(new Data() { X = 2, Y = 3, Z = 2 });
RowSize = 3;
ColumnSize = 3;
```

Now you can add the elements required for the above scenario to the Surface instance created.

#### *Add data to surface chart*

After you populate the data to the chart, you can bind the properties as shown in the following code example.

### XML

```
<syncfusion:SfSurfaceChart ItemsSource="{Binding DataValues}"
XBindingPath="X"
YBindingPath="Y" ZBindingPath="Z"
RowSize="{Binding RowSize}"
ColumnSize="{Binding ColumnSize}" >
```

#### *Add Header to the surface chart*

The header acts as the title of the surface chart created, to identify its purpose.

### XML

```
<syncfusion:SfSurfaceChart Header="Simple Surface" FontSize="20" />
```

#### *Add Axes to surface chart*

The following code example illustrates how to add XAxis, YAxis and ZAxis to the Surface chart.

### XML

```
<syncfusion:SfSurfaceChart ItemsSource="{Binding DataValues}"
XBindingPath="X"
YBindingPath="Y" ZBindingPath="Z" RowSize="{Binding RowSize}"
```

```

ColumnSize="{Binding ColumnSize}">
<syncfusion:SfSurfaceChart.XAxis>
<syncfusion:SurfaceAxis Header="X-Axis" />
</syncfusion:SfSurfaceChart.XAxis>
<syncfusion:SfSurfaceChart.YAxis>
<syncfusion:SurfaceAxis Header="Y-Axis" LabelFormat="0.0"/>
</syncfusion:SfSurfaceChart.YAxis>
<syncfusion:SfSurfaceChart.ZAxis>
<syncfusion:SurfaceAxis Header="Z-Axis"/>
</syncfusion:SfSurfaceChart.ZAxis>
</syncfusion:SfSurfaceChart>

```

**Note:** SfSurfaceChart supports default axes, all the axes are generated automatically, even when it has not been defined explicitly.

#### Add Surface Type

The following code example illustrates how to add surface type to the surface chart.

#### XML

```

<syncfusion:SfSurfaceChart Type="Surface" />

```

#### Add Color bar to the surface chart

The following code example illustrates how to add color bar to the surface chart

#### XML

```

<syncfusion:SfSurfaceChart.ColorBar>
<syncfusion:ChartColorBar ShowLabel="True" DockPosition="Right"/>
</syncfusion:SfSurfaceChart.ColorBar>

```

Now you have created a simple surface chart. The following code example is for XAML.

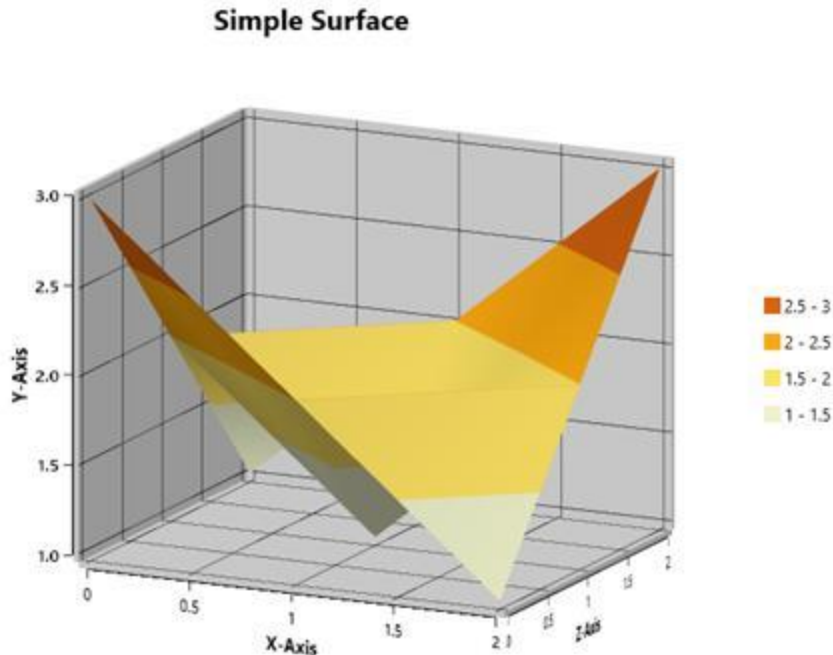
#### XML

```

<syncfusion:SfSurfaceChart Type="Surface" Tilt="15" Rotate="30"
ItemsSource="{Binding DataValues}" XBindingPath="X"
YBindingPath="Y" ZBindingPath="Z"
RowSize="{Binding RowSize}"
ColumnSize="{Binding ColumnSize}">
<syncfusion:SfSurfaceChart.XAxis>
<syncfusion:SurfaceAxis Header="X-Axis" />
</syncfusion:SfSurfaceChart.XAxis>
<syncfusion:SfSurfaceChart.YAxis>
<syncfusion:SurfaceAxis Header="Y-Axis" LabelFormat="0.0"/>
</syncfusion:SfSurfaceChart.YAxis>
<syncfusion:SfSurfaceChart.ZAxis>
<syncfusion:SurfaceAxis Header="Z-Axis"/>
</syncfusion:SfSurfaceChart.ZAxis>
<syncfusion:SfSurfaceChart.ColorBar>
<syncfusion:ChartColorBar DockPosition="Right"/>
</syncfusion:SfSurfaceChart.ColorBar>
</syncfusion:SfSurfaceChart>

```

The following output is displayed as a result of the above code example.



Create a simple surface chart from Code behind.

This section demonstrates how to create a surface chart using SfSurfaceChart control from code behind. For that, you can add the **SfChart** assembly reference to your application.

*Add Assembly reference*

1. Open the Add Reference window from your project.
2. Choose Windows > Extensions > Syncfusion.SfChart.WPF.
3. Add the following namespace in code behind

**C#**

```
using Syncfusion.UI.Xaml.Charts;
```

The following section demonstrates a simple surface chart with the data discussed in the above table.

*Initialize the surface chart*

**C#**

```
SfSurfaceChart surface = new SfSurfaceChart();
```

*Add Data to surface chart*

You can set the surface chart data in code behind, by directly adding data points to the Data property of surface chart using **AddPoints(x,y,z)** method as shown in the following code example.

**C#**

```
SfSurfaceChart surface = new SfSurfaceChart();  
//First Row  
surface.Data.AddPoints(0, 3, 0);  
surface.Data.AddPoints(0, 2, 1);
```

```
surface.Data.AddPoints(0, 2, 2);  
//Second Row  
surface.Data.AddPoints(1, 2, 0);  
surface.Data.AddPoints(1, 1, 1);  
surface.Data.AddPoints(1, 2, 2);  
//Third Row  
surface.Data.AddPoints(2, 1, 0);  
surface.Data.AddPoints(2, 2, 1);  
surface.Data.AddPoints(2, 3, 2);  
surface.RowSize = 3;  
surface.ColumnSize = 3;  
grid.Children.Add(surface);
```

#### Add Header to the Surface chart

The header acts as the title of the surface chart created, to identify its purpose.

#### C#

```
SfSurfaceChart chart = new SfSurfaceChart();  
chart.Header = "Simple Surface";  
chart.FontSize = 20;  
grid.Children.Add(chart);
```

#### Add Axes to surface chart

The following code example illustrates how to add XAxis, YAxis and ZAxis to the surface chart.

#### C#

```
SfSurfaceChart chart = new SfSurfaceChart();  
chart.SetBinding(SfSurfaceChart.ItemsSourceProperty, "DataValues");  
chart.SetBinding(SfSurfaceChart.RowSizeProperty, "RowSize");  
chart.SetBinding(SfSurfaceChart.ColumnSizeProperty, "ColumnSize");  
chart.XBindingPath = "X";  
chart.YBindingPath = "Y";  
chart.ZBindingPath = "Z";  
//Add X axis to surface  
SurfaceAxis xAxis = new SurfaceAxis();  
xAxis.Header = "X-Axis";  
chart.XAxis = xAxis;  
//Add Y axis to surface  
SurfaceAxis yAxis = new SurfaceAxis();  
yAxis.Header = "Y-Axis";  
yAxis.LabelFormat = "0.0";  
chart.YAxis = yAxis;  
//Add Z axis to surface  
SurfaceAxis zAxis = new SurfaceAxis();  
zAxis.Header = "Z-Axis";  
chart.ZAxis = zAxis;  
grid.Children.Add(chart);
```

---

**Note:** SfSurfaceChart supports default axes, where all the axes are generated automatically, even when they are not defined explicitly.

---

#### Add Surface Type

The following code example illustrates how to add surface type to the surface chart.

**C#**

```
SfSurfaceChart chart = new SfSurfaceChart();  
chart.Type = SurfaceType.Surface;  
grid.Children.Add(chart);
```

*Add ColorBar to the surface chart*

The following code example illustrates how to add color bar to the surface chart.

**C#**

```
ChartColorBar colorBar = new ChartColorBar();  
colorBar.DockPosition = ChartDock.Right;  
colorBar.ShowLabel = true;  
chart.ColorBar = colorBar;
```

Now, you have created a simple surface chart. The following code example is for code behind and data can be populated either by using the [ItemsSource](#) property or by using the [AddPoints](#) method of [Data](#) property in surface chart.

**C#**

```
SfSurfaceChart chart = new SfSurfaceChart();  
chart.SetBinding(SfSurfaceChart.ItemsSourceProperty, "DataValues");  
chart.SetBinding(SfSurfaceChart.RowSizeProperty, "RowSize");  
chart.SetBinding(SfSurfaceChart.ColumnSizeProperty, "ColumnSize");  
chart.XBindingPath = "X";  
chart.YBindingPath = "Y";  
chart.ZBindingPath = "Z";  
grid.Children.Add(chart);  
or  
SfSurfaceChart surface = new SfSurfaceChart();  
surface.Header = "Simple Surface";  
surface.Tilt = 15;  
surface.Rotate = 30;  
surface.Type = SurfaceType.Surface;  
//First Row  
surface.Data.AddPoints(0, 3, 0);  
surface.Data.AddPoints(0, 2, 1);  
surface.Data.AddPoints(0, 2, 2);  
//Second Row  
surface.Data.AddPoints(1, 2, 0);  
surface.Data.AddPoints(1, 1, 1);  
surface.Data.AddPoints(1, 2, 2);  
//Third Row  
surface.Data.AddPoints(2, 1, 0);  
surface.Data.AddPoints(2, 2, 1);  
surface.Data.AddPoints(2, 3, 2);  
surface.RowSize = 3;  
surface.ColumnSize = 3;  
//Add X axis to surface  
SurfaceAxis xAxis = new SurfaceAxis();  
xAxis.Header = "X-Axis";  
surface.XAxis = xAxis;  
//Add Y axis to surface  
SurfaceAxis yAxis = new SurfaceAxis();
```

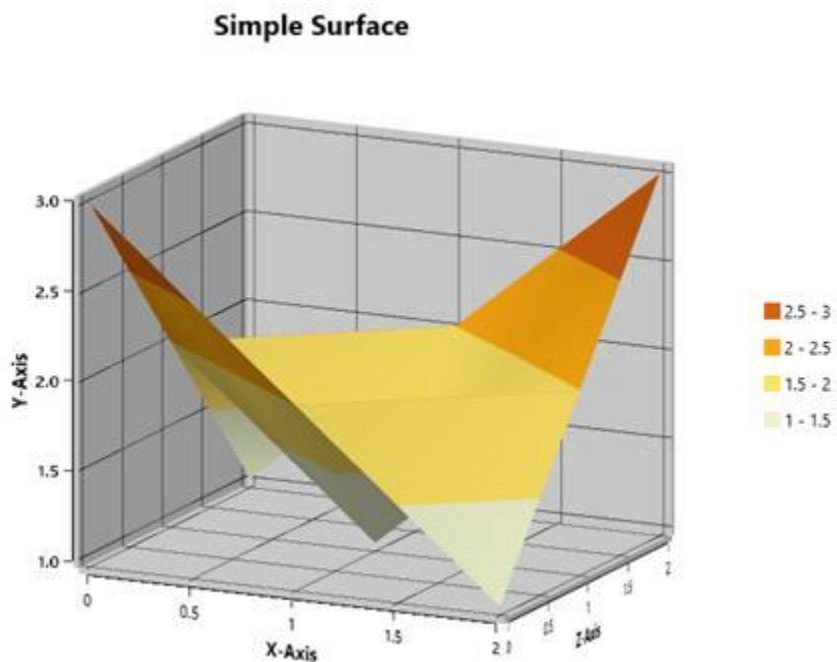
```

yAxis.Header = "Y-Axis";
surface.YAxis = yAxis;
//Add Z axis to surface
SurfaceAxis zAxis = new SurfaceAxis();
zAxis.Header = "Z-Axis";
surface.ZAxis = zAxis;
ChartColorBar colorBar = new ChartColorBar();
colorBar.DockPosition = ChartDock.Right;
colorBar.ShowLabel = true;
surface.ColorBar = colorBar;
this.Content = surface;

```

The following output is displayed as a result of the above code example.

You can get the complete getting started sample [here](#).



### Data Binding in WPF Surface Chart (SfSurfaceChart)

In surface chart, you can apply data in a grid table format, that contains the number of rows and columns as shown in the following table.

	$Z_0$	$Z_1$	$Z_2$	$Z_n$
$X_1$	$Y_{00}$	$Y_{01}$	$Y_{02}$	$Y_{0n}$
$X_2$	$Y_{10}$	$Y_{11}$	$Y_{12}$	$Y_{1n}$
$X_n$	$Y_{n0}$	$Y_{n1}$	$Y_{n2}$	$Y_{nn}$

You can apply the data in surface in two ways.

- Using ItemsSource property
- Directly passing value through Data.AddPoints method.

### Using ItemsSource

You can bind the IEnumerable collection property to the [ItemsSource](#) property of a surface chart. Each item holds the model properties that are used to map surface [XBindingPath](#), [YBindingPath](#) and [ZBindingPath](#) property.

Also, you must set the given data row and column size to surface chart [RowSize](#) and [ColumnSize](#) Properties.

### XML

```
<Grid.DataContext>
<local:ViewModel />
</Grid.DataContext>
<chart:SfSurfaceChart ItemsSource="{Binding DataValue}" XBindingPath="X"
YBindingPath="Y" ZBindingPath="Z" RowSize="{Binding RowSize}"
ColumnSize="{Binding ColumnSize}">
</chart:SfSurfaceChart>
```

### C#

```
SfSurfaceChart chart = new SfSurfaceChart();
chart.SetBinding(SfSurfaceChart.ItemsSourceProperty, "DataValue");
chart.SetBinding(SfSurfaceChart.RowSizeProperty, "RowSize");
chart.SetBinding(SfSurfaceChart.ColumnSizeProperty, "ColumnSize");
chart.XBindingPath = "X";
chart.YBindingPath = "Y";
chart.ZBindingPath = "Z";
ChartColorBar colorBar = new ChartColorBar();
colorBar.DockPosition = ChartDock.Right;
colorBar.ShowLabel = true;
chart.ColorBar = colorBar;
grid.Children.Add(chart);
public class Data
{
    public double X { get; set; }
    public double Y { get; set; }
    public double Z { get; set; }
}
public class ViewModel
{
    public ViewModel()
    {
        DataValue = new ObservableCollection<Data>();
        for (double x = -10; x < 10; x++)
        {
            for (double z = -10; z < 10; z++)
            {
                double y = x*Math.Sin(z) + z*Math.Sin(x);
                DataValue.Add(new Data() { X = x, Y = y, Z = z });
            }
        }
    }
}
```



```
}  
public ObservableCollection<Data> DataValue { get; set; }  
public int RowSize = 20;  
public int ColumnSize = 20;  
}
```

#### Using Data.AddPoints method

In this, you can directly pass the data points to the [Data](#) property [AddPoints\(x,y,z\)](#) method. Here, you no need to create items source and its member path. But, you need to specify provided data rows and column size.

#### XML

```
<chart:SfSurfaceChart x:Name="surface" />
```

#### C#

```
public MainWindow()  
{  
    InitializeComponent();  
    SetData();  
}  
private void SetData()  
{  
    for (double x = -10; x < 10; x++)  
    {  
        for (double z = -10; z < 10; z++)  
        {  
            double y = x*Math.Sin(z) + z*Math.Sin(x);  
            surface.Data.AddPoints(x,y,z); //here we can directly pass data  
        }  
    }  
    surface.RowSize = 20;  
    surface.ColumnSize = 20;  
}
```

### Surface Types in WPF Surface Chart (SfSurfaceChart)

Essential Surface Chart provides the following types to plot three dimensional data points.

- Surface
- WireframeSurface
- Contour
- WireframeContour

#### Surface

Surface charts are used to explore the relationship between three dimensional data.

The following code shows how to set the type of surface.

#### XML

```
<chart:SfSurfaceChart ItemsSource="{Binding DataValues}" XBindingPath="X"  
Type="Surface"
```

```
YBindingPath="Y" ZBindingPath="Z" RowSize="{Binding RowSize}"
ColumnSize="{Binding ColumnSize}" >
</chart:SfSurfaceChart>
```

**C#**

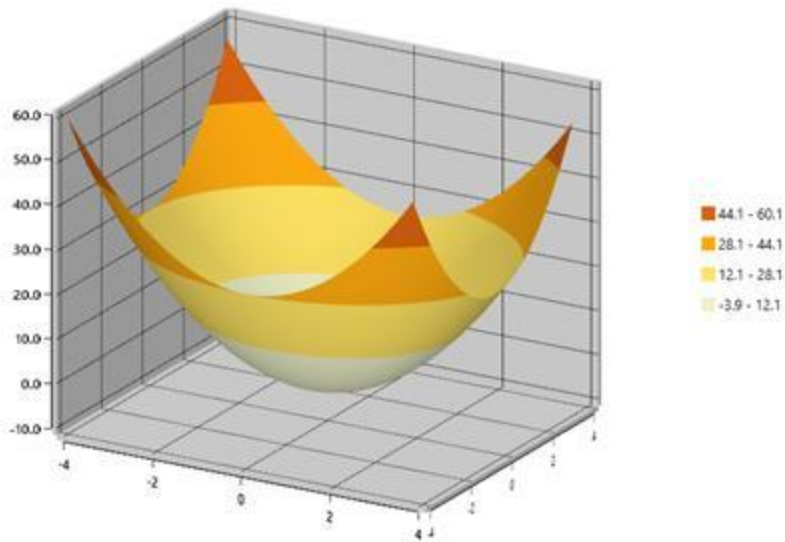
```
SfSurfaceChart chart = new SfSurfaceChart();
chart.Type = SurfaceType.Surface;
chart.SetBinding(SfSurfaceChart.ItemsSourceProperty, "DataValues");
chart.SetBinding(SfSurfaceChart.RowSizeProperty, "RowSize");
chart.SetBinding(SfSurfaceChart.ColumnSizeProperty, "ColumnSize");
chart.XBindingPath = "X";
chart.YBindingPath = "Y";
chart.ZBindingPath = "Z";
grid.Children.Add(chart);
```

The following code specifies the view model data bound to the chart for surface type.

**C#**

```
public class Data
{
    public double X { get; set; }
    public double Y { get; set; }
    public double Z { get; set; }
}

public class DataViewModel
{
    public ObservableCollection<Data> DataValues { get; set; }
    public int RowSize { get; set; }
    public int ColumnSize { get; set; }
    public DataViewModel()
    {
        DataValues = new ObservableCollection<Data>();
        DataValues = new ObservableCollection<Data>();
        double inc = 8.0 / 35;
        for (double x = -4; x < 4; x += inc)
        {
            for (double z = -4; z < 4; z += inc)
            {
                double y = 2 * (x * x) + 2 * (z * z) - 4;
                DataValues.Add(new Data() { X = x, Y = y, Z = z });
            }
        }
        RowSize = 35;
        ColumnSize = 35;
    }
}
```



### WireframeSurface

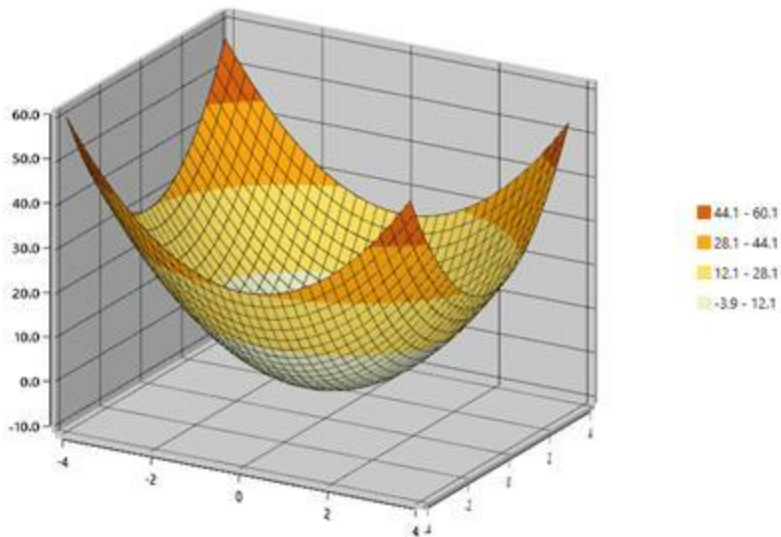
You can draw the wireframe or mesh, for the surface chart.

### XML

```
<chart:SfSurfaceChart ItemsSource="{Binding DataValues}" XBindingPath="X"
Type="WireframeSurface"
YBindingPath="Y" ZBindingPath="Z" RowSize="{Binding RowSize}"
ColumnSize="{Binding ColumnSize}" >
</chart:SfSurfaceChart>
```

### C#

```
SfSurfaceChart chart = new SfSurfaceChart();
chart.Type = SurfaceType.WireframeSurface;
chart.SetBinding(SfSurfaceChart.ItemsSourceProperty, "DataValues");
chart.SetBinding(SfSurfaceChart.RowSizeProperty, "RowSize");
chart.SetBinding(SfSurfaceChart.ColumnSizeProperty, "ColumnSize");
chart.XBindingPath = "X";
chart.YBindingPath = "Y";
chart.ZBindingPath = "Z";
grid.Children.Add(chart);
```



### Contour

Viewing the surface chart from the top is called contour. It is a graphical technique that represents the three dimensional surface in a two dimensional format.

### XML

```
<chart:SfSurfaceChart ItemsSource="{Binding DataValues}" XBindingPath="X"
Type="Contour" Rotate="0"
YBindingPath="Y" ZBindingPath="Z" RowSize="{Binding RowSize}"
ColumnSize="{Binding ColumnSize}" >
<chart:SfSurfaceChart.ColorBar>
<chart:ChartColorBar ShowLabel="True" DockPosition="Right"/>
</chart:SfSurfaceChart.ColorBar>
</chart:SfSurfaceChart>
```

### C#

```
SfSurfaceChart chart = new SfSurfaceChart();
chart.Type = SurfaceType.Contour;
chart.Rotate = 0;
ChartColorBar colorBar = new ChartColorBar();
colorBar.DockPosition = ChartDock.Right;
colorBar.ShowLabel = true;
chart.ColorBar = colorBar;
SetData();
grid.Children.Add(chart);
```

The following code specifies the view model data bound to the chart for contour type.

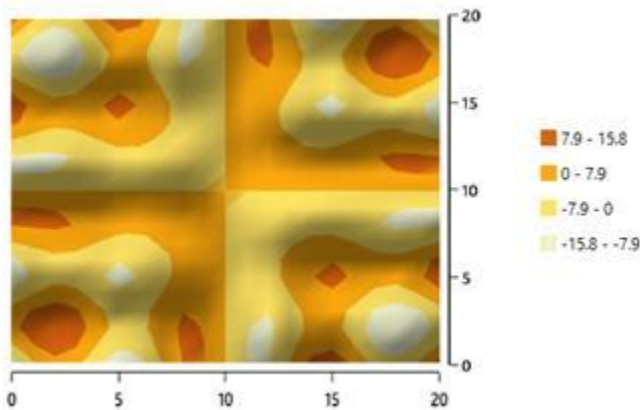
### C#

```
private void SetData()
{
    int x = 0;
    for (double i = -10; i <= 10; i++, x++)
    {
```

```

int z = 0;
for (double j = -10; j <= 10; j++, z++)
{
    double y = i * Math.Sin(j) + j * Math.Sin(i);
    chart.Data.AddPoints(x, y, z);
}
}
chart.RowSize = 21;
chart.ColumnSize = 21;
}

```



### WireframeContour

You can draw the wireframe or mesh for the contour chart

### XML

```

<chart:SfSurfaceChart ItemsSource="{Binding DataValues}" XBindingPath="X"
Type="WireframeContour"
YBindingPath="Y" ZBindingPath="Z" RowSize="{Binding RowSize}" Rotate="0"
ColumnSize="{Binding ColumnSize}" >
<chart:SfSurfaceChart.ColorBar>
<chart:ChartColorBar ShowLabel="True" DockPosition="Right"/>
</chart:SfSurfaceChart.ColorBar>
</chart:SfSurfaceChart>

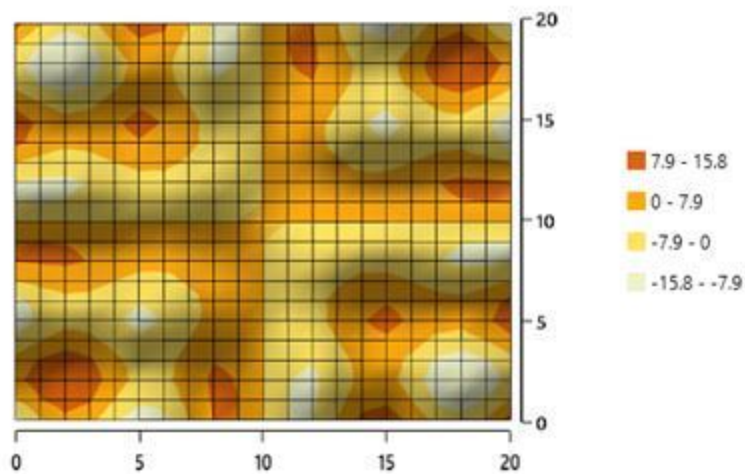
```

### C#

```

SfSurfaceChart chart = new SfSurfaceChart();
chart.Type = SurfaceType.WireframeContour;
chart.Rotate = 0;
ChartColorBar colorBar = new ChartColorBar();
colorBar.DockPosition = ChartDock.Right;
colorBar.ShowLabel = true;
chart.ColorBar = colorBar;
SetData();
grid.Children.Add(chart);

```



### Surface Axis in WPF Surface Chart (SfSurfaceChart)

SurfaceAxis is used to locate a data point inside the surface area. In surface, you require three axis to locate data points, such as X-Axis, Y-Axis and Z-Axis. You can define axis in surface using the following code example. If you do not define the axis, then it automatically takes the default axis with default properties values.

XAML:

#### XML

```
<chart:SfSurfaceChart>
  <chart:SfSurfaceChart.XAxis>
    <chart:SurfaceAxis />
  </chart:SfSurfaceChart.XAxis>
  <chart:SfSurfaceChart.YAxis>
    <chart:SurfaceAxis />
  </chart:SfSurfaceChart.YAxis>
  <chart:SfSurfaceChart.ZAxis>
    <chart:SurfaceAxis />
  </chart:SfSurfaceChart.ZAxis>
</chart:SfSurfaceChart />
```

#### C#

```
SfSurfaceChart surface = new SfSurfaceChart();
surface.XAxis = new SurfaceAxis();
surface.YAxis = new SurfaceAxis();
surface.ZAxis = new SurfaceAxis();
```

### Axis customization.

Axis of the surface chart can be customized using the following properties.

The following APIs are used to customize the surface axis.

*Properties*

Name	Definition
<a href="#">Header</a>	Gets or sets the object that represents the content of a surface axis header. This property is used to specify any object as Header for surface axis.
<a href="#">HeaderTemplate</a>	Gets or sets template for surface axis header.
<a href="#">LabelTemplate</a>	Gets or sets template for surface axis label.
<a href="#">LabelFormat</a>	Gets or sets format for surface axis label.
<a href="#">Minimum</a>	Gets or sets the double that represents the minimum value for the axis.
<a href="#">Maximum</a>	Gets or sets the double that represents the maximum value for the axis.
<a href="#">RangePadding</a>	Gets or sets NumericalPadding that specifies how to render the surface in surface area.
<a href="#">EdgeLabelsDrawingMode</a>	Gets or sets EdgeLabelsDrawingMode that specifies how to place edge axis label.
<a href="#">Interval</a>	Gets or sets the double that represents the interval between labels.
<a href="#">SmallTicksPerInterval</a>	Gets or sets the double that represents the small ticks per interval.
<a href="#">TickLineSize</a>	Gets or sets the double that represents the axis tick line size.
<a href="#">ShowGridLines</a>	Gets or sets bool that represent whether displaying the axis grid lines.
<a href="#">GridLineStroke</a>	Gets or sets the brush for grid line stroke.
<a href="#">GridLineThickness</a>	Gets or sets the double for grid line thickness.
<a href="#">AxisLineStyle</a>	Gets or sets the style for axis line.
<a href="#">MajorTickLineStyle</a>	Gets or sets the style for axis major tick lines.
<a href="#">MinorTickLineStyle</a>	Gets or sets the style for axis minor tick lines.

The following code snippet explains how to customize the axis using the above properties.

#### XML

```
<Grid.Resources>
<DataTemplate x:Key="labelTemplate">
<TextBlock Text="{Binding LabelContent}" Foreground="Red"></TextBlock>
</DataTemplate>
<DataTemplate x:Key="headerTemplate">
<TextBlock Text="YAxis" FontFamily="Comic Sans MS"></TextBlock>
</DataTemplate>
<Style TargetType="Line" x:Key="lineStyle">
<Setter Property="Stroke" Value="Green"/>
<Setter Property="StrokeThickness" Value="2"/>
</Style>
</Grid.Resources>
<chart:SfSurfaceChart ItemsSource="{Binding DataValues}" XBindingPath="X"
Type="Surface"
YBindingPath="Y" ZBindingPath="Z" RowSize="{Binding RowSize}"
ColumnSize="{Binding ColumnSize}">
<chart:SfSurfaceChart.XAxis>
<chart:SurfaceAxis GridLineStroke="Red" GridLineThickness="1"
SmallTicksPerInterval="1" Header="XAxis" AxisLineStyle="{StaticResource
lineStyle }"
LabelTemplate="{StaticResource labelTemplate}">
</chart:SurfaceAxis>
</chart:SfSurfaceChart.XAxis>
<chart:SfSurfaceChart.YAxis>
<chart:SurfaceAxis HeaderTemplate="{StaticResource headerTemplate}" />
</chart:SfSurfaceChart.YAxis>
<chart:SfSurfaceChart.ZAxis>
<chart:SurfaceAxis />
</chart:SfSurfaceChart.ZAxis>
</chart:SfSurfaceChart>
```

#### C#

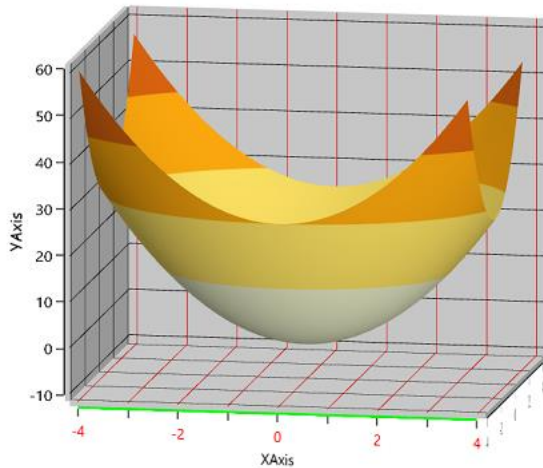
```
chart = new SfSurfaceChart();
chart.Type = SurfaceType.Surface;
chart.SetBinding(SfSurfaceChart.ItemsSourceProperty, "DataValues");
chart.SetBinding(SfSurfaceChart.RowSizeProperty, "RowSize");
chart.SetBinding(SfSurfaceChart.ColumnSizeProperty, "ColumnSize");
chart.XBindingPath = "X";
chart.YBindingPath = "Y";
chart.ZBindingPath = "Z";
SurfaceAxis xAxis = new SurfaceAxis();
xAxis.Header = "X-Axis";
chart.XAxis = xAxis;
xAxis.GridLineStroke = new SolidColorBrush(Colors.Red);
xAxis.GridLineThickness = 1;
xAxis.SmallTicksPerInterval = 1;
xAxis.AxisLineStyle = grid.Resources["lineStyle"] as Style;
xAxis.LabelTemplate = grid.Resources["labelTemplate"] as DataTemplate;
SurfaceAxis yAxis = new SurfaceAxis();
yAxis.Header = "Y-Axis";
```



```

chart.YAxis = yAxis;
yAxis.HeaderTemplate = grid.Resources["headerTemplate"] as DataTemplate;
SurfaceAxis zAxis = new SurfaceAxis();
zAxis.Header = "Z-Axis";
chart.ZAxis = zAxis;
grid.Children.Add(chart);

```



### ColorBar in WPF Surface Chart (SfSurfaceChart)

[ColorBar](#) is used to represent the value range in surface via colors. You can define ColorBar for surface chart as shown in the following code example.

Color bar position can be customized using the [DockPosition](#) property.

Color bar can either show or hide the labels and this can be done using the [ShowLabel](#) property.

#### XML

```

<chart:SfSurfaceChart ItemsSource="{Binding DataValues}" XBindingPath="X"
YBindingPath="Y" ZBindingPath="Z" RowSize="{Binding RowSize}"
ColumnSize="{Binding ColumnSize}">
  <chart:SfSurfaceChart.ColorBar>
    <chart:ChartColorBar ShowLabel="True"
DockPosition="Right"></chart:ChartColorBar>
  </chart:SfSurfaceChart.ColorBar>
</chart:SfSurfaceChart>

```

#### C#

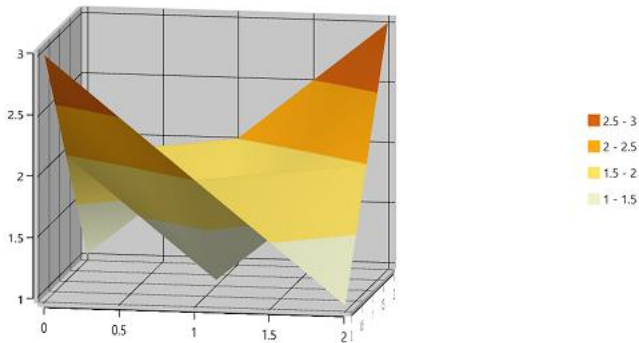
```

SfSurfaceChart chart = new SfSurfaceChart();
chart.SetBinding(SfSurfaceChart.ItemsSourceProperty, "DataValues");
chart.SetBinding(SfSurfaceChart.RowSizeProperty, "RowSize");
chart.SetBinding(SfSurfaceChart.ColumnSizeProperty, "ColumnSize");
chart.XBindingPath = "X";
chart.YBindingPath = "Y";
chart.ZBindingPath = "Z";
ChartColorBar colorBar = new ChartColorBar();
colorBar.DockPosition = ChartDock.Right;
colorBar.ShowLabel = true;
chart.ColorBar = colorBar;

```

```
grid.Children.Add(chart);
```

The following image represents the color bar at the right side of the surface chart.



### Palettes in WPF Surface Chart (SfSurfaceChart)

[WPF Surface Chart](#) provides options to apply different kinds of palettes.

Some of the predefined palettes include:

- Metro
- AutumnBrights
- FloraHues
- Pineapple
- TomotoSpectrum
- RedChrome
- PurpleChrome
- BlueChrome
- GreenChrome
- Elite
- LightCandy
- SandyBeach

### Applying Predefined Brushes

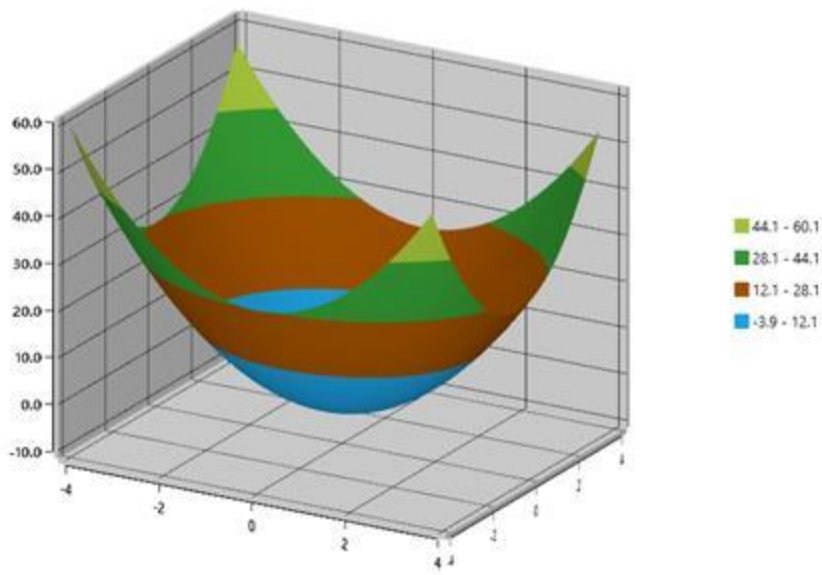
Using the above palette you can apply a set of predefined brushes to surface chart as shown in the following code example.

#### XML

```
<chart:SfSurfaceChart Palette="Metro">
</chart:SfSurfaceChart>
```

#### C#

```
SfSurfaceChart chart = new SfSurfaceChart();
chart.Palette = ChartColorPalette.Metro;
grid.Children.Add(chart);
```



### Applying Custom Brushes

The custom palette option enables you to define your own color brushes for the Palette using ColorModel property as given in the following code example.

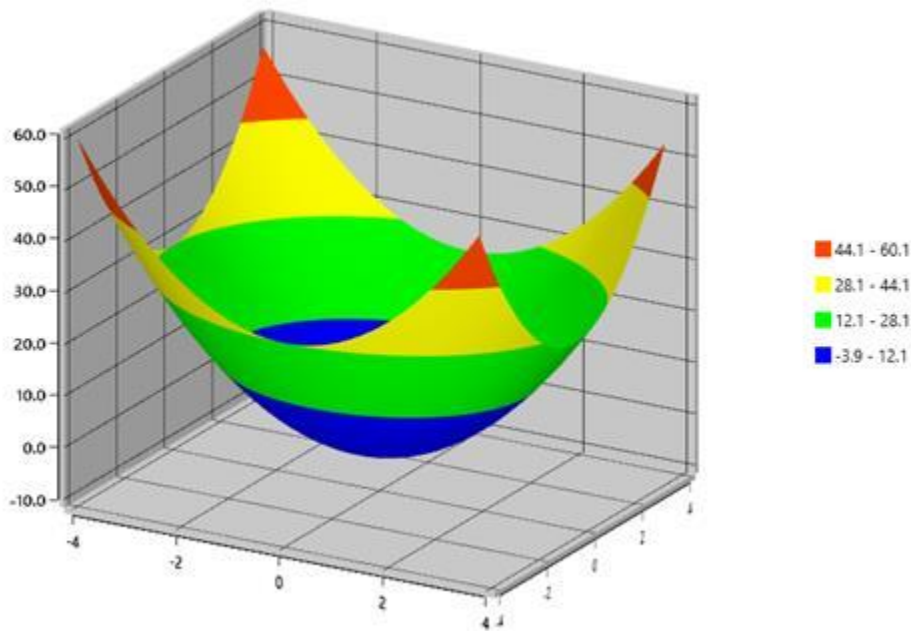
#### XML

```
<chart:SfSurfaceChart Palette="Custom" >
  <chart:SfSurfaceChart.ColorModel>
    <chart:ChartColorModel>
      <chart:ChartColorModel.CustomBrushes>
        <SolidColorBrush Color="Blue"/>
        <SolidColorBrush Color="Lime"/>
        <SolidColorBrush Color="Yellow"/>
        <SolidColorBrush Color="Blue"/>
        <SolidColorBrush Color="Lime"/>
        <SolidColorBrush Color="Yellow"/>
        <SolidColorBrush Color="OrangeRed"/>
      </chart:ChartColorModel.CustomBrushes>
    </chart:ChartColorModel>
  </chart:SfSurfaceChart.ColorModel>
</chart:SfSurfaceChart />
```

#### C#

```
SfSurfaceChart chart = new SfSurfaceChart();
chart.Palette = ChartColorPalette.Custom;
ChartColorModel colorModel = new ChartColorModel();
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.Blue));
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.Lime));
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.Yellow));
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.Blue));
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.Lime));
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.Yellow));
colorModel.CustomBrushes.Add(new SolidColorBrush(Colors.OrangeRed));
chart.ColorModel = colorModel;
```

```
grid.Children.Add(chart);
```



Surface Area in WPF Surface Chart (SfSurfaceChart)

Surface area represents the entire surface chart and all its elements. It's a virtual rectangle area that includes all the surface elements like axis, color bar, wall etc.

The surface chart can be customized to enrich your application’s look and feel. SfSurfaceChart provides API’s to customize surface area based on your requirement. This section explains about the elements and API for common customization of surface area.

Properties

Name	Description
<a href="#">EnableRotation</a>	Gets or sets the bool value that represent the value to enable the rotation for surface chart.
<a href="#">Rotate</a>	Gets or sets the double value that represents a rotation value for surface chart.
<a href="#">Tilt</a>	Gets or sets the double value that represents a tilt value for surface chart.
<a href="#">CameraProjection</a>	Gets or sets the value that represents type of camera projection in surface chart.
<a href="#">Header</a>	Gets or sets the object that represents the content of a surface header. This property is used to specify any object as Header for surface chart.

<a href="#">LeftWallBrush</a>	Gets or sets a brush for left wall.
<a href="#">BottomWallBrush</a>	Gets or sets a brush for bottom wall.
<a href="#">BackWallBrush</a>	Gets or sets a brush for back wall.
<a href="#">WallThickness</a>	Gets or sets the wall thickness for surface chart.
<a href="#">WireframeStrokeThickness</a>	Gets or sets the double value that represents wireframe stroke thickness.
<a href="#">WireframeStroke</a>	Gets or sets a brush for wireframe stroke.
<a href="#">Palette</a>	Gets or sets the color palette for the chart surface chart.
<a href="#">ZoomLevel</a>	Gets or sets double value that represent the zoom level value for surface chart.
<a href="#">Type</a>	Gets or sets type for choosing surface chart type.
<a href="#">XAxis</a>	Gets or sets the X-Axis for the surface chart.
<a href="#">YAxis</a>	Gets or sets the Y-Axis for the surface chart.
<a href="#">ZAxis</a>	Gets or sets the Z-Axis for the surface chart.
<a href="#">ItemsSource</a>	Gets or sets items source for surface chart.
<a href="#">XBindingPath</a>	Gets or sets a string that represents the X member path value of items source.
<a href="#">YBindingPath</a>	Gets or sets a string that represents the Y member path value of items source.
<a href="#">ZBindingPath</a>	Gets or sets a string that represents the Z member path value of items source.
<a href="#">ColorBar</a>	Gets or sets color bar for surface chart.
<a href="#">ApplyGradientBrush</a>	Gets or sets the bool value that represents a value for applying the gradient brush in surface chart.
<a href="#">EnableZooming</a>	Gets or sets the bool value that represents a value to enable zooming in surface chart.

<a href="#">ShowContourLine</a>	Gets or sets the bool value that represents a value whether to show the contour line for surface chart.
<a href="#">BrushCount</a>	Gets or sets the double value that represents a value of brush count in surface chart.

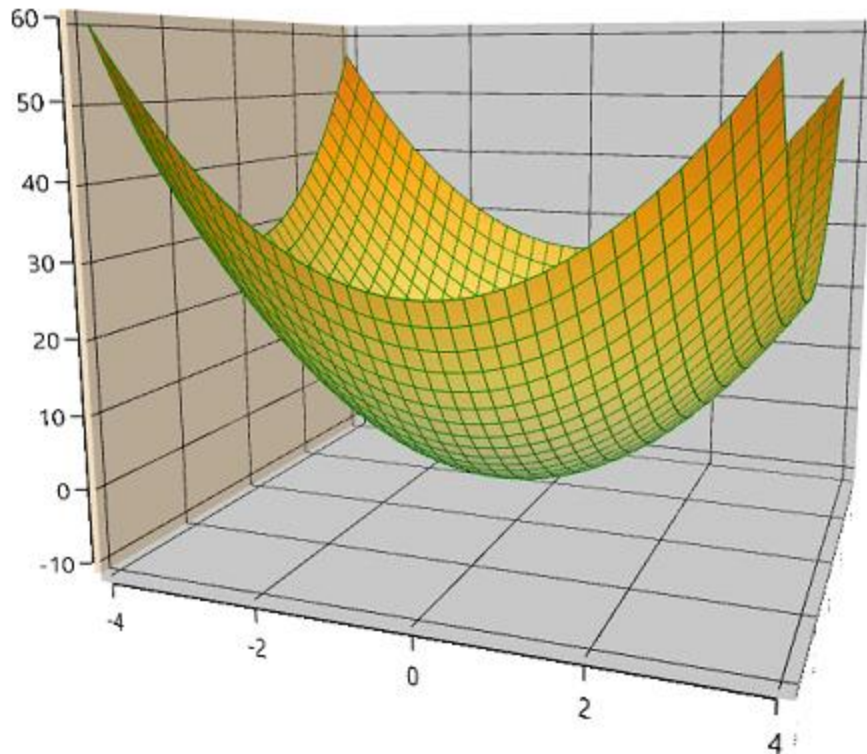
The following code sample explains the customization of surface area.

#### XML

```
<chart:SfSurfaceChart ItemsSource="{Binding DataValues}" XBindingPath="X"
Type="WireframeSurface"
YBindingPath="Y" ZBindingPath="Z" RowSize="{Binding RowSize}"
ColumnSize="{Binding ColumnSize}" ApplyGradientBrush="True" BrushCount="4"
LeftWallBrush="BlanchedAlmond" WireframeStroke="Green"
WireframeStrokeThickness="1"
CameraProjection="Perspective" Tilt="10" Rotate="20">
</chart:SfSurfaceChart>
```

#### C#

```
SfSurfaceChart chart = new SfSurfaceChart();
chart.Type = SurfaceType.WireframeSurface;
chart.SetBinding(SfSurfaceChart.ItemsSourceProperty, "DataValues");
chart.SetBinding(SfSurfaceChart.RowSizeProperty, "RowSize");
chart.SetBinding(SfSurfaceChart.ColumnSizeProperty, "ColumnSize");
chart.XBindingPath = "X";
chart.YBindingPath = "Y";
chart.ZBindingPath = "Z";
chart.BrushCount = 4;
chart.LeftWallBrush = new SolidColorBrush(Colors.BlanchedAlmond);
chart.WireframeStroke = new SolidColorBrush(Colors.Green);
chart.WireframeStrokeThickness = 1;
chart.CameraProjection = CameraProjection.Perspective;
chart.Tilt = 10;
chart.Rotate = 20;
grid.Children.Add(chart);
```



### User interaction in WPF Surface Chart (SfSurfaceChart)

The essential surface chart allows you to zoom and rotate the chart for a better visualization of all the axis and their points.

#### Zooming

Zooming of the surface chart is controlled with the help of [EnableZooming](#) property.

The following code shows how to enable zooming on surface chart.

#### XML

```
<chart:SfSurfaceChart EnableZooming="True" >  
</chart:SfSurfaceChart>
```

#### C#

```
SfSurfaceChart chart = new SfSurfaceChart();  
chart.EnableZooming = true;  
grid.Children.Add(chart);
```

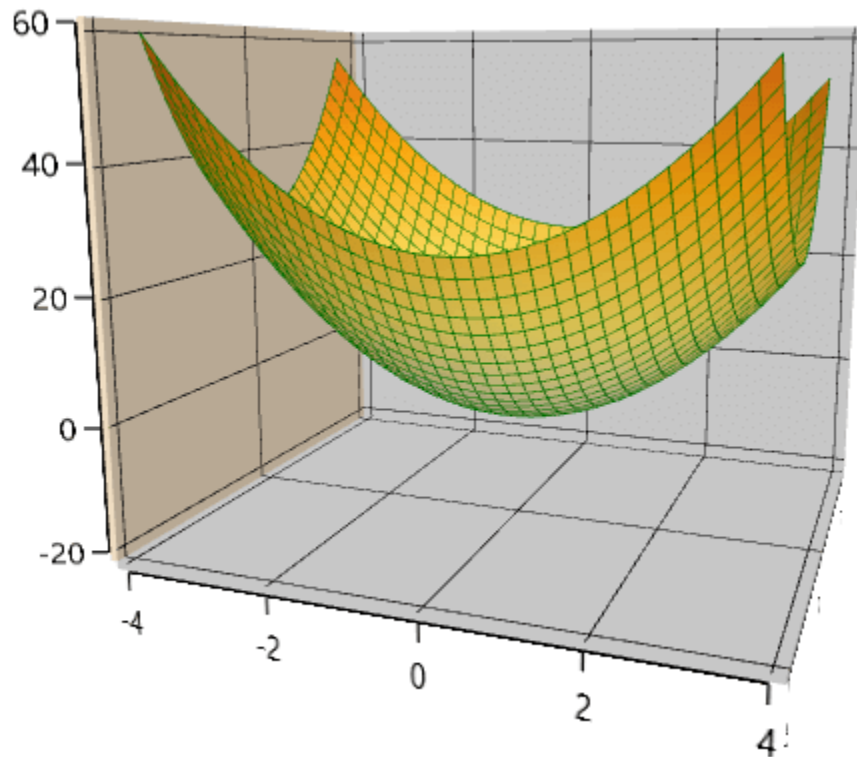
Zooming can either be done with pinch zooming or by programmatically with the [ZoomLevel](#) property.

#### XML

```
<chart:SfSurfaceChart EnableZooming="True" ZoomLevel="0.5">  
</chart:SfSurfaceChart>
```

#### C#

```
SfSurfaceChart chart = new SfSurfaceChart();  
chart.EnableZooming = true;  
chart.ZoomLevel = 0.5;  
grid.Children.Add(chart);
```



### Rotation

Surface chart can be rotated for better visualization of all the axis and their plots.

This can be controlled with the help of [EnableRotation](#) property.

### XML

```
<chart:SfSurfaceChart EnableRotation="True">  
</chart:SfSurfaceChart>
```

### C#

```
SfSurfaceChart chart = new SfSurfaceChart();  
chart.EnableRotation = true;  
grid.Children.Add(chart);
```

Rotation can either be done with interaction or by programmatically using [Rotate](#) property.

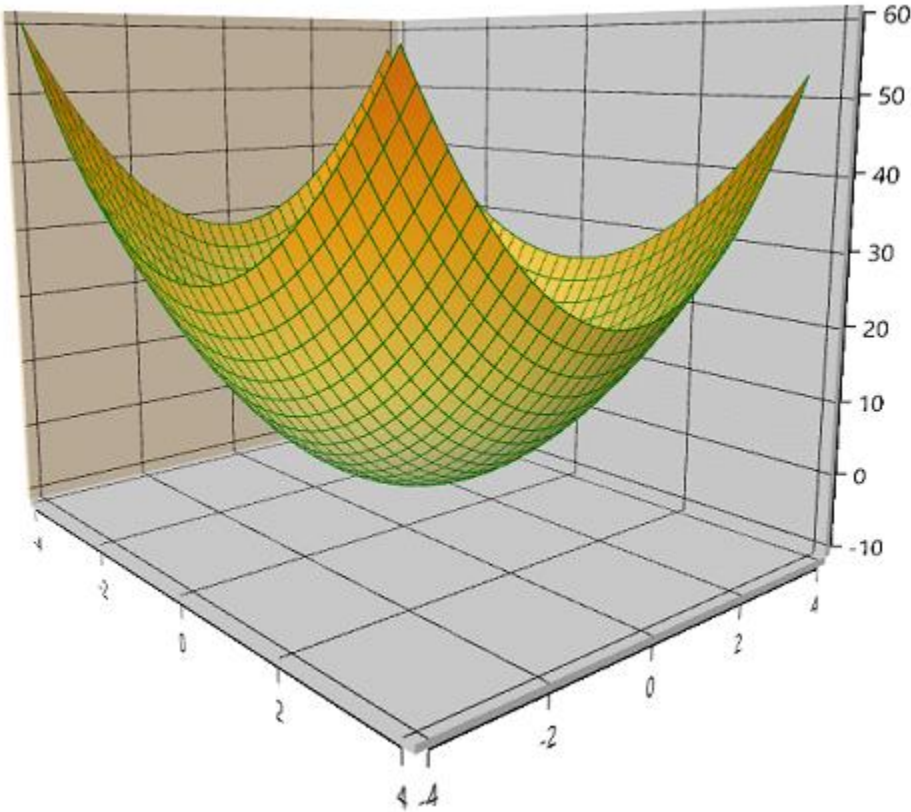
### XML

```
<chart:SfSurfaceChart EnableRotation="True" Rotate="50">  
</chart:SfSurfaceChart>
```



**C#**

```
SfSurfaceChart chart = new SfSurfaceChart();  
chart.EnableRotation = true;  
chart.Rotate = 50;  
grid.Children.Add(chart);
```

**Tilt**

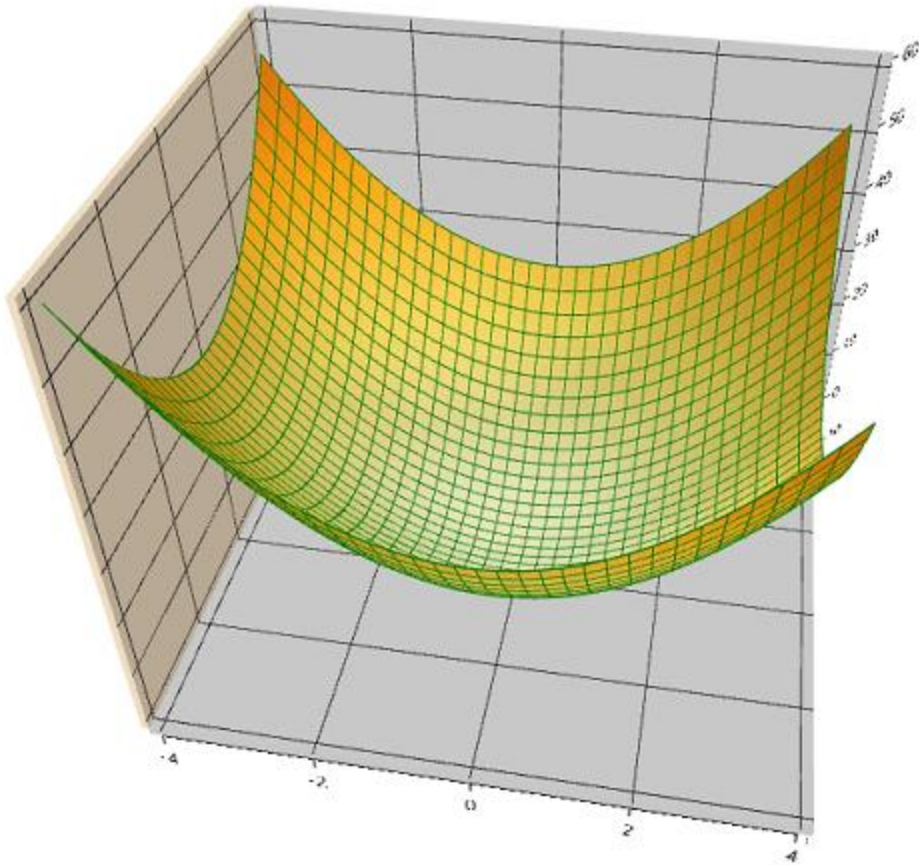
Surface chart can be tilted to a certain angle using the [Tilt](#) property.

**XML**

```
<chart:SfSurfaceChart Tilt="40">  
</chart:SfSurfaceChart>
```

**C#**

```
SfSurfaceChart chart = new SfSurfaceChart();  
chart.Tilt = 40;  
grid.Children.Add(chart);
```



## SfDateTimeRangeNavigator

### WPF Range Selector (SfDateTimeRangeNavigator) Overview

The date-time range navigator control is a time bound data visualization control. Its purpose is to allow scrolling and navigation through large periods of time. This control can be easily combined with other controls such as chart and grid view to create rich and powerful dashboards.

#### Key features

- Supports interactive features such as zooming and scrolling through large data.
- Calculates smart labels when zooming.
- Allows users to select a particular region in large period of time.
- Gets the data from a selected region.

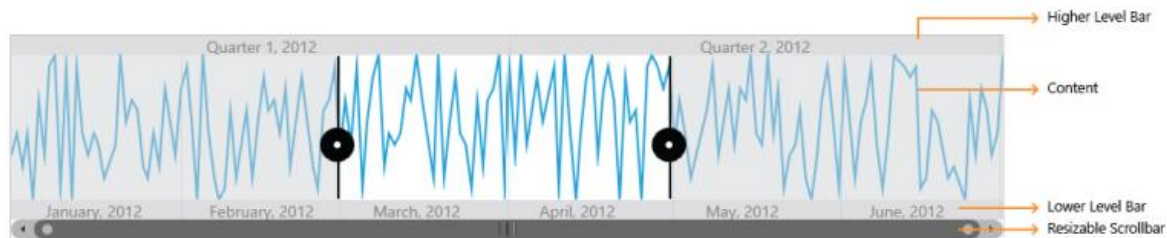
### Getting Started with WPF Range Selector (SfDateTimeRangeNavigator)

#### Visual structure

The date-time range navigator control is composed of various elements such as higher level bar, lower level bar, content, and resizable scroll bar.

- Higher level bar: Contains timespan format that is one level higher than date-time values of lower level bar, e.g. the higher level bar contains year format (yyyy) and the lower level bar contains month format (MMM).

- Lower level bar: Contains timespan format that is one level lower than date-time values of higher level bar, e.g. the lower level bar contains month format (MMM) and the higher level bar contains year format (yyyy).
- Content: Holds any type of UI element inside the navigator.
- Resizable scroll bar: Allows users to zoom and scroll the content and label bars.



### Create SfDateTimeRangeNavigator

The following section explains how to create SfDateTimeRangeNavigator.

#### Add the assembly reference

1. Open the Add Reference window in your project.
2. Choose our assemblies by following the given steps depending upon the developing environment.
  - If you use VS 2012, choose Assemblies > Extensions > Syncfusion.SfChart.WPF.dll.
  - If you use VS 2010, choose .Net > Syncfusion.SfChart.WPF.dll.
3. Add the following namespace in your XAML page:

#### XML

```
xmlns:Syncfusion="clr-namespace:Syncfusion.UI.Xaml.Charts"
```

#### Initialize the SfDateTimeRangeNavigator

#### XML

```
<Syncfusion:SfDateTimeRangeNavigator>
</Syncfusion:SfDateTimeRangeNavigator >
```

#### C#

```
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator();
```

#### Set ItemsSource for SfDateTimeRangeNavigator

Since the above steps are enough to create only empty date-time range navigator, you need to set the [ItemsSource](#) and [XBindingPath](#) for the SfDateTimeRangeNavigator. The ItemsSource must implement the IEnumerable interface.

#### XML

```
<Syncfusion:SfDateTimeRangeNavigator ItemsSource="{Binding UsersList}"
XBindingPath="Date" >
```

```
</Syncfusion:SfDateTimeRangeNavigator >
```

C#

```
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator()  
{  
    ItemsSource = new ViewModel().UsersList,  
    XBindingPath = "Date"  
};
```

Add content

Add content that needs to be displayed inside the date-time range navigator.

Property

Property	Description
<a href="#">ItemsSource</a>	Sets the ItemsSource for SfDateTimeRangeNavigator.
<a href="#">XBindingPath</a>	Represents the date-time x values.
	Adds any UI content inside the date-time range navigator.
<a href="#">Content</a>	

XML

```
<Syncfusion:SfDateTimeRangeNavigator ItemsSource="{Binding UsersList}"  
XBindingPath="Date" >  
    <Syncfusion:SfDateTimeRangeNavigator.Content>  
        <Syncfusion:SfChart x:Name="Chart">  
            <Syncfusion:SfChart.PrimaryAxis>  
                <Syncfusion:DateTimeAxis/>  
            </Syncfusion:SfChart.PrimaryAxis>  
            <Syncfusion:SfChart.SecondaryAxis>  
                <Syncfusion:NumericalAxis/>  
            </Syncfusion:SfChart.SecondaryAxis>  
            <Syncfusion:LineSeries  
                ItemsSource="{Binding UsersList}"  
                XBindingPath="Date"  
                YBindingPath="NoOfUsers">  
            </Syncfusion:LineSeries>  
        </Syncfusion:SfChart>  
    </Syncfusion:SfDateTimeRangeNavigator.Content>  
</Syncfusion:SfDateTimeRangeNavigator>
```

```
</Syncfusion:SfDateTimeRangeNavigator.Content >  
</Syncfusion:SfDateTimeRangeNavigator >
```

### C#

```
SfChart chart = new SfChart();  
chart.PrimaryAxis = new DateTimeAxis();  
chart.SecondaryAxis = new NumericalAxis();  
LineSeries series = new LineSeries()  
{  
    ItemsSource = new ViewModel().UsersList,  
    XBindingPath = "Date",  
    YBindingPath = "NoOfUsers"  
};  
chart.Series.Add(series);  
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator()  
{  
    ItemsSource = new ViewModel().StockPriceDetails,  
    XBindingPath = "Date"  
};  
rangeNavigator.Content = chart;
```

The following screenshot illustrates the result of the above code sample.

![[Getting started for WPF SfDateTimeRangNavigator ](Getting-Startedimages/Getting-Startedimg2.png)]

### Create SfDateTimeRangeNavigator from code-behind

This section explains how to create an application using the date-time range navigator from code-behind.

#### Add assembly reference

1. Open the Add Reference window in your project.
2. Choose Windows > Extensions > Syncfusion.SfChart.WPF.
3. Add the following namespace in your C# file: MainPage.xaml.cs.

### C#

```
using Syncfusion.UI.Xaml.Charts;
```

Initialize the Range Navigator

### C#

```
SfDateTimeRangNavigator rangenavigator = new SfDateTimeRangNavigator ();
```

Create a Sample Data Source

### C#

```
public class ItemsSource  
{  
    public DateTime Date { get; set; }  
    public double NoOfUsers { get; set; }  
}
```

```
}  
public class UsersViewModel  
{  
public UsersViewModel()  
{  
    this.UsersList = new ObservableCollection<ItemsSource>();  
    DateTime date = DateTime.Today;  
    UsersList.Add(new ItemsSource { Timestamp = date.AddHours(0.5), NoOfUsers =  
        3000 });  
    UsersList.Add(new ItemsSource { Timestamp = date.AddHours(0.5), NoOfUsers =  
        5000 });  
    UsersList.Add(new ItemsSource { Timestamp = date.AddHours(0.5), NoOfUsers =  
        2000 });  
    UsersList.Add(new ItemsSource { Timestamp = date.AddHours(0.5), NoOfUsers =  
        7000 });  
    UsersList.Add(new ItemsSource { Timestamp = date.AddHours(0.5), NoOfUsers =  
        6000 });  
    UsersList.Add(new ItemsSource { Timestamp = date.AddHours(0.5), NoOfUsers =  
        3000 });  
    public ObservableCollection<ItemsSource> UsersList  
    {  
        get; set;  
    }  
}
```

#### Define ItemsSource

Define the ItemsSource for the date-time range navigator as demonstrated in the following code sample.

---

**Note:** You can set any IEnumerable collection as ItemsSource.

---

#### C#

```
//Initialize the SfDateTimeRangNavigator.  
SfDateTimeRangNavigator rangenavigator = new SfDateTimeRangNavigator ();  
rangenavigator.ItemsSource = UsersList;  
rangenavigator.XBindingPath = "Date";
```

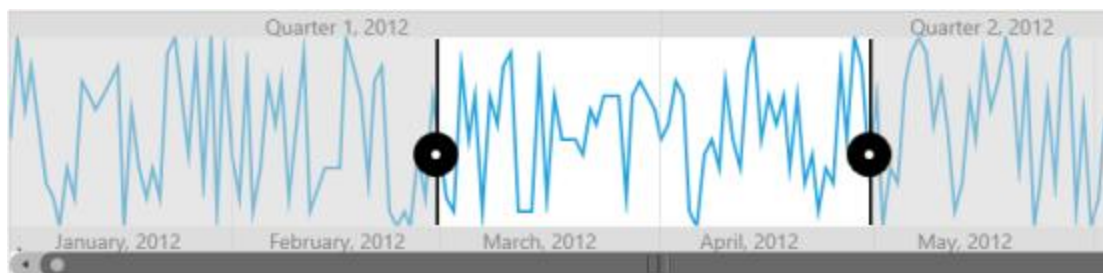
#### Add content

Add the content that needs to be displayed inside the date-time range navigator control using the [Content](#) property.

#### C#

```
//Initialize the SfChart  
SfChart chart = new SfChart();  
LineSeries series = new LineSeries();  
series.ItemsSource = UsersList;  
series.XBindingPath = "Date";  
series.YBindingPath = "NoOfUsers";  
var content = chart.Series.add(series);  
//Add content to navigator  
rangenavigator.Content = content;
```

The following screenshot illustrates the result of the above code sample.

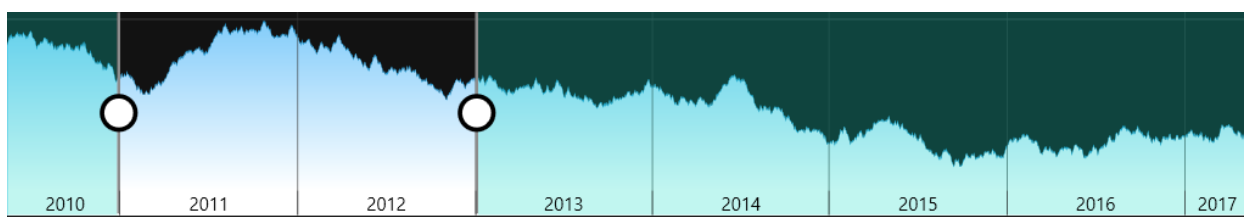


**Note:** The [SelectedData](#) property of the date-time range navigator control returns the collection that represents the data between the selected ranges.

### Theme

SfDateTimeRangeNavigator supports various built-in themes. Refer to the below links to apply themes for the SfDateTimeRangeNavigator,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Interactivity in WPF Range Selector (SfDateTimeRangeNavigator)

The date-time range navigator control provides interactive features such as zooming and panning. This control has a resizable scrollbar, which is used to zoom in large amount of data and navigate to particular timespan by moving the scroll bar.

The ZoomPosition and ZoomFactor properties of the chart axis can be bound with the SfDateTimeRangeNavigator.

### Properties

Property	Description
<a href="#">ZoomPosition</a>	Represents the zoom position of the selected range.
<a href="#">ZoomFactor</a>	Represents the zoom factor of the selected range.
<a href="#">SelectedData</a>	Gets the selected data between selected ranges.
<a href="#">ShowGridLines</a>	Shows the gridlines inside the content.
<a href="#">Minimum</a>	Sets the minimum or starting range of the navigator.
<a href="#">Maximum</a>	Sets the maximum or ending range of the navigator.

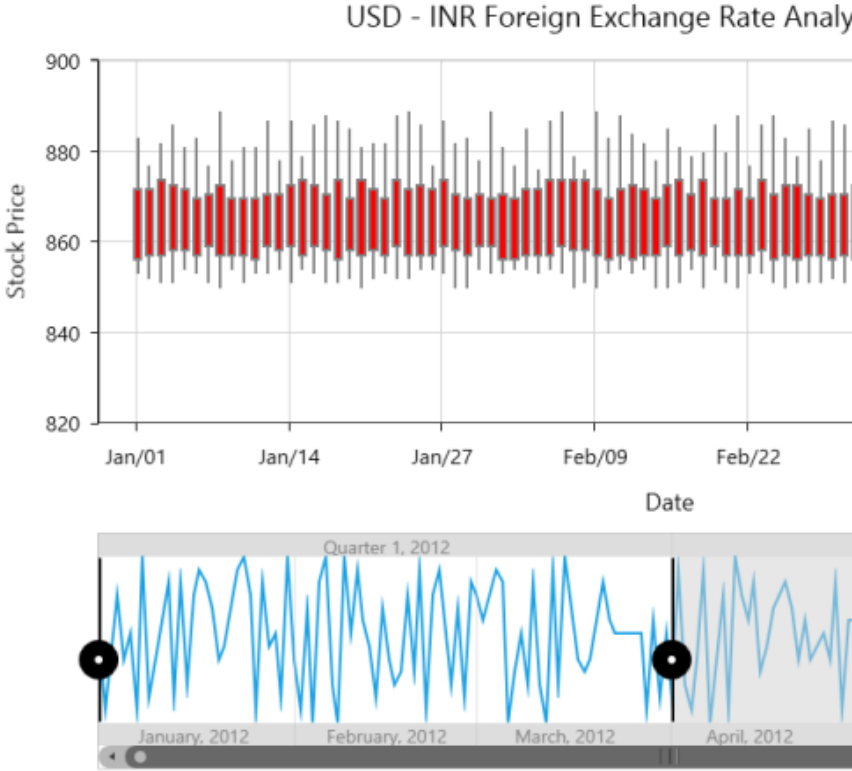


<a href="#">ViewRangeStart</a>	Gets the start value of the selected range.
<a href="#">ViewRangeEnd</a>	Gets the end value of the selected range.
<a href="#">EnableDeferredUpdate</a>	Enables deferred scrolling and panning.
<a href="#">DeferredUpdateDelay</a>	Gets or sets the delay value when the EnableDeferredUpdate is enabled.

## Events

Event	Parameters	Description
<a href="#">ValueChanged</a>	ValueChanged (Object sender, EventArgs e)	This event is triggered when the position of the scrollbar is changed.
<a href="#">LowerBarLabelsCreated</a>	LowerBarLabelsCreated (Object sender, LowerBarLabelsCreatedEventArgs e)	This event is triggered when the lower bar labels gets created.



 <p>UpperBarLabelsCreated</p>	<p>UpperBarLabelsCreated(Object sender, UpperBarLabelsCreatedEventArgs e)</p>	<p>This event is triggered when the upper bar labels gets created.</p>
---	---	--

**XML**

```

<chart:SfChart x:Name="financialChart">
  <chart:SfChart.PrimaryAxis>
    <chart:CategoryAxis Name="axis1" ZoomPosition="{Binding
      ElementName=RangeNavigator, Path=ZoomPosition, Mode=TwoWay}"
      ZoomFactor="{Binding ElementName=RangeNavigator, Path=ZoomFactor,
        Mode=TwoWay}"
      Header="Date" LabelFormat="MMM/dd"
      LabelTemplate="{StaticResource labelTemplate}" />
    </chart:SfChart.PrimaryAxis>
    <chart:SfChart.SecondaryAxis>
      <chart:NumericalAxis />
    </chart:SfChart.SecondaryAxis>
    <chart:CandleSeries Name="series" ItemsSource="{Binding StockPriceDetails}"
      XBindingPath="_Date" High="High" Open="Open" Close="Close" Low="Low"
      Label="CandleSeries">
    </chart:CandleSeries>
  </chart:SfChart>
  <chart:SfDateTimeRangeNavigator x:Name="RangeNavigator"
    ItemsSource="{Binding StockPriceDetails}" XBindingPath="_Date" >
    <chart:SfDateTimeRangeNavigator.Content>
      <chart:SfLineSparkline ItemsSource="{Binding StockPriceDetails}"
        YBindingPath="High" >
      </chart:SfLineSparkline>
    </chart:SfDateTimeRangeNavigator.Content>
  </chart:SfDateTimeRangeNavigator>

```

**C#**

```
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    VerticalAlignment = VerticalAlignment.Top
};
SfLineSparkline sparkline = new SfLineSparkline()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    YBindingPath = "High"
};
rangeNavigator.Content = sparkline;
Grid.SetColumn(rangeNavigator, 1);
SfChart chart = new SfChart();
chart.PrimaryAxis = new CategoryAxis()
{
    PlotOffset = 25,
    Header = "Date",
    LabelFormat = "MMM/dd",
    ZoomPosition = rangeNavigator.ZoomPosition,
    ZoomFactor = rangeNavigator.ZoomFactor
};
chart.SecondaryAxis = new NumericalAxis();
CandleSeries candleSeries=new CandleSeries ()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    High ="High", Low = "Low",
    Open ="Open", Close ="Close",
    XBindingPath ="Date",
    Label ="CandleSeries"
};
chart.Series.Add(candleSeries);
```

The following screenshot illustrates selecting one quarter of data.

![Zooming support in WPF SfDateTimeRangeNavigator](Interactivity  
USD - INR Foreign Exchange Rate Analysis



images/Interactivityimg1.png)

The following screenshot illustrates the control after zooming into weeks of data from 6 months of data.

![Zooming support in WPF SfDateTimeRangeNavigator](Interactivityimages/Interactivityimg2.png)

Thumb style customization

SfDateTimeRangeNavigator [LeftThumbStyle](#) and [RightThumbStyle](#) can be customized by using the following properties:

[SymbolTemplate](#)

[SymbolTemplate](#) can be used for gets or sets the data template for the symbol.

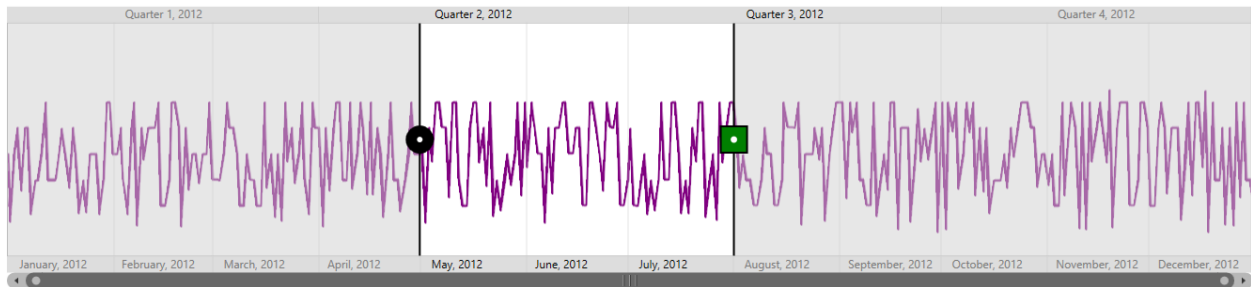
### XML

```
<syncfusion:SfDateTimeRangeNavigator x:Name="navigator">
  <syncfusion:SfRangeNavigator.Resources>
    <DataTemplate x:Key="symbolTemplate">
      <Grid>
        <Ellipse Height="40" Width="40" Fill="Green" Stroke="Black"
          VerticalAlignment="Center" StrokeThickness="2"/>
        <Ellipse Height="7" Width="7" Fill="White"
          VerticalAlignment="Center"/>
      </Grid>
    </DataTemplate>
  </syncfusion:SfRangeNavigator.Resources>
  <syncfusion:SfDateTimeRangeNavigator.RightThumbStyle>
    <syncfusion:ThumbStyle SymbolTemplate="{StaticResource symbolTemplate}"/>
  </syncfusion:SfDateTimeRangeNavigator.RightThumbStyle>
</syncfusion:SfDateTimeRangeNavigator>
```

**C#**

```
ThumbStyle thumbStyle = new ThumbStyle()
{
    SymbolTemplate = grid.Resources["symbolTemplate"] as DataTemplate
};
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator()
{
    ItemsSource = new ViewModel().StockPriceDetails,
    XBindingPath = "Date",
    RightThumbStyle = thumbStyle
};
```

The following screenshot illustrates the control after customizing the right thumb.

**LineStyle**

[LineStyle](#) can be used for gets or sets the style for the thumb line.

**XML**

```
<Grid x:Name="grid">
  <Grid.Resources>
    <Style TargetType="Line" x:Key="lineStyle1">
      <Setter Property="Stroke" Value="Red"></Setter>
      <Setter Property="StrokeThickness" Value="3"></Setter>
      <Setter Property="StrokeDashArray" Value="2,1"></Setter>
    </Style>
    <Style TargetType="Line" x:Key="lineStyle2">
      <Setter Property="Stroke" Value="Red"></Setter>
      <Setter Property="StrokeThickness" Value="3"></Setter>
      <Setter Property="StrokeDashArray" Value="2,1"></Setter>
    </Style>
  </Grid.Resources>
  <syncfusion:SfDateTimeRangeNavigator>
    <syncfusion:SfDateTimeRangeNavigator.LeftThumbStyle>
      <syncfusion:ThumbStyle LineStyle="{StaticResource lineStyle1}" />
    </syncfusion:SfDateTimeRangeNavigator.LeftThumbStyle>
    <syncfusion:SfDateTimeRangeNavigator.RightThumbStyle>
      <syncfusion:ThumbStyle LineStyle="{StaticResource lineStyle2}" />
    </syncfusion:SfDateTimeRangeNavigator.RightThumbStyle>
  </syncfusion:SfDateTimeRangeNavigator>
</Grid>
```

**C#**

```
ThumbStyle thumbLineStyle1 = new ThumbStyle()
```

```

{
LineStyle = grid.Resources["lineStyle1"] as DataTemplate
};
ThumbStyle thumbLineStyle2 = new ThumbStyle()
{
LineStyle = grid.Resources["lineStyle2"] as DataTemplate
};
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator()
{
LineStyle = thumbLineStyle1,
LineStyle = thumbLineStyle2
};

```



#### Higher and lower bars customization

Property	Description
<a href="#">HigherBarGridLineStyle</a>	Defines the style for gridlines of upper labels bar.
<a href="#">LowerBarGridLineStyle</a>	Defines the style for gridlines of lower labels bar.
<a href="#">HigherBarTickLineStyle</a>	Defines the style for ticklines of upper labels bar.
<a href="#">LowerBarTickLineStyle</a>	Defines the style for ticklines of lower labels bar.

#### Customization in WPF Range Selector (SfDateTimeRangeNavigator)

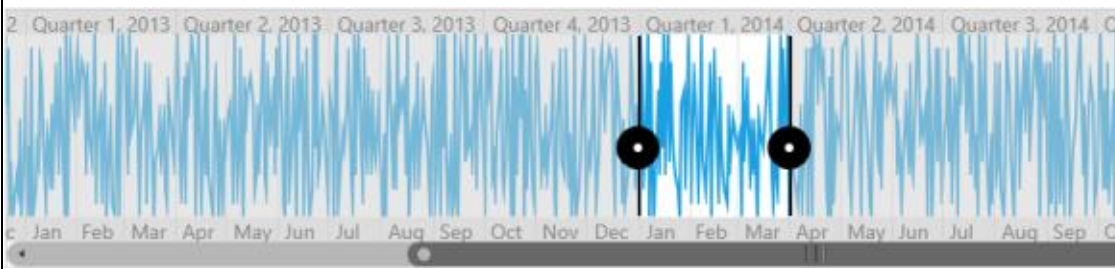
##### Interval customization

The date-time range navigator control helps users to visualize large data in a simplified manner. The timespan of data is represented in the higher level bar and lower level bar. The timespan calculates data smartly and provides suitable date-time format and interval, by default.

Users can also set the interval as needed for their data. This can be done by using the interval as demonstrated in the following code sample.

##### Properties

Property	Description
<a href="#">IntervalType</a>	Sets the interval type that needs to be displayed

	in the navigator.
 <p><a href="#">LabelFormatters</a></p>	Gets or sets string collection to set the label format for the navigator labels.

**XML**

```

<syncfusion:SfDateTimeRangeNavigator x:Name="rangepicker"
ItemsSource="{Binding power}" XBindingPath="Date" >
<syncfusion:SfDateTimeRangeNavigator.Intervals>
<syncfusion:Interval IntervalType="Quarter"/>
<syncfusion:Interval IntervalType="Month"/>
</syncfusion:SfDateTimeRangeNavigator.Intervals>
<syncfusion:SfDateTimeRangeNavigator.Content>
<syncfusion:SfChart>
<syncfusion:SfChart.PrimaryAxis>
<syncfusion:CategoryAxis Visibility="Collapsed" />
</syncfusion:SfChart.PrimaryAxis>
<syncfusion:SfChart.SecondaryAxis>
<syncfusion:NumericalAxis Visibility="Collapsed" />
</syncfusion:SfChart.SecondaryAxis>
<syncfusion:FastLineBitmapSeries XBindingPath="Date" ItemsSource="{Binding
power}" YBindingPath="Value">
</syncfusion:FastLineBitmapSeries>
</syncfusion:SfChart>
</syncfusion:SfDateTimeRangeNavigator.Content>
</syncfusion:SfDateTimeRangeNavigator>

```

**C#**

```

SfChart chart = new SfChart();
chart.PrimaryAxis = new CategoryAxis() { Visibility = Visibility.Collapsed
};
chart.SecondaryAxis = new NumericalAxis() { Visibility =
Visibility.Collapsed };
FastLineBitmapSeries candleSeries = new FastLineBitmapSeries()
{
ItemsSource = new ViewModel().Power,
XBindingPath = "Date",
YBindingPath = "Value"
};
chart.Series.Add(candleSeries);
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator()
{
ItemsSource = new ViewModel().Power,

```

```

XBindingPath = "Date"
};
rangeNavigator.Intervals.Add(new Interval() { IntervalType =
NavigatorIntervalType.Quarter });
rangeNavigator.Intervals.Add(new Interval() { IntervalType =
NavigatorIntervalType.Month });
rangeNavigator.Content = chart;

```

The following screenshot illustrates only Quarter and Month intervals in the navigator.

![Label customization in WPF SfDateTimeRangeNavigator](Label-Customizationimages/Label-Customizationimg1.png)

The interval can be set in the following types:

- [Year](#)
- [Quarter](#)
- [Month](#)
- [Week](#)
- [Day](#)
- [Hour](#)

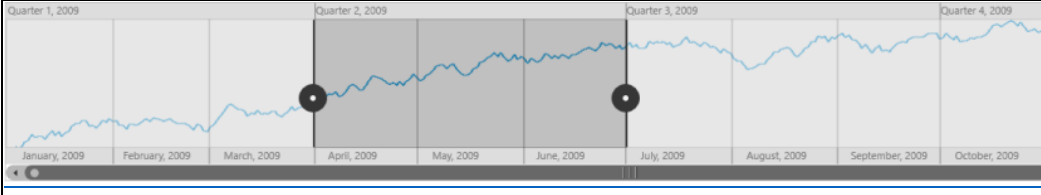
The auto timespan format simplifies the visual representation of data when zooming in with the following formats:

#### Intervals

Intervals	Examples
HourInterval	7/21/2011 12:00:00 AM > 12 AM7/21/2011 12:00:00 AM > 12A
DayInterval	7/21/2011 12:00:00 AM > Thursday, July 21, 20117/21/2011 12:00:00 AM > Thu, Jul 21, 20117/21/2011 12:00:00 AM > Thursday, 217/21/2011 12:00:00 AM > Thu, 21
WeekInterval	7/21/2011 12:00:00 AM > Week 29, July, 20117/21/2011 12:00:00 AM > Week 29, Jul, 20117/21/2011 12:00:00 AM > Week 297/21/2011 12:00:00 AM > W29
MonthInterval	7/21/2011 12:00:00 AM > July, 20117/21/2011 12:00:00 AM > July7/21/2011 12:00:00 AM > Jul7/21/2011 12:00:00 AM > J
QuarterInterval	7/21/2011 12:00:00 AM > Quarter 3, 20117/21/2011 12:00:00 AM > Quarter 37/21/2011 12:00:00 AM > Q3, 20117/21/2011 12:00:00 AM > Q3
YearInterval	7/21/2011 12:00:00 AM > 2011

#### Label style customization

Property	Description
LabelBarStyle	Allows to customize the label style using the

	LabelBarStyle property, and this property can be applied to the <a href="#">HigherLevelBarStyle</a> or <a href="#">LowerLevelBarStyle</a> .
<a href="#">SelectedLabelStyle</a>	Defines the label style for labels in the selected region.
	Positions the upper and lower labels inside or outside the label bar.
<a href="#">Position</a>	

## XML

```
<syncfusion:SfDateTimeRangeNavigator x:Name="navigator">
  <syncfusion:SfRangeNavigator.Resources>
    <Style TargetType="TextBlock" x:Key="labelStyle">
      <Setter Property="FontSize" Value="10"/>
    </Style>
  </syncfusion:SfRangeNavigator.Resources>
  <syncfusion:SfDateTimeRangeNavigator.HigherLevelBarStyle>
    <syncfusion:LabelBarStyle Background="Red"
      LabelHorizontalAlignment="Left"
      SelectedLabelStyle="{StaticResource labelStyle}"/>
  </syncfusion:SfDateTimeRangeNavigator.HigherLevelBarStyle>
</syncfusion:SfDateTimeRangeNavigator>
```

## C#

```
LabelBarStyle barStyle = new LabelBarStyle()
{
    LabelHorizontalAlignment = HorizontalAlignment.Left,
    Background = new SolidColorBrush(Colors.Red),
    SelectedLabelStyle = grid.Resources["labelStyle"] as Style
};
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator()
{
    HigherLevelBarStyle = barStyle
};
```

The following screenshot illustrates setting the Label HorizontalAlignment to left.



![[Label customization in WPF SfDateTimeRangeNavigator](Label-Customizationimages/Label-Customizationimg2.png)]

### Visibility of label bar

SfDateTimeRangeNavigator provides support to customize the visibility of lower bar and upper bar using the `LowerLabelBarVisibility` and `UpperLabelBarVisibility` types.

### RangePadding Customization

The date-time range navigator supports the following types of padding.

- None
- Round

#### None

When the value of the `RangePadding` is `None`, padding is not applied to the range.

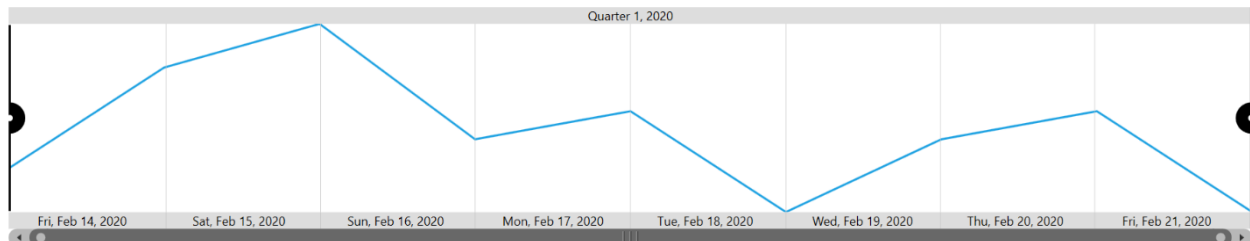
The following screenshot demonstrates a `RangePadding` set to `None`.

#### XML

```
<Syncfusion:SfDateTimeRangeNavigator RangePadding="None">
</Syncfusion:SfDateTimeRangeNavigator >
```

#### C#

```
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator()
{
    RangePadding = NavigatorRangePadding.None
};
```



#### Round

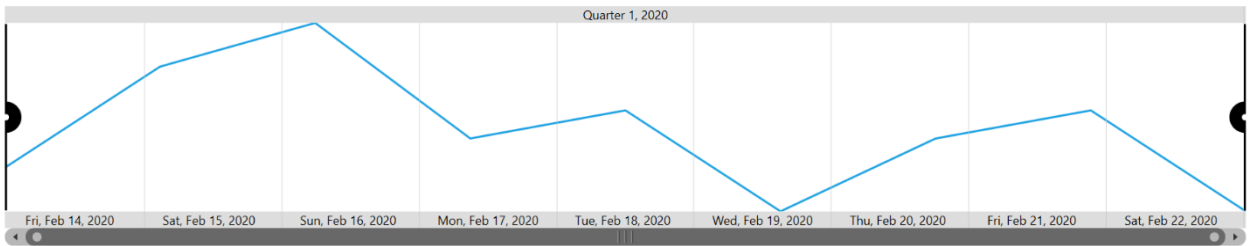
When the value of `RangePadding` property is `Round`, range will be rounded to the nearest possible value by the interval.

#### XML

```
<Syncfusion:SfDateTimeRangeNavigator RangePadding="Round">
</Syncfusion:SfDateTimeRangeNavigator >
```

#### C#

```
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator()
{
    RangePadding = NavigatorRangePadding.Round
};
```



ToolTip Support in WPF Range Selector (SfDateTimeRangeNavigator)

The date-time range navigator control provides tooltip support to sliders. Sliders are used to select a particular region of data in the control. Tooltips for sliders show the selected start and end date-time values. You can view the exact date values with the milliseconds precision.

Properties

The following properties are used to customize the tooltip settings:

Property

Property Name	Description
ShowToolTip	Shows or hides the tooltip.
ToolTipLabelFormat	Sets the date-time label format for the tooltip.
LeftToolTipTemplate	Sets the data template for the left tooltip.
RightToolTipTemplate	Sets the data template for the right tooltip.

XML

```
<chart:SfDateTimeRangeNavigator x:Name="RangeNavigator"
ItemsSource="{Binding StockPriceDetails}" XBindingPath="_Date"
ShowToolTip="true" ToolTipLabelFormat ="MMM/dd/yyyy">
<chart:SfDateTimeRangeNavigator.LeftToolTipTemplate>
<DataTemplate>
-----
</DataTemplate>
</chart:SfDateTimeRangeNavigator.LeftToolTipTemplate>
<chart:SfDateTimeRangeNavigator.Content>
</chart:SfDateTimeRangeNavigator.Content>
</chart:SfDateTimeRangeNavigator>
```

**C#**

```
SfDateTimeRangeNavigator rangeNavigator = new SfDateTimeRangeNavigator()  
{  
    ItemsSource = new ViewModel().StockPriceDetails,  
    XBindingPath = "Date",  
    ShowToolTip = true,  
    ToolTipLabelFormat = "yyyy/MMM/dd",  
    LeftToolTipTemplate = grid.Resources["tooltipTemplate"] as DataTemplate  
};
```

![ToolTip support in WPF SfDateTimeRangeNavigator](ToolTip-Supportimages/ToolTip-Supportimg1.png)

## SfSparkline

### WPF Sparkline (SfSparkline) Overview

A sparkline is a very small chart, typically drawn without axes or coordinates. It presents the general shape of data's in a simple and highly condensed way.

#### Key features

- Data binding support.
- Range band support.
- Track ball support.
- Support 4 different types of sparkline.
- Animation support.
- Dynamic update.
- Marker support for line and area sparkline.
- Empty point support.

### Getting Started with WPF Sparkline (SfSparkline)

#### Creating sparkline

Following steps explain how to create sparkline,

#### Adding the assembly reference:

- Open the [Add Reference](#) window from your project.
- To Choose our assemblies follow the below step depending upon the developing environment.
- If using VS 2012 choose Assemblies > Extensions > Syncfusion.SfChart.WPF.dll
- If using VS 2010 choose .Net>Syncfusion.SfChart.WPF.dll
- Add the following namespace in your XAML page:

**XML**

```
xmlns:Syncfusion="clr-namespace:Syncfusion.UI.Xaml.Charts"
```

**C#**

```
using Syncfusion.UI.Xaml.Charts;
```

### *Initialize the sparkline*

You need to initialize the sparkline represented by the following class Syncfusion.UI.Xaml.Charts.SfChart,

### **XML**

```
<Syncfusion:SfLineSparkline>  
</Syncfusion:SfLineSparkline>
```

### **C#**

```
SfLineSparkline sparkline = new SfLineSparkline()
```

### *Create a Sample Data Source*

Since the above step will produce only an empty sparkline, we need to add some data to the sparkline for plotting. In this step, let's create a sample data source.

### **C#**

```
public class UserProfile  
{  
    public DateTime TimeStamp { get; set; }  
    public double NoOfUsers { get; set; }  
}  
public class UsersViewModel  
{  
    public UsersViewModel()  
    {  
        this.UsersList = new ObservableCollection<UserProfile>();  
        DateTime date = DateTime.Today;  
        UsersList.Add(new UserProfile { TimeStamp = date.AddHours(0.5), NoOfUsers = 3000 });  
        UsersList.Add(new UserProfile { TimeStamp = date.AddHours(0.5), NoOfUsers = 5000 });  
        UsersList.Add(new UserProfile { TimeStamp = date.AddHours(0.5), NoOfUsers = -3000 });  
        UsersList.Add(new UserProfile { TimeStamp = date.AddHours(0.5), NoOfUsers = -4000 });  
        UsersList.Add(new UserProfile { TimeStamp = date.AddHours(0.5), NoOfUsers = 2000 });  
        UsersList.Add(new UserProfile { TimeStamp = date.AddHours(0.5), NoOfUsers = 3000 });  
    }  
    public ObservableCollection<UserProfile> UsersList  
    {  
        get; set;  
    }  
}
```

**Note:** Syncfusion sparkline also supports items source as collection of double values and collection inherited from IEnumerable.

### *Binding Data to sparkline*

We need to add the above UsersViewModel to the DataContext of the sparkline, bind the data source to the ItemsSource property of the SfLineSparkline, and then map the data using YBindingPath and XBindingPath.

### **XML**

```
<Grid.DataContext>
<local:UsersViewModel/>
</Grid.DataContext>
<Syncfusion:SfLineSparkline
ItemsSource="{Binding UsersList}"
YBindingPath="NoOfUsers">
</Syncfusion:SfLineSparkline >
```

### **C#**

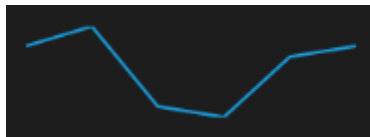
```
SfLineSparkline sparkline = new SfLineSparkline()
{
    ItemsSource = new SparkViewModel().UsersList,
    YBindingPath = "NoOfUsers"
};
```

---

**Note:** if we do not map the XBindingPath means sparkline data positioned as indexed.

---

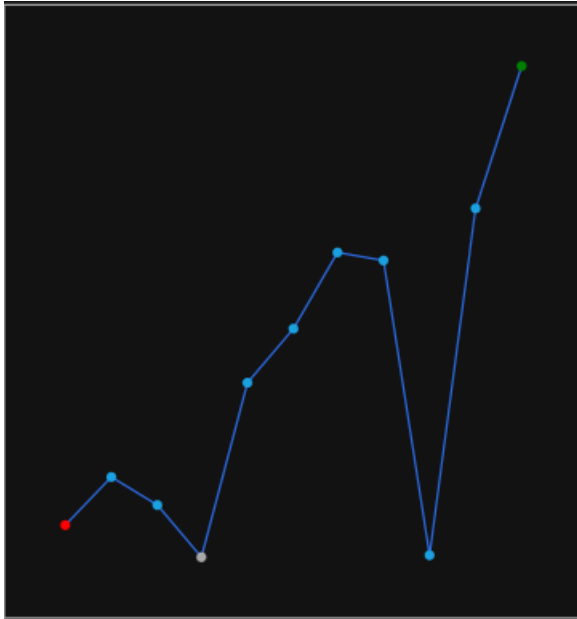
The following illustrates the result of the above code sample,



### Theme

SfSparkline supports various built-in themes. Refer to the below links to apply themes for the SfSparkline,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



## Sparkline Types in WPF Sparkline (SfSparkline)

### Line Sparkline

Line sparkline rendered using polyline and the following code is used to create line sparkline,

#### XML

```
<Grid.DataContext>
<local:UsersViewModel/>
</Grid.DataContext>
<Syncfusion:SfLineSparkline ItemsSource="{Binding UsersList}"
YBindingPath="NoOfUsers">
</Syncfusion:SfLineSparkline >
```

#### C#

```
SfLineSparkline sparkline = new SfLineSparkline()
{
    ItemsSource = new SparkViewModel().UsersList,
    YBindingPath = "NoOfUsers"
};
```

The following illustrates the result of the above code sample,



### Column Sparkline

Column sparkline used to visualize the raw data as a rectangle and following code is used to create column sparkline,

#### XML

```
<Syncfusion:SfColumnSparkline ItemsSource="{Binding UsersList}"  
YBindingPath="NoOfUsers" >  
</Syncfusion:SfColumnSparkline>
```

### C#

```
SfColumnSparkline sparkline = new SfColumnSparkline()  
{  
    ItemsSource = new SparkViewModel().UsersList,  
    YBindingPath = "NoOfUsers"  
};
```

Following is the snapshot for Column Sparkline,



### Area sparkline

Following code is used to create area sparkline and all the line sparkline features are applicable for area sparkline,

### XML

```
<Syncfusion:SfAreaSparkline ItemsSource="{Binding UsersList}"  
YBindingPath="NoOfUsers">  
</Syncfusion:SfAreaSparkline >
```

### C#

```
SfAreaSparkline sparkline = new SfAreaSparkline()  
{  
    ItemsSource = new SparkViewModel().UsersList,  
    YBindingPath = "NoOfUsers"  
};
```

Following is the snapshot for area sparkline,



### WinLoss Sparkline

WinLoss sparkline render as a column segment and it show the positive, negative and neutral values.

#### XML

```
<Page.DataContext>
<local:MatchDetailsViewModel/>
</Page.DataContext>
<Syncfusion:SfWinLossSparkline x:Name="sparkline" ItemsSource="{Binding
Match}" YBindingPath="Result" >
</Syncfusion:SfWinLossSparkline>
```

#### C#

```
SfWinLossSparkline sparkline = new SfWinLossSparkline()
{
    ItemsSource = new SparkViewModel().Match,
    YBindingPath = "Result"
};
public class MatchDetailsModel
{
    public double Result { get; set; }
    public string Status { get; set; }
}
public class MatchDetailsViewModel
{
    public MatchDetailsViewModel()
    {
        this.Match = new ObservableCollection<MatchDetailsModel>();
    }
}
```

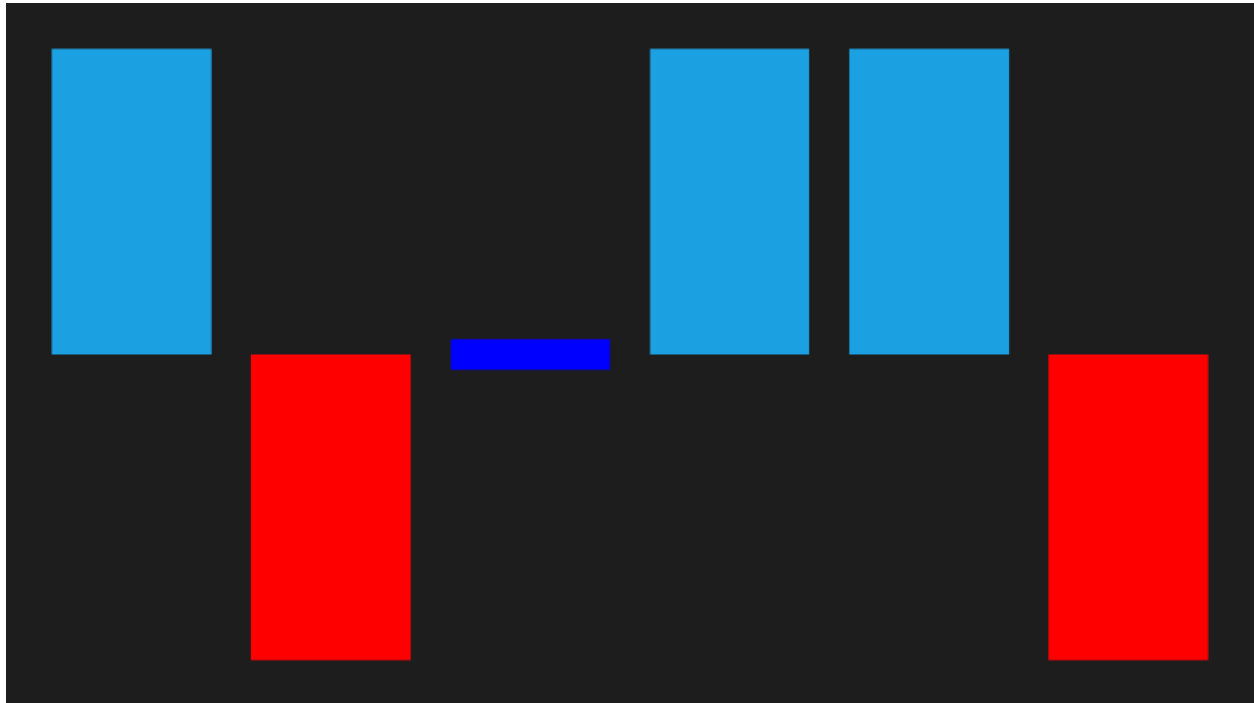


```

this.Match.Add(new MatchDetailsModel() { Result = 1, Status = "Win" });
this.Match.Add(new MatchDetailsModel() { Result = -1, Status = "Loss" });
this.Match.Add(new MatchDetailsModel() { Result = 0, Status = "Draw" });
this.Match.Add(new MatchDetailsModel() { Result = 1, Status = "Win" });
this.Match.Add(new MatchDetailsModel() { Result = 1, Status = "Win" });
this.Match.Add(new MatchDetailsModel() { Result = -1, Status = "Loss" });
}
public ObservableCollection<MatchDetailsModel> Match { get; set; }
}

```

Execute the above code to render the following output.



### Range Band in WPF Sparkline (SfSparkline)

Range band feature used to highlight the particular mentioned range along Y axis.

#### XML

```

<Syncfusion:SfLineSparkline
ItemsSource="{Binding UsersList}"
BandRangeStart="2000"
BandRangeEnd="-1000" RangeBandBrush="Green"
YBindingPath="NoOfUsers">
</Syncfusion:SfLineSparkline >

```

#### C#

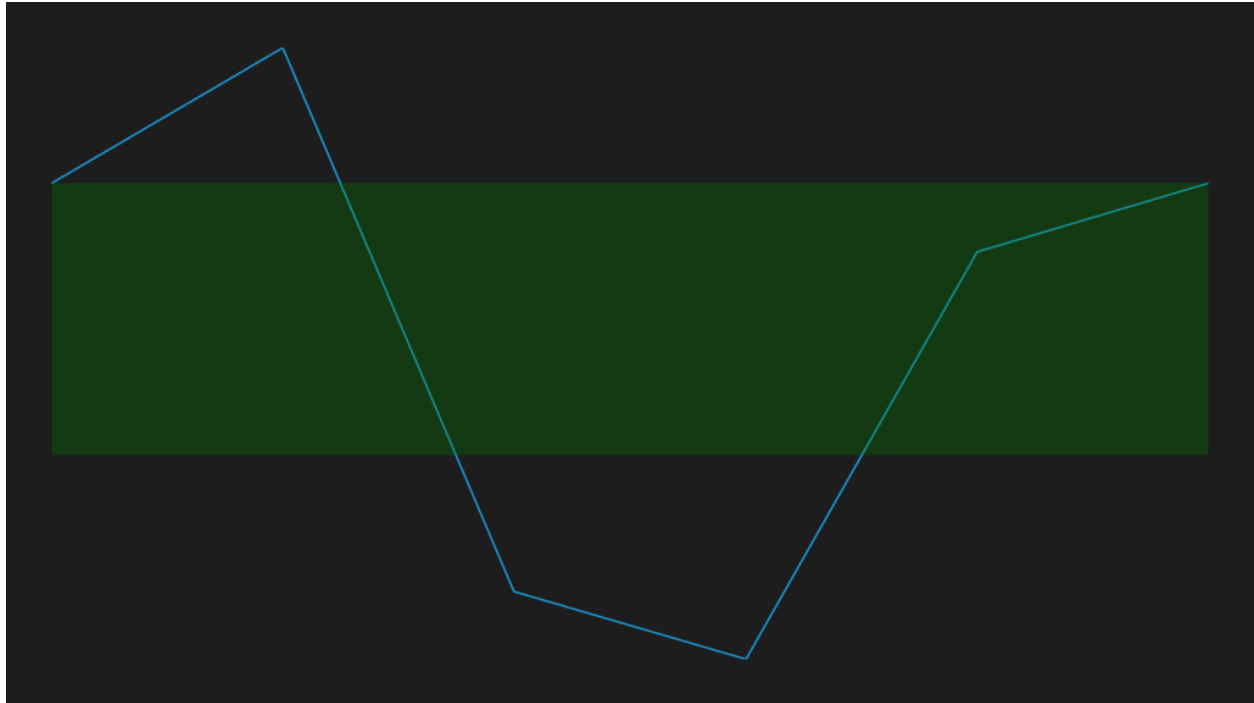
```

SfLineSparkline sparkline = new SfLineSparkline()
{
    ItemsSource = new SparkViewModel().UsersList,
    YBindingPath = "NoOfUsers",
    BandRangeStart = 2000,
    BandRangeEnd = -1000,
}

```

```
RangeBandBrush = new SolidColorBrush(Colors.Green)
};
```

Following is the snapshot for range band,



### Track ball in WPF Sparkline (SfSparkline)

This is used to indicate the value point on mouse move and this feature is applicable for line and area sparkline.

#### XML

```
<Syncfusion:SfLineSparkline
ItemsSource="{Binding UsersList}"
ShowTrackBall="True"
YBindingPath="NoOfUsers">
</Syncfusion:SfLineSparkline >
```

#### C#

```
SfLineSparkline sparkline = new SfLineSparkline()
{
    ItemsSource = new SparkViewModel().UsersList,
    YBindingPath = "NoOfUsers",
    ShowTrackBall = true
};
```

Following is the snapshot for track ball,



### Customizing TrackBall

You can customize the default appearance of the Trackball style by using the TrackballStyle property.

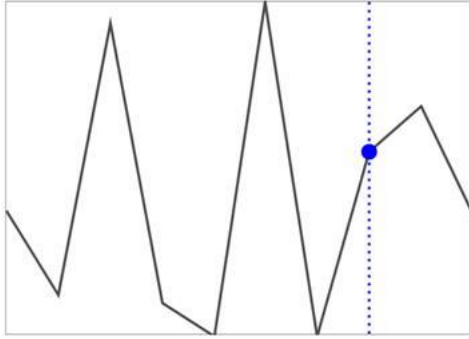
The following code shows how to apply the style for the Trackball line.

#### XML

```
<Grid.Resources>
<Style TargetType="Ellipse" x:Key="lineStyle1">
<Setter Property="Fill" Value="Blue"></Setter>
<Setter Property="Height" Value="12"></Setter>
<Setter Property="Width" Value="12"></Setter>
</Style>
<Style TargetType="Line" x:Key="lineStyle2">
<Setter Property="Stroke" Value="Blue"/>
<Setter Property="StrokeThickness" Value="2"></Setter>
<Setter Property="StrokeDashArray" Value="1,2"></Setter>
</Style>
</Grid.Resources>
<Syncfusion:SfLineSparkline Height="250" Width="350" Interior="#4a4a4a"
BorderBrush="DarkGray" BorderThickness="1"
ItemsSource="{Binding UsersList}" ShowTrackBall="True"
TrackBallStyle="{StaticResource lineStyle1}"
LineStyle="{StaticResource lineStyle2}"
YBindingPath="NoOfUsers">
</Syncfusion:SfLineSparkline >
```

#### C#

```
SfLineSparkline sparkline = new SfLineSparkline()
{
    ItemsSource = new SparkViewModel().Data,
    YBindingPath = "Day",
    ShowTrackBall = true,
    Interior = new SolidColorBrush(Colors.Gray),
    BorderBrush = new SolidColorBrush(Colors.DarkGray),
    BorderThickness = new Thickness(1),
    LineStyle = this.Resources["lineStyle1"] as Style,
    TrackBallStyle = this.Resources["lineStyle2"] as Style
};
```



### Markers in WPF Sparkline (SfSparkline)

Markers are used to indicate the value point for line and area series, and we can customize with different template.

#### XML

```
<Syncfusion:SfLineSparkline
ItemsSource="{Binding UsersList}"
MarkerVisibility="Visible"
YBindingPath="NoOfUsers">
</Syncfusion:SfLineSparkline >
```

#### C#

```
SfLineSparkline sparkline = new SfLineSparkline()
{
    ItemsSource = new SparkViewModel().UsersList,
    YBindingPath = "NoOfUsers",
    MarkerVisibility = Visibility.Visible
};
```

Following is the snapshot for markers,



### Marker Customization in WPF Sparkline (SfSparkline)

We can customize the markers by initializing the marker template selector class, and we can differentiate the first, last, high, low, negative points.

#### XML

```
<Syncfusion:SfLineSparkline
ItemsSource="{Binding UsersList}"
MarkerVisibility="Visible" Padding="20"
YBindingPath="NoOfUsers">
<Syncfusion:SfLineSparkline.MarkerTemplateSelector>
<Syncfusion:MarkerTemplateSelector FirstPointBrush="Yellow"
LastPointBrush="Yellow"
HighPointBrush="Red" MarkerHeight="15" MarkerWidth="15"/>
</Syncfusion:SfLineSparkline.MarkerTemplateSelector>
</Syncfusion:SfLineSparkline >
```

**C#**

```
SfLineSparkline sparkline = new SfLineSparkline()
{
    ItemsSource = new SparkViewModel().Data,
    YBindingPath = "Day",
    MarkerVisibility = Visibility.Visible,
    Padding = new Thickness(20)
};
SegmentTemplateSelector selector = new SegmentTemplateSelector()
{
    FirstPointBrush = new SolidColorBrush(Colors.Yellow),
    LastPointBrush = new SolidColorBrush(Colors.Yellow),
    HighPointBrush = new SolidColorBrush(Colors.Red),
    MarkerHeight = 15,
    MarkerWidth = 15
};
sparkline.MarkerTemplateSelector = selector;
```

Following is the snapshot above code,

**Marker Template**

You can customize default appearance of the marker symbol by using the MarkerTemplate property in the sparkline.

The following code shows how to apply the template for the Marker.

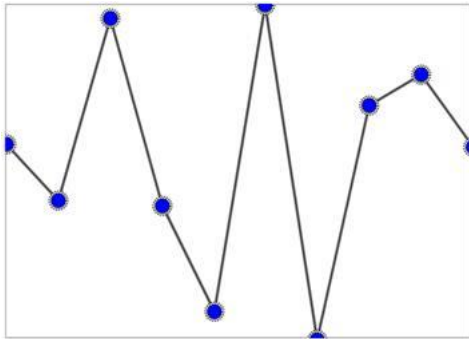
**XML**

```
<syncfusion:SfLineSparkline Interior="#4a4a4a"
    BorderBrush="DarkGray"
    MarkerVisibility="Visible"
    BorderThickness="1"
    ItemsSource="{Binding UsersList}"
    YBindingPath="NoOfUsers">
    <syncfusion:SfLineSparkline.Resources>
    <DataTemplate x:Key="markerTemplate">
    <Grid>
    <Ellipse Height="15" Width="15"
        Fill="LightGoldenrodYellow"
        Stroke="Black" StrokeDashArray="1,1"
        StrokeThickness="1" />
    <Ellipse Height="12" Width="12" Fill="Blue" Stroke="Black"
        StrokeDashArray="1,1"
        StrokeThickness="1"/>
    </Grid>
    </DataTemplate>
    </syncfusion:SfLineSparkline.Resources>
    <syncfusion:SfLineSparkline.MarkerTemplateSelector>
```

```
<syncfusion:MarkerTemplateSelector MarkerTemplate="{StaticResource
markerTemplate}"/>
</syncfusion:SfLineSparkline.MarkerTemplateSelector>
</syncfusion:SfLineSparkline>
```

**C#**

```
SfLineSparkline sparkline = new SfLineSparkline()
{
    ItemsSource = new SparkViewModel().Data,
    YBindingPath = "Day",
    MarkerVisibility = Visibility.Visible,
    Interior = new SolidColorBrush(Colors.Gray),
    BorderBrush = new SolidColorBrush(Colors.DarkGray),
    BorderThickness = new Thickness (1)
};
MarkerTemplateSelector selector = new MarkerTemplateSelector()
{
    MarkerTemplate = sparkline.Resources["markerTemplate"] as DataTemplate
};
sparkline.MarkerTemplateSelector = selector;
```

**Highlight Segment in WPF Sparkline (SfSparkline)**

This feature enable to highlight the column segments on mouse move and this is applicable for column and win-loss sparkline.

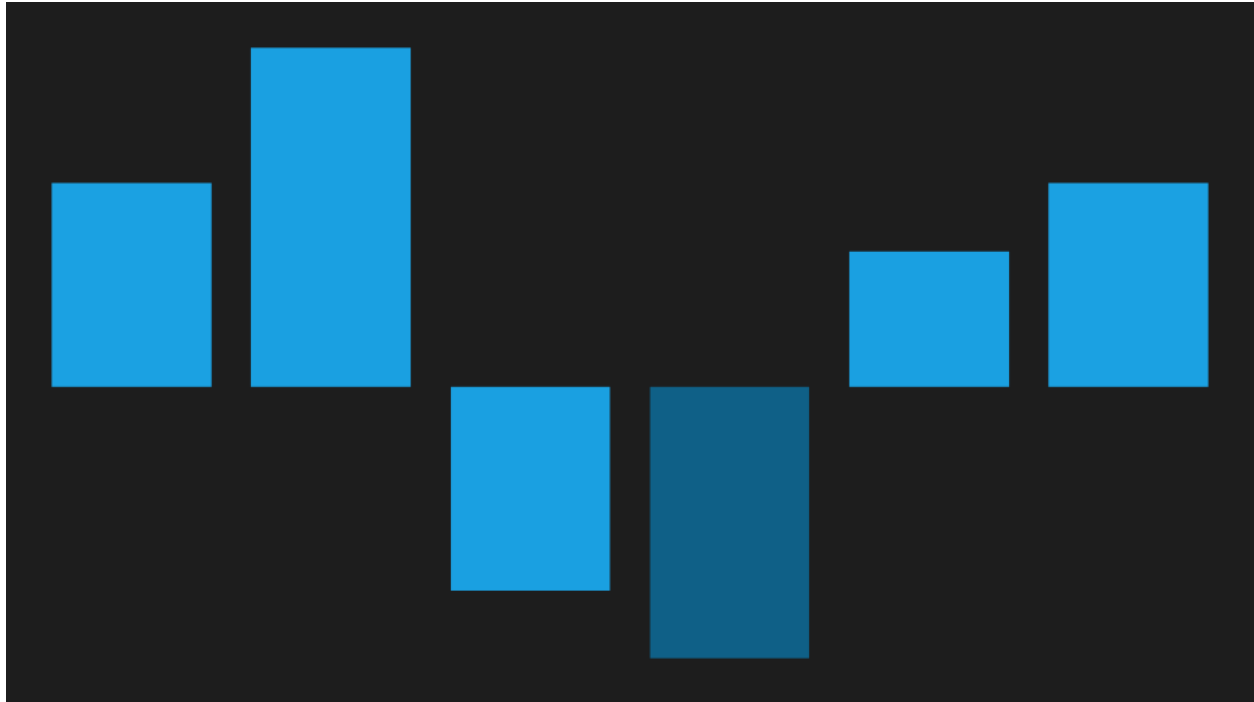
**XML**

```
<Syncfusion:SfColumnSparkline HighlightSegment="True" ItemsSource="{Binding
UsersList}" YBindingPath="NoOfUsers" >
</Syncfusion:SfColumnSparkline>
```

**C#**

```
SfColumnSparkline sparkline = new SfColumnSparkline()
{
    ItemsSource = new SparkViewModel().UsersList,
    YBindingPath = "NoOfUsers",
    HighlightSegment = true
};
```

Following is the snapshot for highlight segment,



### Customize segment brush in WPF Sparkline (SfSparkline)

We can able to customize the first, last, negative, high and low point brushes as like markers in area and line sparkline.

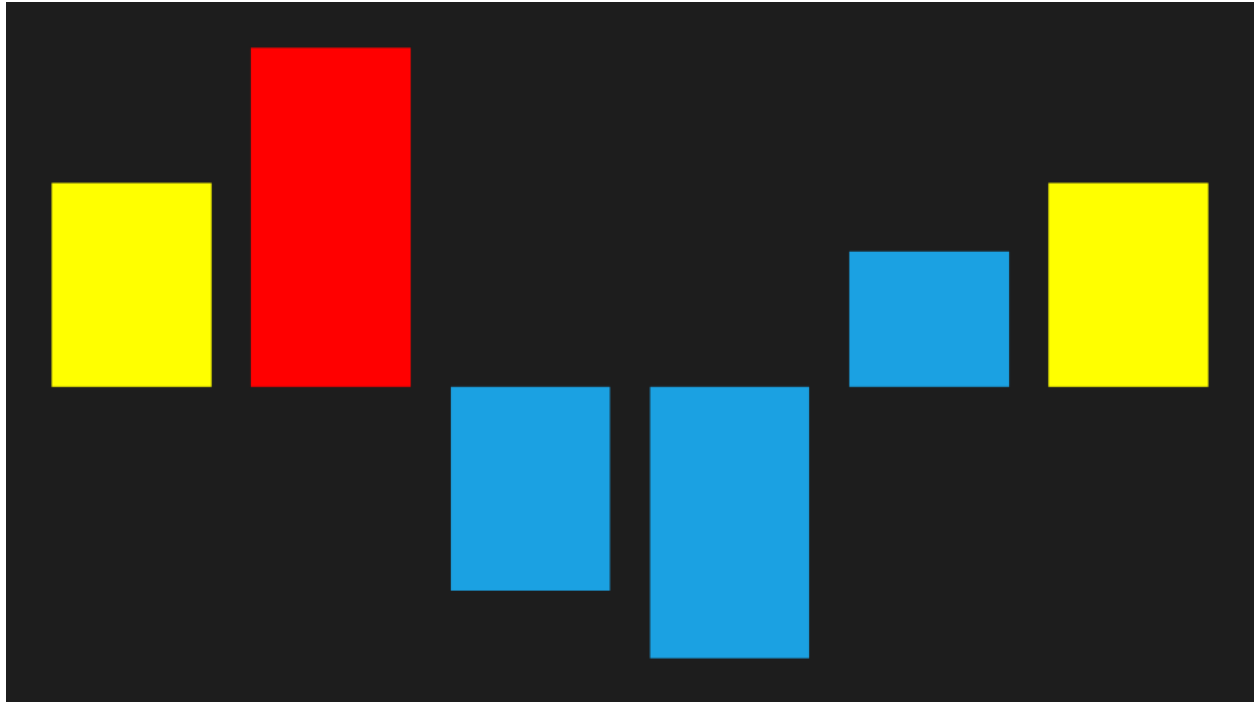
#### XML

```
<Syncfusion:SfColumnSparkline ItemsSource="{Binding UsersList}"
YBindingPath="NoOfUsers" >
  <Syncfusion:SfColumnSparkline.SegmentTemplateSelector>
    <Syncfusion:SegmentTemplateSelector FirstPointBrush="Yellow"
    LastPointBrush="Yellow" HighPointBrush="Red"/>
  </Syncfusion:SfColumnSparkline.SegmentTemplateSelector>
</Syncfusion:SfColumnSparkline>
```

#### C#

```
SfColumnSparkline sparkline = new SfColumnSparkline()
{
    ItemsSource = new SparkViewModel().UsersList,
    YBindingPath = "NoOfUsers"
};
SegmentTemplateSelector selector = new SegmentTemplateSelector()
{
    FirstPointBrush = new SolidColorBrush(Colors.Yellow),
    LastPointBrush = new SolidColorBrush(Colors.Yellow),
    HighPointBrush = new SolidColorBrush(Colors.Red)
};
sparkline.SegmentTemplateSelector = selector;
```

Following is the snapshot for customize segment,



### ShowHide Axis in WPF Sparkline (SfSparkline)

Following code used to enable the axis and this feature applicable for all the sparkline except WinLoss sparkline, also you can style the axis by AxisStyle property and position the axis by AxisOrigin property.

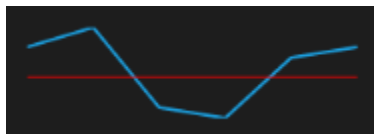
#### XML

```
<Syncfusion:SfLineSparkline ShowAxis="True" ItemsSource="{Binding
UsersList}" YBindingPath="NoOfUsers" >
</Syncfusion:SfLineSparkline>
```

#### C#

```
SfLineSparkline sparkline = new SfLineSparkline()
{
    ItemsSource = new SparkViewModel().UsersList,
    YBindingPath = "NoOfUsers",
    ShowAxis = true
};
```

Following is the snapshot for axis visibility,



#### Axis Origin

#### XML

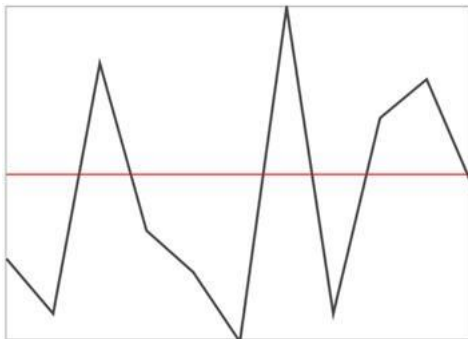
```
<Syncfusion:SfLineSparkline Height="250" Width="350" Interior="#4a4a4a"
BorderBrush="DarkGray" BorderThickness="1">
```



```
ShowAxis="True"
AxisOrigin="2" ItemsSource="{Binding UsersList}"
YBindingPath="NoOfUsers">
</Syncfusion:SfLineSparkline >
```

**C#**

```
SfLineSparkline sparkline = new SfLineSparkline()
{
    ItemsSource = new SparkViewModel().UsersList,
    YBindingPath = "NoOfUsers",
    ShowAxis = true,
    AxisOrigin = 2,
    Interior = new SolidColorBrush(Color.FromRgb(0x4a, 0x4a, 0x4a)),
    BorderBrush = new SolidColorBrush(Colors.DarkGray),
    BorderThickness = new Thickness(1)
};
```

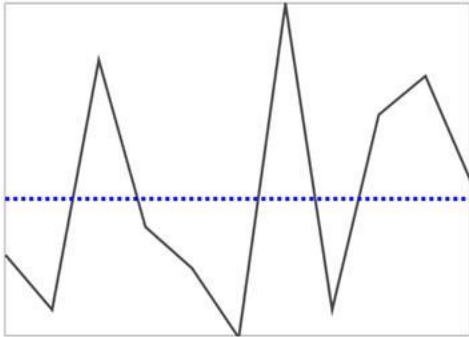
**Axis LineStyle****XML**

```
<Grid.Resources>
<Style TargetType="Line" x:Key="lineStyle2">
<Setter Property="Stroke" Value="Blue"/>
<Setter Property="StrokeThickness" Value="2"/></Setter>
<Setter Property="StrokeDashArray" Value="1,1"/></Setter>
</Style>
</Grid.Resources>
<Syncfusion:SfLineSparkline Height="250" Width="350" Interior="#4a4a4a"
BorderBrush="DarkGray" BorderThickness="1"
ShowAxis="True" AxisStyle="{StaticResource lineStyle2}"
AxisOrigin="1" ItemsSource="{Binding UsersList}"
YBindingPath="NoOfUsers">
</Syncfusion:SfLineSparkline >
```

**C#**

```
SfLineSparkline sparkline = new SfLineSparkline()
{
    ItemsSource = new SparkViewModel().UsersList,
    YBindingPath = "NoOfUsers",
    ShowAxis = true,
    AxisOrigin = 1,
```

```
AxisStyle = grid.Resources["lineStyle2"] as Style,
Interior = new SolidColorBrush(Color.FromRgb(0x4a, 0x4a, 0x4a)),
BorderBrush = new SolidColorBrush(Colors.DarkGray),
BorderThickness = new Thickness(1)
};
```



## How to

Customize the marker for specific data point

We can customize the marker for specific data point with custom template for LineSparkline and AreaSparkline, in order to customize the marker we need to inherit the MarkerTemplateSelector class and override the SelectTemplate method.

## XML

```
<Syncfusion:SfLineSparkline x:Name="sparkline" MarkerVisibility="Visible"
ItemsSource="{Binding UsersList}" YBindingPath="NoOfUsers" >
<Syncfusion:SfLineSparkline.MarkerTemplateSelector>
<local:CustomMarkersTemplateSelector MarkerHeight="10" MarkerWidth="10"/>
</Syncfusion:SfLineSparkline.MarkerTemplateSelector>
</Syncfusion:SfLineSparkline>
```

## C#

```
public class CustomMarkersTemplateSelector : MarkerTemplateSelector
{
    protected override DataTemplate SelectTemplate(double x, double y)
    {
        if (y == MaximumY)
        {
            DataTemplate markerTemplate =
            Application.Current.Resources["markerTemplate"] as DataTemplate;
            return markerTemplate;
        }
        else
            return base.SelectTemplate(x, y);
    }
}
```

Following is the snapshot for custom marker position,



### Add labels for track ball

We can add labels for track ball to show the corresponding values. In order to add labels for the trackball, you need to subscribe the event `OnSparklineMouseMove` and you can get the following data's from event argument.

### XML

```
<Page.DataContext>
<local:UsersViewModel/>
</Page.DataContext>
<Syncfusion:SfLineSparkline ShowTrackBall="True"
OnSparklineMouseMove="SfLineSparkline_OnSparklineMouseMove"
x:Name="sparkline" ItemsSource="{Binding UsersList}"
YBindingPath="NoOfUsers" >
</Syncfusion:SfLineSparkline>
```

### C#

```
ContentPresenter info;
private void SfLineSparkline_OnSparklineMouseMove(object src,
SparklineMouseMoveEventArgs args)
{
    if (!args.RootPanel.Children.Contains(info))
    {
        info=new ContentPresenter();
        args.RootPanel.Children.Add(info);
        TextBlock.SetForeground(info, new SolidColorBrush(Colors.Red));
        TextBlock.SetFontSize(info, 25);
    }
    info.Content = args.Value.Y;
    info.Arrange(new
    Rect(args.Coordinate.X, args.Coordinate.Y, info.ActualWidth, info.ActualHeight)
    );
}
```

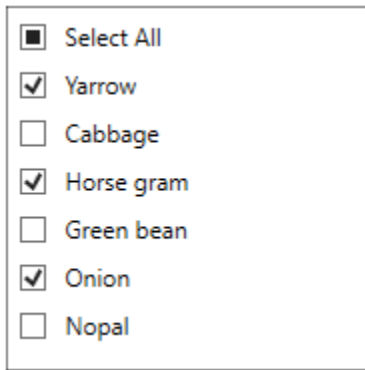
Following is the snapshot for track ball labels,



## CheckListBox

### WPF CheckListBox (CheckListBox) Overview

The [CheckListBox](#) control implements a classic list box with check list box items. The control displays items with a check box to enable multiple selection of items. Custom templates are defined to customize its appearance.



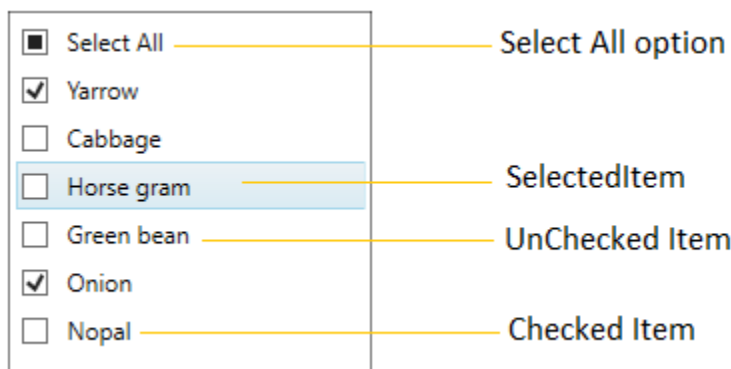
### Features

- Select items on first mouse click
- Align the check box to the right or left side of the CheckListBox Item
- Grouping and Sorting support
- SelectAll support
- Built-in Visual Styles and Themes support

### Getting Started with WPF CheckedListBox (CheckListBox)

This section explains how to display and select the required items using the [WPF CheckListBox](#) control.

#### Control Structure



#### Assembly deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as reference to use the control in any application.

Further information on installing the NuGet package can be found in the following link: [How to install nuget packages](#).

You can also use the [Syncfusion Reference Manager](#) to refer the CheckListBox's dependent assemblies.

#### Creating simple application with CheckListBox control

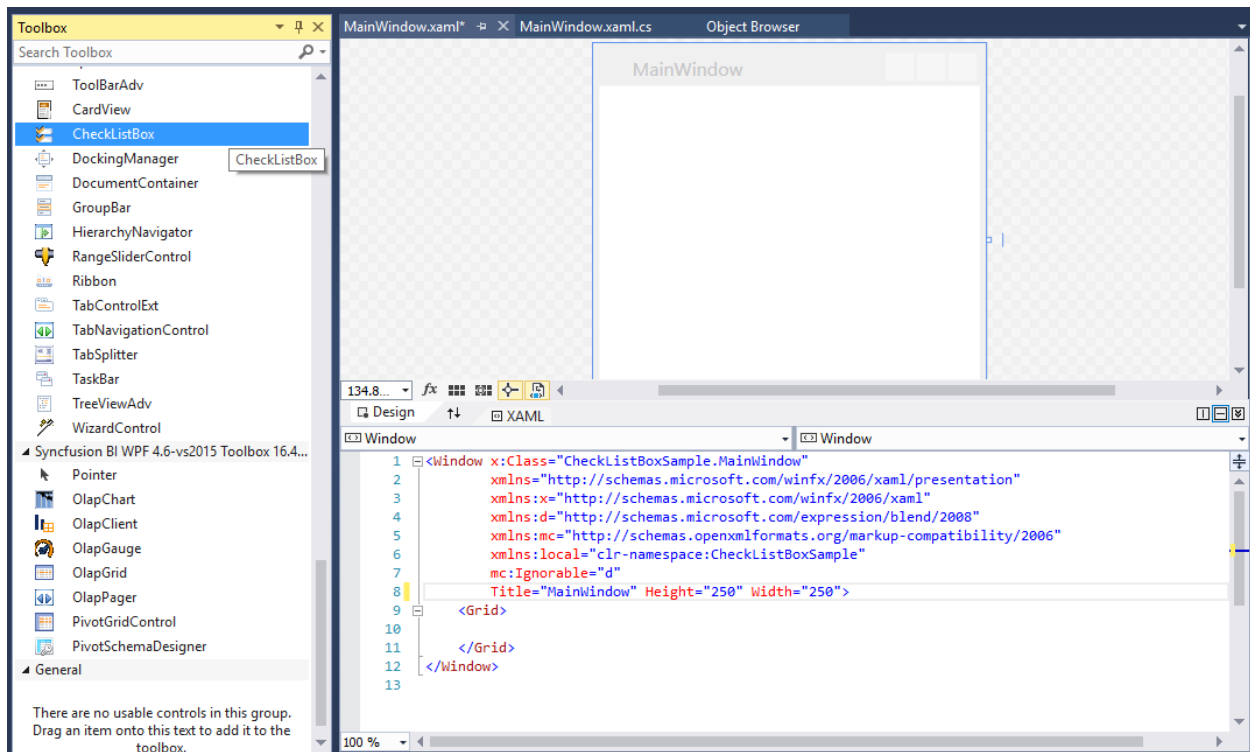
In this walk through, the user will create a WPF application that contains the [CheckListBox](#) control.

#### Creating project

In Visual Studio, create a new WPF project to show the features of the [CheckListBox](#) control and add the following namespace to the added assemblies.

## Adding control via designer

The WPF [CheckListBox](#) control can be added to the application by dragging it from Toolbox and dropping it in the designer. The required [assemblies](#) will be added automatically.



## Adding control manually in XAML

To add the WPF [CheckListBox](#) control manually in XAML, follow these steps,

- 1.Add the following assembly references to the project. *Syncfusion.Shared.WPF* *Syncfusion.Tools.WPF*
- 2.Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page. 3.Declare the [CheckListBox](#) control in XAML page.

## XML

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:GettingStartedComboBox"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="GettingStarted.MainWindow"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<syncfusion:CheckListBox x:Name="checkListBox" Width="200" Height="300"/>
</Grid>
</Window>
```

## Adding control manually in C#

To add the [CheckListBox](#) control manually in C#, follow these steps,

1. Add the following assembly references to the project. *Syncfusion.Shared.WPF* *Syncfusion.Tools.WPF*
2. Import CheckListBox namespace **Syncfusion.Windows.Tools.Controls**.
3. Create the **CheckListBox** control instance and add it to the window.

### C#

```
using System.Windows;
using Syncfusion.Windows.Tools.Controls;
namespace ComboBox
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            //Creating an instance of CheckListBox control.
            CheckListBox checkListBox = new CheckListBox();
            //Adding CheckListBox as window content.
            this.Content = checkListBox;
        }
    }
}
```

### Populating items using CheckListBoxItem

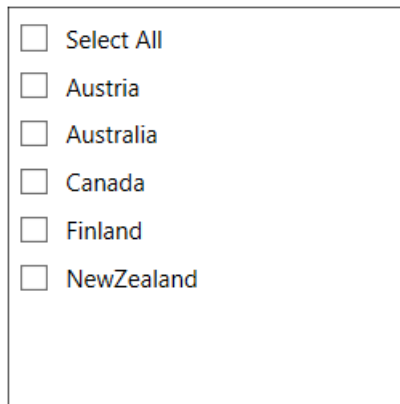
You can add the items inside the WPF [CheckListBox](#) control using the [CheckListBoxItem](#).

### XML

```
<syncfusion:CheckListBox Height="200" Width="200">
<syncfusion:CheckListBoxItem Content="Austria" />
<syncfusion:CheckListBoxItem Content="Australia"/>
<syncfusion:CheckListBoxItem Content="Canada"/>
<syncfusion:CheckListBoxItem Content="Finland"/>
<syncfusion:CheckListBoxItem Content="NewZealand"/>
</syncfusion:CheckListBox>
```

### C#

```
CheckListBox checkListBox = new CheckListBox();
checkListBox.Height = 200;
checkListBox.Width = 200;
CheckListBoxItem item1 = new CheckListBoxItem() { Content = "Austria" };
CheckListBoxItem item2 = new CheckListBoxItem() { Content = "Australia" };
CheckListBoxItem item3 = new CheckListBoxItem() { Content = "Canada" };
CheckListBoxItem item4 = new CheckListBoxItem() { Content = "Finland" };
CheckListBoxItem item5 = new CheckListBoxItem() { Content = "NewZealand" };
checkListBox.Items.Add(item1);
checkListBox.Items.Add(item2);
checkListBox.Items.Add(item3);
checkListBox.Items.Add(item4);
checkListBox.Items.Add(item5);
this.Content = checkListBox;
```



### Populating items by DataBinding

You can populate the items to the WPF [CheckListBox](#) control by using the `ItemsSource` property. The `DisplayMemberPath` property is used to the name or path of the property displayed for each data item in the control.

1. Create Model and populate it with required properties. Create the ViewModel class and populate ObservableCollection object with the Model objects.

### C#

```
//Model.cs
public class Model
{
    public string Name { get; set; }
    public string Description { get; set; }
}

//ViewModel.cs
public class ViewModel
{
    private ObservableCollection<Model> checkList;
    public ObservableCollection<Model> CheckListItems
    {
        get
        {
            return checkList;
        }
        set
        {
            checkList = value;
        }
    }
    public ViewModel()
    {
        CheckListItems = new ObservableCollection<Model>();
        populateItem();
    }
    private void populateItem()
    {
        CheckListItems.Add(new Model() { Name="Mexico", Description="Mexico"});
        CheckListItems.Add(new Model() { Name="Canada", Description="Canada"});
    }
}
```

```
CheckListItems.Add(new Model() { Name="Bermuda", Description="Bermuda"});  
CheckListItems.Add(new Model() { Name="Beize", Description="Beize"});  
CheckListItems.Add(new Model() { Name="Panama", Description="Panama"});  
}  
}
```

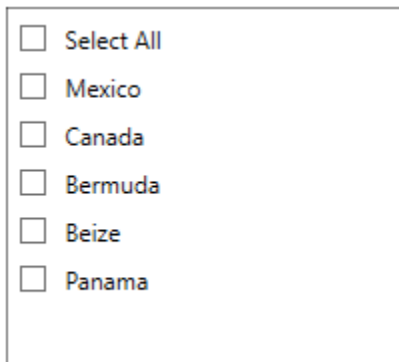
2. Now create an instance of ViewModel in **DataContext** property of the CheckListBox control in **MainWindow.xaml** and bind the collection property from ViewModel to the **ItemSource** property of CheckListBox. Set the property from Model class to be displayed in the **DisplayMemberPath** property.

### XML

```
<!--Adding CheckListBox control -->  
<syncfusion:CheckListBox ItemsSource="{Binding CheckListItems}"  
    DisplayMemberPath="Name"  
    x:Name="checkListBox">  
    <syncfusion:CheckListBox.DataContext>  
    <local:ViewModel />  
    </syncfusion:CheckListBox.DataContext>  
</syncfusion:CheckListBox>
```

### C#

```
CheckListBox checkListBox = new CheckListBox();  
checkListBox.DataContext = new ViewModel();  
checkListBox.ItemsSource = (checkListBox.DataContext as  
ViewModel).CheckListItems;  
checkListBox.DisplayMemberPath = "Name";
```

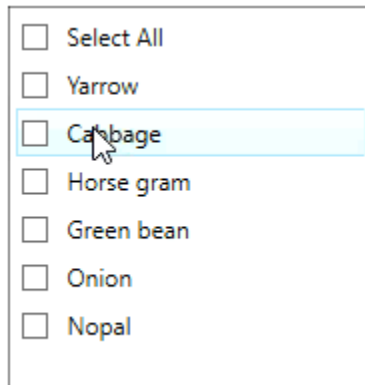


### Check or Uncheck items

You can check or uncheck the items in the [CheckListBox](#) by clicking on the CheckBox or the content of the item. You can use **Space** key to uncheck or check the previously selected item.

You can programmatically check the items in CheckListBox by adding the items in the [SelectedItems](#) property.





### Checked event notification

When the checked state of an item is changed, it will be notified by using the [ItemChecked](#) event. You can get the details about the checked item in the [ItemCheckedEventArgs](#).

### XML

```
<!--Adding CheckListBox control -->
<syncfusion:CheckListBox ItemChecked="CheckListBox_ItemChecked"
x:Name="checkListBox">
</syncfusion:CheckListBox>
```

### C#

```
CheckListBox checkListBox = new CheckListBox();
checkListBox.ItemChecked += CheckListBox_ItemChecked;
```

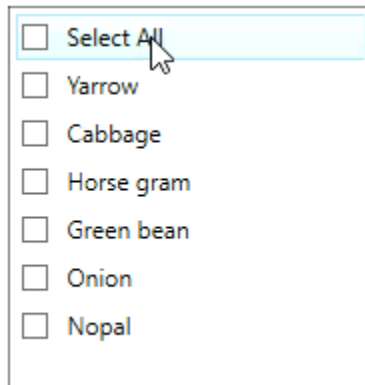
You can handle the event as follows:

### C#

```
private void CheckListBox_ItemChecked(object sender, ItemCheckedEventArgs e)
{
    MessageBox.Show((e.Item as CheckListBoxItem).Content.ToString() + " is
checked ");
}
```

### Get list of checked items

The [CheckListBox](#) control gets all the checked items using the [SelectedItems](#) property. You can also get the currently selected item which is in either checked or unchecked state by using the [SelectedItem](#) property.



[View Sample in GitHub](#)

### Localization support

Localization is the process of translating the application resources into different language for the specific cultures. You can localize the `Select All` `CheckListBoxItem` in `CheckListBox` control by adding resource file for each language.

To localize the `CheckListBox` based on culture using resource files, follow the below steps.

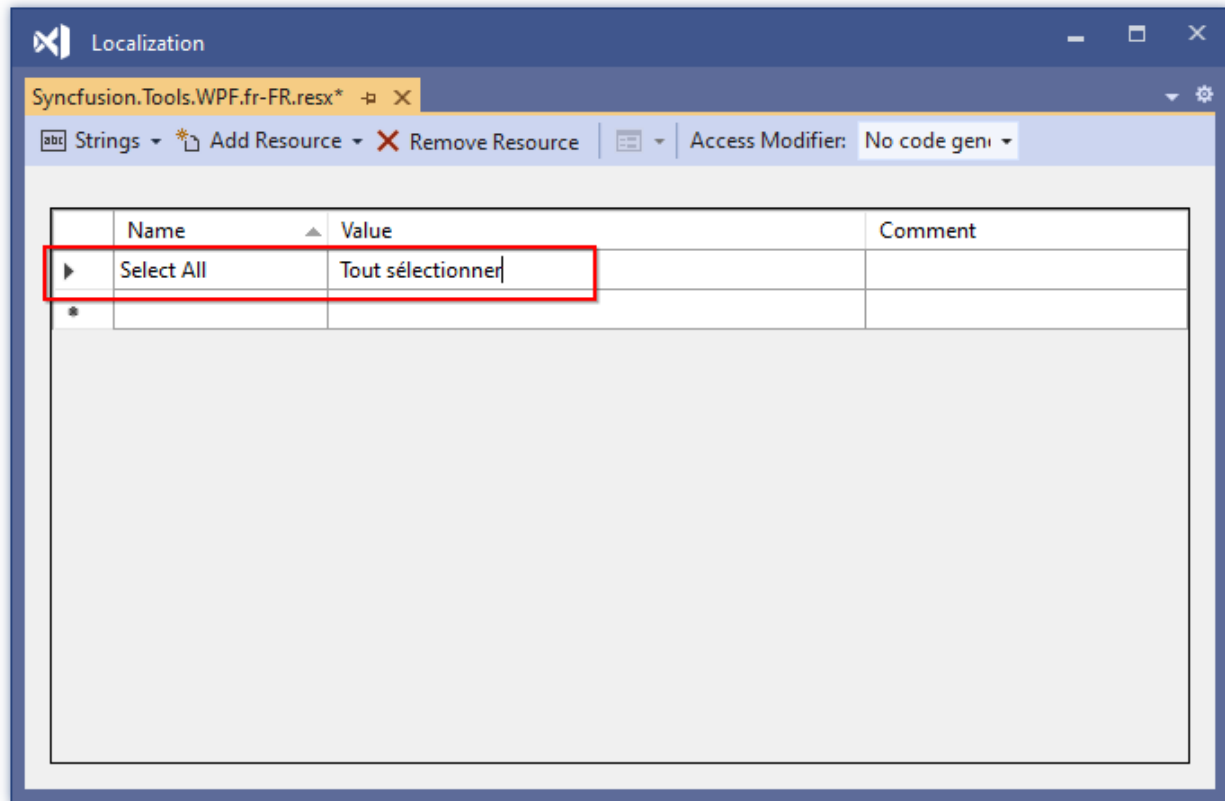
1. Change application culture and create `.resx` files.

---

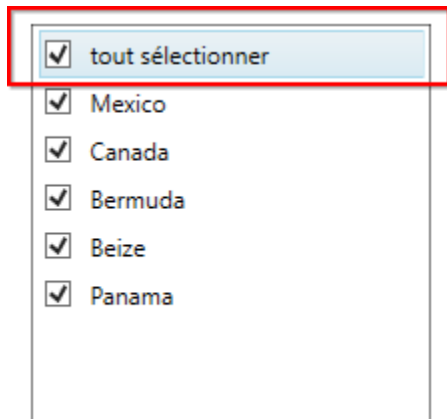
**Note:** Refer [Localization of Syncfusion WPF Controls](#) page to know more about how to Change application culture and create `.resx` files into a application.

---

2. After creating `.resx` file, add the Name/Value pair in Resource Designer of created `.resx` file and change its corresponding value to corresponding culture.



The following screenshot shows the localized CheckListBox control.

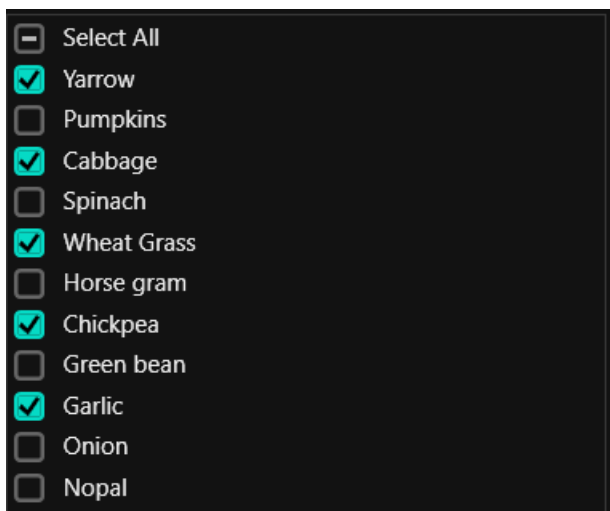


[View Sample in GitHub](#)

#### Theme

CheckListBox supports various built-in themes. Refer to the below links to apply themes for the CheckListBox,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



[View Sample in GitHub](#)

### Sorting Items in WPF CheckedListBox (CheckListBox)

By default, the [CheckListBox](#) are arranged based on the order they added in to the `Items` collection. We can sort the `CheckListBox` items by adding the sorting direction to the `CollectionView.SortDescriptions` collection. The `CheckListBox` items can be sorted either in ascending or descending order based on the user's perspective.

#### C#

```
//Model.cs
class Vegetable {
public string Category { get; set; }
public int Price { get; set; }
public string Name { get; set; }
}

//ViewModel.cs
class ViewModel {
public ObservableCollection<Vegetable> Vegetables { get; set; }
public ICommand LoadedCommand { get; set; }
public void OnLoaded(object param) {
CollectionView view =
(CollectionView)CollectionViewSource.GetDefaultView(Vegetables);
//Adding sort description
view.SortDescriptions.Add(new SortDescription
{
PropertyName = "Name",
Direction = ListSortDirection.Ascending
});
}

public ViewModel() {
Vegetables = new ObservableCollection<Vegetable>();
Vegetables.Add(new Vegetable { Price = 10, Name = "Yarrow", Category =
"Leafy and Salad" });
Vegetables.Add(new Vegetable { Price = 20, Name = "Cabbage", Category =
"Leafy and Salad" });
Vegetables.Add(new Vegetable { Price = 30, Name = "Horse gram", Category =
"Beans" });
}
```

```

Vegetables.Add(new Vegetable { Price = 20, Name = "Green bean", Category =
"Beans" });
Vegetables.Add(new Vegetable { Price = 10, Name = "Onion", Category = "Bulb
and Stem" });
Vegetables.Add(new Vegetable { Price = 30, Name = "Nopal", Category = "Bulb
and Stem" });
//Initialize the CheckListBox LoadedCommand
LoadedCommand = new DelegateCommand<object>(OnLoaded);
}
}

```

## XML

```

<syncfusion:CheckListBox ItemsSource="{Binding Vegetables}"
DisplayMemberPath="Name"
Name="checkListBox">
<syncfusion:CheckListBox.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:CheckListBox.DataContext>
<i:Interaction.Triggers>
<i:EventTrigger EventName="Loaded">
<i:InvokeCommandAction Command="{Binding LoadedCommand}" />
</i:EventTrigger>
</i:Interaction.Triggers>
</syncfusion:CheckListBox>

```

Here, the **Vegetables** items are sorted in ascending order based on their name.

### Custom Sorting: Ascending

☐ Select All  
☐ Cabbage  
☐ Green bean  
☐ Horse gram  
☐ Nopal  
☐ Onion  
☐ Yarrow

### Default Sorting order

☐ Select All  
☐ Yarrow  
☐ Cabbage  
☐ Horse gram  
☐ Green bean  
☐ Onion  
☐ Nopal

Click [here](#) to download the sample that showcases the sorting support in the **CheckListBox**.

## Grouping Items in WPF CheckListBox (CheckListBox)

By default, the **CheckListBox** items are in the listed view. We can group the **CheckListBox** items by adding the group description to the **CollectionView.GroupDescriptions** collection.

The selection state of group header varies based on the checked or unchecked state of the child items. Group can be expanded or collapsed and the child items present in the group can be checked or unchecked based on the user's perspective.

## C#

```

//Model.cs
class Vegetable {
public string Category { get; set; }
public int Price { get; set; }
public string Name { get; set; }
}
//ViewModel.cs
class ViewModel {
public ObservableCollection<Vegetable> Vegetables { get; set; }
public ICommand LoadedCommand { get; set; }
public void OnLoaded(object param) {
CollectionView view =
(CollectionView)CollectionViewSource.DefaultView(Vegetables);
//Adding group description
view.GroupDescriptions.Add(new PropertyGroupDescription("Price"));
}
public ViewModel() {
Vegetables = new ObservableCollection<Vegetable>();
Vegetables.Add(new Vegetable { Price=10, Name="Yarrow", Category="Leafy and
Salad"});
Vegetables.Add(new Vegetable { Price=20, Name="Cabbage", Category="Leafy and
Salad"});
Vegetables.Add(new Vegetable { Price=30, Name="Horse gram",
Category="Beans"});
Vegetables.Add(new Vegetable { Price=20, Name="Green bean",
Category="Beans"});
Vegetables.Add(new Vegetable { Price=10, Name="Onion", Category="Bulb and
Stem"});
Vegetables.Add(new Vegetable { Price=30, Name="Nopal", Category="Bulb and
Stem"});
//Initialize the CheckListBox LoadedCommand
LoadedCommand = new DelegateCommand<object>(OnLoaded);
}
}

```

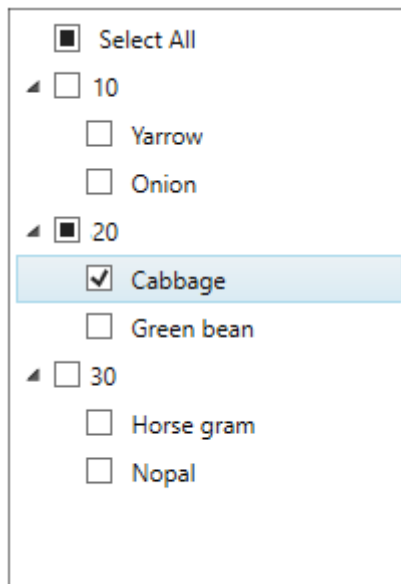
**XML**

```

<syncfusion:CheckListBox ItemsSource="{Binding Vegetables}"
DisplayMemberPath="Name"
Name="checkListBox">
<syncfusion:CheckListBox.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:CheckListBox.DataContext>
<i:Interaction.Triggers>
<i:EventTrigger EventName="Loaded">
<i:InvokeCommandAction Command="{Binding LoadedCommand}" />
</i:EventTrigger>
</i:Interaction.Triggers>
</syncfusion:CheckListBox>

```

Here, the **Vegetables** items are grouped based on their price.



Click [here](#) to download the sample that showcases the grouping support in the CheckListBox.

### Nested Grouping

We can create a multi-level nested groups for the CheckListBox items by adding the two or more group descriptions to the CollectionView.GroupDescriptions collection. Child level groups are created by order of group descriptions added to the collection.

### C#

```
//Model.cs
class Vegetable {
public string Category { get; set; }
public int Price { get; set; }
public string Name { get; set; }
}

//ViewModel.cs
class ViewModel {
public ObservableCollection<Vegetable> Vegetables { get; set; }
public ICommand LoadedCommand { get; set; }
public void OnLoaded(object param) {
CollectionView view =
(CollectionView)CollectionViewSource.GetDefaultView(Vegetables);
//Adding the multiple group description
view.GroupDescriptions.Add(new PropertyGroupDescription("Category"));
view.GroupDescriptions.Add(new PropertyGroupDescription("Price"));
}

public ViewModel() {
Vegetables = new ObservableCollection<Vegetable>();
Vegetables.Add(new Vegetable { Price=10, Name="Yarrow", Category="Leafy and Salad"});
Vegetables.Add(new Vegetable { Price=20, Name="Pumpkins", Category="Leafy and Salad"});
Vegetables.Add(new Vegetable { Price=30, Name="Cabbage", Category="Leafy and Salad"});
Vegetables.Add(new Vegetable { Price=10, Name="Spinach", Category="Leafy and Salad"});
}
```

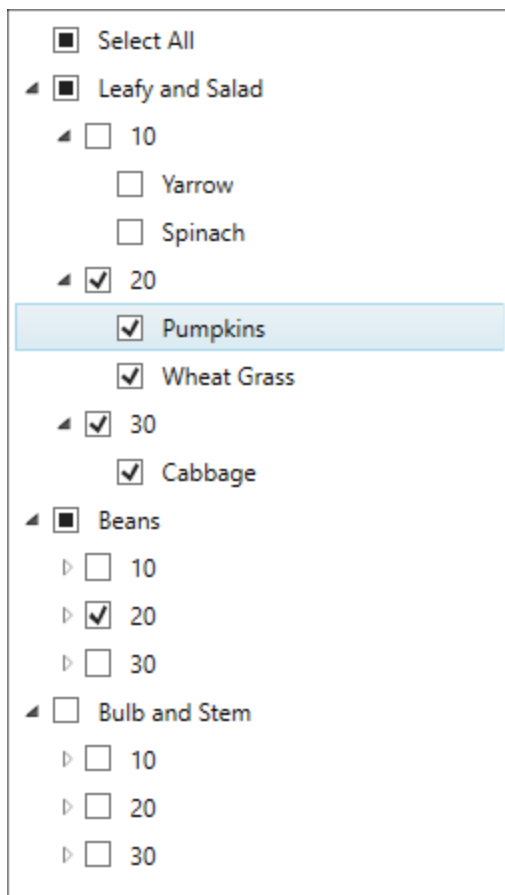
```
Vegetables.Add(new Vegetable { Price=20, Name="Wheat Grass", Category="Leafy and Salad"});  
Vegetables.Add(new Vegetable { Price=10, Name="Horse gram", Category="Beans"});  
Vegetables.Add(new Vegetable { Price=20, Name="Chickpea", Category="Beans"});  
Vegetables.Add(new Vegetable { Price=30, Name="Green bean", Category="Beans"});  
Vegetables.Add(new Vegetable { Price=10, Name="Garlic", Category="Bulb and Stem"});  
Vegetables.Add(new Vegetable { Price=20, Name="Onion", Category="Bulb and Stem"});  
Vegetables.Add(new Vegetable { Price=30, Name="Nopal", Category="Bulb and Stem"});  
//Initialize the checklistbox LoadedCommand  
LoadedCommand = new DelegateCommand<object>(OnLoaded);  
}  
}
```

## XML

```
<syncfusion:CheckListBox ItemsSource="{Binding Vegetables}"  
DisplayMemberPath="Name"  
Name="checkListBox">  
  <syncfusion:CheckListBox.DataContext>  
    <local:ViewModel></local:ViewModel>  
  </syncfusion:CheckListBox.DataContext>  
  <i:Interaction.Triggers>  
    <i:EventTrigger EventName="Loaded">  
      <i:InvokeCommandAction Command="{Binding LoadedCommand}" />  
    </i:EventTrigger>  
  </i:Interaction.Triggers>  
</syncfusion:CheckListBox>
```

Here, the **Vegetables** items are grouped based on their Category and then, the categorized properties are grouped by price.





Click [here](#) to download the sample that showcases the nested grouping support in the **CheckListBox**.

#### Custom grouping

**CheckListBox** allows you to group the items based on custom logic when the common grouping functionality doesn't meet your requirement. We can achieve this by using the **IValueConverter**.

#### C#

```
//Model.cs
public class Employee {
    public string Name { get; set; }
    public int Age { get; set; }
    public DateTime DOB { get; set; }
}

//EmployeesCollection.cs
public class EmployeesCollection : ObservableCollection<Employee> {
}

//ViewModel.cs
public class ViewModel : NotificationObject {
    private ObservableCollection<GroupDescription> groupDescriptions;
    private EmployeesCollection employeesCollection;
    public ObservableCollection<GroupDescription> GroupDescriptions {
        get {
            return groupDescriptions;
        }
        set {
            groupDescriptions = value;
        }
    }
}
```

```

RaisePropertyChanged("GroupDescriptions");
}
}
public EmployeesCollection EmployeesCollection {
get {
return employeesCollection;
}
set {
employeesCollection = value;
RaisePropertyChanged("EmployeesCollection");
}
}
public ViewModel() {
EmployeesCollection = new EmployeesCollection();
GroupDescriptions = new ObservableCollection<GroupDescription>();
}
}

```

**C#**

```

//Converter for the Custom Grouping
public class DateConverter : IValueConverter {
public string Category { get; set; }
public object Convert(object value, Type targetType, object parameter,
CultureInfo culture) {
DateTime result;
string MyString = value.ToString();
DateTime.TryParse(MyString, out result);
if (Category == "Year") {
//Grouping the employee who are all born on 1980-1990
if ((DateTime.Compare(new DateTime(1980, 01, 01), result)) <= 0 &&
(DateTime.Compare(new DateTime(1990, 12, 31), result)) >= 0)
{
return "Birth Year(1980 - 1990)";
}
//Grouping the employee who are all born on 1990-2000
else if ((DateTime.Compare(new DateTime(1991, 01, 01), result)) <= 0 &&
(DateTime.Compare(new DateTime(2000, 12, 31), result)) >= 0)
{
return "Birth Year(1991 - 2000)";
}
}
else if (Category == "Month") {
//Sub-Grouping the employee who are all born on Jan-Mar
if (result.Month >= 1 && result.Month <= 3) {
return "Quarter-1";
}
//Sub-Grouping the employee who are all born on Apr-Jun
else if (result.Month >= 4 && result.Month <= 6) {
return "Quarter-2";
}
//Sub-Grouping the employee who are all born on Jul-Sep
else if (result.Month >= 7 && result.Month <= 9) {
return "Quarter-3";
}
}
//Sub-Grouping the employee who are all born on Oct-Dec
}
}

```

```

else {
    return "Quarter-4";
}
}
return "Others";
}
public object ConvertBack(object value, Type targetType, object parameter,
    CultureInfo culture) {
    throw new NotImplementedException();
}
}

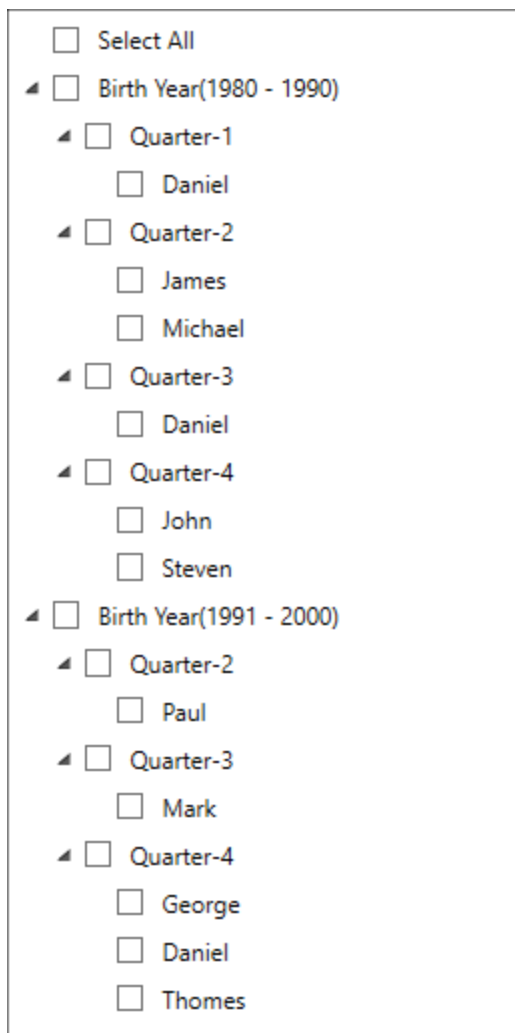
```

## XML

```

<Window.Resources>
    <!-- Define employee details-->
    <local:EmployeesCollection x:Key="employees">
        <local:Employee Name="Daniel" DOB="02/15/1983" Age="37"/>
        <local:Employee Name="James" DOB="06/29/1988" Age="32"/>
        <local:Employee Name="Michael" DOB="06/10/1987" Age="33"/>
        <local:Employee Name="Daniel" DOB="08/23/1980" Age="40"/>
        <local:Employee Name="John" DOB="12/22/1990" Age="30"/>
        <local:Employee Name="Paul" DOB="04/08/1997" Age="23"/>
        <local:Employee Name="Mark" DOB="08/05/1994" Age="26"/>
        <local:Employee Name="George" DOB="10/01/1998" Age="22"/>
        <local:Employee Name="Daniel" DOB="10/01/1998" Age="24"/>
        <local:Employee Name="Thomes" DOB="10/08/1995" Age="25"/>
        <local:Employee Name="Steven" DOB="12/15/1982" Age="38"/>
    </local:EmployeesCollection>
    <!-- Define view model with group description -->
    <local:DateConverter Category="Year" x:Key="year"/>
    <local:DateConverter Category="Month" x:Key="month"/>
    <local:ViewModel x:Key="viewModel">
        <local:ViewModel.GroupDescriptions>
            <PropertyGroupDescription PropertyName="DOB"
                Converter="{StaticResource year}"/>
            <PropertyGroupDescription PropertyName="DOB"
                Converter="{StaticResource month}"/>
        </local:ViewModel.GroupDescriptions>
    </local:ViewModel>
</Window.Resources>
<Grid>
    <syncfusion:CheckListBox ItemsSource="{StaticResource
        ResourceKey=employees}"
        GroupDescriptions="{Binding GroupDescriptions}"
        DataContext="{StaticResource viewModel}"
        DisplayMemberPath="Name"
        Name="checkListBox"
        Margin="20"/>
</Grid>

```



Click [here](#) to download the sample that showcases the custom grouping support.

### Check or uncheck all items in WPF CheckedListBox (CheckListBox)

The [CheckListBox](#) allows the users to check or uncheck all the items by a single click on the **SelectAll** option. The **SelectAll** option can be enabled or disabled using the [IsSelectAllEnabled](#) property. Selection mode of the **SelectAll** item varies based on the checked state of the **CheckListBox** items. Initially, the **SelectAll** is unchecked because of no item is checked. It is checked only on when all items are checked, otherwise it is in intermediate state.

#### C#

```
//ViewModel.cs
class ViewModel {
public ObservableCollection<string> DaysCollection { get; set; }
public ViewModel() {
//Days added in the collection
DaysCollection = new ObservableCollection<string>();
DaysCollection.Add("Sunday");
DaysCollection.Add("Monday");
DaysCollection.Add("Tuesday");
DaysCollection.Add("Wednesday");
```

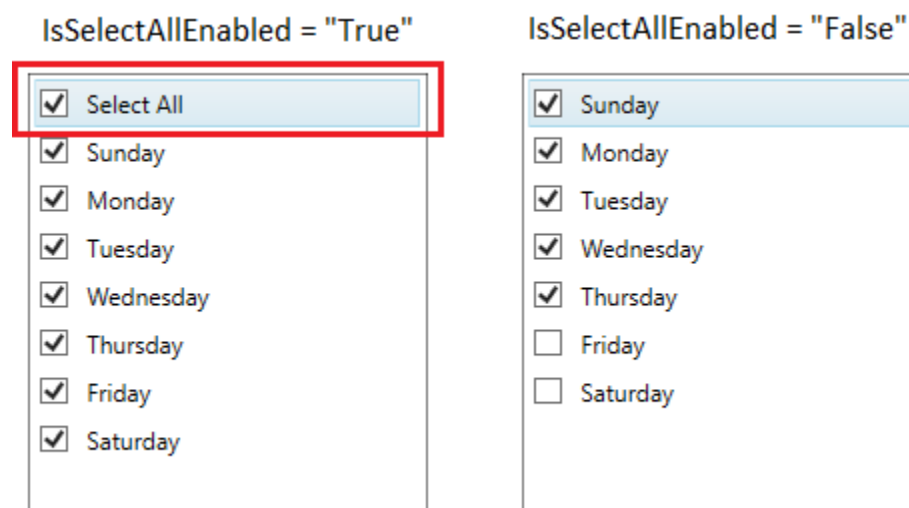
```
DaysCollection.Add("Thursday");  
DaysCollection.Add("Friday");  
DaysCollection.Add("Saturday");  
}  
}
```

### XML

```
<syncfusion:CheckListBox IsSelectAllEnabled="False"  
ItemsSource="{Binding DaysCollection}"  
Name="checkListBox">  
  <syncfusion:CheckListBox.DataContext>  
    <local:ViewModel></local:ViewModel>  
  </syncfusion:CheckListBox.DataContext>  
</syncfusion:CheckListBox>
```

### C#

```
CheckListBox checkListBox = new CheckListBox();  
checkListBox.DataContext = new ViewModel();  
checkListBox.ItemsSource = (checkListBox.DataContext as  
ViewModel).DaysCollection;
```



Click [here](#) to download the sample that showcases the SelectAll option in the **CheckListBox**.

### Check Items in WPF CheckedListBox (CheckListBox)

In [CheckListBox](#), items present in the control can be checked or unchecked either by using any one of the following ways:

1. Using Collection
2. Using Property
3. Using Mouse
4. Using Keyboard

Check items programmatically

You can check or uncheck the items by using collection or property.

*Check items using Collection*

We can check or uncheck the particular item by add or remove that items into the [SelectedItems](#) collection.

**C#**

```
//Model.cs
class Vegetable {
public string Category { get; set; }
public int Price { get; set; }
public string Name { get; set; }
}

//ViewModel.cs
class ViewModel : NotificationObject {
private ObservableCollection<Vegetable> vegetables =
new ObservableCollection<Vegetable>();
private ObservableCollection<object> checkedVegetables =
new ObservableCollection<object>();
//Vegetables collection
public ObservableCollection<Vegetable> Vegetables {
get {
return vegetables;
}
set {
vegetables = value;
this.RaisePropertyChanged("Vegetables");
}
}

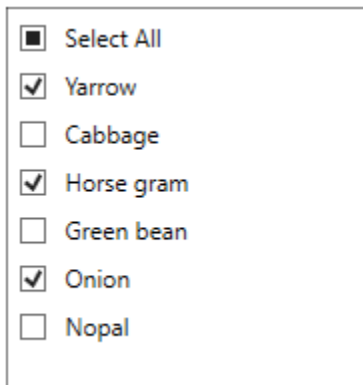
//Selected vegetable collection
public ObservableCollection<object> CheckedVegetables {
get {
return checkedVegetables;
}
set {
checkedVegetables = value;
this.RaisePropertyChanged("CheckedVegetables");
}
}

public ViewModel() {
//Adding a vegetables details
Vegetables = new ObservableCollection<Vegetable>();
Vegetables.Add(new Vegetable { Price = 10,
Name = "Yarrow", Category = "Leafy and Salad" });
Vegetables.Add(new Vegetable { Price = 20,
Name = "Cabbage", Category = "Leafy and Salad" });
Vegetables.Add(new Vegetable { Price = 30,
Name = "Horse gram", Category = "Beans" });
Vegetables.Add(new Vegetable { Price = 20,
Name = "Green bean", Category = "Beans" });
Vegetables.Add(new Vegetable { Price = 10,
Name = "Onion", Category = "Bulb and Stem" });
Vegetables.Add(new Vegetable { Price = 30,
Name = "Nopal", Category = "Bulb and Stem" });
//Adding a selected vegetable
```

```
CheckedVegetables = new ObservableCollection<object>();
CheckedVegetables.Add(Vegetables[0]);
CheckedVegetables.Add(Vegetables[2]);
CheckedVegetables.Add(Vegetables[4]);
}
```

**XML**

```
<syncfusion:CheckListBox ItemsSource="{Binding Vegetables}"
SelectedItems="{Binding CheckedVegetables}"
DisplayMemberPath="Name"
Margin="30"
Name="checkListBox">
<syncfusion:CheckListBox.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:CheckListBox.DataContext>
</syncfusion:CheckListBox>
```



Click [here](#) to download the sample that showcases check the items by using the collection.

*Check items using Property*

We can change the item's checked state by using the [CheckListBoxItem.IsChecked](#) property bounded with any boolean property. There is limitation when using this approach. During virtualization, views will not be loaded, so it does not know the bounded value of [CheckListBoxItem.IsChecked](#) property.

Hence, [SelectedItems](#) will not be in synchronized. However we can turnoff virtualization if you want both selected items to be in sync with bounded boolean property.

**C#**

```
//Model.cs
public class GroupItem {
public string Name { get; set; }
public string GroupName { get; set; }
public bool IsChecked { get; set; }
}

//ViewModel.cs
public class ViewModel : NotificationObject {
private ObservableCollection<GroupItem> virtualCollection;
private ObservableCollection<GroupDescription> groupDescriptions;
public ObservableCollection<GroupItem> VirtualCollection {
```

```

get {
    return virtualCollection;
}
set {
    virtualCollection = value;
    RaisePropertyChanged("VirtualCollection");
}
}
public ObservableCollection<GroupDescription> GroupDescriptions {
    get {
        return groupDescriptions;
    }
    set {
        groupDescriptions = value;
        RaisePropertyChanged("GroupDescriptions");
    }
}
public ViewModel() {
    GroupDescriptions = new ObservableCollection<GroupDescription>();
    VirtualCollection = new ObservableCollection<GroupItem>();
    for (int i = 0; i < 1000; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            GroupItem myitem = new GroupItem()
            {
                Name = "Module " + i.ToString(),
                GroupName = "Group" + j.ToString()
            };
            if (i % 2 == 0)
            {
                //Define a checked state for items
                myitem.IsChecked = true;
            }
            VirtualCollection.Add(myitem);
        }
    }
}
}

```

**XML**

```

<Window.Resources>
    <!-- Define view model with group description -->
    <local:ViewModel x:Key="viewModel">
        <local:ViewModel.GroupDescriptions>
            <!-- Define group details -->
            <PropertyGroupDescription PropertyName="GroupName" />
        </local:ViewModel.GroupDescriptions>
    </local:ViewModel>
</Window.Resources>
<Grid>
    <syncfusion:CheckListBox ItemsSource="{Binding VirtualCollection}"
        GroupDescriptions="{Binding GroupDescriptions}"
        DataContext="{StaticResource viewModel}"
        DisplayMemberPath="Name"

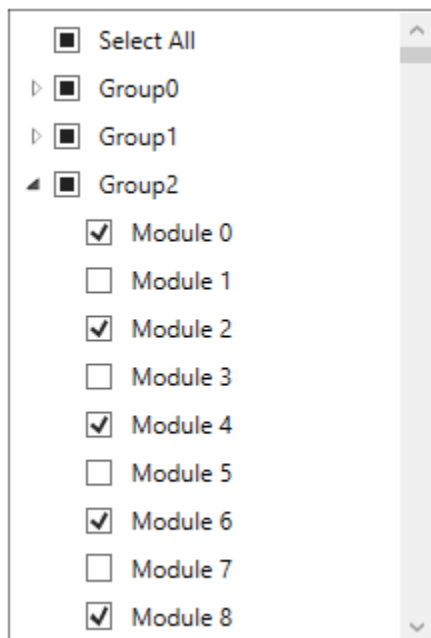
```



```

Name="checkListBox"
Margin="20">
<!--Binding the IsChecked property from ViewModel-->
<syncfusion:CheckListBox.ItemContainerStyle>
<Style TargetType="syncfusion:CheckListBoxItem">
<Setter Property="IsChecked" Value="{Binding IsChecked}"/>
</Style>
</syncfusion:CheckListBox.ItemContainerStyle>
<!--Disable the Virtualization to update the checked item-->
<syncfusion:CheckListBox.ItemsPanel>
<ItemsPanelTemplate>
<StackPanel></StackPanel>
</ItemsPanelTemplate>
</syncfusion:CheckListBox.ItemsPanel>
</syncfusion:CheckListBox>
</Grid>

```



Click [here](#) to download the sample that showcases check the items by using the `IsChecked` property.

### Check items using Mouse

The `CheckListBox` items can be checked or unchecked in a single click either by clicking the `CheckBox` or clicking the content of the item. We can check or uncheck a `GroupHeader` or `SelectAll` item to check or uncheck a group of items or all items. By default, the items are checked or unchecked by the single mouse click. If we want to check or uncheck the items only on mouse double click use the `IsCheckOnFirstClick` property as `false`.

### C#

```

//ViewModel.cs
class ViewModel {
public ObservableCollection<string> DaysCollection { get; set; }
public ViewModel() {
//Days added in the collection

```

```

DaysCollection = new ObservableCollection<string>();
DaysCollection.Add("Sunday");
DaysCollection.Add("Monday");
DaysCollection.Add("Tuesday");
DaysCollection.Add("Wednesday");
DaysCollection.Add("Thursday");
DaysCollection.Add("Friday");
DaysCollection.Add("Saturday");
}
}

```

```
<syncfusion:CheckListBox IsCheckOnFirstClick="False"
```

```
ItemsSource="{Binding DaysCollection}"
```

```
Name="checkListBox">
```

```
<syncfusion:CheckListBox.DataContext>
```

```
<local:ViewModel></local:ViewModel>
```

```
</syncfusion:CheckListBox.DataContext>
```

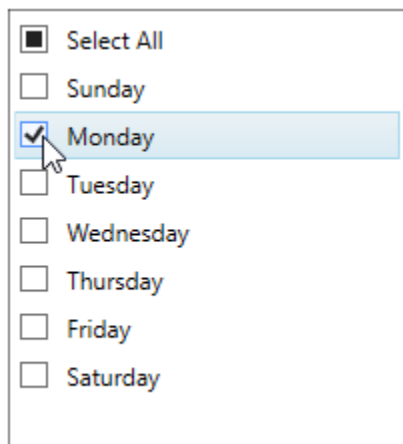
```
</syncfusion:CheckListBox>
```

### C#

```

CheckListBox checkListBox = new CheckListBox();
checkListBox.DataContext = new ViewModel();
checkListBox.ItemsSource = (checkListBox.DataContext as
ViewModel).DaysCollection;

```



### Check items using Keyboard

**Space bar** key allows the users to check or uncheck the selected item. By default, the selected item is checked or unchecked by the single **Space bar** key press. If we want to check or uncheck the items only on double **Space bar** key press use the **IsCheckOnFirstClick** property as **false**. We can check or uncheck a **GroupHeader** or **SelectAll** item to check or uncheck a group of items or all items

### C#

```
//ViewModel.cs
```

```
class ViewModel {  
    public ObservableCollection<string> DaysCollection { get; set; }  
    public ViewModel() {  
        //Days added in the collection  
        DaysCollection = new ObservableCollection<string>();  
        DaysCollection.Add("Sunday");  
        DaysCollection.Add("Monday");  
        DaysCollection.Add("Tuesday");  
        DaysCollection.Add("Wednesday");  
        DaysCollection.Add("Thursday");  
        DaysCollection.Add("Friday");  
        DaysCollection.Add("Saturday");  
    }  
}
```

<syncfusion:CheckListBox IsCheckOnFirstClick="False"

ItemsSource="{Binding DaysCollection}"

Name="checkListBox">

<syncfusion:CheckListBox.DataContext>

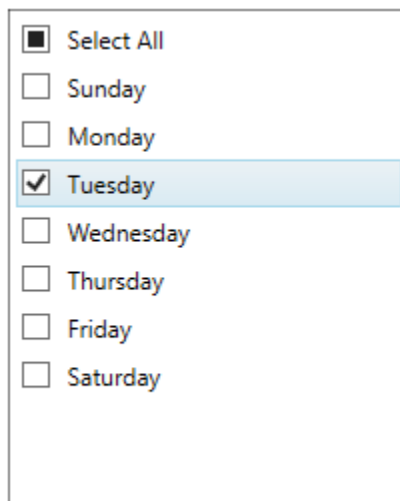
<local:ViewModel></local:ViewModel>

</syncfusion:CheckListBox.DataContext>

</syncfusion:CheckListBox>

### C#

```
CheckListBox checkListBox = new CheckListBox();  
checkListBox.DataContext = new ViewModel();  
checkListBox.ItemsSource = (checkListBox.DataContext as  
ViewModel).DaysCollection;
```



Based on the checked state of a **GroupHeader** and **SelectAll** item, their corresponding child items will be updated and vice versa.

### Checked state changed notification

When the checked state of an item is changed, it will be notified by using the [ItemChecked](#) event. You can get the details about the checked item in [ItemCheckedEventArgs](#).

#### XML

```
<!--Adding CheckListBox control -->
<syncfusion:CheckListBox ItemChecked="CheckListBox_ItemChecked"
x:Name="checkListBox">
</syncfusion:CheckListBox>
```

#### C#

```
CheckListBox checkListBox = new CheckListBox();
checkListBox.ItemChecked += CheckListBox_ItemChecked;
```

You can handle the event as follows:

#### C#

```
private void CheckListBox_ItemChecked(object sender, ItemCheckedEventArgs e)
{
    Console.WriteLine("ItemChecked event is raised");
}
```

### Selection changed notification

When the selected item is changed, it will be notified by using the [SelectionChanged](#) event. The [SelectionChangedEventArgs](#) has the following members that provides information for the [SelectionChanged](#) event:

- **Added items** : Gets a collection of the underlying data objects in which the selection has to be processed.
- **Removed items** : Gets a collection of the underlying data objects in which the selection has to be removed.

#### XML

```
<!--Adding CheckListBox control -->
<syncfusion:CheckListBox SelectionChanged = "CheckListBox_SelectionChanged"
x:Name="checkListBox">
</syncfusion:CheckListBox>
```

#### C#

```
CheckListBox checkListBox = new CheckListBox();
checkListBox.SelectionChanged += CheckListBox_SelectionChanged;
```

You can handle the event as follows:

#### C#

```
private void CheckListBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
```

```
{  
    Console.WriteLine("SelectionChanged event is raised");  
}
```

Click [here](#) to download the sample that showcases check the items by using the mouse and **Space** key.

### Virtualization in WPF CheckedListBox (CheckListBox)

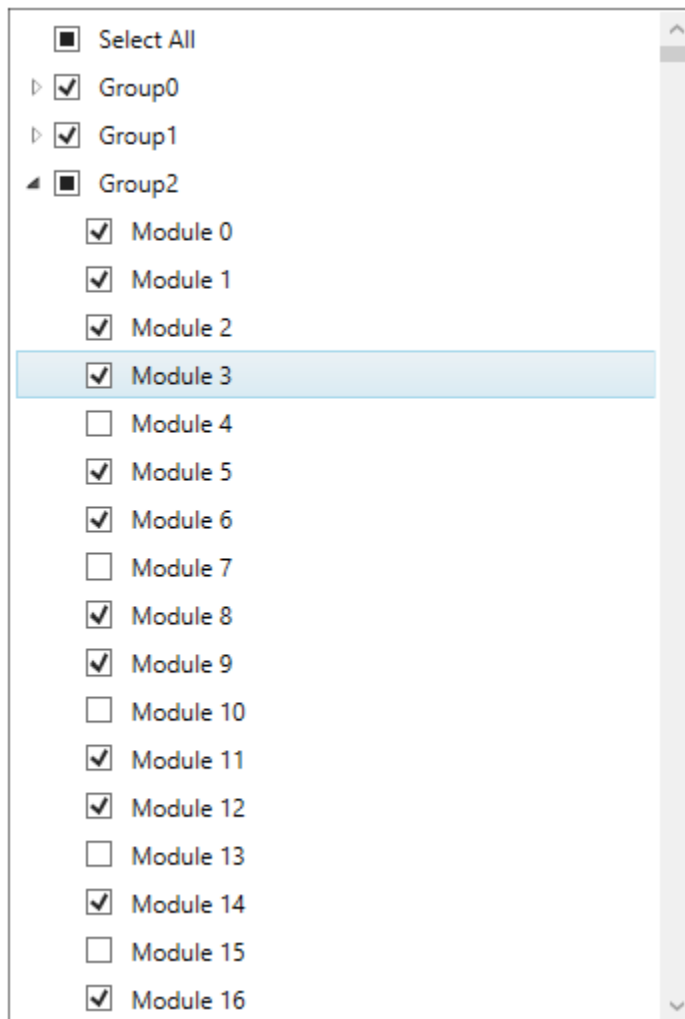
UI Virtualization support is enabled by default in [CheckListBox](#), which allows the users to load large sets of data without affecting loading or scrolling performance. This feature allows users to reduce the loading time of [CheckListBox](#) items regardless of items count.

### C#

```
//Model.cs  
public class Model {  
    public string Name { get; set; }  
    public string GroupName { get; set; }  
}  
  
//ViewModel.cs  
public class ViewModel : NotificationObject {  
    private ObservableCollection<GroupDescription> groupDescriptions;  
    private ObservableCollection<Model> collection = new  
        ObservableCollection<Model>();  
    public ObservableCollection<Model> Collection {  
        get {  
            return collection;  
        }  
        set {  
            collection = value;  
        }  
    }  
    public ObservableCollection<GroupDescription> GroupDescriptions {  
        get {  
            return groupDescriptions;  
        }  
        set {  
            groupDescriptions = value;  
            RaisePropertyChanged("GroupDescriptions");  
        }  
    }  
    public ViewModel() {  
        GroupDescriptions = new ObservableCollection<GroupDescription>();  
        //Define virtualisation items  
        Collection = new ObservableCollection<Model>();  
        for (int i = 0; i < 1000; i++) {  
            for (int j = 0; j < 10; j++) {  
                Collection.Add( new Model()  
                {  
                    Name = "Module " + i.ToString(),  
                    GroupName = "Group" + j.ToString()  
                });  
            }  
        }  
    }  
}
```

## XML

```
<Window.Resources>
<local:ViewModel x:Key="viewModel">
<local:ViewModel.GroupDescriptions>
<PropertyGroupDescription PropertyName="GroupName" />
</local:ViewModel.GroupDescriptions>
</local:ViewModel>
</Window.Resources>
<Grid>
<syncfusion:CheckListBox ItemsSource="{Binding Collection}"
GroupDescriptions="{Binding GroupDescriptions}"
DataContext="{StaticResource viewModel}"
DisplayMemberPath="Name"
Name="checkListBox"
Margin="20">
</syncfusion:CheckListBox>
</Grid>
```



Click [here](#) to download the sample that showcases the virtualization support in the CheckListBox.

### Disable the Virtualization

We can only load the particular items to visible at a time. If we want to make some items as checked that are not in view, then we need to disable the virtualization, otherwise the items will not be checked until they are in view. We can disable the virtualization by using the `ItemsPanel` template.

#### XML

```
<syncfusion:CheckListBox Name="checkListBox">
  <syncfusion:CheckListBox.ItemsPanel>
    <ItemsPanelTemplate>
      <StackPanel></StackPanel>
    </ItemsPanelTemplate>
  </syncfusion:CheckListBox.ItemsPanel>
</syncfusion:CheckListBox>
```

#### C#

```
CheckListBox checkListBox = new CheckListBox();
checkListBox.ItemsPanel = new ItemsPanelTemplate(new
FrameworkElementFactory(typeof(StackPanel)));
```

Click [here](#) to download the sample that showcases disable the virtualization.

### Appearance in WPF CheckedListBox (CheckListBox)

This section explains different UI customization, styling, theming options available in [CheckListBox](#) control.

#### Setting the Foreground

We can change the foreground color of `CheckListBox` items by setting the `Foreground` property. The default color value of `Foreground` property is `Black`.

#### XML

```
<syncfusion:CheckListBox Foreground="Red"
Name="checkListBox">
  <syncfusion:CheckListBoxItem Content="Mexico"/>
  <syncfusion:CheckListBoxItem Content="Canada" />
  <syncfusion:CheckListBoxItem Content="Bermuda" />
  <syncfusion:CheckListBoxItem Content="Belize" />
  <syncfusion:CheckListBoxItem Content="Panama" />
</syncfusion:CheckListBox>
```

#### C#

```
CheckListBox checkListBox = new CheckListBox();
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Mexico" });
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Canada" });
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Bermuda" });
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Belize" });
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Panama" });
//Setting the foreground brush
checkListBox.Foreground=Brushes.Red;
```



### Setting the Background

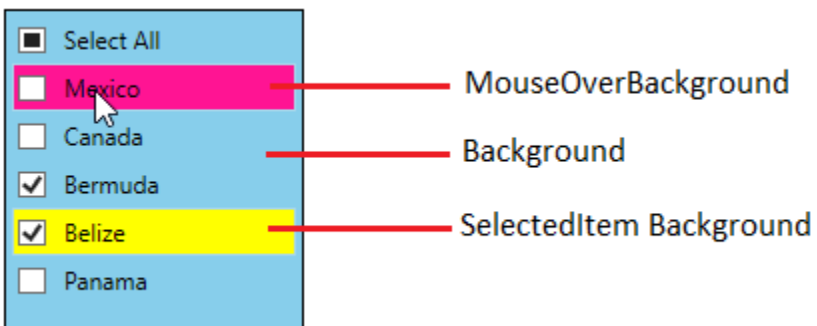
We can change the background color of **CheckListBox** items by setting the **Background** property. If we want to differentiate mouse hovered item or currently selected item with other items, we can do this by using [MouseOverBackground](#) and [SelectedItemBackground](#) properties.

### XML

```
<syncfusion:CheckListBox Background="SkyBlue"
MouseOverBackground="DeepPink"
SelectedItemBackground="Yellow"
Name="checkListBox">
<syncfusion:CheckListBoxItem Content="Mexico"/>
<syncfusion:CheckListBoxItem Content="Canada" />
<syncfusion:CheckListBoxItem Content="Bermuda" />
<syncfusion:CheckListBoxItem Content="Belize" />
<syncfusion:CheckListBoxItem Content="Panama" />
</syncfusion:CheckListBox>
```

### C#

```
CheckListBox checkListBox = new CheckListBox();
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Mexico" });
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Canada" });
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Bermuda" });
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Belize" });
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Panama" });
//Setting various background.
checkListBox.Background = Brushes.SkyBlue;
checkListBox.MouseOverBackground = Brushes.DeepPink;
checkListBox.SelectedItemBackground = Brushes.Yellow;
```





### Change flow direction

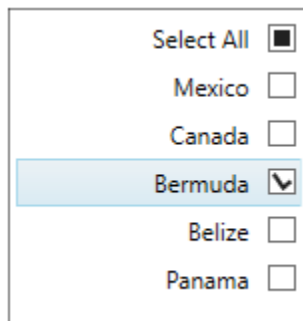
We can change the flow direction of the `CheckListBox` layout from right to left by setting the `FlowDirection` property value as `RightToLeft`. The Default value of `FlowDirection` property is `LeftToRight`.

### XML

```
<syncfusion:CheckListBox FlowDirection="RightToLeft"
Name="checkListBox">
<syncfusion:CheckListBoxItem Content="Mexico"/>
<syncfusion:CheckListBoxItem Content="Canada" />
<syncfusion:CheckListBoxItem Content="Bermuda" />
<syncfusion:CheckListBoxItem Content="Belize" />
<syncfusion:CheckListBoxItem Content="Panama" />
</syncfusion:CheckListBox>
```

### C#

```
CheckListBox checkListBox = new CheckListBox();
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Mexico" });
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Canada" });
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Bermuda" });
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Belize" });
checkListBox.Items.Add( new CheckListBoxItem() { Content = "Panama" });
//Setting flow direction as RTL
checkListBox.FlowDirection = FlowDirection.RightToLeft;
```



**Note:** [View Sample in GitHub](#)

### Item Template

We change the `DataTemplate` of the each `CheckListBox` items by using the `ItemTemplate` property.

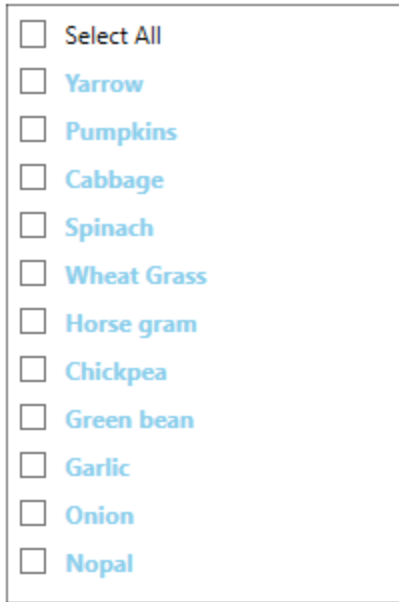
### C#

```
//Model.cs
class Vegetable {
public string Category { get; set; }
public int Price { get; set; }
public string Name { get; set; }
}
//ViewModel.cs
class ViewModel {
public ObservableCollection<Vegetable> Vegetables { get; set; }
public ViewModel() {
```

```
Vegetables = new ObservableCollection<Vegetable>();
Vegetables.Add(new Vegetable { Price=10, Name="Yarrow", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=20, Name="Pumpkins", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=30, Name="Cabbage", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=10, Name="Spinach", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=20, Name="Wheat Grass", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=30, Name="Horse gram", Category="Beans" });
Vegetables.Add(new Vegetable { Price=10, Name="Chickpea", Category="Beans" });
Vegetables.Add(new Vegetable { Price=20, Name="Green bean", Category="Beans" });
Vegetables.Add(new Vegetable { Price=30, Name="Garlic", Category="Bulb and Stem" });
Vegetables.Add(new Vegetable { Price=10, Name="Onion", Category="Bulb and Stem" });
Vegetables.Add(new Vegetable { Price=20, Name="Nopal", Category="Bulb and Stem" });
}
```

## XML

```
<syncfusion:CheckListBox x:Name="ListBox"
ItemsSource="{Binding Vegetables}">
<syncfusion:CheckListBox.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:CheckListBox.DataContext>
<!--DataTemplate for the CheckListBox items-->
<syncfusion:CheckListBox.ItemTemplate>
<DataTemplate>
<StackPanel Orientation="Horizontal">
<TextBlock FontWeight="Bold"
Foreground="SkyBlue"
Text="{Binding Name}" />
</StackPanel>
</DataTemplate>
</syncfusion:CheckListBox.ItemTemplate>
</syncfusion:CheckListBox>
```



### Group Template

We change the `DataTemplate` of the group header items by using the `GroupTemplate` property.

### C#

```
//Model.cs
class Vegetable {
public string Category { get; set; }
public int Price { get; set; }
public string Name { get; set; }
}

//ViewModel.cs
class ViewModel {
public ObservableCollection<Vegetable> Vegetables { get; set; }
public ICommand LoadedCommand { get; set; }
public void OnLoaded(object param) {
CollectionView view =
(CollectionView)CollectionViewSource.DefaultView(Vegetables);
//Adding group description
view.GroupDescriptions.Add(new PropertyGroupDescription("Category"));
}
public ViewModel() {
Vegetables = new ObservableCollection<Vegetable>();
Vegetables.Add(new Vegetable { Price=10, Name="Yarrow", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=20, Name="Pumpkins", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=30, Name="Cabbage", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=10, Name="Spinach", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=20, Name="Wheat Grass", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=30, Name="Horse gram", Category="Beans" });
});
}
```

```

Vegetables.Add(new Vegetable { Price=10, Name="Chickpea", Category="Beans"
});
Vegetables.Add(new Vegetable { Price=20, Name="Green bean", Category="Beans"
});
Vegetables.Add(new Vegetable { Price=30, Name="Garlic", Category="Bulb and
Stem" });
Vegetables.Add(new Vegetable { Price=10, Name="Onion", Category="Bulb and
Stem" });
Vegetables.Add(new Vegetable { Price=20, Name="Nopal", Category="Bulb and
Stem" });
//Initialize the checklistbox LoadedCommand
LoadedCommand = new DelegateCommand<object>(OnLoaded);
}
}

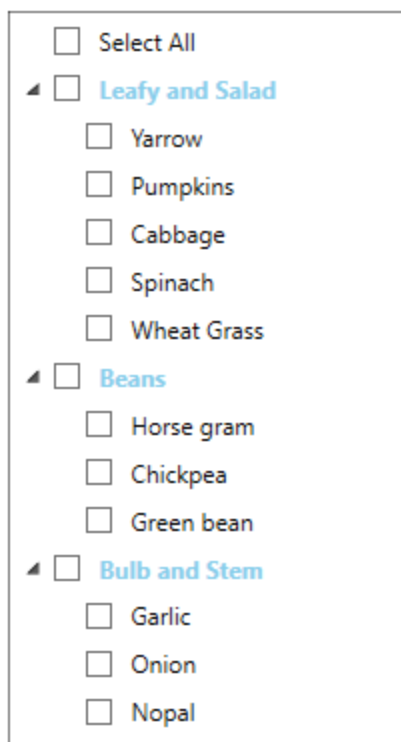
```

**XML**

```

<syncfusion:CheckListBox x:Name="ListBox"
ItemsSource="{Binding Vegetables}"
DisplayMemberPath="Name">
<syncfusion:CheckListBox.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:CheckListBox.DataContext>
<i:Interaction.Triggers>
<i:EventTrigger EventName="Loaded">
<i:InvokeCommandAction Command="{Binding LoadedCommand}" />
</i:EventTrigger>
</i:Interaction.Triggers>
<!--DataTemplate for the GroupHeader items-->
<syncfusion:CheckListBox.GroupTemplate>
<DataTemplate>
<TextBlock Text="{Binding Name}"
Foreground="Green"
FontWeight="Bold">
</TextBlock>
</DataTemplate>
</syncfusion:CheckListBox.GroupTemplate>
</syncfusion:CheckListBox>

```



### SelectAll Template

We change the `DataTemplate` of the `SelectAll` item by using the `SelectAllTemplate` property.

### C#

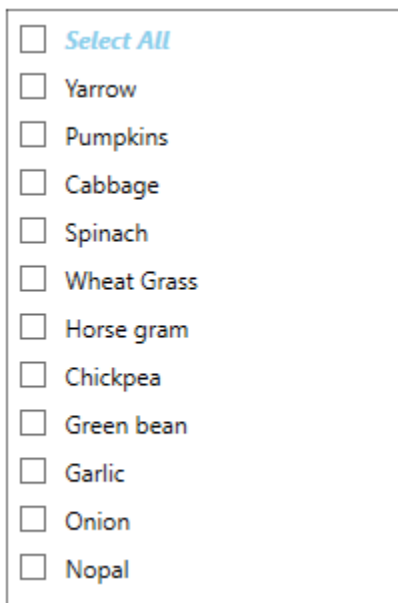
```
//Model.cs
class Vegetable {
public string Category { get; set; }
public int Price { get; set; }
public string Name { get; set; }
}

//ViewModel.cs
class ViewModel {
public ObservableCollection<Vegetable> Vegetables { get; set; }
public ViewModel() {
Vegetables = new ObservableCollection<Vegetable>();
Vegetables.Add(new Vegetable { Price=10, Name="Yarrow", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=20, Name="Pumpkins", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=30, Name="Cabbage", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=10, Name="Spinach", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=20, Name="Wheat Grass", Category="Leafy and Salad" });
Vegetables.Add(new Vegetable { Price=30, Name="Horse gram", Category="Beans" });
Vegetables.Add(new Vegetable { Price=10, Name="Chickpea", Category="Beans" });
Vegetables.Add(new Vegetable { Price=20, Name="Green bean", Category="Beans" });
};
}
```

```
Vegetables.Add(new Vegetable { Price=30, Name="Garlic", Category="Bulb and Stem" });  
Vegetables.Add(new Vegetable { Price=10, Name="Onion", Category="Bulb and Stem" });  
Vegetables.Add(new Vegetable { Price=20, Name="Nopal", Category="Bulb and Stem" });  
}  
}
```

## XML

```
<syncfusion:CheckListBox x:Name="ListBox"  
ItemsSource="{Binding Vegetables}"  
DisplayMemberPath="Name">  
  <syncfusion:CheckListBox.DataContext>  
  <local:ViewModel></local:ViewModel>  
  </syncfusion:CheckListBox.DataContext>  
  <!--DataTemplate for the SelectAll items-->  
  <syncfusion:CheckListBox.SelectAllTemplate>  
  <DataTemplate>  
  <TextBlock FontWeight="Bold"  
    Foreground="SkyBlue"  
    Text="Select All"  
    FontStyle="Italic"/>  
  </DataTemplate>  
  </syncfusion:CheckListBox.SelectAllTemplate>  
</syncfusion:CheckListBox>
```



**Note:** [View Sample in GitHub](#)

## ItemContainerStyle

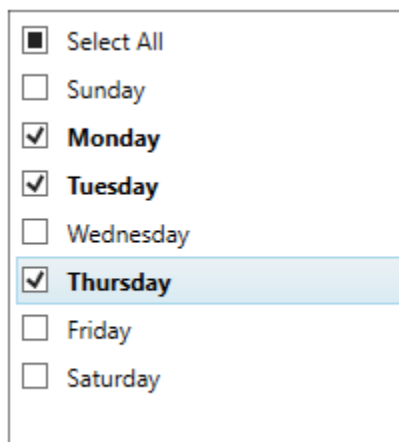
We can change the item container style of `CheckListBox` by using the `ItemContainerStyle` which is applied to the container element that generated for each item. The default value of `ItemContainerStyle` is null.

**C#**

```
class ViewModel {
    private ObservableCollection<string> daysCollection = new
    ObservableCollection<string>();
    public ObservableCollection<string> DaysCollection {
        get { return daysCollection; }
        set { daysCollection = value; }
    }
    public ViewModel() {
        //Days added in the collection
        DaysCollection.Add("Sunday");
        DaysCollection.Add("Monday");
        DaysCollection.Add("Tuesday");
        DaysCollection.Add("Wednesday");
        DaysCollection.Add("Thursday");
        DaysCollection.Add("Friday");
        DaysCollection.Add("Saturday");
    }
}
```

**XML**

```
<syncfusion:CheckListBox ItemsSource="{Binding DaysCollection}"
    x:Name="checkListBox">
    <syncfusion:CheckListBox.DataContext>
    <local:ViewModel></local:ViewModel>
    </syncfusion:CheckListBox.DataContext>
    <!--Setting ItemContainerStyle for the CheckListBoxItems-->
    <syncfusion:CheckListBox.ItemContainerStyle>
    <Style TargetType="syncfusion:CheckListBoxItem">
    <Style.Triggers>
    <Trigger Property="IsChecked" Value="True">
    <Setter Property="FontWeight" Value="Bold"></Setter>
    </Trigger>
    </Style.Triggers>
    </Style>
    </syncfusion:CheckListBox.ItemContainerStyle>
</syncfusion:CheckListBox>
```



Here, the Checked items are contains a bold font.

**Note:** [View Sample in GitHub](#)

### ItemTemplateSelector

We can change the various `DataTemplate` for the `CheckListBox` items based on the provided logic by using the `ItemTemplateSelector`.

### C#

```
//Model.cs
public class Vegetable {
    public string Category { get; set; }
    public int Price { get; set; }
    public string Name { get; set; }
}

//ViewModel.cs
class ViewModel {
    public ObservableCollection<Vegetable> Vegetables { get; set; }
    public ICommand LoadedCommand { get; set; }
    public void OnLoaded(object param) {
        CollectionView view =
            (CollectionView)CollectionViewSource.GetDefaultView(Vegetables);
        //Adding group description
        view.GroupDescriptions.Add(new PropertyGroupDescription("Category"));
    }

    public ViewModel() {
        Vegetables = new ObservableCollection<Vegetable>();
        Vegetables.Add(new Vegetable { Price=10, Name="Yarrow", Category="Leafy and Salad" });
        Vegetables.Add(new Vegetable { Price=20, Name="Pumpkins", Category="Leafy and Salad" });
        Vegetables.Add(new Vegetable { Price=30, Name="Cabbage", Category="Leafy and Salad" });
        Vegetables.Add(new Vegetable { Price=10, Name="Spinach", Category="Leafy and Salad" });
        Vegetables.Add(new Vegetable { Price=20, Name="Wheat Grass", Category="Leafy and Salad" });
        Vegetables.Add(new Vegetable { Price=30, Name="Horse gram", Category="Beans" });
        Vegetables.Add(new Vegetable { Price=10, Name="Chickpea", Category="Beans" });
        Vegetables.Add(new Vegetable { Price=20, Name="Green bean", Category="Beans" });
        Vegetables.Add(new Vegetable { Price=30, Name="Garlic", Category="Bulb and Stem" });
        Vegetables.Add(new Vegetable { Price=10, Name="Onion", Category="Bulb and Stem" });
        Vegetables.Add(new Vegetable { Price=20, Name="Nopal", Category="Bulb and Stem" });
        //Initialize the checklistbox LoadedCommand
        LoadedCommand = new DelegateCommand<object>(OnLoaded);
    }
}

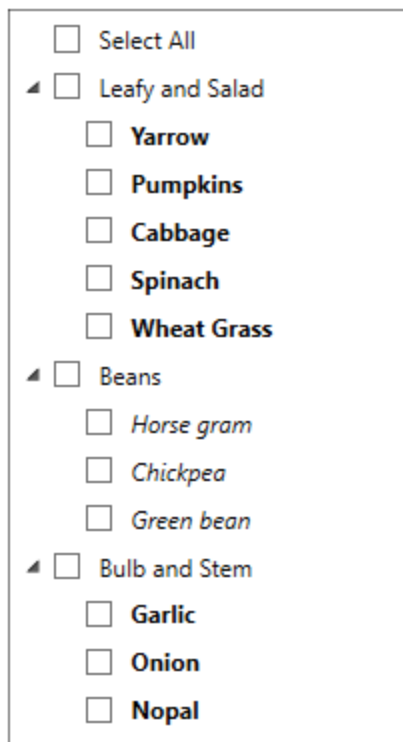
//ItemTemplateSelector class for select a DataTemplate
public class MyTemplateSelector : DataTemplateSelector {
    public DataTemplate Template { get; set; }
}
```



```
public DataTemplate itemTemplate { get; set; }
public override DataTemplate SelectTemplate(object item, DependencyObject
container) {
    if (item is Vegetable && (item as Vegetable).Category == "Beans")
        return itemTemplate;
    else
        return Template;
}
```

## XML

```
<Window.Resources>
<local:MyTemplateSelector x:Key="Mytemplate">
<local:MyTemplateSelector.Template>
<DataTemplate>
<TextBlock Text="{Binding Name}" FontWeight="Bold"></TextBlock>
</DataTemplate>
</local:MyTemplateSelector.Template>
<local:MyTemplateSelector.itemTemplate>
<DataTemplate>
<TextBlock Text="{Binding Name}" FontStyle="Italic"></TextBlock>
</DataTemplate>
</local:MyTemplateSelector.itemTemplate>
</local:MyTemplateSelector>
</Window.Resources>
<syncfusion:CheckListBox ItemTemplateSelector="{StaticResource Mytemplate}"
ItemsSource="{Binding Vegetables}"
Name="checkListBox">
<syncfusion:CheckListBox.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:CheckListBox.DataContext>
<i:Interaction.Triggers>
<i:EventTrigger EventName="Loaded">
<i:InvokeCommandAction Command="{Binding LoadedCommand}" />
</i:EventTrigger>
</i:Interaction.Triggers>
</syncfusion:CheckListBox>
```



Here, the Leaf and Salad and Bulb and Stem group items have same data template and the Beans group items have separate data template.

**Note:** [View Sample in GitHub](#)

#### ItemContainerStyleSelection

We can choose the style for the CheckListBox items based on the provided logic by using the ItemContainerStyleSelector.

#### C#

```
//Model.cs
public class Vegetable {
    public string Category { get; set; }
    public int Price { get; set; }
    public string Name { get; set; }
}

//ViewModel.cs
class ViewModel {
    public ObservableCollection<Vegetable> Vegetables { get; set; }
    public ICommand LoadedCommand { get; set; }
    public void OnLoaded(object param) {
        CollectionView view =
            (CollectionView)CollectionViewSource.GetDefaultView(Vegetables);
        //Adding group description
        view.GroupDescriptions.Add(new PropertyGroupDescription("Category"));
    }
    public ViewModel() {
        Vegetables = new ObservableCollection<Vegetable>();
        Vegetables.Add(new Vegetable { Price=10, Name="Yarrow", Category="Leafy and Salad" });
    }
}
```

```

Vegetables.Add(new Vegetable { Price=20, Name="Pumpkins", Category="Leafy
and Salad" });
Vegetables.Add(new Vegetable { Price=30, Name="Cabbage", Category="Leafy and
Salad" });
Vegetables.Add(new Vegetable { Price=10, Name="Spinach", Category="Leafy and
Salad" });
Vegetables.Add(new Vegetable { Price=20, Name="Wheat Grass", Category="Leafy
and Salad" });
Vegetables.Add(new Vegetable { Price=30, Name="Horse gram", Category="Beans"
});
Vegetables.Add(new Vegetable { Price=10, Name="Chickpea", Category="Beans"
});
Vegetables.Add(new Vegetable { Price=20, Name="Green bean", Category="Beans"
});
Vegetables.Add(new Vegetable { Price=30, Name="Garlic", Category="Bulb and
Stem" });
Vegetables.Add(new Vegetable { Price=10, Name="Onion", Category="Bulb and
Stem" });
Vegetables.Add(new Vegetable { Price=20, Name="Nopal", Category="Bulb and
Stem" });
//Initialize the checklistbox LoadedCommand
LoadedCommand = new DelegateCommand<object>(OnLoaded);
}
}
// A class that choose style for for the items
public class VegetableStyleSelector : StyleSelector {
public Style GroupStyle { get; set; }
public Style ItemStyle { get; set; }
public override Style SelectStyle(object item, DependencyObject container) {
if (item is Vegetable)
return ItemStyle;
else
return GroupStyle;
}
}

```

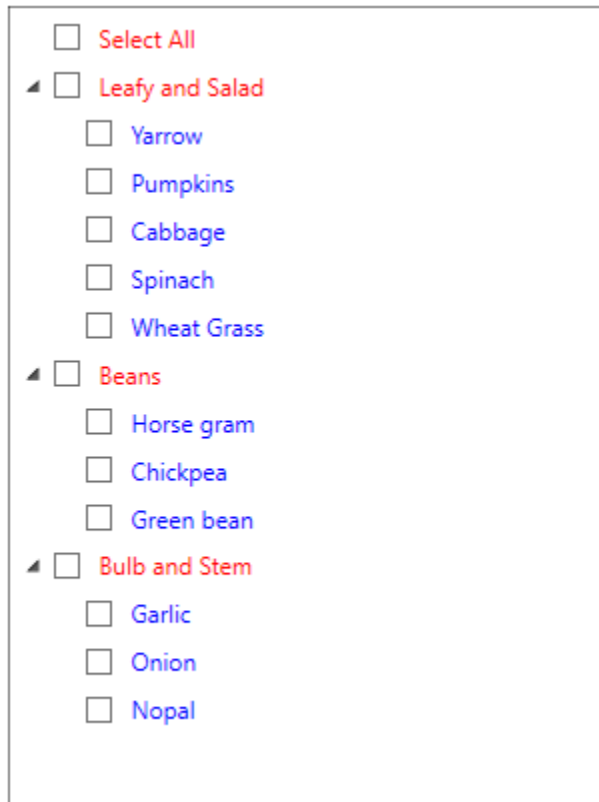
## XML

```

<Window.Resources>
<Style TargetType="syncfusion:CheckListBoxItem" x:Key="Groupstyle">
<Setter Property="Foreground" Value="Red"></Setter>
</Style>
<Style TargetType="syncfusion:CheckListBoxItem" x:Key="ItemStyle">
<Setter Property="Foreground" Value="Blue"></Setter>
</Style>
<local:VegetableStyleSelector x:Key="StyleSelector"
GroupStyle="{StaticResource Groupstyle}"
ItemStyle="{StaticResource ItemStyle}">
</local:VegetableStyleSelector>
</Window.Resources>
<syncfusion:CheckListBox ItemContainerStyleSelector="{StaticResource
StyleSelector}"
ItemsSource="{Binding Vegetables}"
DisplayMemberPath="Name"
Name="checkListBox"
Margin="20">

```

```
<syncfusion:CheckListBox.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:CheckListBox.DataContext>
<i:Interaction.Triggers>
<i:EventTrigger EventName="Loaded">
<i:InvokeCommandAction Command="{Binding LoadedCommand}" />
</i:EventTrigger>
</i:Interaction.Triggers>
</syncfusion:CheckListBox>
```



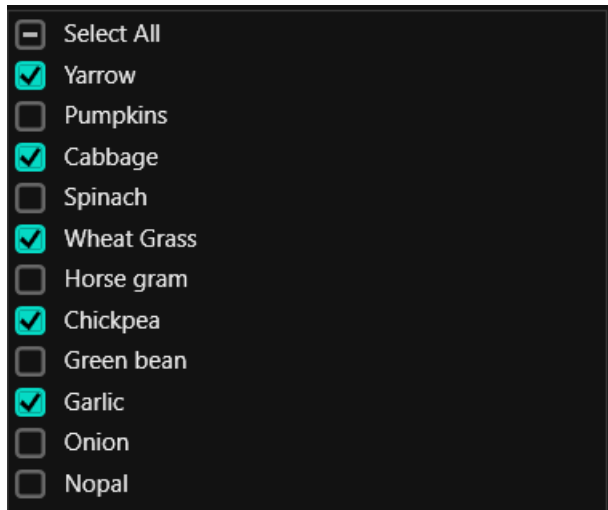
Here, the `GroupStyle` is applied to the group header and the `ItemStyle` is applied to the child items.

**Note:** [View Sample in GitHub](#)

### Theme

CheckListBox supports various built-in themes. Refer to the below links to apply themes for the CheckListBox,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



## How to

### Check the Item in CheckListBox when Initiating

To check the items when initiating the CheckListBox control, items need to be added in the SelectedItems collection. The following code illustrates this:

#### C#

```
this.ListBox.SelectedItems.Add(this.ListBox.Items[4]);
```

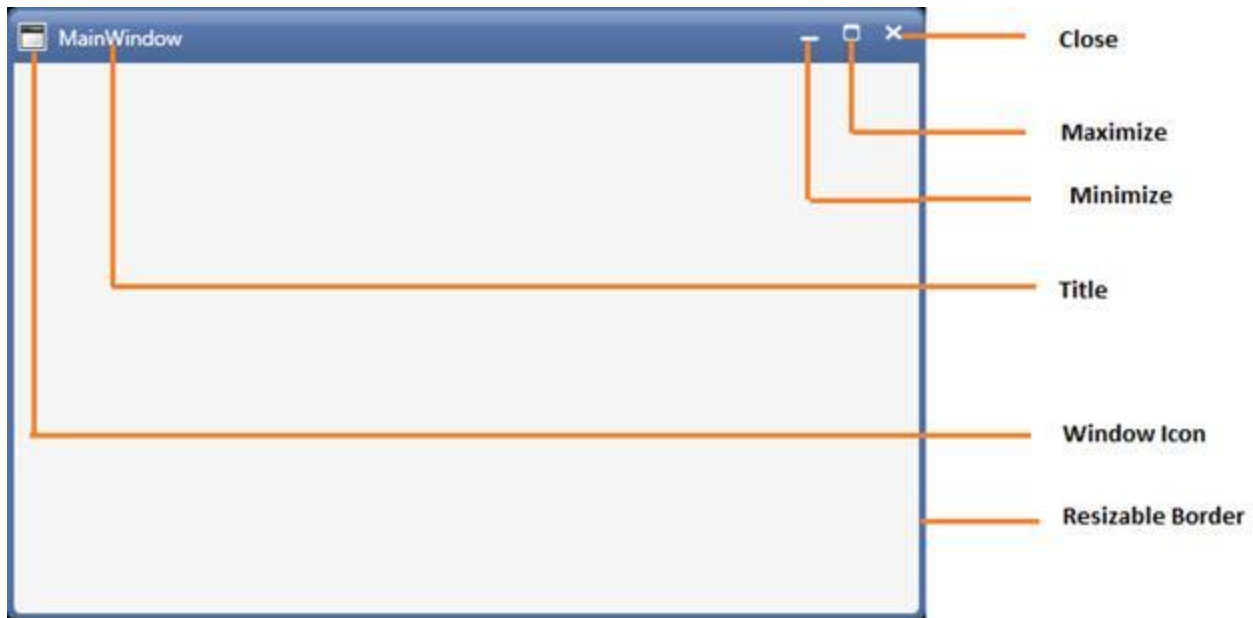
## ChromelessWindow

### WPF Chromeless Window Overview

ChromelessWindow is used to create customizable window for the end user's applications. With these windows, user can have the complete control over the entire window. The features offered includes resizing, dragging and moving the window. It implements several other features to present an appealing User Interface to the end users. It also supports various built in skins and let the user to control its behavior and appearance. The features are described in detail in the further topics

### Visual Structure in WPF Chromeless Window

The ChromelessWindow's visual elements are explained in the below snapshot



## Getting Started with WPF Chromeless Window

This section explains how to implement the ChromelessWindow control. It describes the following:

### Assembly deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as reference to use the control in any application.

You can find more details about installing the NuGet packages in a WPF application in the following link:

[How to install nuget packages](#)

### Creating simple application with ChromelessWindow

You can create a WPF application with ChromelessWindow using the following steps:

1. [Create a project.](#)
2. [Add ChromelessWindow.](#)
3. [Customize title bar.](#)
4. [Customize title bar background.](#)
5. [Customize title bar font.](#)
6. [Customize title bar height.](#)
7. [Customize title bar icon.](#)
8. [Customize the border of the ChromelessWindow.](#)

### Creating the project

Create a new WPF project in Visual Studio to display chromeless window.

### Add ChromelessWindow

The following steps help you add ChromelessWindow:

1. Create a WPF project, and refer the following assemblies: \* Syncfusion.Shared.WPF.dll.
2. Include an XML namespace for the above assemblies to the Main window.

### XML

```
<Window x:Class="Chromelesswindow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525"
xmlns:syncfusion="clr-namespace:Syncfusion.Windows.Shared;assembly=Syncfusion.Shared.WPF">
</Window>
```

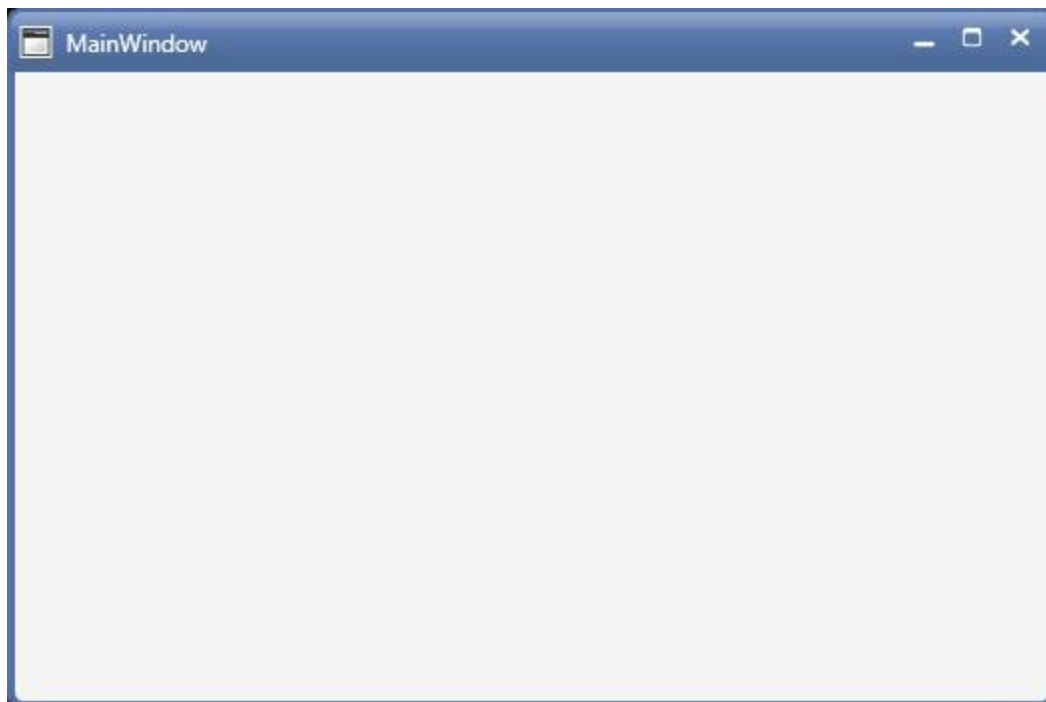
3. Change the Window to ChromelessWindow.

```
<syncfusion:ChromelessWindow x:Class="Chromelesswindow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525"
xmlns:syncfusion="clr-namespace:Syncfusion.Windows.Shared;assembly=Syncfusion.Shared.WPF">
</syncfusion:ChromelessWindow>
```

4. Add the **Syncfusion.Windows.Shared** namespace, and inherit MainWindow from ChromelessWindow in code behind.

#### C#

```
using Syncfusion.Windows.Shared;
public partial class MainWindow : ChromelessWindow
{
    public MainWindow()
    {
        InitializeComponent();
    }
}
```



Customizing title bar

Title bar background

You can customize the background of the title bar by setting the [TitleBarBackground](#) property of ChromelessWindow.

## XML

```
<syncfusion:ChromelessWindow
x:Class="WPF_CalendarEdit.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
TitleBarBackground="Red"
syncfusion:SkinStorage.VisualStudio="Metro"
Title="ChromelessWindow Sample" Height="350" Width="525">
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```



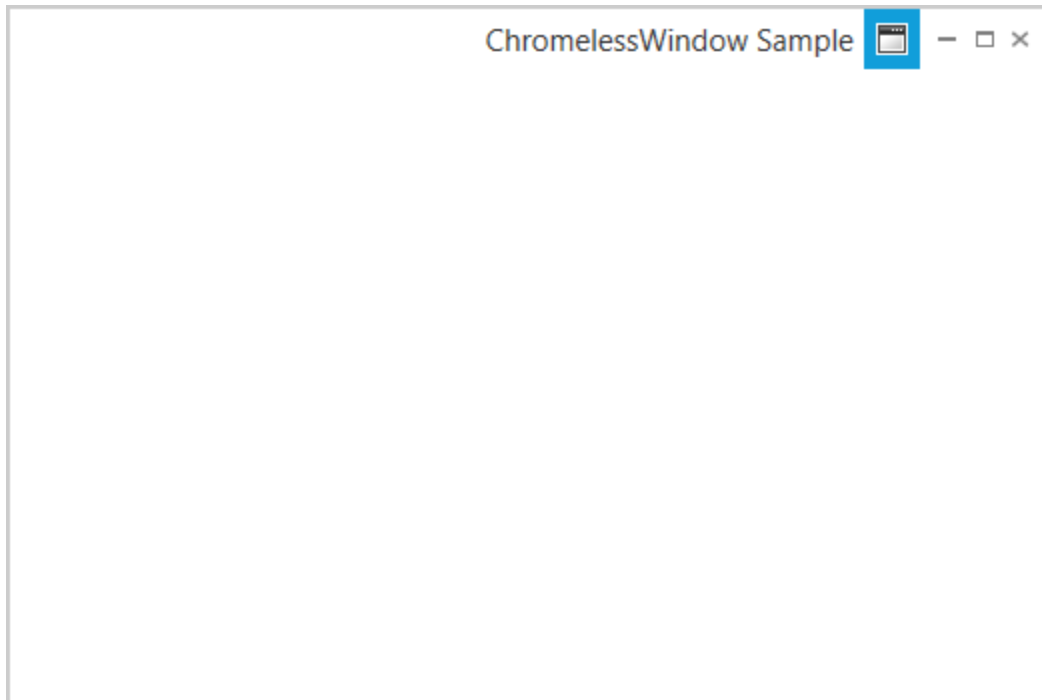
Title bar font

The font of the caption in the title bar can be customized using the [TitleFontSize](#) property.

## XML

```
<syncfusion:ChromelessWindow
x:Class="WPF_CalendarEdit.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
TitleFontSize="15"
syncfusion:SkinStorage.VisualStudio="Metro"
Title="ChromelessWindow Sample" Height="350" Width="525">
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```



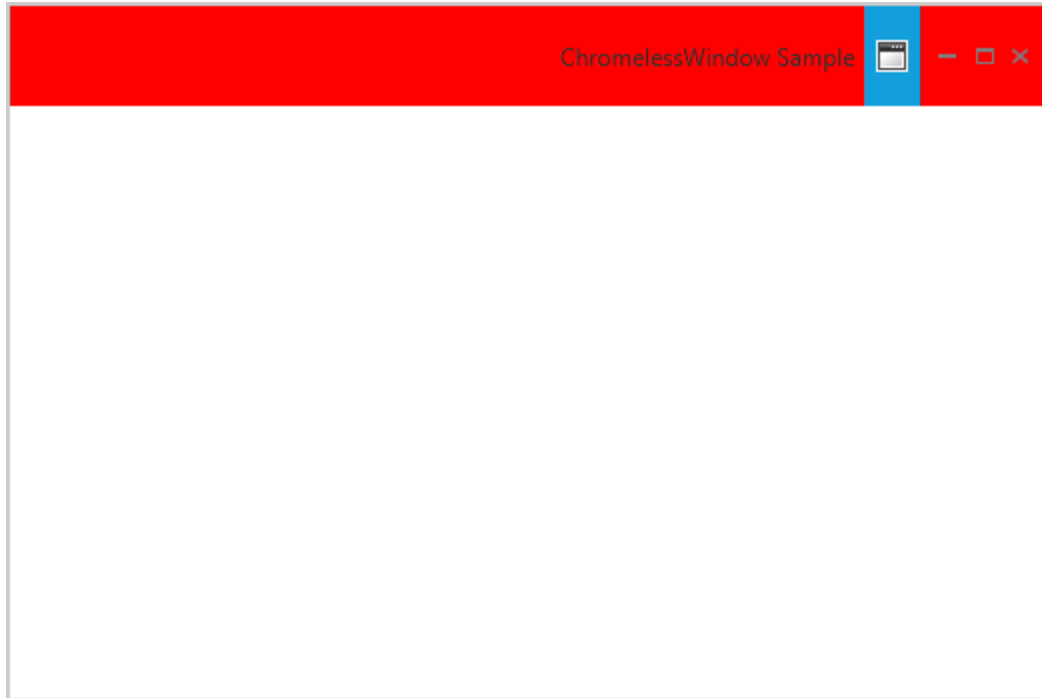


### Title bar height

You can customize the caption height by setting the [TitleBarHeight](#) of ChromelessWindow.

### XML

```
<syncfusion:ChromelessWindow
x:Class="WPF_CalendarEdit.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:local="clr-namespace:WPF_CalendarEdit"
TitleBarHeight="50"
syncfusion:SkinStorage.VisualStudio="Metro"
Title="ChromelessWindow Sample" Height="350" Width="525">
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```



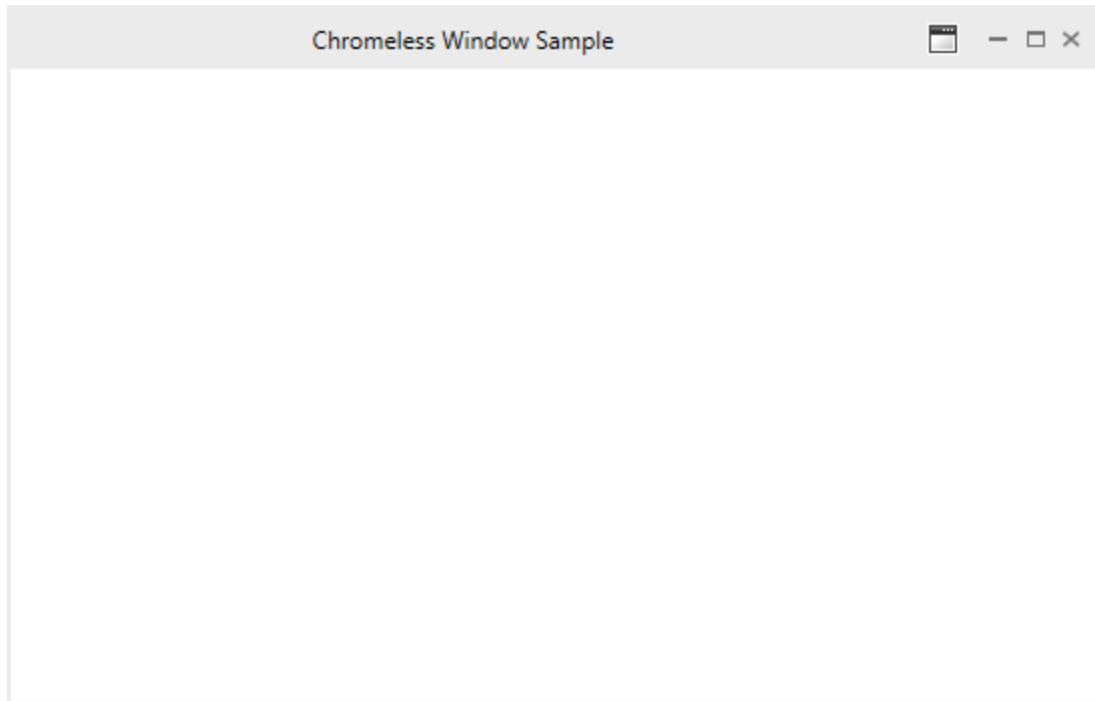
#### Title bar text alignment

The text alignment of the title can be set using the [TitleTextAlignment](#) property in the Chromeless Window. The various options of the `TitleTextAlignment` are listed below.

- Left
- Right
- Center
- Stretch

#### XML

```
<syncfusion:ChromelessWindow x:Class="Chromeless_Window_Sample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:Chromeless_Window_Sample"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="Office2016Colorful"
TitleTextAlignment="Center"
Title="Chromeless Window Sample"
Height="350" Width="550">
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```

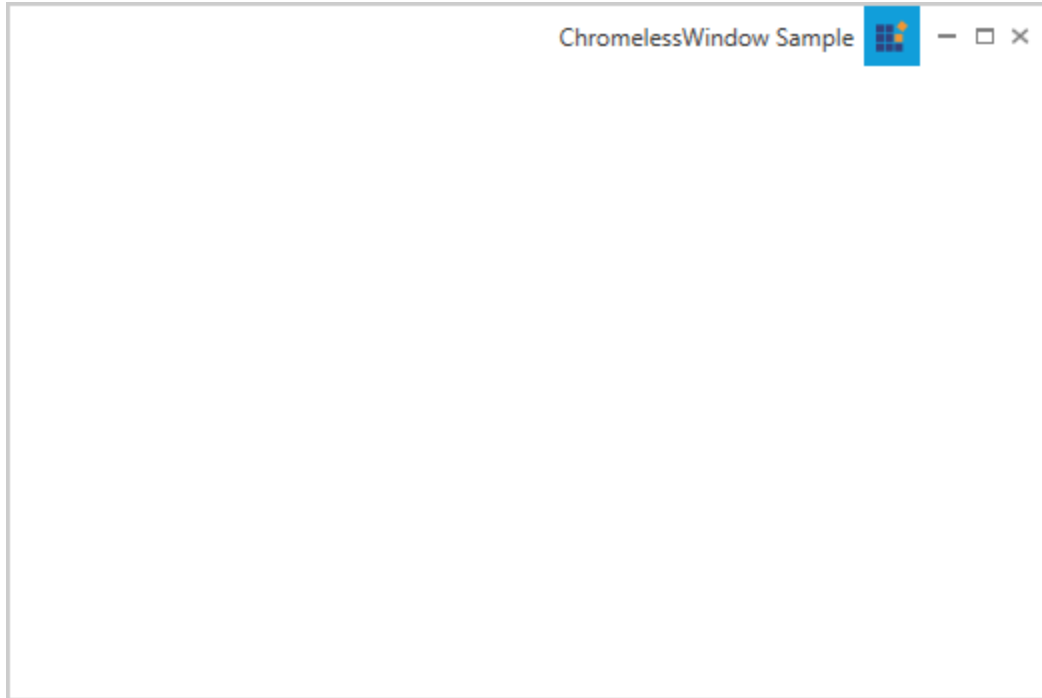


Title bar icon

You can set the caption icon by setting the [Icon](#) property.

#### XML

```
<syncfusion:ChromelessWindow
x:Class="WPF_CalendarEdit.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Icon="App.ico"
syncfusion:SkinStorage.VisualStudio="Metro"
Title="ChromelessWindow Sample" Height="350" Width="525">
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```



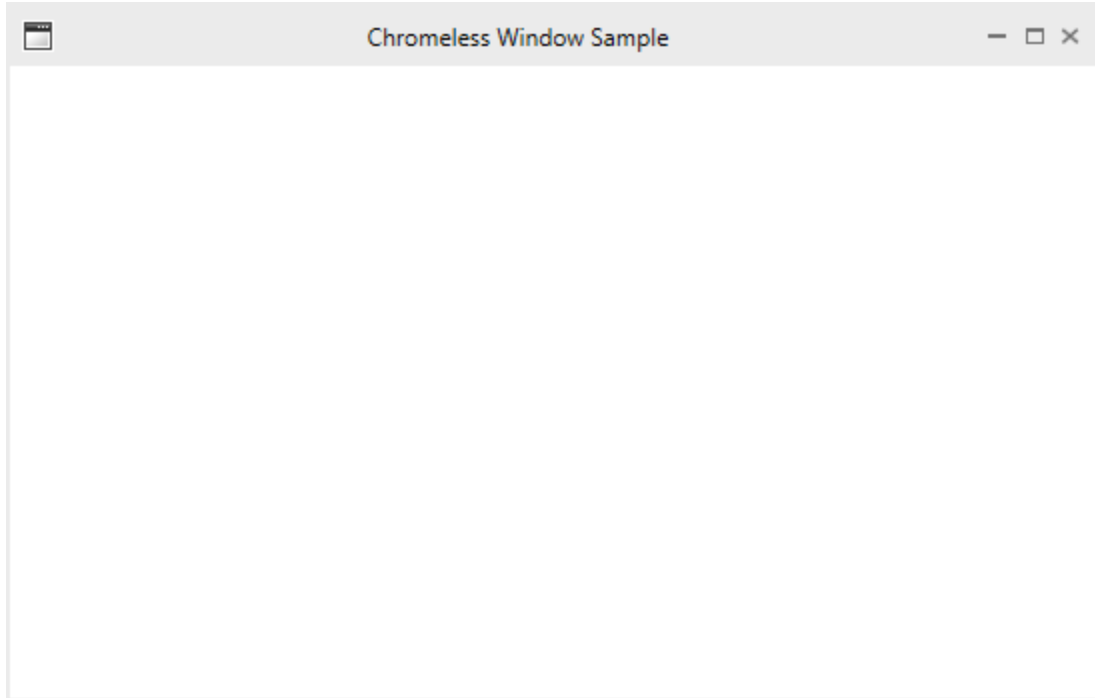
#### Title bar icon alignment

The title bar icon can be aligned to **Left** or **Right** using the [IconAlignment](#) property in the Chromeless Window. Its various options are listed below.

- Left
- Right

#### XML

```
<syncfusion:ChromelessWindow x:Class="Chromeless_Window_Sample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:Chromeless_Window_Sample"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="Office2016Colorful"
TitleTextAlignment="Center"
Title="Chromeless Window Sample" IconAlignment="Left"
Height="350" Width="550">
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```

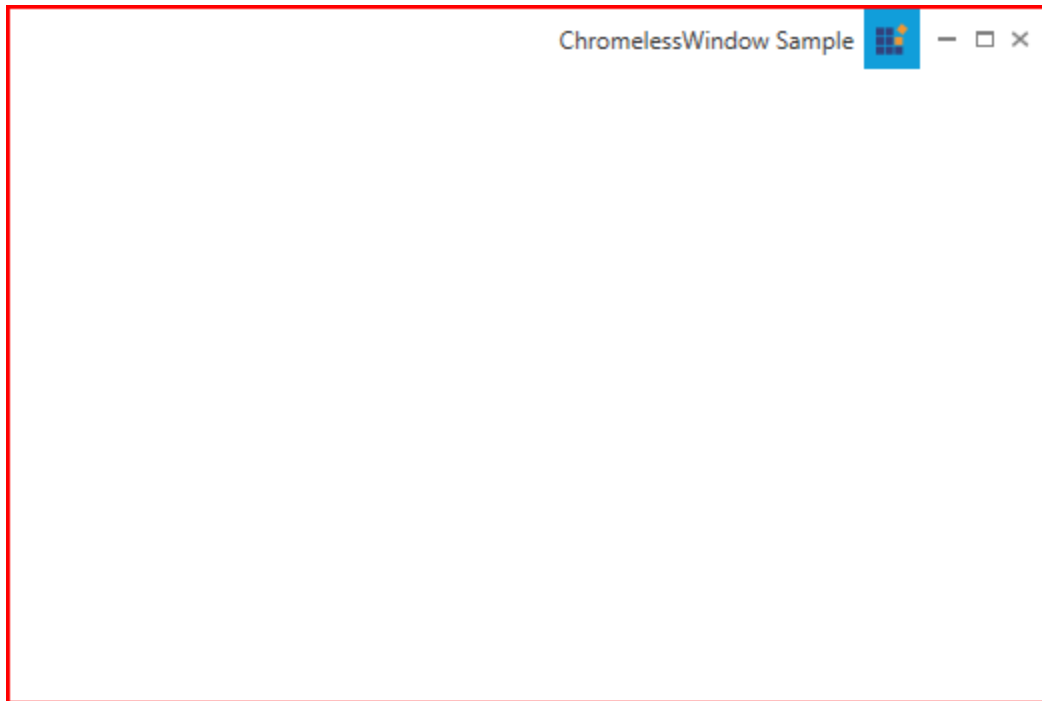


### Customizing the border of ChromelessWindow

You can change the border color of the chromeless window by setting the [ResizeBorderBrush](#) property.

### XML

```
<syncfusion:ChromelessWindow
x:Class="WpfApplication2.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
syncfusion:SkinStorage.VisualStudio="Metro"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
ResizeBorderBrush="Red"
Title="ChromelessWindow Sample" Height="350" Width="525">
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```



### Theme

ChromelessWindow supports various built-in themes. Refer to the below links to apply themes for the ChromelessWindow,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



## Set Visual Styles in WPF Chromeless Window

### Theme

ChromelessWindow supports various built-in themes. Refer to the below links to apply themes for the ChromelessWindow,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Corner Radius in WPF Chromeless Window

Corner Radius indicates the degree to which the corners of the border can be rounded. To create curved borders for the windows, use **CornerRadius** property of the ChromelessWindow.

The default value is zero, which implies sharp corners

#### XML

```
<syncfusion:ChromelessWindow x:Class="Chromelesswindow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="ChromelessWindow" Height="350" Width="525" CornerRadius="8"
AllowsTransparency="True"
syncfusion:SkinStorage.VisualStudio="Metro" x:Name="_chromelessWindow"
xmlns:syncfusion="clr-namespace:Syncfusion.Windows.Shared;assembly=Syncfusion.Shared.WPF" >
</syncfusion:ChromelessWindow>
```

#### C#

```
_chromelessWindow.CnerRadius = new CornerRadius(8);
```

#### VB.NET



```
chromelessWindow.CnerRadius = New CornerRadius(8)
```



### UseNativeChrome in WPF Chromeless Window

Windows can be arranged side by side by using **UseNativeChrome** property, which is very helpful while moving and comparing the windows. To enable this docking behavior, set **UseNativeChrome** property to **True**. This property allows the **ChromelessWindow** to behave like the MS Window. Windows can be docked to the left and right side of the Window to resize them to half of the screen. Also it supports the window maximization when docked at the top of the screen.

To set this property, use the below code.

#### XML

```
<syncfusion:ChromelessWindow x:Class="Chromelesswindow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="ChromelessWindow" Height="350" Width="525" UseNativeChrome="True"
syncfusion:SkinStorage.VisualStudio="Metro"
x:Name="_chromelessWindow" xmlns:syncfusion="clr-
namespace:Syncfusion.Windows.Shared;assembly=Syncfusion.Shared.WPF" >
</syncfusion:ChromelessWindow>
```

#### C#

```
_chromelessWindow.UseNativeChrome = true;
```

#### VB.NET

```
_chromelessWindow.UseNativeChrome = True
```

## Title bar Customization in WPF Chromeless Window

### Customizing the background

The [TitleBarBackground](#) property can be used to set the background for the Title bar.

#### XML

```
<syncfusion:ChromelessWindow x:Class="Chromelesswindow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525" TitleBarBackground="Red"
x:Name="_chromelessWindow"
syncfusion:SkinStorage.VisualStudio="Metro"
xmlns:syncfusion="clr-namespace:Syncfusion.Windows.Shared;assembly=Syncfusion.Shared.WPF" >
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```

#### C#

```
this.TitleBarBackground = new SolidColorBrush(Colors.Red);
```

#### VB.NET

```
Me.TitleBarBackground = New SolidColorBrush(Colors.Red)
```

The following screen shots illustrate the title bar background changes.



### Customizing the height

The [TitleBarHeight](#) property can be used to change the height of the Title bar. The default value is 30.

### XML

```
<syncfusion:ChromelessWindow x:Class="Chromelesswindow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="ChromelessWindow" Height="350" Width="525" TitleBarHeight="60"
TitleBarBackground="Pink"
syncfusion:SkinStorage.VisualStudio="Metro" x:Name="_chromelessWindow"
xmlns:syncfusion="clr-namespace:Syncfusion.Windows.Shared;assembly=Syncfusion.Shared.WPF" >
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```

### C#

```
this.TitleBarHeight = 60;
```

### VB.NET

```
Me.TitleBarHeight = 60
```



### Customizing the font size

The font size of the ChromelessWindow title bar can be changed by using [TitleFontSize](#) property. The default value is 12.

### XML

```
<syncfusion:ChromelessWindow x:Class="Chromelesswindow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
Title="ChromelessWindow" Height="350" Width="525" TitleFontSize="25"
syncfusion:SkinStorage.VisualStudio="Metro"
x:Name="_chromelessWindow" xmlns:syncfusion="clr-
namespace:Syncfusion.Windows.Shared;assembly=Syncfusion.Shared.WPF" >
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```

### C#

```
this.TitleFontSize = 25;
```

### VB.NET

```
Me.TitleFontSize = 25
```



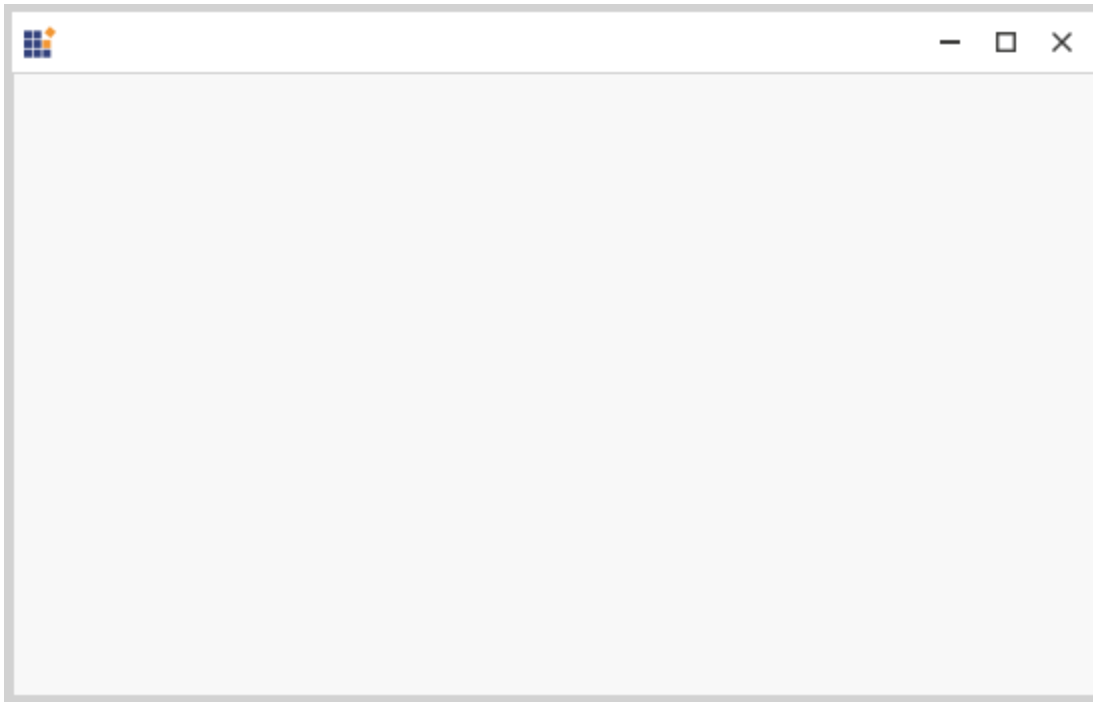
Show or hide the title bar text

The visibility of the title can be customized using the [ShowTitle](#) property. The default value of `ShowTitle` property is true.

### XML

```
<syncfusion:ChromelessWindow x:Class="Chromeless_Window_Sample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:Chromeless_Window_Sample"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
```

```
skin:SfSkinManager.VisualStudio="Office2019Colorful"
TitleTextAlignment="Center"
Icon="App.ico"
ShowTitle="False"
Title="Getting Started"
Height="350" Width="550">
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```



#### Show or hide the maximize and minimize buttons

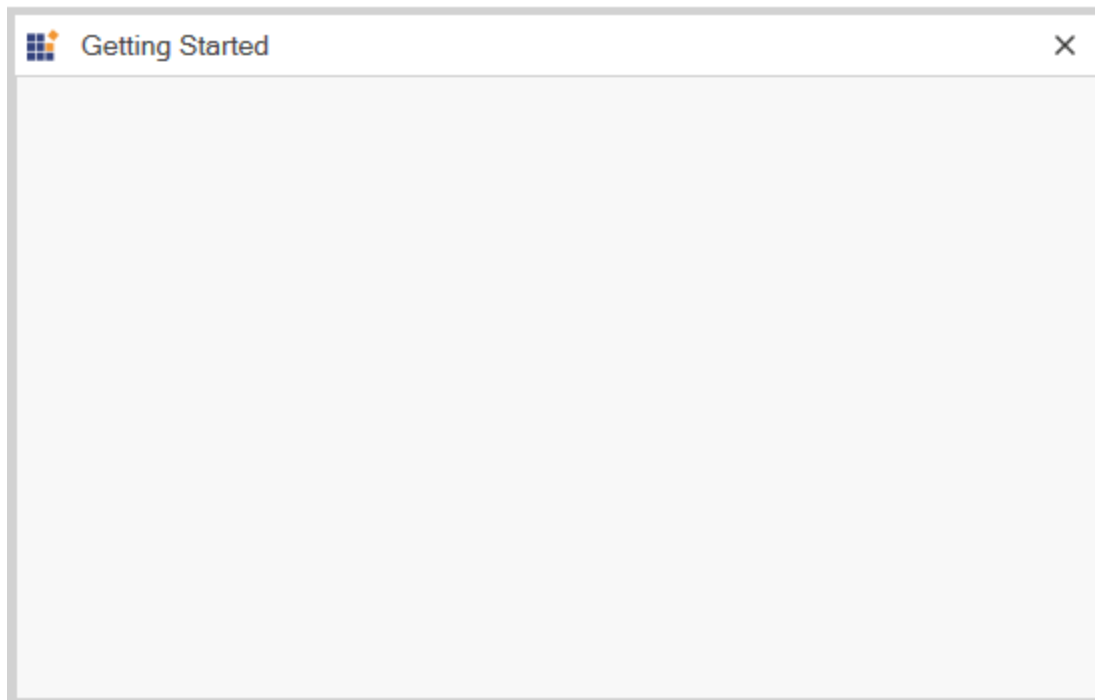
The visibility of maximize and minimize buttons can be customized using the [ShowMaximizeButton](#) and [ShowMinimizeButton](#) properties respectively. The default value of [ShowMaximizeButton](#) and [ShowMinimizeButton](#) properties are true.

**Note:** By default, the maximize and minimize buttons will not be displayed in [ResizeMode -NoResize](#) even when the value of [ShowMaximizeButton](#) and [ShowMaximizeButton](#) set as True. The maximize and minimize buttons will be displayed only in the remaining resize modes such as [CanMinimize](#), [CanResize](#), [CanResizeWithGrip](#) based on the [ShowMaximizeButton](#) and [ShowMinimizeButton](#) properties.

#### XML

```
<syncfusion:ChromelessWindow x:Class="Chromeless_Window_Sample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:Chromeless_Window_Sample"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
```

```
xmlns:skin="clr-  
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"  
skin:SfSkinManager.VisualStudio="Office2019Colorful"  
TitleTextAlignment="Center"  
Icon="App.ico"  
ShowMinimizeButton="False"  
ShowMaximizeButton="False"  
Title="Getting Started"  
Height="350" Width="550">  
<Grid>  
</Grid>  
</syncfusion:ChromelessWindow>
```



### Adding controls in the Title bar in WPF Chromeless Window

Provided the option to add adaptive controls such as Button, TextBox, Label, etc on either side of the title bar in the Chromeless Window. The [LeftHeaderItemsSource](#) property holds the controls to be shown on the left side of the title bar likewise the [RightHeaderItemsSource](#) property holds the controls to be shown on the right side of the title bar.

#### XML

```
<syncfusion:ChromelessWindow x:Class="Chromeless_Window_Sample.MainWindow"  
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
xmlns:local="clr-namespace:Chromeless_Window_Sample"  
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"  
xmlns:skin="clr-  
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"  
skin:SfSkinManager.VisualStudio="Office2016Colorful"  
TitleTextAlignment="Left"
```

```

Title="Chromeless Window Sample" IconAlignment="Left" Height="350"
Width="550" x:Name="window"
RightHeaderItemsSource="{DynamicResource rightHeaderItems}">
<syncfusion:ChromelessWindow.Resources>
<local:MyObservableCollection x:Key="rightHeaderItems">
<syncfusion:SfTextBoxExt Watermark="Search..." Width="60" Height="20"/>
<Button x:Name="help" Width="22" Height="22" Background="Transparent"
Command="{Binding Path=HelpCommand, ElementName=window}" BorderThickness="0"
>
<Path Data="M3.9400001,13.238 L5.5349999,13.238 5.5349999,14.833
3.9400001,14.833 z M4.7539988,0 C6.2060028,8.8817842E-16
7.3750015,0.39599991 8.2300044,1.1770003 9.0749989,1.9500008
9.5039998,2.8959999 9.5039998,3.9900017 9.5039998,4.6220016
9.3539982,5.2110023 9.0599995,5.743 8.7549992,6.2900009 8.1419993,6.9750023
7.2350021,7.7770004 6.5800033,8.3570023 6.1620041,8.7770004
5.9580017,9.0590019 5.7429972,9.3530006 5.5830012,9.6930008
5.4789973,10.070999 5.3929988,10.394001 5.3399974,10.871002
5.3170024,11.521999 L4.0500041,11.521999 C4.0479975,11.409
4.0459986,11.316002 4.0459986,11.244999 4.0459986,10.528 4.1480036,9.9029999
4.3499995,9.387001 4.4899989,9.0110016 4.7289973,8.618 5.0599986,8.2180023
5.310998,7.9189987 5.7679992,7.4770012 6.4180008,6.9049988
7.1190048,6.2859993 7.5660034,5.7989998 7.7829991,5.4169998
8.0100032,5.0200005 8.1240016,4.5839996 8.1240016,4.1189995
8.1240016,3.288002 7.796999,2.5480003 7.1510025,1.9220008
6.5110031,1.2989998 5.7139979,0.98400116&#xd;&#xa;4.784997,0.9840011
3.8870018,0.98400116 3.1250005,1.2709999 2.5199972,1.8380011
1.9710011,2.3500004 1.5930027,3.1230011 1.3939974,4.1389999 L0,3.9729996
C0.19999708,2.7350006 0.6869967,1.7670002 1.4499972,1.0950012
2.2720037,0.36900139 3.3850029,8.8817842E-16 4.7539988,0 z"
Fill="#FF595858" HorizontalAlignment="Left" Height="11" Stretch="Fill"
VerticalAlignment="Top" Width="7.504" />
</Button>
</local:MyObservableCollection>
</syncfusion:ChromelessWindow.Resources>
<Grid>
</Grid>
</syncfusion:ChromelessWindow>

```

**C#**

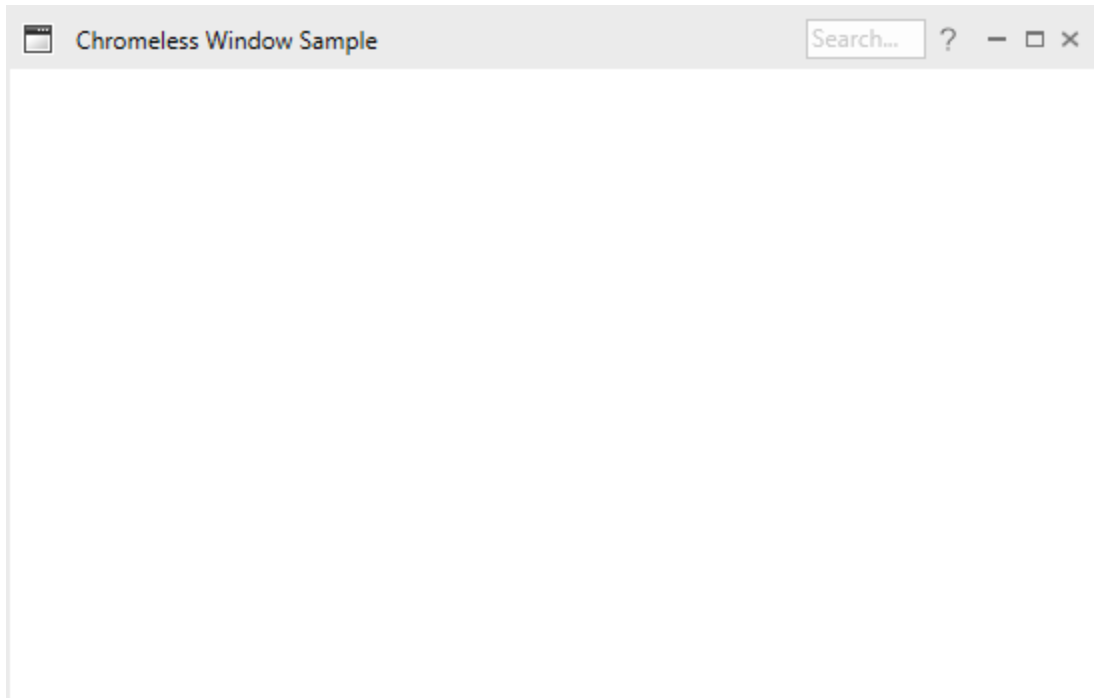
```

public class MyObservableCollection : ObservableCollection<object> { }
/// <summary>
/// Command for the help button.
/// </summary>
public DelegateCommand HelpCommand
{
    get
    {
        return new DelegateCommand(HelpCommandAction);
    }
}
/// <summary>
/// Action that is performed when clicking the help button.
/// </summary>
private void HelpCommandAction(object param)
{

```

```
System.Diagnostics.Process.Start("https://help.syncfusion.com/wpf/chromeless-  
-window/adding-controls-in-the-titlebar");  
}  
/// <summary>  
/// A class that defines the interface for the command.  
/// </summary>  
public class DelegateCommand : ICommand  
{  
    private readonly Action<object> _execute;  
    /// <summary>  
    /// Raises when changes occur and specifies whether or not the command  
    /// should be executed.  
    /// </summary>  
    public event EventHandler CanExecuteChanged;  
    /// <summary>  
    /// Constrcutor of the delegate command.  
    /// </summary>  
    /// <param name="execute"></param>  
    public DelegateCommand(Action<object> execute)  
    {  
        _execute = execute;  
    }  
    /// <summary>  
    /// Defines the method that determines whether the command can be executed  
    /// in its current state.  
    /// </summary>  
    public bool CanExecute(object parameter)  
    {  
        return true;  
    }  
    /// <summary>  
    /// Defines the method to be called when the command is invoked.  
    /// </summary>  
    public void Execute(object parameter)  
    {  
        _execute(parameter);  
    }  
    /// <summary>  
    /// Defines the method to be called when changes occur that affect the  
    /// command.  
    /// </summary>  
    public void RaiseCanExecuteChanged()  
    {  
        if (CanExecuteChanged != null)  
        {  
            CanExecuteChanged(this, EventArgs.Empty);  
        }  
    }  
}
```





We can also apply data template in-order to define the visual appearance of the items stored in the [LeftHeaderItemsSource](#) and [RightHeaderItemsSource](#). The [LeftHeaderItemTemplate](#) is used to define the visual appearance of the items stored in the [LeftHeaderItemsSource](#) while the [RightHeaderItemTemplate](#) is used to define the visual appearance of the items stored in the [RightHeaderItemsSource](#).

In the illustration below, we have defined the [RightHeaderItemTemplate](#) with a button control and set the “Utilities” collection holding sign-in and help details to the [RightHeaderItemsSource](#). Now, the properties of the button are mapped to the properties of the “Item” class objects maintained in the “Utilities” collection to create a visual appearance. When this is done, the items in the “Utilities” collection are visually converted as a sign-in and help button's and are added to the right side of the title bar.

### XML

```
<syncfusion:ChromelessWindow x:Class="Chromeless_Window_Sample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:Chromeless_Window_Sample"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:skin="clr-namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
skin:SfSkinManager.VisualStudio="Office2016Colorful"
TitleTextAlignment="Left"
Title="Chromeless Window Sample" IconAlignment="Left" Height="350"
Width="550"
RightHeaderItemsSource="{Binding Utilities}"
RightHeaderItemTemplate="{DynamicResource Part_RightItemsTemplate}">
<syncfusion:ChromelessWindow.DataContext>
<local:UtilityViewModel/>
```

```

</syncfusion:ChromelessWindow.DataContext>
<syncfusion:ChromelessWindow.Resources>
<DataTemplate x:Key="Part_RightItemsTemplate">
<Grid>
<syncfusion:ButtonAdv SmallIcon="{Binding Path = Icon, Mode = TwoWay}"
Label="{Binding Path= Text, Mode=TwoWay}" SizeMode="{Binding Path=Mode}"
Command="{Binding Path=Command}" Background="Transparent"
BorderThickness="0" />
</Grid>
</DataTemplate>
</syncfusion:ChromelessWindow.Resources>
<Grid>
</Grid>
</syncfusion:ChromelessWindow>

```

**C#**

```

/// <summary>
/// A model class view containing the details of the items to be bound in
the title bar.
/// </summary>
public class UtilityViewModel
{
    /// <summary>
    /// Constructor of the UtilityViewModel class.
    /// </summary>
    public UtilityViewModel()
    {
        LoadUtilities();
    }

    /// <summary>
    /// Collection containing the complete details of the items to be bound in
the title bar.
    /// </summary>
    public ObservableCollection<Item> Utilities
    {
        get;
        set;
    }

    /// <summary>
    /// Command for the Help button.
    /// </summary>
    public DelegateCommand HelpCommand
    {
        get
        {
            return new DelegateCommand(HelpCommandAction);
        }
    }

    /// <summary>
    /// Action that is performed when clicking the help button.
    /// </summary>
    private void HelpCommandAction(object param)
    {
        System.Diagnostics.Process.Start("https://help.syncfusion.com/wpf/chromeless-window/adding-controls-in-the-titlebar");
    }
}

```

```
}
/// <summary>
/// Commmand for the Sign in button.
/// </summary>
public DelegateCommand SignInCommand
{
    get
    {
        return new DelegateCommand(SignInCommandAction);
    }
}
/// <summary>
/// Action that is performed when clicking the Sign in button.
/// </summary>
private void SignInCommandAction(object param)
{
    MessageBox.Show("Sign in successful");
}
/// <summary>
/// Create and add items along with the necessary details for the
collection.
/// </summary>
public void LoadUtilities()
{
    ObservableCollection<Item> utilities = new ObservableCollection<Item>();
    var outPutDirectory =
        System.IO.Path.GetDirectoryName(Assembly.GetExecutingAssembly().CodeBase);
    var logoimage = System.IO.Path.Combine(outPutDirectory, "Images\\Help.png");
    utilities.Add(new Item { Name = "Account", Icon = null, Text = "Sign in",
        Mode = SizeMode.Normal, Command=SignInCommand });
    utilities.Add(new Item { Name = "Help", Icon = new BitmapImage(new
        Uri(logoimage.ToString())), Text="", Mode = SizeMode.Small,
        Command=HelpCommand });
    Utilities = utilities;
}
}
/// <summary>
/// A model class that shows the outline of items, such as the name of the
item, the source path of the icon, the text display, the size and the
command of the image.
/// </summary>
public class Item
{
    public string Name
    {
        get; set;
    }
    public ImageSource Icon
    {
        get; set;
    }
    public string Text
    {
        get; set;
    }
    public SizeMode Mode
    {

```

```
get; set;
}
public ICommand Command
{
    get; set;
}
}
/// <summary>
/// A class that defines the interface for the command.
/// </summary>
public class DelegateCommand : ICommand
{
    private readonly Action<object> _execute;
    /// <summary>
    /// Raises when changes occur and specifies whether or not the command
    /// should be executed.
    /// </summary>
    public event EventHandler CanExecuteChanged;
    /// <summary>
    /// Constrcutor of the Delegate command
    /// </summary>
    /// <param name="execute"></param>
    public DelegateCommand(Action<object> execute)
    {
        _execute = execute;
    }
    /// <summary>
    /// Defines the method that determines whether the command can be executed
    /// in its current state.
    /// </summary>
    public bool CanExecute(object parameter)
    {
        return true;
    }
    /// <summary>
    /// Defines the method to be called when the command is invoked.
    /// </summary>
    public void Execute(object parameter)
    {
        _execute(parameter);
    }
    /// <summary>
    /// Defines the method to be called when changes occur that affect the
    /// command.
    /// </summary>
    public void RaiseCanExecuteChanged()
    {
        if (CanExecuteChanged != null)
        {
            CanExecuteChanged(this, EventArgs.Empty);
        }
    }
}
```



## Customizing Border of the ChromelessWindow in WPF Chromeless Window

### BorderBrush

The Borders of the resizable window can be painted by using the **ResizeBorderBrush** property.

To set the ResizeBorderBrush property, use the below code

### XML

```
<syncfusion:ChromelessWindow x:Class="Chromelesswindow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="ChromelessWindow" Height="350" Width="525" ResizeBorderBrush="Maroon"
syncfusion:SkinStorage.VisualStudio="Metro"
x:Name="_chromelessWindow" xmlns:syncfusion="clr-
namespace:Syncfusion.Windows.Shared;assembly=Syncfusion.Shared.WPF" >
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```

### C#

```
this.ResizeBorderBrush = new SolidColorBrush(Colors.Maroon);
```

### VB.NET

```
Me.ResizeBorderBrush = New SolidColorBrush(Colors.Maroon)
```



### BorderThickness

To set the thickness for the Resizable border, use **ResizableBorderThickness** property of the ChromelessWindow. This property virtually sets the region for the resize pointer to appear. Hence, greater the region, easier it is to resize.

To set this property, use the following code.

#### XML

```
<syncfusion:ChromelessWindow x:Class="Chromelesswindow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="ChromelessWindow" Height="350" Width="525" ResizeBorderThickness="8"
syncfusion:SkinStorage.VisualStudio="Metro"
x:Name="_chromelessWindow" xmlns:syncfusion="clr-
namespace:Syncfusion.Windows.Shared;assembly=Syncfusion.Shared.WPF" >
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```

#### C#

```
this.ResizeBorderThickness = new Thickness(8);
```

#### VB.NET

```
Me.ResizeBorderThickness = New Thickness(8)
```



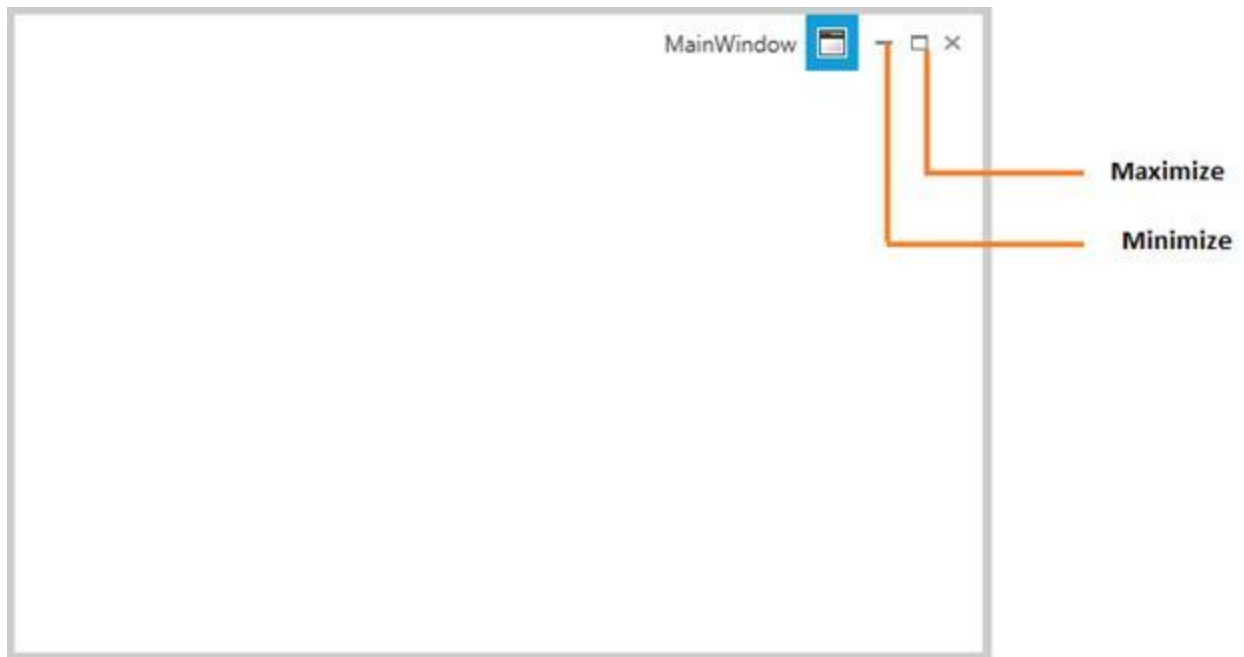
### End user capabilities in WPF Chromeless Window

The user can perform the below listed operations through the UI provided by the ChromelessWindow.

1. Maximize/Minimize
2. Restore
3. Close
4. Resize

### Maximize/Minimize

The end user can maximize or minimize the buttons by using the Maximize and Minimize Buttons at the Top-Right corner of the ChromelessWindow



### Restore

After minimized or maximized the respective button is replaced by a **Restore** button. By clicking on this restore button the user can bring the **ChromelessWindow** to its **Normal** state.



### Close

The user can close the Window by using the Close Button at the Top-Right Corner of the ChromelessWindow





### Resize

The Window can be resize by clicking and dragging the Resizable Border

ChromelessWindow supports the following four Resize Modes

1. NoResize
2. CanMinimize
3. CanResize
4. CanResizeWithGrip

### NoResize

In NoResize mode, the window cannot able to resize. The Minimize and Maximize buttons are not displayed in the title bar

**CanMinimize**

In this mode the window can only be minimize, since the Minimize button alone enabled.

**CanResize**

In CanResize Mode, a window can be resized. The Minimize and Maximize buttons are both shown and enabled



### CanResizeWithGrip

In `CanResizeWithGrip` mode, a window can be resized. A resize grip appears in the bottom-right corner of the window. The Minimize and Maximize buttons are both shown and enabled



### Styling the ChromelessWindow in WPF Chromeless Window

This section deals with the Styles and Templates supported by `ChromelessWindow` control.

### Custom template for the TitleBar

ChromelessWindow allows the end-user to write own templates for the TitleBar. The TitleBar can be customized by using the **TitleBarTemplate** property. The following code snippet illustrates how to set the TitleBarTemplate property.

#### XML

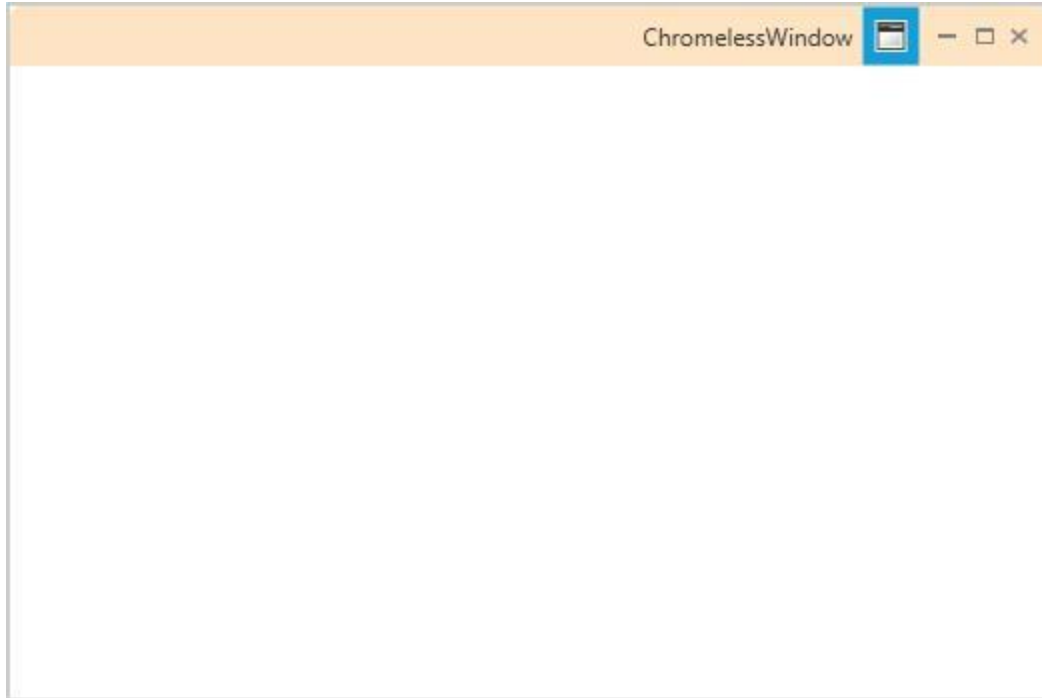
```
<syncfusion:ChromelessWindow x:Class="Chromelesswindow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="ChromelessWindow" Height="350" Width="525"
TitleBarTemplate="{StaticResource TitleBarTemplateKey}"
syncfusion:SkinStorage.VisualStudio="Metro" x:Name="_chromelessWindow"
xmlns:syncfusion="clr-
namespace:Syncfusion.Windows.Shared;assembly=Syncfusion.Shared.WPF" >
</syncfusion:ChromelessWindow>
```

For example, use the following code for a TitleBarTemplate.

#### XML

```
<ControlTemplate x:Key="TitleBarTemplateKey" TargetType="{x:Type
syncfusion:TitleBar}">
<Border x:Name="MainGrid" Height="30" CornerRadius="5,5,0,0"
Background="Bisque" >
<Border BorderBrush="Transparent" Background="Transparent"
BorderThickness=".5" Width="Auto" CornerRadius="0,0,0,0">
<ContentPresenter HorizontalAlignment="Stretch" VerticalAlignment="Center"
/>
</Border>
</Border>
</ControlTemplate>
```

**Note:** The drag and drop feature for the ChromelessWindow is available only with the TitleBar,to incorporate this feature,include a TitleBar into your ChromelessWindow.Also it is necessary to specify the name of the TitleBar as PART\_TitleBar",to enable the drag and drop feature.



### Custom template for the TitleButton

ChromelessWindow enables the user to write their own templates for the Title button that is used in the TitleBar. **Maximize , Minimize , Restore and Close** Button can be edited with **MaximizeButtonTemplate , MinimizeButtonTemplate , RestoreButtonTemplate , CloseButtonTemplate** property respectively. The below code example is illustrated to change the template of the Maximize Button using **MaximizeButtonTemplate** property

### XML

```
<syncfusion:ChromelessWindow x:Class="Chromelesswindow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="ChromelessWindow" Height="350" Width="525"
MaximizeButtonTemplate="{StaticResource MaxTemplateKey}"
syncfusion:SkinStorage.VisualStudio="Metro" x:Name="_chromelessWindow"
xmlns:syncfusion="clr-namespace:Syncfusion.Windows.Shared;assembly=Syncfusion.Shared.WPF" >
<Grid>
</Grid>
</syncfusion:ChromelessWindow>
```

For example, use the following code for a MaximizeButtonTemplate.

### XML

```
<ControlTemplate x:Key="MaxTemplateKey" TargetType="{x:Type
syncfusion:TitleButton}">
<Border SnapsToDevicePixels="true" x:Name="maxborder" Width="15" Height="15"
Background="Coral"
BorderThickness="0" BorderBrush="Transparent">
<Grid SnapsToDevicePixels="true" x:Name="grid" HorizontalAlignment="Center"
VerticalAlignment="Center" Width="9" Height="8">
```

```
<Path Name="pathButton" SnapsToDevicePixels="True" Stretch="Fill"
StrokeThickness="1"
Data="M1,1 L8,1 L8,8 L1,8 z M1,2 L8,2" Stroke="Green"
HorizontalAlignment="Center" VerticalAlignment="Center" Width="9"
Height="8">
</Path>
</Grid>
</Border>
</ControlTemplate>
```



Similarly, the template for the desired button can be changed by using the corresponding button property

#### Overriding the default style

ChromelessWindow can be customized by editing the default template. The default style can be downloaded from the following link

<http://www.syncfusion.com/downloads/support/directtrac/general/ze/DefaultStyle-1700710349>

This default style need to be included in the App.xaml file of your application and the override this style as you desire.

---

**Note:** All the templates should be written in App.xaml file only.

---

The below example is illustrated to override the ChromelessWindow's full style

#### XML

```
<ControlTemplate x:Key="TitleBarTemplateKey" TargetType="{x:Type
syncfusion:TitleBar}">
<Border Name="border" Background="#AA161616" Height="30">
<ContentPresenter HorizontalAlignment="Stretch" VerticalAlignment="Bottom"
Margin="0,0,0,0"/>
```

```

</Border>
</ControlTemplate>
<ControlTemplate x:Key="ChromelessWindowTemplate" TargetType="{x:Type
syncfusion:ChromelessWindow}">
<AdornerDecorator>
<Border Name="OuterBorder" Background="#F1401013" BorderThickness="2"
CornerRadius="{Binding ElementName=CornerRadiusSlider,Path=Value}"
BorderBrush="#401013">
.....
.....
.....
<syncfusion:TitleBar Name="PART_TitleBar" Grid.Row="0"
Template="{StaticResource TitleBarTemplateKey}" >
<Grid VerticalAlignment="Top" Height="30">
<StackPanel Orientation="Horizontal">
<Image x:Name="PART_Icon" Source="{Binding RelativeSource={RelativeSource
FindAncestor, AncestorType={x:Type syncfusion:ChromelessWindow}}},
Path=Icon}" VerticalAlignment="Center" HorizontalAlignment="Left"
Margin="4,4,2,4" MaxWidth="16" MaxHeight="16" MinWidth="16" MinHeight="16"
/>
<ContentControl Foreground="White" Content="{TemplateBinding Title}"
VerticalAlignment="Center" HorizontalAlignment="Left"
x:Name="TitlePresenter" Margin="5,5,5,5" />
</StackPanel>
.....
.....
</Grid>
</syncfusion:TitleBar>
.....
.....
</Grid>
</Border>
</AdornerDecorator>
</ControlTemplate>

```

Then apply this style to the ChromelessWindow using Template Property

#### XML

```

<syncfusion:ChromelessWindow x:Class="Chromelesswindow.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="ChromelessWindow" Height="350" Width="525" Template="{StaticResource
ChromelessWindowTemplate}"
syncfusion:SkinStorage.VisualStudio="Metro" x:Name="_chromelessWindow"
xmlns:syncfusion="clr-
namespace:Syncfusion.Windows.Shared;assembly=Syncfusion.Shared.WPF" >
</syncfusion:ChromelessWindow>

```

The following screen shot shows the Custom ChromelessWindow control created using the preceding code.



## SfCircularGauge

### WPF Radial Gauge (SfCircularGauge) Overview

The circular gauge helps to visualize numeric values on a circular scale. The appearance of the gauge is fully customized to integrate your applications without fault.

#### Key features

The [CircularGauge](#) is composed of several scales. The scales will be an integrated UI part of the circular gauge.

The [CircularGauge](#) is a composite UI element with the following subparts:

- Scales
- Ranges
- Pointers
- Header

The circular gauge control is highly customizable with variety of simple APIs to modify its basic look and feel. You can position the ranges, ticks, labels, and range pointers as needed.

#### Scales

The [Scales](#) contain labels, tick marks, and a rim to customize its basic look and feel. It defines the start angle, sweep direction, sweep angle, overall minimum and maximum values, the frequency of labels, and tick marks.

#### Ranges

The [Ranges](#) are visual elements that depict the start and end values of inner divisions within the scale's range. Each scale is capable of displaying one or more ranges, and each range can depict different zones or regions of same metrics, such as high, low, and average temperatures.

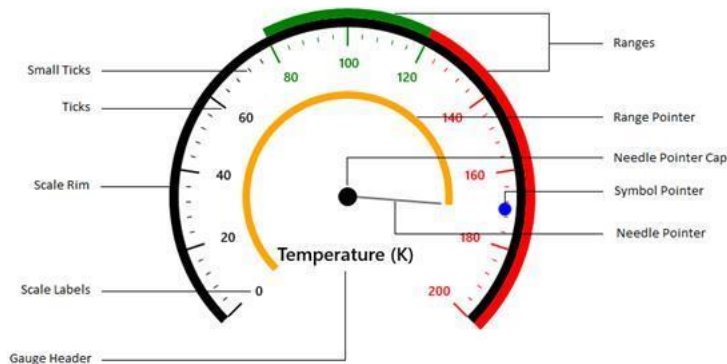


### Pointers

The [Pointers](#) are elements that point out the values of the bound property on a scale. A circular scale can contain one or more pointers that can be used to measure different values. Each pointer has the [Value](#) property, which informs the current value to users visually.

### Header

The [Header](#) can be used to set a unique header for the circular gauge. You can add text as the header in the circular gauge.



## Getting Started with WPF Radial Gauge (SfCircularGauge)

This section explains the steps required to configure the [SfCircularGauge](#) and add basic elements to it using various APIs.

### Adding gauge references

You can add gauge reference using one of the following methods:

#### Method 1: Adding gauge reference from nuget.org

Syncfusion WPF components are available in [nuget.org](https://nuget.org). To add gauge to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.SfGauge.WPF](#), and then install it.

![Adding gauge reference from NuGet](Getting-Started\_images/Adding gauge reference.png)

#### Method 2: Adding gauge reference from toolbox

You can drag the circular gauge control from the toolbox and drop it to the designer. It will add the required assembly references automatically, and add the namespace to the page.

#### Method 3: Adding gauge assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/WPF/Assemblies

You can refer to [this](#) link to know about the assemblies required for adding gauge to your project.

### Initialize gauge

Import the [SfCircularGauge](#) namespace to your respective Window as follows.

### XML

```
xmlns:gauge="clr-  
namespace:Syncfusion.UI.Xaml.Gauges;assembly=Syncfusion.SfGauge.Wpf"
```

### C#

```
using Syncfusion.UI.Xaml.Gauges;
```

You can initialize an empty [SfCircularGauge](#) control.

### XML

```
<gauge:SfCircularGauge />
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
this.Content = sfCircularGauge;
```

### Adding headers

You can assign a unique header to the [SfCircularGauge](#) by using the [GaugeHeader](#) property.

### XML

```
<gauge:SfCircularGauge Height="250"  
Width="250"  
HeaderAlignment="Custom"  
GaugeHeaderPosition="0.63,0.75">  
  <gauge:SfCircularGauge.GaugeHeader>  
    <TextBlock Text="Speedometer"  
Height="40"  
Width="140"  
FontSize="13"  
Foreground="Black"  
FontWeight="SemiBold" />  
  </gauge:SfCircularGauge.GaugeHeader>  
</gauge:SfCircularGauge>
```

### C#

```
//Initializing circular gauge  
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
sfCircularGauge.Height = 250;  
sfCircularGauge.Width = 250;  
//Adding header  
sfCircularGauge.HeaderAlignment = HeaderAlignment.Custom;  
sfCircularGauge.GaugeHeaderPosition = new Point(0.63, 0.75);  
TextBlock textBlock = new TextBlock();  
textBlock.Text = "Speedometer";  
textBlock.Height = 40;  
textBlock.Width = 140;  
textBlock.FontSize = 13;  
textBlock.Foreground = new SolidColorBrush(Colors.Black);  
textBlock.FontWeight = FontWeights.SemiBold;  
sfCircularGauge.GaugeHeader = textBlock;
```

```
this.Content = sfCircularGauge;
```

### Configuring scales

You can configure the [CircularScale](#) elements by using the following APIs:

- StartAngle
- SweepAngle
- StartValue
- EndValue
- Interval
- TickStroke
- LabelStroke

### XML

```
<gauge:SfCircularGauge.Scales>  
<gauge:CircularScale />  
</gauge:SfCircularGauge.Scales>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
CircularScale mainscale = new CircularScale();  
sfCircularGauge.Scales.Add(mainscale);  
this.Content = sfCircularGauge;
```

### Adding ranges

You can add ranges to the [SfCircularGauge](#) by creating ranges collection using the [CircularRange](#) property.

### XML

```
<gauge:SfCircularGauge>  
<gauge:SfCircularGauge.Scales>  
<gauge:CircularScale>  
<gauge:CircularScale.Ranges>  
<gauge:CircularRange StartValue="0"  
EndValue="60"  
Stroke="Gray" />  
</gauge:CircularScale.Ranges>  
</gauge:CircularScale>  
</gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
CircularScale mainscale = new CircularScale();  
CircularRange circularRange = new CircularRange();  
circularRange.StartValue = 0;  
circularRange.EndValue = 60;  
circularRange.Stroke = new SolidColorBrush(Colors.Gray);
```

```
mainscale.Ranges.Add(circularRange);  
sfCircularGauge.Scales.Add(mainscale);  
this.Content = sfCircularGauge;
```

### Adding a needle pointer

Create a **Needle Pointer**, and associate it with a scale that is to be displayed the current value.

#### XML

```
<gauge:SfCircularGauge>  
  <gauge:SfCircularGauge.Scales>  
    <gauge:CircularScale>  
      <gauge:CircularScale.Pointers>  
        <gauge:CircularPointer PointerType="NeedlePointer"  
          Value="60"  
          NeedleLengthFactor="0.5"  
          NeedlePointerType="Triangle"  
          PointerCapDiameter="12"  
          NeedlePointerStroke="#757575"  
          KnobFill="#757575"  
          KnobStroke="#757575"  
          NeedlePointerStrokeThickness="7" />  
      </gauge:CircularScale.Pointers>  
    </gauge:CircularScale>  
  </gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
CircularScale mainscale = new CircularScale();  
CircularPointer circularPointer = new CircularPointer();  
circularPointer.PointerType = PointerType.NeedlePointer;  
circularPointer.Value = 60;  
circularPointer.NeedleLengthFactor = 0.5;  
circularPointer.NeedlePointerType = NeedlePointerType.Triangle;  
circularPointer.PointerCapDiameter = 12;  
circularPointer.NeedlePointerStroke = (SolidColorBrush) new  
BrushConverter().ConvertFrom("#757575");  
circularPointer.KnobFill = (SolidColorBrush) new  
BrushConverter().ConvertFrom("#757575");  
circularPointer.KnobStroke = (SolidColorBrush) new  
BrushConverter().ConvertFrom("#757575");  
circularPointer.NeedlePointerStrokeThickness = 7;  
mainscale.Pointers.Add(circularPointer);  
sfCircularGauge.Scales.Add(mainscale);  
this.Content = sfCircularGauge;
```

### Adding a range pointer

The **Range Pointer** provides an alternative way to indicate the current value.

#### XML

```
<gauge:SfCircularGauge>  
  <gauge:SfCircularGauge.Scales>
```

```
<gauge:CircularScale>
<gauge:CircularScale.Pointers>
<gauge:CircularPointer PointerType="RangePointer"
Value="40"
RangePointerStrokeThickness="5"
RangePointerStroke="#27beb6" />
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer1 = new CircularPointer();
circularPointer1.PointerType = PointerType.RangePointer;
circularPointer1.Value = 40;
circularPointer1.RangePointerStrokeThickness = 5;
circularPointer1.RangePointerStroke = (SolidColorBrush)new
BrushConverter().ConvertFrom("#27beb6");
mainscale.Pointers.Add(circularPointer1);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
```

### Adding a symbol pointer

The **Symbol Pointer** points to the current value in a scale.

### XML

```
<gauge:SfCircularGauge>
<gauge:SfCircularGauge.Scales>
<gauge:CircularScale>
<gauge:CircularScale.Pointers>
<gauge:CircularPointer PointerType="SymbolPointer"
Value="70"
SymbolPointerHeight="12"
SymbolPointerWidth="12"
Symbol="InvertedTriangle"
SymbolPointerStroke="#757575" />
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer2 = new CircularPointer();
circularPointer2.PointerType = PointerType.SymbolPointer;
circularPointer2.Value = 70;
circularPointer2.SymbolPointerHeight = 12;
circularPointer2.SymbolPointerWidth = 12;
circularPointer2.Symbol = Symbol.InvertedTriangle;
```

```

circularPointer2.SymbolPointerStroke = (SolidColorBrush) new
BrushConverter().ConvertFrom("#757575");
mainscale.Pointers.Add(circularPointer2);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;

```

The following code example is the complete code of the previous configurations.

#### XML

```

<gauge:SfCircularGauge Height="250"
Width="250"
HeaderAlignment="Custom"
GaugeHeaderPosition="0.63,0.75">
  <gauge:SfCircularGauge.GaugeHeader>
    <TextBlock Text="Speedometer"
Height="40"
Width="140"
FontSize="13"
Foreground="Black"
FontWeight="SemiBold" />
  </gauge:SfCircularGauge.GaugeHeader>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale ShowRim="True"
RimStroke="LightGray"
RimStrokeThickness="3"
LabelOffset="0.1">
      <gauge:CircularScale.MajorTickSettings>
        <gauge:MajorTickSetting Length="10"
StrokeThickness="1" />
      </gauge:CircularScale.MajorTickSettings>
      <gauge:CircularScale.MinorTickSettings>
        <gauge:MinorTickSetting Length="5"
StrokeThickness="1" />
      </gauge:CircularScale.MinorTickSettings>
      <gauge:CircularScale.Ranges>
        <gauge:CircularRange StartValue="0"
EndValue="60"
Stroke="Gray" />
      </gauge:CircularScale.Ranges>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="NeedlePointer"
Value="60"
NeedleLengthFactor="0.5"
NeedlePointerType="Triangle"
PointerCapDiameter="12"
NeedlePointerStroke="#757575"
KnobFill="#757575"
KnobStroke="#757575"
NeedlePointerStrokeThickness="7" />
        <gauge:CircularPointer PointerType="RangePointer"
Value="40"
RangePointerStrokeThickness="5"
RangePointerStroke="#27beb6" />
        <gauge:CircularPointer PointerType="SymbolPointer"
Value="70"

```

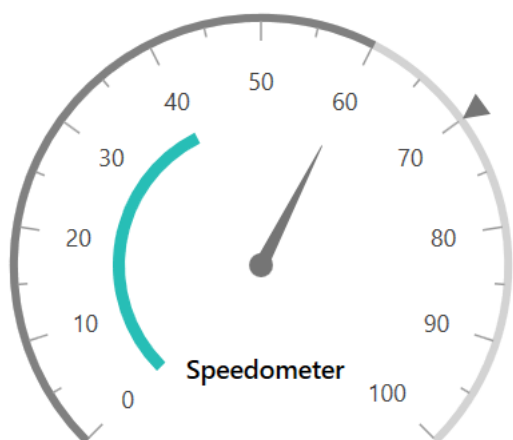
```
SymbolPointerHeight="12"
SymbolPointerWidth="12"
Symbol="InvertedTriangle"
SymbolPointerStroke="#757575" />
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

## C#

```
//Initializing circular gauge
SfCircularGauge sfCircularGauge = new SfCircularGauge();
sfCircularGauge.Height = 250;
sfCircularGauge.Width = 250;
//Adding header
sfCircularGauge.HeaderAlignment = HeaderAlignment.Custom;
sfCircularGauge.GaugeHeaderPosition = new Point(0.63, 0.75);
TextBlock textBlock = new TextBlock();
textBlock.Text = "Speedometer";
textBlock.Height = 40;
textBlock.Width = 140;
textBlock.FontSize = 13;
textBlock.Foreground = new SolidColorBrush(Colors.Black);
textBlock.FontWeight = FontWeights.SemiBold;
sfCircularGauge.GaugeHeader = textBlock;
//Initializing scales for circular gauge
CircularScale mainscale = new CircularScale();
mainscale.RimStroke = new SolidColorBrush(Colors.LightGray);
mainscale.RimStrokeThickness = 3;
mainscale.LabelOffset = 0.1;
MajorTickSetting majorTickSetting = new MajorTickSetting();
majorTickSetting.StrokeThickness = 1;
majorTickSetting.Length = 10;
mainscale.MajorTickSettings = majorTickSetting;
MinorTickSetting minorTickSetting = new MinorTickSetting();
minorTickSetting.StrokeThickness = 1;
minorTickSetting.Length = 5;
mainscale.MinorTickSettings = minorTickSetting;
//Adding range
CircularRange circularRange = new CircularRange();
circularRange.StartValue = 0;
circularRange.EndValue = 60;
circularRange.Stroke = new SolidColorBrush(Colors.Gray);
mainscale.Ranges.Add(circularRange);
//Adding needle pointer
CircularPointer circularPointer = new CircularPointer();
circularPointer.PointerType = PointerType.NeedlePointer;
circularPointer.Value = 60;
circularPointer.NeedleLengthFactor = 0.5;
circularPointer.NeedlePointerType = NeedlePointerType.Triangle;
circularPointer.PointerCapDiameter = 12;
circularPointer.NeedlePointerStroke = (SolidColorBrush)new
BrushConverter().ConvertFrom("#757575");
circularPointer.KnobFill = (SolidColorBrush)new
BrushConverter().ConvertFrom("#757575");
```

```
circularPointer.KnobStroke = (SolidColorBrush)new  
BrushConverter().ConvertFrom("#757575");  
circularPointer.NeedlePointerStrokeThickness = 7;  
mainscale.Pointers.Add(circularPointer);  
//Adding range pointer  
CircularPointer circularPointer1 = new CircularPointer();  
circularPointer1.PointerType = PointerType.RangePointer;  
circularPointer1.Value = 40;  
circularPointer1.RangePointerStrokeThickness = 5;  
circularPointer1.RangePointerStroke = (SolidColorBrush)new  
BrushConverter().ConvertFrom("#27beb6");  
mainscale.Pointers.Add(circularPointer1);  
//Adding symbol pointer  
CircularPointer circularPointer2 = new CircularPointer();  
circularPointer2.PointerType = PointerType.SymbolPointer;  
circularPointer2.Value = 70;  
circularPointer2.SymbolPointerHeight = 12;  
circularPointer2.SymbolPointerWidth = 12;  
circularPointer2.Symbol = Symbol.InvertedTriangle;  
circularPointer2.SymbolPointerStroke = (SolidColorBrush)new  
BrushConverter().ConvertFrom("#757575");  
mainscale.Pointers.Add(circularPointer2);  
sfCircularGauge.Scales.Add(mainscale);  
this.Content = sfCircularGauge;
```

The following screenshot illustrates the result of the previous codes.



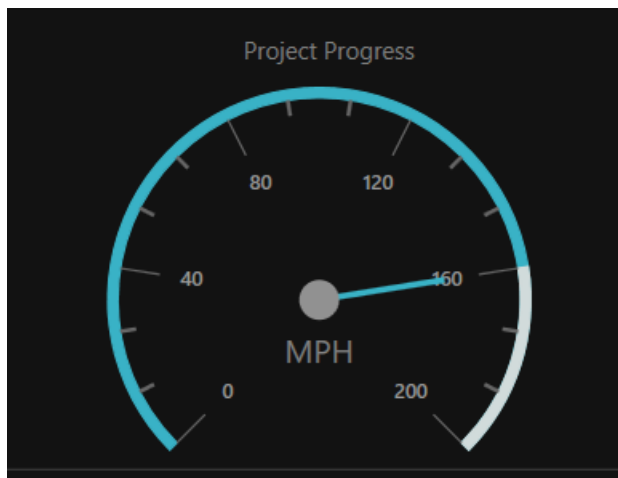
You can find the complete getting started sample from this [link](#).

#### Theme

SfCircularGauge supports various built-in themes. Refer to the below links to apply themes for the SfCircularGauge,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)





See also

[How to apply themes for SfCircularGauge](#)

[How to create dynamic application driven by excelling formulas using SfCircularGauge](#)

### Header in WPF Radial Gauge (SfCircularGauge)

Header allows you to show text or any UI content inside the gauge control using [GaugeHeader](#) option. This provides information about the data that is being plotted in the circular gauge.

#### Setting Header for Circular Gauge

The [GaugeHeader](#) is an object that can be used to set a unique header for the circular gauge. You can add text and images as header in the circular gauge. Only one header can be added in a circular gauge.

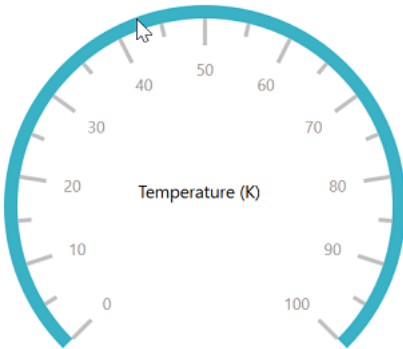
#### XML

```
<gauge:SfCircularGauge HeaderAlignment="Center">
  <gauge:SfCircularGauge.GaugeHeader>
    <TextBlock Text="Temperature (K)"
      Height="40" Width="100"
      FontSize="13" Foreground="Black"/>
  </gauge:SfCircularGauge.GaugeHeader>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
TextBlock textBlock = new TextBlock();
textBlock.Text = "Temperature (K)";
textBlock.Height = 40;
textBlock.Width = 100;
textBlock.FontSize = 13;
textBlock.Foreground = new SolidColorBrush(Colors.Black);
```

```
sfCircularGauge.GaugeHeader = textBlock;
sfCircularGauge.HeaderAlignment = HeaderAlignment.Center;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
```



### Setting alignment for header

The gauge header can be positioned by using the [HeaderAlignment](#) property. The default value of this property is **Left**.

It includes the following options:

- Left
- Right
- Top
- Bottom
- Center
- TopLeft
- TopRight
- BottomLeft
- BottomRight
- Custom

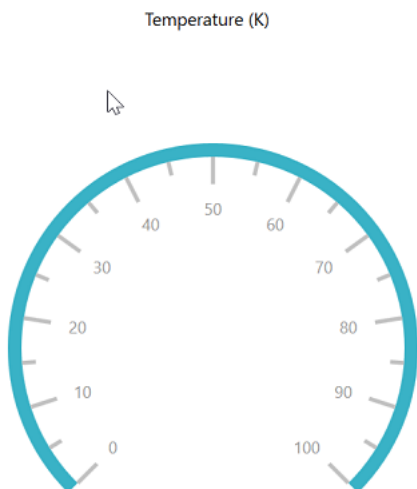
### XML

```
<gauge:SfCircularGauge HeaderAlignment="Top">
  <gauge:SfCircularGauge.GaugeHeader>
    <TextBlock Text="Temperature (K)"
      Height="40" Width="100"
      FontSize="13" Foreground="Black"/>
  </gauge:SfCircularGauge.GaugeHeader>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
```

```
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
TextBlock textBlock = new TextBlock();
textBlock.Text = "Temperature (K)";
textBlock.Height = 40;
textBlock.Width = 100;
textBlock.FontSize = 13;
textBlock.Foreground = new SolidColorBrush(Colors.Black);
sfCircularGauge.GaugeHeader = textBlock;
sfCircularGauge.HeaderAlignment = HeaderAlignment.Top;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
```

**Setting position for header**

The [GaugeHeaderPosition](#) property is used to place header in the circular gauge. The value for [GaugeHeaderPosition](#) should be specified in offset value. In the point value, which has been given for the [GaugeHeaderPosition](#), the first value represent x-coordinate and the second value represents y-coordinate. First, set the [HeaderAlignment](#) to custom, then set the position of header.

**XML**

```
<gauge:SfCircularGauge HeaderAlignment="Custom"
GaugeHeaderPosition="0.5,0.8">
  <gauge:SfCircularGauge.GaugeHeader>
    <TextBlock Text="Temperature (K)"
      Height="40" Width="100"
      FontSize="13" Foreground="Black"/>
  </gauge:SfCircularGauge.GaugeHeader>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
```

```

<gauge:CircularScale.Pointers>
<gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

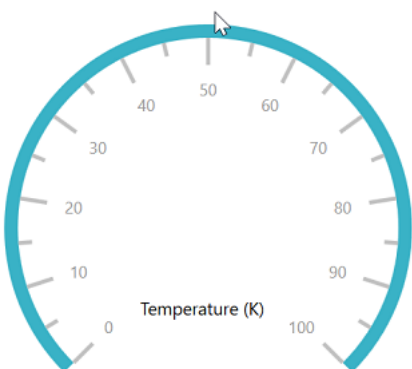
```

### C#

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
TextBlock textBlock = new TextBlock();
textBlock.Text = "Temperature (K)";
textBlock.Height = 40;
textBlock.Width = 100;
textBlock.FontSize = 13;
textBlock.Foreground = new SolidColorBrush(Colors.Black);
sfCircularGauge.GaugeHeader = textBlock;
sfCircularGauge.HeaderAlignment = HeaderAlignment.Custom;
sfCircularGauge.GaugeHeaderPosition = new Point(0.5, 0.8);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;

```



### Customization of header font

You can customize the header's text by using the `FontFamily`, `FontStyle`, `FontSize`, and `Foreground` properties.

### XML

```

<gauge:SfCircularGauge HeaderAlignment="Custom"
GaugeHeaderPosition="0.5,0.8"
FontFamily="Monotype Corsiva" FontSize="15"
FontStyle="Italic" Foreground="Blue">
<gauge:SfCircularGauge.GaugeHeader>
<TextBlock Text="Temperature (K)"
Height="40" Width="100" />
</gauge:SfCircularGauge.GaugeHeader>
<gauge:SfCircularGauge.Scales>
<gauge:CircularScale >

```

```

<gauge:CircularScale.Pointers>
<gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

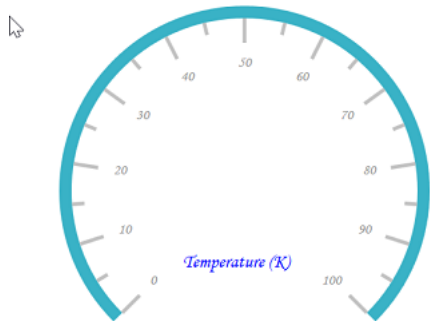
```

## C#

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
TextBlock textBlock = new TextBlock();
textBlock.Text = "Temperature (K)";
textBlock.Height = 40;
textBlock.Width = 100;
sfCircularGauge.GaugeHeader = textBlock;
sfCircularGauge.HeaderAlignment = HeaderAlignment.Custom;
sfCircularGauge.GaugeHeaderPosition = new Point(0.5, 0.8);
sfCircularGauge.FontSize = 15;
sfCircularGauge.FontFamily = new FontFamily("Monotype Corsiva");
sfCircularGauge.FontStyle = FontStyles.Italic;
sfCircularGauge.Foreground = new SolidColorBrush(Colors.Blue);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;

```



## Scales in WPF Radial Gauge (SfCircularGauge)

Scales contain a collection of [CircularScale](#) elements, which integrate labels, tick marks, and a rim to customize the basic look and feel of the circular gauge.

### Scale

The [CircularScale](#) contains sub elements such as rim, ticks, labels, ranges, and pointers. They define the radius, start angle, sweep direction, sweep angle, overall minimum and maximum values, frequency of labels, and tick marks. A scale will have multiple ranges.

A range is a visual element, which begins and ends at the specified values within a [CircularScale](#). A range will have one or more pointers to point out the values in a scale.

## XML

```

<gauge:SfCircularGauge>

```

```
<gauge:SfCircularGauge.Scales>  
<gauge:CircularScale />  
</gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
CircularScale circularScale = new CircularScale();  
sfCircularGauge.Scales.Add(circularScale);
```



Setting start and end values for scale

The [StartValue](#) and [EndValue](#) properties allow you to set the start and end values for a scale.

### XML

```
<gauge:SfCircularGauge>  
<gauge:SfCircularGauge.Scales>  
<gauge:CircularScale StartValue="-30" EndValue="50">  
<gauge:CircularScale.Pointers>  
<gauge:CircularPointer NeedlePointerVisibility="Hidden"/>  
</gauge:CircularScale.Pointers>  
</gauge:CircularScale>  
</gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
```

```

CircularScale circularScale = new CircularScale();
circularScale.StartValue = -30;
circularScale.EndValue = 50;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
circularScale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(circularScale);
this.Content = sfCircularGauge;

```



Setting start and sweep angles for scale

The [StartAngle](#) and [SweepAngle](#) properties allow you to set the start and end angles for a scale.

#### XML

```

<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale StartAngle="185" SweepAngle="270">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

#### C#

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale circularScale = new CircularScale();
circularScale.StartAngle = 185;
circularScale.SweepAngle = 270;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
circularScale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(circularScale);
this.Content = sfCircularGauge;

```



### Setting interval for scale

The [Interval](#) property allows you to set the interval for a scale. The default value of `Interval` is `Auto`, it defines the count of the scale labels based on `StartValue` and `EndValue` of scale.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale StartValue="0" EndValue="500" Interval="100">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale circularScale = new CircularScale();
circularScale.StartValue = 0;
circularScale.EndValue = 500;
circularScale.Interval = 100;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
circularScale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(circularScale);
this.Content = sfCircularGauge;
```





**Note:** You can specify the interval value upto 5 decimal places while showing the labels of linear scale.

#### Setting sweep direction for scale

The [SweepDirection](#) property allows you to render the gauge scale in either clockwise or counterclockwise direction.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale SweepDirection="Counterclockwise">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale circularScale = new CircularScale();
circularScale.SweepDirection = SweepDirection.Counterclockwise;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
circularScale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(circularScale);
this.Content = sfCircularGauge;
```



Setting multiple scales for circular gauge

You can add multiple scales to the same circular gauge and customize all the scales in a [Scales](#) collection. The [SpacingMargin](#) property determines the size of the circular gauge, which ranges from 0.1 to 1.

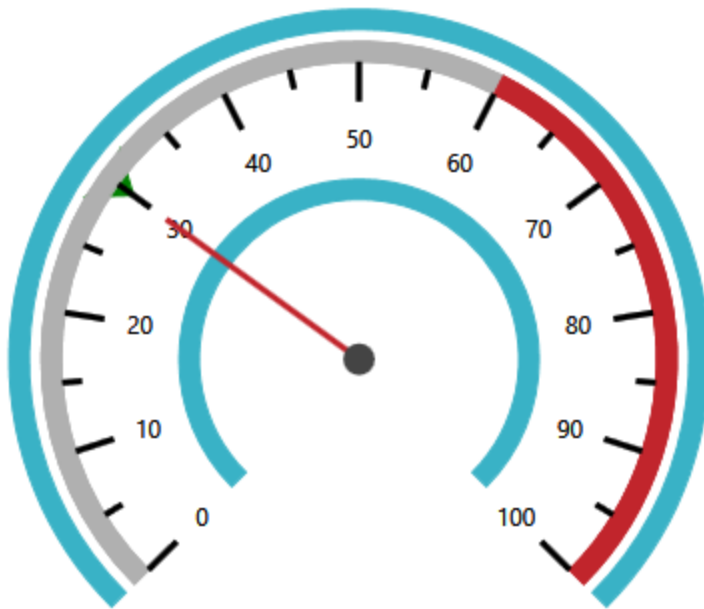
#### XML

```
<gauge:SfCircularGauge SpacingMargin="0.7">
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale Radius="175">
      <gauge:CircularScale.Ranges>
        <gauge:CircularRange StartValue="0" EndValue="60"
          Stroke="#B0B0B0" StrokeThickness="5" />
        <gauge:CircularRange StartValue="60" EndValue="100"
          Stroke="#C1252C" StrokeThickness="5" />
      </gauge:CircularScale.Ranges>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer Value="30" PointerType="SymbolPointer"
          Symbol="InvertedArrow"
          SymbolPointerWidth="30"
          SymbolPointerHeight="20"
          SymbolPointerStroke="Green" />
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
    <gauge:CircularScale Radius="90" LabelStroke="Black">
      <gauge:CircularScale.MajorTickSettings>
        <gauge:MajorTickSetting Stroke="Black" />
      </gauge:CircularScale.MajorTickSettings>
      <gauge:CircularScale.MinorTickSettings>
        <gauge:MinorTickSetting Stroke="Black" />
      </gauge:CircularScale.MinorTickSettings>
      <gauge:CircularScale.Ranges>
        <gauge:CircularRange StartValue="0" EndValue="60"
          Stroke="#B0B0B0" StrokeThickness="5" />
        <gauge:CircularRange StartValue="60" EndValue="100"
          Stroke="#C1252C" StrokeThickness="5" />
      </gauge:CircularScale.Ranges>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer Value="30" PointerType="NeedlePointer"
          NeedlePointerStroke="#C1252C" />
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
circularGauge.SpacingMargin = 0.7;
CircularScale scale1 = new CircularScale();
scale1.LabelStroke = new SolidColorBrush(Colors.Black);
scale1.Radius = 175;
CircularScale scale2 = new CircularScale();
scale2.LabelStroke = new SolidColorBrush(Colors.Black);
scale2.Radius = 90;
scale2.MajorTickSettings.Stroke = new SolidColorBrush(Colors.Black);
scale2.MinorTickSettings.Stroke = new SolidColorBrush(Colors.Black);
```

```
CircularRange circularRange = new CircularRange();
circularRange.StartValue = 0;
circularRange.EndValue = 60;
circularRange.Stroke = new SolidColorBrush(Color.FromRgb(176, 176, 176));
circularRange.StrokeThickness = 5;
scale1.Ranges.Add(circularRange);
CircularRange circularRange1 = new CircularRange();
circularRange1.StartValue = 60;
circularRange1.EndValue = 100;
circularRange1.Stroke = new SolidColorBrush(Color.FromArgb(0xFF, 0xC1, 0x25, 0x2C));
circularRange1.StrokeThickness = 5;
scale1.Ranges.Add(circularRange1);
CircularPointer circularPointer = new CircularPointer();
circularPointer.Value = 30;
circularPointer.PointerType = PointerType.SymbolPointer;
circularPointer.Symbol = Syncfusion.UI.Xaml.Gauges.Symbol.InvertedArrow;
circularPointer.SymbolPointerWidth = 30;
circularPointer.SymbolPointerHeight = 20;
circularPointer.SymbolPointerStroke = new SolidColorBrush(Colors.Green);
scale1.Pointers.Add(circularPointer);
CircularRange circularRange2 = new CircularRange();
circularRange2.StartValue = 0;
circularRange2.EndValue = 60;
circularRange2.Stroke = new SolidColorBrush(Color.FromRgb(176, 176, 176));
circularRange2.StrokeThickness = 5;
scale2.Ranges.Add(circularRange2);
CircularRange circularRange3 = new CircularRange();
circularRange3.StartValue = 60;
circularRange3.EndValue = 100;
circularRange3.Stroke = new SolidColorBrush(Color.FromArgb(0xFF, 0xC1, 0x25, 0x2C));
circularRange3.StrokeThickness = 5;
scale2.Ranges.Add(circularRange3);
CircularPointer circularPointer2 = new CircularPointer();
circularPointer2.Value = 30;
circularPointer2.PointerType = PointerType.NeedlePointer;
circularPointer2.NeedlePointerStroke = new
SolidColorBrush(Color.FromArgb(0xFF, 0xC1, 0x25, 0x2C));
scale2.Pointers.Add(circularPointer2);
circularGauge.Scales.Add(scale1);
circularGauge.Scales.Add(scale2);
this.Content = circularGauge;
```



### Rim in WPF Radial Gauge (SfCircularGauge)

Scale determines the structure of a circular gauge by using a circular rim. By setting the [StartAngle](#) and [SweepAngle](#) properties, you can change the shape of the circular gauge to a full-circular gauge, half-circular gauge, or quarter-circular gauge.

The [StartValue](#) and [EndValue](#) properties determine the overall range of the circular rim.

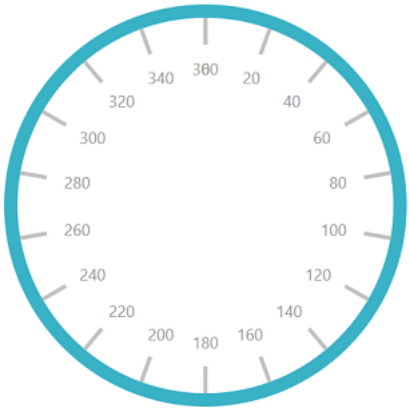
#### XML

```
<gauge:SfCircularGauge HeaderAlignment="Bottom">
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale StartAngle="270" SweepAngle="360" StartValue="0"
      EndValue="360"
      Interval="20" MinorTicksPerInterval="0" >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale circularScale = new CircularScale();
circularScale.StartAngle = 270;
circularScale.SweepAngle = 360;
circularScale.StartValue = 0;
circularScale.EndValue = 360;
circularScale.Interval = 20;
circularScale.MinorTicksPerInterval = 0;
CircularPointer circularPointer = new CircularPointer();
```

```
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
circularScale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(circularScale);
this.Content = sfCircularGauge;
```



### Rim customization

The color and thickness of the rim can be set by using the [RimStroke](#) and [RimStrokeThickness](#) properties.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale RadiusFactor="1" RimStrokeThickness="40"
      RimStroke="SkyBlue" >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale circularScale = new CircularScale();
circularScale.RadiusFactor = 1;
circularScale.RimStrokeThickness = 40;
circularScale.RimStroke = new SolidColorBrush(Colors.SkyBlue);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
circularScale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(circularScale);
this.Content = sfCircularGauge;
```



Setting a position for rim

You can customize the position of [Scales](#) in the following two ways:

1. [RadiusFactor](#) property.
2. [Radius](#) property.

*Setting a radius factor for rim*

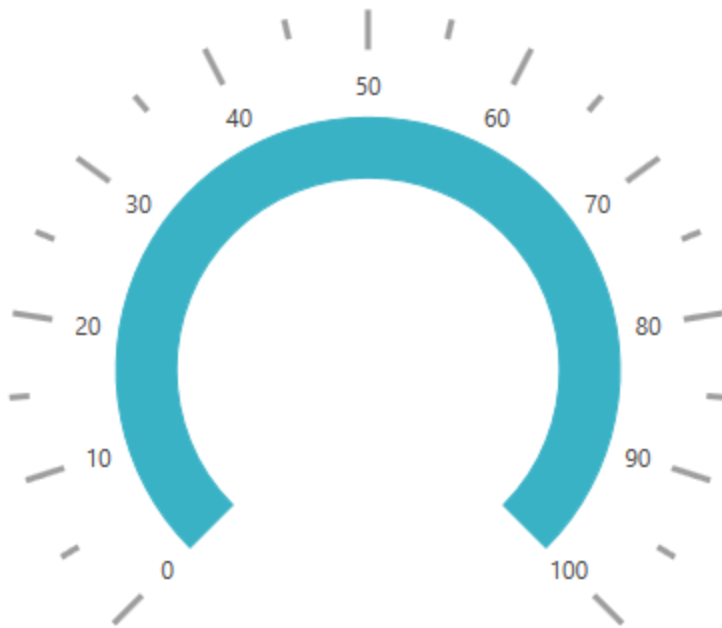
The value for `RadiusFactor` should be specified in offset value. Its value should be 0 to 1. `RadiusFactor` sets the scale responsive with the window.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale RadiusFactor="0.6" RimStrokeThickness="30" >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale circularScale = new CircularScale();
circularScale.RadiusFactor = 0.6;
circularScale.RimStrokeThickness = 30;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
circularScale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(circularScale);
this.Content = sfCircularGauge;
```



#### Setting a radius for rim

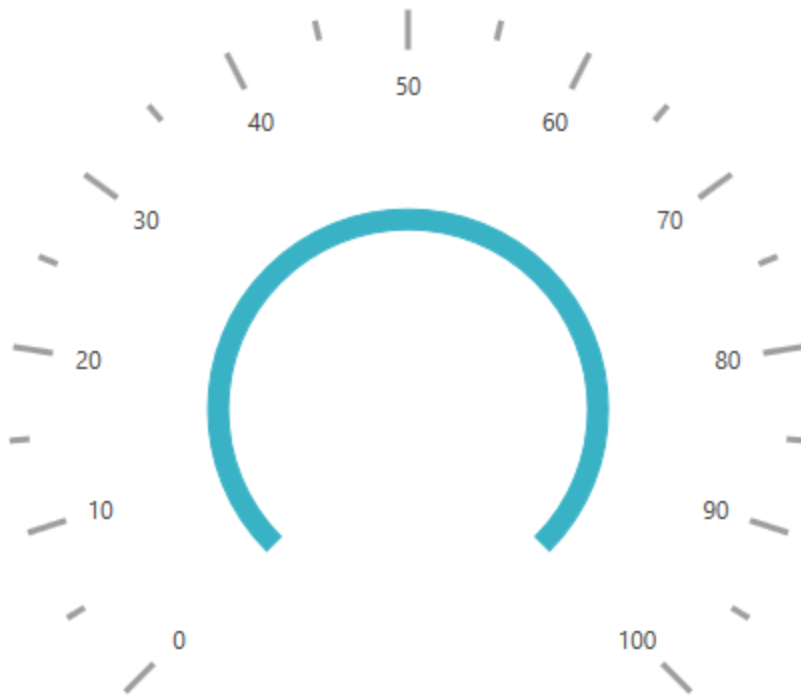
You can set the **Radius** of rim in pixel value. **Radius** sets the scale fixed with the given pixel value.

#### XML

```
<gauge:SfCircularGauge HeaderAlignment="Bottom">
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale Radius="100" >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale circularScale = new CircularScale();
circularScale.Radius = 100;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
circularScale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(circularScale);
this.Content = sfCircularGauge;
```



#### Setting rim visibility

The [ShowRim](#) property is a Boolean property, which is used to enable or disable the rim in circular gauge.

---

**Note:** Default value of the ShowRim property is true.

---

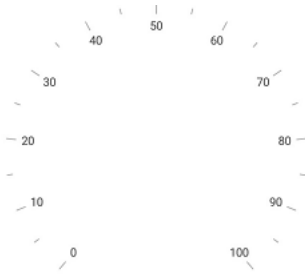
#### XML

```
<gauge:SfCircularGauge HeaderAlignment="Bottom">
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale ShowRim="False" x:Name="scale" >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.ShowRim = false;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
```





*Set offset value for a responsive size in rim*

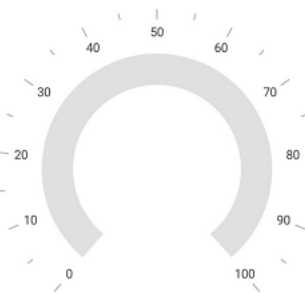
- [RimStartOffset](#) - Sets the rim start offset position. Its range is from 0 to 1.
- [RimEndOffset](#) - Sets the rim end offset position. Its range is from 0 to 1.

### XML

```
<gauge:SfCircularGauge x:Name="gauge">
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale x:Name="scale" RimStartOffset="0.6" RimEndOffset="0.7"
      RimStroke="LightGray" >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RimStroke = new SolidColorBrush(Colors.LightGray);
mainscale.RimStartOffset = 0.6;
mainscale.RimEndOffset = 0.7;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
```



### Ticks in WPF Radial Gauge (SfCircularGauge)

Ticks help you identify the gauge's data value by marking the gauge scale in regular increments.

### Tick customization

The Interval property is used to calculate the tick count for a scale. Similar major ticks, small ticks are calculated using the [MinorTicksPerInterval](#) property.

The length, stroke, and stroke thickness of a major ticks and minor ticks are set by using the [Length](#), [Stroke](#), and [StrokeThickness](#) properties, respectively.

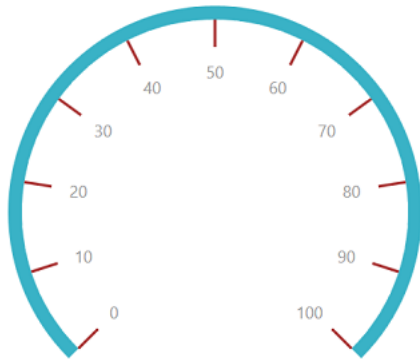
*Customize major ticks for scale*

#### XML

```
<gauge:SfCircularGauge x:Name="gauge">
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale x:Name="scale" MinorTicksPerInterval="3" >
      <gauge:CircularScale.MajorTickSettings>
        <gauge:MajorTickSetting Length="20" Stroke="Brown" StrokeThickness="2" />
      </gauge:CircularScale.MajorTickSettings>
      <gauge:CircularScale.MinorTickSettings>
        <gauge:MinorTickSetting Stroke="White" />
      </gauge:CircularScale.MinorTickSettings>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.MinorTicksPerInterval = 3;
MajorTickSetting majorTickSetting = new MajorTickSetting();
majorTickSetting.Length = 20;
majorTickSetting.Stroke = new SolidColorBrush(Colors.Brown);
majorTickSetting.StrokeThickness = 2;
mainscale.MajorTickSettings = majorTickSetting;
MinorTickSetting minorTickSetting = new MinorTickSetting();
minorTickSetting.Stroke = new SolidColorBrush(Colors.White);
mainscale.MinorTickSettings = minorTickSetting;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
```



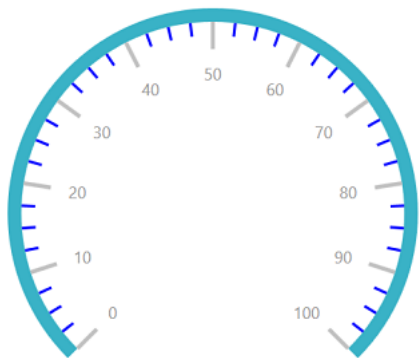
*Customize minor ticks for scale*

#### **XML**

```
<gauge:SfCircularGauge x:Name="gauge">
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale x:Name="scale" MinorTicksPerInterval="3" >
      <gauge:CircularScale.MinorTickSettings>
        <gauge:MinorTickSetting Length="10" Stroke="Blue" StrokeThickness="2" />
      </gauge:CircularScale.MinorTickSettings>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### **C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.MinorTicksPerInterval = 3;
MinorTickSetting minorTickSetting = new MinorTickSetting();
minorTickSetting.Length = 10;
minorTickSetting.Stroke = new SolidColorBrush(Colors.Blue);
minorTickSetting.StrokeThickness = 2;
mainscale.MinorTickSettings = minorTickSetting;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



### Setting shape for tick

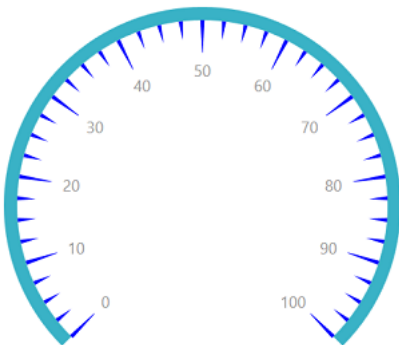
The [TickShape](#) is an enum property that provides an option to select shape of the circular mark ticks, which contains several shapes such as rectangle, ellipse, and triangle.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale TickShape="Triangle" MinorTicksPerInterval="3">
      <gauge:CircularScale.MajorTickSettings>
        <gauge:MajorTickSetting Stroke="Blue"/>
      </gauge:CircularScale.MajorTickSettings>
      <gauge:CircularScale.MinorTickSettings>
        <gauge:MinorTickSetting Stroke="Blue"/>
      </gauge:CircularScale.MinorTickSettings>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.TickShape = TickShape.Triangle;
mainscale.MinorTicksPerInterval = 3;
mainscale.MajorTickSettings.Stroke = new SolidColorBrush(Colors.Blue);
mainscale.MinorTickSettings.Stroke = new SolidColorBrush(Colors.Blue);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
```



### Setting position for tick

The major and minor ticks can be positioned far away from the rim using the following ways:

- Setting direct [TickPosition](#) property. This place the tick inside, outside or cross of the rim.
- Setting [MajorTickSettings](#) and [MinorTickSettings Offset](#) and [Length](#) properties. This tick positions is responsive for all the window size. But tick length is fixed.
- Setting [MajorTickSettings](#) and [MinorTickSettings StartOffset](#), and [EndOffset](#) properties. This tick position and length are responsive to all size of the window.

### Setting direct tick position

Placing the ticks inside or outside the scale, or across the scale by selecting one of the options available in the [TickPosition](#) property. They are:

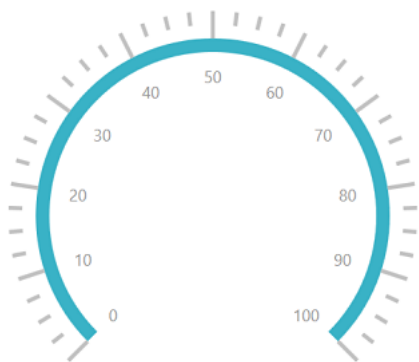
1. Inside (Default)
2. Outside
3. Cross
4. Custom

### XML

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale TickPosition="Outside" MinorTicksPerInterval="3">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.TickPosition = TickPosition.Outside;
mainscale.MinorTicksPerInterval = 3;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
```



### Setting scale tick Offset value

For relative position you can use [Offset](#) and [Length](#) property. First, set the [TickPosition](#) property to custom, and then set the offset and length of the tick.

### XML

```
<gauge:SfCircularGauge x:Name="gauge">
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale TickPosition="Custom"
      x:Name="scale" MinorTicksPerInterval="3" >
      <gauge:CircularScale.MajorTickSettings>
        <gauge:MajorTickSetting Offset="0.5" Length="20"/>
      </gauge:CircularScale.MajorTickSettings>
      <gauge:CircularScale.MinorTickSettings>
        <gauge:MinorTickSetting Offset="0.5" Length="5"/>
      </gauge:CircularScale.MinorTickSettings>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.TickPosition = TickPosition.Custom;
mainscale.MinorTicksPerInterval = 3;
MajorTickSetting majorTickSetting = new MajorTickSetting();
majorTickSetting.Offset = 0.5;
majorTickSetting.Length = 20;
mainscale.MajorTickSettings = majorTickSetting;
MinorTickSetting minorTickSetting = new MinorTickSetting();
minorTickSetting.Offset = 0.5;
minorTickSetting.Length = 5;
mainscale.MinorTickSettings = minorTickSetting;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
```



#### Setting start and end Offset value for scale tick

For absolute position you can use [StartOffset](#), [EndOffset](#) properties of [MajorTickSettings](#) and [MinorTickSettings](#). First, set the [TickPosition](#) property to custom, and then set the [StartOffset](#) and [EndOffset](#) of the tick.

#### XML

```
<gauge:SfCircularGauge x:Name="gauge">
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale x:Name="scale" TickPosition="Custom"
      MinorTicksPerInterval="3" >
      <gauge:CircularScale.MajorTickSettings>
        <gauge:MajorTickSetting StartOffset="0.5" EndOffset="0.7"
          Length="20" Stroke="Brown" StrokeThickness="2" />
      </gauge:CircularScale.MajorTickSettings>
      <gauge:CircularScale.MinorTickSettings>
        <gauge:MinorTickSetting Stroke="Brown" StartOffset="0.6"
          EndOffset="0.7" StrokeThickness="2" />
      </gauge:CircularScale.MinorTickSettings>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
```

```

CircularScale mainscale = new CircularScale();
mainscale.TickPosition = TickPosition.Custom;
mainscale.MinorTicksPerInterval = 3;
MajorTickSetting majorTickSetting = new MajorTickSetting();
majorTickSetting.StartOffset = 0.5;
majorTickSetting.EndOffset = 0.7;
majorTickSetting.Length = 20;
majorTickSetting.Stroke = new SolidColorBrush(Colors.Brown);
majorTickSetting.StrokeThickness = 2;
mainscale.MajorTickSettings = majorTickSetting;
MinorTickSetting minorTickSetting = new MinorTickSetting();
minorTickSetting.StartOffset = 0.6;
minorTickSetting.EndOffset = 0.7;
minorTickSetting.Stroke = new SolidColorBrush(Colors.Brown);
minorTickSetting.StrokeThickness = 2;
mainscale.MinorTickSettings = minorTickSetting;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;

```



#### Setting ticks visibility in scale

The [ShowTicks](#) property allows you to enable or disable the ticks of circular gauge.

**Note:** Default value of the ShowTicks property is true.

#### XML

```

<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale ShowTicks="False" x:Name="scale" RimStroke="LightGray"
    >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```



```

</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

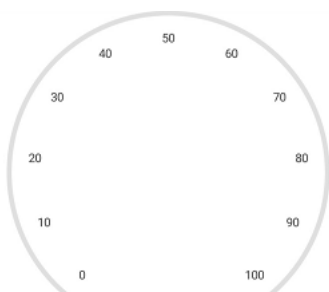
```

**C#**

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RimStroke = new SolidColorBrush(Colors.LightGray);
mainscale.ShowTicks = false;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;

```



## Labels in WPF Radial Gauge (SfCircularGauge)

The **Scale** labels associate numeric values with major scale tick marks.

## Label stroke customization

The label color can be changed using the [LabelStroke](#) property.

**XML**

```

<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale LabelStroke="DeepPink">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

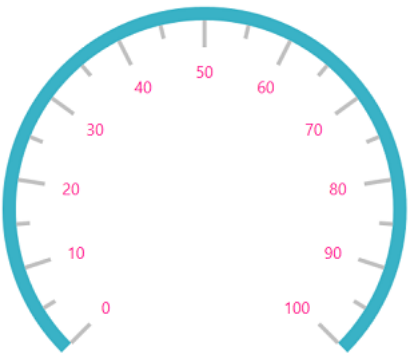
```

**C#**

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.LabelStroke = new SolidColorBrush(Colors.DeepPink);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);

```



### Label font customization

The label font can be customized using the `FontSize`, `FontFamily`, and `FontStyle` properties.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale FontFamily="Monotype Corsiva" FontSize="20"
      FontStyle="Italic" >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
sfCircularGauge.FontSize = 20;
sfCircularGauge.FontFamily = new FontFamily("Monotype Corsiva");
sfCircularGauge.FontStyle = FontStyles.Italic;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



### Setting a position for labels

The **Labels** can be placed inside the scale, outside the scale, or custom position using the following two ways:

#### Inside or Outside label Position

Placing the labels inside or outside the scale by selecting the [LabelPosition](#) property is **Inside** or **Outside**. Please find the values available in **LabelPosition** property below.

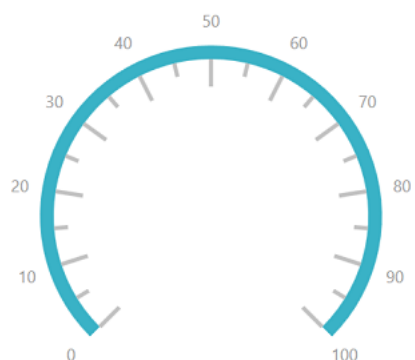
1. Inside (Default)
2. Outside
3. Custom

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale LabelPosition="Outside" >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.LabelPosition = LabelPosition.Outside;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



### Setting custom position for labels

Positioning the labels far away from the ticks using the [LabelOffset](#) property. First, set the [LabelPosition](#) to custom, and then position the label using the **LabelOffset** property.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale LabelPosition="Custom" LabelOffset="0.5" >
      <gauge:CircularScale.Pointers>
```

```
<gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.LabelPosition = LabelPosition.Custom;
mainscale.LabelOffset = 0.5;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```

**Label rotation**

Circular gauge labels can be rotated based on its corresponding angle. This can be controlled by [CanRotateLabels](#) property in the scale.

Setting false to this property will display all the labels without rotation.

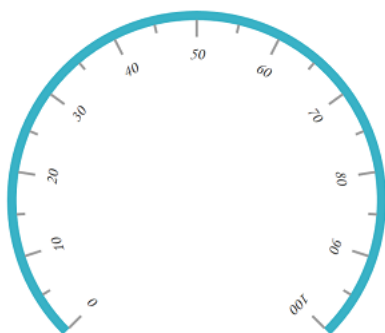
**XML**

```
<gauge:SfCircularGauge>
<gauge:SfCircularGauge.Scales>
<gauge:CircularScale FontFamily="Monotype Corsiva" FontSize="20"
CanRotateLabels="True"
FontStyle="Italic" >
<gauge:CircularScale.Pointers>
<gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
sfCircularGauge.FontSize = 20;
```

```
sfCircularGauge.FontFamily = new FontFamily("Monotype Corsiva");
sfCircularGauge.FontStyle = FontStyles.Italic;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
mainscale.CanRotateLabels = true;
sfCircularGauge.Scales.Add(mainscale);
```



### Setting a smart labels

The [EnableSmartLabels](#) property is a Boolean property that enables or disables the smart label feature of the circular gauge.

This property allow to change the [NumericScaleType](#) of the labels displayed in a gauge scale, and customize the labels by adding prefixes or suffixes to the scale labels.

The [NumericScaleType](#) property allows to set the type of label. The following types can be applied to labels:

- Auto
- Thousands
- Millions
- Billions
- Trillions
- Quadrillions
- Quintillions

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale StartValue="0" EndValue="500"
      EnableSmartLabels="True" NumericScaleType="Thousands">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.StartValue = 0;
mainscale.EndValue = 500;
mainscale.EnableSmartLabels = true;
mainscale.NumericScaleType = NumericScaleType.Thousands;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



Setting a number of fraction digits for labels

The [NoOfFractionalDigit](#) property is used to set the number of fractional digits to be displayed in the scale labels.

**XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale NoOfFractionalDigit="3">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.NoOfFractionalDigit = 3;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



### Setting a postfix and prefix for labels

You can postfix or prefix values to the scale labels using the [LabelPostfix](#) and [LabelPrefix](#) properties, respectively.

#### *Setting postfix value for labels*

The [LabelPostfix](#) property allows to postfix the values to the scale labels.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale LabelPostfix="k">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.LabelPostfix = "k";
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



*Setting prefix value for labels*

The `LabelPrefix` property allows to prefix the values to the scale labels.

**XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale LabelPrefix="$">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.LabelPrefix = "$";
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
```

*Setting scale labels visibility*

The `ShowLabels` property is a Boolean property, which is used to enable or disable the labels in circular gauge.

**Note:** Default value of the `ShowLabels` property is true.

**XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale ShowLabels="False" x:Name="scale" RimStroke="LightGray"
    >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```



**C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.ShowLabels = false;
mainscale.RimStroke = new SolidColorBrush(Colors.LightGray);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```

*Edge label customization in scale*

You can customize the edge label by using the [ShowFirstLabel](#) and [ShowLastLabel](#) properties, which are Boolean properties.

- **ShowFirstLabel** property is used to enable or disable the first label. Default value of the ShowFirstLabel property is true.
- **ShowLastLabel** property is used to enable or disable the last label. Default value of the ShowLastLabel property is true.

**XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale ShowFirstLabel = "False" StartValue = "0" EndValue =
    "12"
    Interval = "1" MinorTicksPerInterval = "5" StartAngle = "270"
    SweepAngle = "360" x:Name="scale" RimStroke="LightGray" >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

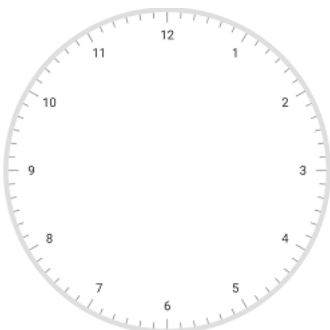
**C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.StartValue = 0;
mainscale.Interval = 1;
```

```

mainscale.MinorTicksPerInterval = 5;
mainscale.EndValue = 12;
mainscale.StartAngle = 270;
mainscale.SweepAngle = 360;
mainscale.ShowFirstLabel = false;
mainscale.RimStroke = new SolidColorBrush(Colors.LightGray);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);

```



## Events

You can change the default label by hooking the [LabelCreated](#) event. Based on requirements, the labels can be changed by using the [LabelText](#) property of [LabelCreatedEventArgs](#).

## XML

```

<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale x:Name="scale" LabelCreated="scale_LabelCreated"
      SweepAngle="360" StartAngle="270" StartValue="0" EndValue="16"
      Interval="2" RimStroke="LightGray" MinorTicksPerInterval="1"
      ShowLastLabel="False" >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

## C#

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.StartValue = 0;
mainscale.Interval = 2;
mainscale.MinorTicksPerInterval = 1;
mainscale.EndValue = 16;
mainscale.StartAngle = 270;
mainscale.SweepAngle = 360;
mainscale.LabelCreated += scale_LabelCreated;
mainscale.ShowLastLabel = false;
mainscale.RimStroke = new SolidColorBrush(Colors.LightGray);
CircularPointer circularPointer = new CircularPointer();

```

```
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainScale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainScale);
private void scale_LabelCreated(object sender, LabelCreatedEventArgs e)
{
    switch ((string)e.LabelText)
    {
        case "0":
            e.LabelText = "N";
            break;
        case "2":
            e.LabelText = "NE";
            break;
        case "4":
            e.LabelText = "E";
            break;
        case "6":
            e.LabelText = "SE";
            break;
        case "8":
            e.LabelText = "S";
            break;
        case "10":
            e.LabelText = "SW";
            break;
        case "12":
            e.LabelText = "W";
            break;
        case "14":
            e.LabelText = "NW";
            break;
    }
}
```



## Ranges in WPF Radial Gauge (SfCircularGauge)

Range is a visual element, which begins and ends at the specified values within a scale.

Setting a start and end values for range

The start and end values of ranges are set by using the [StartValue](#) and [EndValue](#) properties.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
      <gauge:CircularScale.Ranges>
        <gauge:CircularRange StartValue="0" EndValue="50"/>
      </gauge:CircularScale.Ranges>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularRange circularRange = new CircularRange();
circularRange.StartValue = 0;
circularRange.EndValue = 50;
mainscale.Ranges.Add(circularRange);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



## Range customization

A ranges appearance is customized by using the [Stroke](#), [StrokeThickness](#), and [Offset](#) properties. First, set the [RangePosition](#) property to custom in scale, and then set the [Offset](#).

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale RangePosition="Custom">
```

```
<gauge:CircularScale.Ranges>
<gauge:CircularRange StartValue="0" EndValue="50" Offset="0.6" Stroke="Pink"
StrokeThickness="40"/>
</gauge:CircularScale.Ranges>
<gauge:CircularScale.Pointers>
<gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RangePosition = RangePosition.Custom;
CircularRange circularRange = new CircularRange();
circularRange.StartValue = 0;
circularRange.EndValue = 50;
circularRange.Offset = 0.6;
circularRange.Stroke = new SolidColorBrush(Colors.Pink);
circularRange.StrokeThickness = 40;
mainscale.Ranges.Add(circularRange);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



### Setting a width for range

The circular range width is customized by setting the [StartWidth](#) and [EndWidth](#) properties.

#### XML

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale RangePosition="Inside" Radius="200">
      <gauge:CircularScale.Ranges>
        <gauge:CircularRange StartValue="10" EndValue="80" StrokeThickness="10"
          InnerStartOffset="0.95" InnerEndOffset="0.95"
          OuterStartOffset="0.9" OuterEndOffset="0.8"/>
      </gauge:CircularScale.Ranges>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RangePosition = RangePosition.Inside;
CircularRange circularRange = new CircularRange();
circularRange.StartValue = 10;
circularRange.EndValue = 80;
circularRange.StrokeThickness = 10;
circularRange.InnerStartOffset = 0.95;
circularRange.InnerEndOffset = 0.95;
circularRange.OuterStartOffset = 0.9;
circularRange.OuterEndOffset = 0.8;
circularRange.EndWidth = 10;
mainscale.Ranges.Add(circularRange);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
```



Binding range color to scale tick and labels

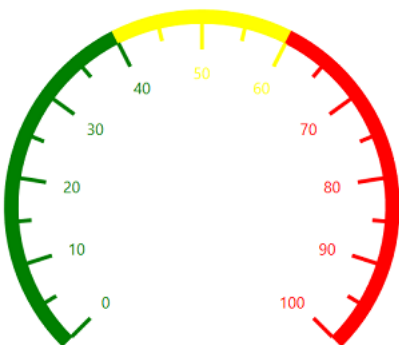
You can bind the ranges stroke to the tick lines and labels within its range by setting the [BindRangeStrokeToLabels](#) and [BindRangeStrokeToTicks](#) properties to true.

#### XML

```
<gauge:SfCircularGauge >
<gauge:SfCircularGauge.Scales>
<gauge:CircularScale BindRangeStrokeToLabels="True"
BindRangeStrokeToTicks="True">
<gauge:CircularScale.Ranges>
<gauge:CircularRange StartValue="0" EndValue="40" Stroke="Green"
StrokeThickness ="10" />
<gauge:CircularRange StartValue="40" EndValue="60" Stroke="Yellow"
StrokeThickness ="10" />
<gauge:CircularRange StartValue="60" EndValue="100" Stroke="Red"
StrokeThickness ="10"/>
</gauge:CircularScale.Ranges>
<gauge:CircularScale.Pointers>
<gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.BindRangeStrokeToLabels = true;
mainscale.BindRangeStrokeToTicks = true;
CircularRange circularRange = new CircularRange();
circularRange.StartValue = 0;
circularRange.EndValue = 40;
circularRange.Stroke = new SolidColorBrush(Colors.Green);
circularRange.StrokeThickness = 10;
mainscale.Ranges.Add(circularRange);
CircularRange circularRange1 = new CircularRange();
circularRange1.StartValue = 40;
circularRange1.EndValue = 60;
circularRange1.Stroke = new SolidColorBrush(Colors.Yellow);
circularRange1.StrokeThickness = 10;
mainscale.Ranges.Add(circularRange1);
CircularRange circularRange2 = new CircularRange();
circularRange2.StartValue = 60;
circularRange2.EndValue = 100;
circularRange2.Stroke = new SolidColorBrush(Colors.Red);
circularRange2.StrokeThickness = 10;
mainscale.Ranges.Add(circularRange2);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
```



### Setting a position for range

The range can be placed inside the scale, outside the scale, or on the scale by using the following two ways:

#### *By setting direct range position*

You can place the range by selecting one of the options available in the [RangePosition](#) property.

1. Inside
2. Outside
3. SetAsGaugeRim (Default)

### XML

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale RangePosition="Outside">
      <gauge:CircularScale.Ranges>
        <gauge:CircularRange StartValue="0" EndValue="40" Stroke="Green"
          StrokeThickness ="10" />
        <gauge:CircularRange StartValue="40" EndValue="60" Stroke="Yellow"
          StrokeThickness ="10" />
        <gauge:CircularRange StartValue="60" EndValue="100" Stroke="Red"
          StrokeThickness ="10"/>
      </gauge:CircularScale.Ranges>
    </gauge:CircularScale>
    <gauge:CircularScale.Pointers>
      <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
    </gauge:CircularScale.Pointers>
  </gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

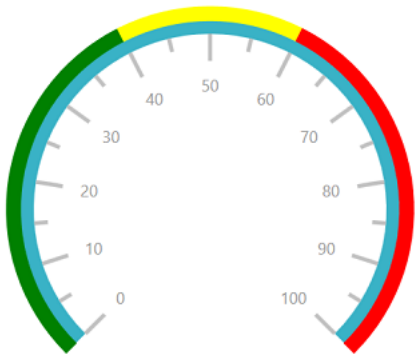
```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RangePosition = RangePosition.Outside;
CircularRange circularRange = new CircularRange();
circularRange.StartValue = 0;
circularRange.EndValue = 40;
circularRange.Stroke = new SolidColorBrush(Colors.Green);
circularRange.StrokeThickness = 10;
mainscale.Ranges.Add(circularRange);
CircularRange circularRange1 = new CircularRange();
circularRange1.StartValue = 40;
circularRange1.EndValue = 60;
circularRange1.Stroke = new SolidColorBrush(Colors.Yellow);
```



```

circularRange1.StrokeThickness = 10;
mainscale.Ranges.Add(circularRange1);
CircularRange circularRange2 = new CircularRange();
circularRange2.StartValue = 60;
circularRange2.EndValue = 100;
circularRange2.Stroke = new SolidColorBrush(Colors.Red);
circularRange2.StrokeThickness = 10;
mainscale.Ranges.Add(circularRange2);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;

```



#### By setting range offset

The range can be placed inside the scale, outside the scale, or on the scale by using the following two ways:

1. Setting `Offset` and `StrokeThickness` properties.
2. Setting `InnerStartOffset`, `InnerEndOffset`, `OuterStartOffset`, and `OuterEndOffset` properties.

---

**Note:** For using this feature need to set the `RangePosition` as custom in the `Rim`.

---

#### By setting `Offset` and `StrokeThickness`

For absolute position, you can use `Offset` and `StrokeThickness` properties of `Range`. For setting the `Offset` and `StrokeThickness` to the range.

#### XML

```

<gauge:SfCircularGauge x:Name="gauge">
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale x:Name="scale" RangePosition="Custom"
      RimStroke="LightGray">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
      <gauge:CircularScale.Ranges>
        <gauge:CircularRange StartValue = "0" EndValue = "100"
          Stroke="MediumTurquoise" Offset = "0.3"
          StrokeThickness="20">

```

```

</gauge:CircularRange>
</gauge:CircularScale.Ranges>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

### C#

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RangePosition = RangePosition.Custom;
CircularRange circularRange = new CircularRange();
circularRange.StartValue = 0;
circularRange.EndValue = 100;
circularRange.Stroke = new SolidColorBrush(Colors.MediumTurquoise);
circularRange.StrokeThickness = 20;
circularRange.Offset = 0.3;
mainscale.Ranges.Add(circularRange);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;

```



By setting `InnerStartOffset`, `InnerEndOffset`, `OuterStartOffset`, and `OuterEndOffset`

For relative position, you can use `InnerStartOffset`, `InnerEndOffset`, `OuterStartOffset`, and `OuterEndOffset` properties of `Range`. This positions is responsive for all the window size.

### XML

```

<gauge:SfCircularGauge x:Name="gauge">
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale x:Name="scale" RangePosition="Custom"
      RimStroke="LightGray">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
      <gauge:CircularScale.Ranges>
        <gauge:CircularRange StartValue="10" EndValue="80" InnerStartOffset = "0.83"
          InnerEndOffset = "0.6" OuterStartOffset = "0.85"
          OuterEndOffset = "0.8" Stroke="MediumTurquoise">
        </gauge:CircularRange>
      </gauge:CircularScale.Ranges>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

```
</gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
CircularScale mainscale = new CircularScale();  
mainscale.RimStroke = new SolidColorBrush(Colors.LightGray);  
mainscale.RangePosition = RangePosition.Custom;  
CircularRange circularRange = new CircularRange();  
circularRange.StartValue = 10;  
circularRange.EndValue = 80;  
circularRange.Stroke = new SolidColorBrush(Colors.MediumTurquoise);  
circularRange.InnerStartOffset = 0.83;  
circularRange.InnerEndOffset = 0.6;  
circularRange.OuterStartOffset = 0.85;  
circularRange.OuterEndOffset = 0.8;  
mainscale.Ranges.Add(circularRange);  
CircularPointer circularPointer = new CircularPointer();  
circularPointer.NeedlePointerVisibility = Visibility.Hidden;  
mainscale.Pointers.Add(circularPointer);  
sfCircularGauge.Scales.Add(mainscale);  
this.Content = sfCircularGauge;
```



### Setting a multiple ranges

In addition to the default range, you can add n number of ranges to a scale by using the [Ranges](#) property.

**XML**

```
<gauge:SfCircularGauge>
<gauge:SfCircularGauge.Scales>
<gauge:CircularScale RangePosition="Inside" Radius="200">
<gauge:CircularScale.Ranges>
<gauge:CircularRange StartValue="0" EndValue="40"
Stroke="Green" StrokeThickness="10"/>
<gauge:CircularRange StartValue="40" EndValue="60"
Stroke="Yellow" StrokeThickness="10"/>
<gauge:CircularRange StartValue="60" EndValue="100"
Stroke="Red" StrokeThickness="10"/>
</gauge:CircularScale.Ranges>
<gauge:CircularScale.Pointers>
<gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RangePosition = RangePosition.Inside;
CircularRange circularRange = new CircularRange();
circularRange.StartValue = 0;
circularRange.EndValue = 40;
circularRange.Stroke = new SolidColorBrush(Colors.Green);
circularRange.StrokeThickness = 10;
circularRange.StartWidth = 1;
circularRange.EndWidth = 10;
mainscale.Ranges.Add(circularRange);
CircularRange circularRange1 = new CircularRange();
circularRange1.StartValue = 40;
circularRange1.EndValue = 60;
circularRange1.Stroke = new SolidColorBrush(Colors.Yellow);
circularRange1.StrokeThickness = 10;
circularRange1.StartWidth = 1;
circularRange1.EndWidth = 10;
mainscale.Ranges.Add(circularRange1);
CircularRange circularRange2 = new CircularRange();
circularRange2.StartValue = 60;
circularRange2.EndValue = 100;
circularRange2.Stroke = new SolidColorBrush(Colors.Red);
circularRange2.StrokeThickness = 10;
circularRange2.StartWidth = 1;
circularRange2.EndWidth = 10;
mainscale.Ranges.Add(circularRange2);
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
```



#### Setting gradient color for range

You can give color transition to range by specifying the different colors using the [GradientStops](#) property of range. By using the `Value` and `Color` properties of `GradientStops`, you can adjust the color transition range for each color.

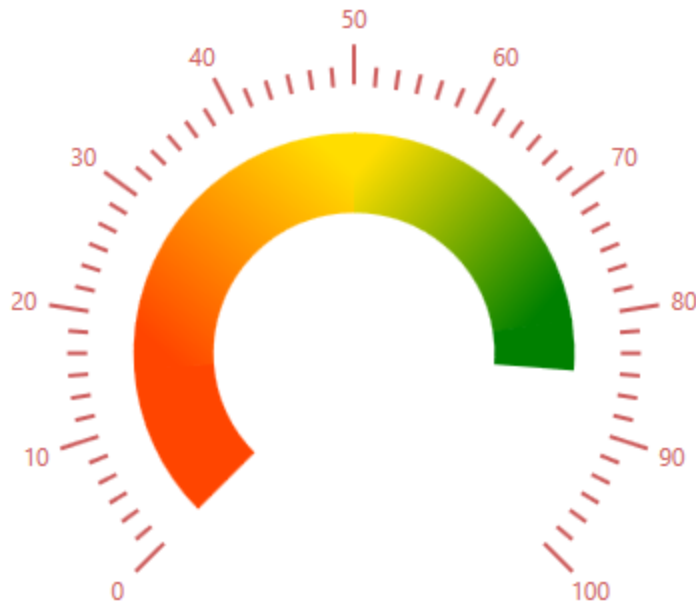
#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale LabelStroke="IndianRed" LabelOffset="0.77"
      LabelPosition="Custom" MinorTicksPerInterval="5"
      RangePosition="Custom" TickPosition="Custom"
      RimStroke="White">
      <gauge:CircularScale.Ranges>
        <gauge:CircularRange StartValue="0" EndValue="85" Offset="0.5"
          StrokeThickness="40" >
          <gauge:CircularRange.GradientStops>
            <gauge:GaugeGradientStop Value="15" Color="OrangeRed"/>
            <gauge:GaugeGradientStop Value="50" Color="#FFDD00"/>
            <gauge:GaugeGradientStop Value="80" Color="Green"/>
          </gauge:CircularRange.GradientStops>
        </gauge:CircularRange>
      </gauge:CircularScale.Ranges>
      <gauge:CircularScale.MajorTickSettings>
        <gauge:MajorTickSetting Length="20" Stroke="IndianRed"
          StrokeThickness="2" Offset="0.7" />
      </gauge:CircularScale.MajorTickSettings>
      <gauge:CircularScale.MinorTickSettings>
        <gauge:MinorTickSetting Stroke="IndianRed" StrokeThickness="2"
          Offset="0.65" />
      </gauge:CircularScale.MinorTickSettings>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

```
<gauge:CircularScale.Pointers>
<gauge:CircularPointer PointerType="NeedlePointer"
Visibility="Collapsed"/>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RimStroke = new SolidColorBrush(Colors.White);
mainscale.LabelStroke = new SolidColorBrush(Colors.IndianRed);
mainscale.LabelOffset = 0.77;
mainscale.LabelPosition = LabelPosition.Custom;
mainscale.MinorTicksPerInterval = 5;
mainscale.RangePosition = RangePosition.Custom;
mainscale.TickPosition = TickPosition.Custom;
CircularRange circularRange = new CircularRange();
circularRange.StartValue = 0;
circularRange.EndValue = 85;
circularRange.Offset = 0.5;
circularRange.StrokeThickness = 40;
ObservableCollection<GaugeGradientStop> gradientColor1 = new
ObservableCollection<GaugeGradientStop>();
GaugeGradientStop gaugeGradientStop = new GaugeGradientStop();
gaugeGradientStop.Value = 15;
gaugeGradientStop.Color = Colors.Green;
circularRange.GradientStops.Add(gaugeGradientStop);
GaugeGradientStop gaugeGradientStop1 = new GaugeGradientStop();
gaugeGradientStop1.Value = 50;
gaugeGradientStop1.Color = Colors.Yellow;
circularRange.GradientStops.Add(gaugeGradientStop1);
GaugeGradientStop gaugeGradientStop2 = new GaugeGradientStop();
gaugeGradientStop2.Value = 80;
gaugeGradientStop2.Color = Colors.Red;
circularRange.GradientStops.Add(gaugeGradientStop2);
mainscale.Ranges.Add(circularRange);
MajorTickSetting majorTickSetting = new MajorTickSetting();
majorTickSetting.Length = 20;
majorTickSetting.Stroke = new SolidColorBrush(Colors.IndianRed);
majorTickSetting.StrokeThickness = 2;
majorTickSetting.Offset = 0.7;
mainscale.MajorTickSettings = majorTickSetting;
MinorTickSetting minorTickSetting = new MinorTickSetting();
minorTickSetting.Stroke = new SolidColorBrush(Colors.IndianRed);
minorTickSetting.StrokeThickness = 2;
minorTickSetting.Offset = 0.65;
mainscale.MinorTickSettings = minorTickSetting;
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



### Pointers in WPF Radial Gauge (SfCircularGauge)

[Pointers](#) are used to indicate values on the scale. Pointer value can be modified using the [Value](#) property.

There are three types of [Pointers](#). You can choose a pointer using the [PointerType](#) property.

#### Needle pointer

Indicate a current value by using the highly customizable needle-type element. A needle pointer contains three parts, a needle, knob and tail that can be placed on a gauge to mark values.

#### XML

```
<gauge:SfCircularGauge >  
  <gauge:SfCircularGauge.Scales>  
    <gauge:CircularScale >  
      <gauge:CircularScale.Pointers>  
        <gauge:CircularPointer PointerType="NeedlePointer"/>  
      </gauge:CircularScale.Pointers>  
    </gauge:CircularScale>  
  </gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
CircularScale mainscale = new CircularScale();  
CircularPointer circularPointer = new CircularPointer();  
circularPointer.PointerType = PointerType.NeedlePointer;  
mainscale.Pointers.Add(circularPointer);  
sfCircularGauge.Scales.Add(mainscale);
```



#### *Different types of needle pointer*

The appearance of the needle pointer can be customized using the [NeedlePointerType](#) property. The default value of this property is Rectangle.

The `NeedlePointerType` is an `enum` property that includes the following options:

1. Rectangle 2. Triangle 3. Tapered 4. Arrow

#### **XML**

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="NeedlePointer"
          NeedlePointerType="Triangle"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### **C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
circularPointer.PointerType = PointerType.NeedlePointer;
circularPointer.NeedlePointerType = NeedlePointerType.Triangle;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```





#### *Needle pointer customization*

The length of a needle is controlled using the [NeedleLengthFactor](#) property. The minimum and maximum bounds of the [NeedleLengthFactor](#) property is 0 to 1. The needle's UI is customized using the [NeedlePointerStroke](#) and [NeedlePointerStrokeThickness](#) properties. The size of the pointer cap can be modified by changing the [PointerCapDiameter](#) property.

#### **XML**

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="NeedlePointer"
          NeedlePointerType="Triangle"
          NeedlePointerStroke="DeepSkyBlue" PointerCapDiameter="20"
          KnobStroke="DeepSkyBlue"
          KnobFill="DeepSkyBlue"
          NeedleLengthFactor="0.5" NeedlePointerStrokeThickness="10"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### **C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
```

```

circularPointer.PointerType = PointerType.NeedlePointer;
circularPointer.NeedlePointerType = NeedlePointerType.Triangle;
circularPointer.NeedlePointerStroke = new
SolidColorBrush(Colors.DeepSkyBlue);
circularPointer.NeedlePointerStrokeThickness = 10;
circularPointer.NeedleLengthFactor = 0.5;
circularPointer.PointerCapDiameter = 20;
circularPointer.KnobStroke = new SolidColorBrush(Colors.DeepSkyBlue);
circularPointer.KnobFill = new SolidColorBrush(Colors.DeepSkyBlue);
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);

```



#### *Needle pointer knob customization*

You can able to customize the needle pointer knob size, fill, stroke and stroke thickness by using [KnobRadiusFactor](#), [KnobFill](#), [KnobStroke](#), [KnobStrokeThickness](#) properties.

#### **XML**

```

<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="NeedlePointer"
KnobStrokeThickness="5"
KnobRadiusFactor="0.2"
KnobStroke="DeepSkyBlue"
KnobFill="White"
Value="60"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

```
</gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
CircularScale mainscale = new CircularScale();  
CircularPointer circularPointer = new CircularPointer();  
circularPointer.PointerType = PointerType.NeedlePointer;  
circularPointer.KnobStroke = new SolidColorBrush(Colors.DeepSkyBlue);  
circularPointer.KnobRadiusFactor = 0.2;  
circularPointer.KnobStrokeThickness = 5;  
circularPointer.KnobFill = new SolidColorBrush(Colors.White);  
circularPointer.Value = 60;  
mainscale.Pointers.Add(circularPointer);  
sfCircularGauge.Scales.Add(mainscale);
```



#### *Setting visibility of needle pointer*

The visibility of the needle pointer can be set using the [NeedlePointerVisibility](#) property.

### XML

```
<gauge:SfCircularGauge >  
<gauge:SfCircularGauge.Scales>  
<gauge:CircularScale >
```

```

<gauge:CircularScale.Pointers>
<gauge:CircularPointer NeedlePointerVisibility="Hidden"/>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

**C#**

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
circularPointer.NeedlePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);

```

*Setting tail for needle pointer*

Tail of the needle pointer can be customized by using the [TailFill](#), [TailLengthFactor](#), [TailStroke](#), and [TailStrokeThickness](#) properties.

- **TailFill** - Fill color to needle pointer's tail.
- **TailLengthFactor** - Length factor of needle pointer's tail. It's range is 0 to 1.
- **TailStroke** - Stroke to needle pointer's tail.
- **TailStrokeThickness** - Set the stroke thickness to needle pointer's tail.

**XML**

```

<gauge:SfCircularGauge >
<gauge:SfCircularGauge.Scales>
<gauge:CircularScale >
<gauge:CircularScale.Pointers>
<gauge:CircularPointer
Value="30"
PointerType="NeedlePointer"
TailLengthFactor="0.3"
TailStroke="#ed7d31"
TailFill="#ed7d31"
NeedlePointerStrokeThickness="3"
NeedlePointerType="Tapered"
KnobFill="White"
NeedlePointerVisibility="Visible"
NeedlePointerStroke="#ed7d31" />
<gauge:CircularPointer
Value="50"
PointerType="NeedlePointer"
TailLengthFactor="0.4"
TailStroke="#ed7d31"
TailFill="#ed7d31"
KnobFill="White"
NeedlePointerStrokeThickness="3"
NeedlePointerType="Tapered"
NeedlePointerVisibility="Visible"
NeedlePointerStroke="#ed7d31" />

```

```
</gauge:CircularScale.Pointers>  
</gauge:CircularScale>  
</gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
CircularScale mainscale = new CircularScale();  
CircularPointer circularPointer = new CircularPointer();  
circularPointer.PointerType = PointerType.NeedlePointer;  
circularPointer.Value = 30;  
circularPointer.TailLengthFactor = 0.3;  
circularPointer.TailStroke = new SolidColorBrush(Color.FromRgb(237, 125, 49));  
circularPointer.TailFill = new SolidColorBrush(Color.FromRgb(237, 125, 49));  
circularPointer.NeedlePointerStrokeThickness = 3;  
circularPointer.NeedlePointerType = NeedlePointerType.Tapered;  
circularPointer.KnobFill = new SolidColorBrush(Colors.White);  
circularPointer.NeedlePointerVisibility = Visibility.Visible;  
circularPointer.NeedlePointerStroke = new SolidColorBrush(Color.FromRgb(237, 125, 49));  
mainscale.Pointers.Add(circularPointer);  
CircularPointer circularPointer2 = new CircularPointer();  
circularPointer2.PointerType = PointerType.NeedlePointer;  
circularPointer2.Value = 50;  
circularPointer2.TailLengthFactor = 0.3;  
circularPointer2.TailStroke = new SolidColorBrush(Color.FromRgb(237, 125, 49));  
circularPointer2.TailFill = new SolidColorBrush(Color.FromRgb(237, 125, 49));  
circularPointer2.NeedlePointerStrokeThickness = 3;  
circularPointer2.NeedlePointerType = NeedlePointerType.Tapered;  
circularPointer2.KnobFill = new SolidColorBrush(Colors.White);  
circularPointer2.NeedlePointerVisibility = Visibility.Visible;  
circularPointer2.NeedlePointerStroke = new SolidColorBrush(Color.FromRgb(237, 125, 49));  
mainscale.Pointers.Add(circularPointer2);  
sfCircularGauge.Scales.Add(mainscale);
```



### Range pointer

A range pointer is used to indicate the current value relative to the start value of a circular scale.

### XML

```
<gauge:SfCircularGauge >  
  <gauge:SfCircularGauge.Scales>  
    <gauge:CircularScale >  
      <gauge:CircularScale.Pointers>  
        <gauge:CircularPointer PointerType="RangePointer" Value="50" />  
      </gauge:CircularScale.Pointers>  
    </gauge:CircularScale>  
  </gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
CircularScale mainscale = new CircularScale();  
CircularPointer circularPointer = new CircularPointer();  
circularPointer.PointerType = PointerType.RangePointer;  
circularPointer.Value = 50;  
mainscale.Pointers.Add(circularPointer);  
sfCircularGauge.Scales.Add(mainscale);
```



### Range pointer customization

The range pointer appearance is customized using the [RangePointerStroke](#) and [RangePointerStrokeThickness](#) properties.

### XML

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="RangePointer" Value="50"
          RangePointerStroke="DarkCyan" RangePointerStrokeThickness="20"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
circularPointer.PointerType = PointerType.RangePointer;
circularPointer.Value = 50;
circularPointer.RangePointerStroke = new SolidColorBrush(Colors.DarkCyan);
circularPointer.RangePointerStrokeThickness = 20;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



*Setting visibility for range pointer*

The [RangePointerVisibility](#) property is used to set the visibility of the range pointer.

**XML**

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer RangePointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

**C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
circularPointer.RangePointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```

*Setting position for range pointer*

The [RangePointer](#) can be positioned anywhere in the gauge by the following ways:

1. Setting direct [RangePointerPosition](#) property to place pointer inside or outside the gauge.



2. Setting `Offset` and `RangePointerStrokeThickness` properties. This positions is responsive for all the window size. But `RangePointer` width is fixed.
3. Setting `RangeStartOffset` and `RangeEndOffset` properties. This position and width are responsive to all size of the window.

#### Setting direct range pointer position

You can directly place the range pointer inside, outside or cross the scale by selecting one of the below option available in the [RangePointerPosition](#) property.

1. Inside (Default) 2. Outside 3. Cross 4. Custom

#### XML

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale RangePointerPosition="Outside">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="RangePointer"
          RangePointerStroke="HotPink" Value="60"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RangePointerPosition = RangePointerPosition.Outside;
CircularPointer circularPointer = new CircularPointer();
circularPointer.PointerType = PointerType.RangePointer;
circularPointer.RangePointerStroke = new SolidColorBrush(Colors.HotPink);
circularPointer.Value = 60;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



#### Setting offset range pointer position with fixed width

This way allows to place the range pointer anywhere inside the gauge. First, set the [RangePointerPosition](#) to custom, and then set the [Offset](#) property for required position. `Offset` value

should be from 0 to 1. This positions is responsive for all the window size. But **RangePointer** width is fixed.

#### XML

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale x:Name="scale" RangePointerPosition="Custom">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="RangePointer" Offset="0.5"
          RangePointerStrokeThickness="20"
          RangePointerStroke="LightGray" Value="60"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RangePointerPosition = RangePointerPosition.Custom;
CircularPointer circularPointer = new CircularPointer();
circularPointer.PointerType = PointerType.RangePointer;
circularPointer.Offset = 0.5;
circularPointer.RangePointerStroke = new SolidColorBrush(Colors.LightGray);
circularPointer.RangePointerStrokeThickness = 20;
circularPointer.Value = 60;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



Setting start and end position of the range pointer with offset

[RangeStartOffset](#) and [RangeEndOffset](#) properties with [RangePointerPosition](#) is custom allows to place the range pointer anywhere inside the gauge. First, set the [RangePointerPosition](#) to custom, and then set the [RangeStartOffset](#) and [RangeEndOffset](#) properties for required position. Offset value should be from 0 to 1. This position and width are responsive to all size of the window.

#### XML

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale x:Name="scale" RangePointerPosition="Custom">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="RangePointer"
          RangeStartOffset="0.5" RangeEndOffset="0.7"
          RangePointerStroke="LightGray" Value="90"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RangePointerPosition = RangePointerPosition.Custom;
CircularPointer circularPointer = new CircularPointer();
circularPointer.PointerType = PointerType.RangePointer;
circularPointer.RangeStartOffset = 0.5;
circularPointer.RangeEndOffset = 0.7;
```

```
circularPointer.RangePointerStroke = new SolidColorBrush(Colors.LightGray);
circularPointer.Value = 90;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



#### *Range pointer start customization*

The [RangeStart](#) property used to customize the range pointer start position in scale.

#### **XML**

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale x:Name="scale" RangePointerPosition="Custom">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="RangePointer"
          RangeStart="20"
          RangeStartOffset="0.5" RangeEndOffset="0.7"
          RangePointerStroke="LightGray" Value="90"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### **C#**

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RangePointerPosition = RangePointerPosition.Custom;
CircularPointer circularPointer = new CircularPointer();
```

```

circularPointer.PointerType = PointerType.RangePointer;
circularPointer.RangeStartOffset = 0.5;
circularPointer.RangeEndOffset = 0.7;
circularPointer.RangeStart = 20;
circularPointer.RangePointerStroke = new SolidColorBrush(Colors.LightGray);
circularPointer.Value = 90;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);

```



#### Setting range cap for range pointer

The [RangeCap](#) property provides options to position the range cap of the RangePointer, which contains the start, end, both, and none options. The [RangeCap](#) property is an enum property.

#### XML

```

<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale x:Name="scale" SweepAngle="360" RimStroke="LightGray"
      RimStrokeThickness="30" RangePointerPosition="Custom">
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="RangePointer"
          RangePointerStrokeThickness="30" RangeCap="Both"
          RangePointerStroke="LightSkyBlue" Value="75"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

#### C#

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RangePointerPosition = RangePointerPosition.Custom;
mainscale.SweepAngle = 360;
mainscale.RimStroke = new SolidColorBrush(Colors.LightGray);
mainscale.RimStrokeThickness = 30;
CircularPointer circularPointer = new CircularPointer();
circularPointer.PointerType = PointerType.RangePointer;
circularPointer.RangePointerStrokeThickness = 30;
circularPointer.RangeCap = RangeCap.Both;
circularPointer.RangePointerStroke = new
SolidColorBrush(Colors.LightSkyBlue);
circularPointer.Value = 75;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);

```



### Symbol pointer

In symbol pointer, the value is pointed out using a symbol on the scale. The symbol is an enum property that provides symbol options for the symbol pointer, which contains several shapes such as rectangle, ellipse, and triangle.

### XML

```

<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="SymbolPointer" Value="60"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

```
</gauge:CircularScale.Pointers>  
</gauge:CircularScale>  
</gauge:SfCircularGauge.Scales>  
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();  
CircularScale mainscale = new CircularScale();  
CircularPointer circularPointer = new CircularPointer();  
circularPointer.PointerType = PointerType.SymbolPointer;  
circularPointer.Value = 60;  
mainscale.Pointers.Add(circularPointer);  
sfCircularGauge.Scales.Add(mainscale);
```



### Symbol pointer types

You can modify the symbol pointer view by choosing available options in [Symbol](#) property. Please find the types of symbol below.

1. Arrow
2. Cross
3. Custom
4. Diamond
5. Ellipse
6. Hexagon
7. InvertedArrow
8. InvertedTriangle
9. Pentagon
10. Rectangle
11. RoundedRectangle
12. Triangle

## XML

```
<gauge:SfCircularGauge >  
<gauge:SfCircularGauge.Scales>  
<gauge:CircularScale>  
<gauge:CircularScale.Pointers>  
<gauge:CircularPointer PointerType="SymbolPointer"
```

```

Symbol="Pentagon"
Value="60"/>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

**C#**

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
circularPointer.PointerType = PointerType.SymbolPointer;
circularPointer.Symbol = Symbol.Pentagon;
circularPointer.Value = 60;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
grid.Children.Add(sfCircularGauge);

```

*Setting custom symbol pointer*

You can customize the **SymbolPointer** by setting **Symbol** as **Custom** and defining **SymbolTemplate**. Please find the custom template section below.

**XML**

```

<gauge:SfCircularGauge >
<gauge:SfCircularGauge.Scales >
<gauge:CircularScale x:Name="scale"
RangePointerPosition="Custom">

```



```

<gauge:CircularScale.Pointers>
<gauge:CircularPointer PointerType="SymbolPointer"
Symbol="Custom"
Value="60">
<gauge:CircularPointer.SymbolPointerTemplate>
<DataTemplate>
<Grid>
<Grid Name="backgroundGrid" Width="24" Height="24" Visibility="Visible">
<Ellipse Fill="White" Name="Fill" Visibility="Visible" />
</Grid>
<Path Stretch="Uniform" Fill="Black" Width="24" Height="24"
Margin="0,0,0,0" RenderTransformOrigin="0.5,0.5">
<Path.Data>
<PathGeometry FillRule="Nonzero"
Figures="M23.9296875,10.6165618896484L20.759765625,11.2200794219971
18.09375,
13.0306243896484 16.283203125,15.6966400146484 15.6796875,18.8665618896484
16.283203125,
22.0423431396484 18.09375,24.7259368896484 20.759765625,26.5540618896484
23.9296875,27.1634368896484 27.1025371551514,
26.5540618896484 29.77734375,24.7259368896484 31.5966796875,22.0423431396484
32.203125,18.8665618896484 31.5966796875,
15.6966400146484 29.77734375,13.0306243896484
27.1025371551514,11.2200794219971 23.9296875,10.6165618896484z M25.265625,
7.3587493896484L26.6953125,9.8665618896484 29.3671875,8.6478118896484
29.765625,11.4837493896484 32.7421875,
11.2728118896484 32.015625,14.1790618896484 34.921875,14.9759368896484
33.1875,17.4134368896484 35.578125,
19.1478118896484 33.140625,20.7884368896484 34.640625,23.3665618896484
31.8046875,23.9759368896484 32.3203125,
26.9759368896484 29.4375,26.5540618896484 28.921875,29.4837493896484
26.25,27.9603118896484 24.75,
30.4681243896484 22.8046875,28.2181243896484 20.5078125,30.0228118896484
19.5703125,27.1634368896484 16.640625,
28.0306243896484 16.875,25.1009368896484 13.875,24.7728118896484
15.140625,22.1478118896484 12.421875,
20.7415618896484 14.5546875,18.6790618896484 12.4921875,16.5228118896484
15.2578125,15.3040618896484 14.203125,
12.5384368896484 17.1328125,12.3978118896484 17.1328125,9.4212493896484
19.921875,10.4056243896484 21.046875,
7.6165618896484 23.296875,9.4915618896484 25.265625,7.3587493896484z" />
</Path.Data>
<Path.RenderTransform>
<TransformGroup>
<TransformGroup.Children>
<RotateTransform Angle="0" />
<ScaleTransform ScaleX="1" ScaleY="1" />
</TransformGroup.Children>
</TransformGroup>
</Path.RenderTransform>
</Path>
</Grid>
</DataTemplate>
</gauge:CircularPointer.SymbolPointerTemplate>
</gauge:CircularPointer>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>

```

```
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```



#### Symbol pointer customization

You can modify the stroke of the symbol by changing the [SymbolPointerStroke](#) property. The [SymbolPointerHeight](#) property is used to set the height of the symbol pointer. The value should be given as a double value. The [SymbolPointerWidth](#) property is used to set the width of the symbol pointer.

#### XML

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="SymbolPointer" Value="60"
          SymbolPointerStroke="Red"
          SymbolPointerHeight="20" SymbolPointerWidth="20"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
circularPointer.PointerType = PointerType.SymbolPointer;
circularPointer.Value = 60;
```

```
circularPointer.SymbolPointerStroke = new SolidColorBrush(Colors.Red);
circularPointer.SymbolPointerHeight = 20;
circularPointer.SymbolPointerWidth = 20;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



#### Setting visibility of symbol pointer

The visibility of the symbol pointer can be set using the [SymbolPointerVisibility](#) Property.

#### XML

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer SymbolPointerVisibility="Hidden"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
circularPointer.SymbolPointerVisibility = Visibility.Hidden;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```

#### Setting offset for SymbolPointers

The [Offset](#) property in the CircularPointer can be placed SymbolPointer in desired position of rim.

#### XML

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="SymbolPointer"
          Offset="0.5" Symbol="Triangle" Value="60"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

```

</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

### C#

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
circularPointer.PointerType = PointerType.SymbolPointer;
circularPointer.Value = 60;
circularPointer.Symbol = Symbol.Triangle;
circularPointer.Offset = 0.5;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
CircularPointer circularPointer1 = new CircularPointer();
circularPointer1.NeedlePointerVisibility = Visibility.Hidden;
circularPointer1.PointerType = PointerType.NeedlePointer;
mainscale.Pointers.Add(circularPointer1);

```



### Setting multiple pointers

In addition to the default pointer, you can add n number of [Pointers](#) to a scale using the [Pointers](#) property.

### XML

```

<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="NeedlePointer"
          NeedleLengthFactor="0.4" Value="60"
          NeedlePointerType="Tapered" KnobStroke="#39B2C6" />
        <gauge:CircularPointer PointerType="RangePointer" Value="100"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale >
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

```
<gauge:CircularPointer PointerType="SymbolPointer" Value="50"
Symbol="Pentagon"
SymbolPointerHeight="20" SymbolPointerWidth="20"
SymbolPointerStroke="#39B2C6"/>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

## C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
circularPointer.PointerType = PointerType.NeedlePointer;
circularPointer.NeedleLengthFactor = 0.4;
circularPointer.Value = 60;
circularPointer.NeedlePointerType = NeedlePointerType.Tapered;
circularPointer.KnobStroke = new SolidColorBrush(Color.FromArgb(0xff, 0x39,
0xb2, 0xc6));
mainscale.Pointers.Add(circularPointer);
CircularPointer circularPointer1 = new CircularPointer();
circularPointer1.PointerType = PointerType.RangePointer;
circularPointer1.Value = 100;
mainscale.Pointers.Add(circularPointer1);
CircularPointer circularPointer2 = new CircularPointer();
circularPointer2.PointerType = PointerType.SymbolPointer;
circularPointer2.Value = 50;
circularPointer2.Symbol = Symbol.Pentagon;
circularPointer2.SymbolPointerHeight = 20;
circularPointer2.SymbolPointerWidth = 20;
circularPointer2.SymbolPointerStroke = new
SolidColorBrush(Color.FromArgb(0xff, 0x39, 0xb2, 0xc6));
mainscale.Pointers.Add(circularPointer2);
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```



### Setting animation for pointer

The [EnableAnimation](#) property is a Boolean property that enables or disables the animation of the [Pointers](#) in circular gauge and defining the corresponding animation speed with [AnimationDuration](#) property.

### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale Radius="150" >
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer EnableAnimation="True" PointerType="NeedlePointer"
          AnimationDuration="500" NeedleLengthFactor="0.4" Value="60"
          NeedlePointerType="Triangle" KnobStroke="#39B2C6" PointerCapDiameter="20"/>
        <gauge:CircularPointer PointerType="RangePointer" AnimationDuration="1000"
          Value="100" RangePointerStroke="#39B2C6"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.Radius = 150;
CircularPointer circularPointer = new CircularPointer();
circularPointer.PointerType = PointerType.NeedlePointer;
circularPointer.EnableAnimation = true;
```

```
circularPointer.NeedleLengthFactor = 0.4;
circularPointer.AnimationDuration = 500;
circularPointer.Value = 60;
circularPointer.NeedlePointerType = NeedlePointerType.Triangle;
circularPointer.PointerCapDiameter = 20;
circularPointer.KnobStroke = new SolidColorBrush(Color.FromArgb(0xff, 0x39,
0xb2, 0xc6));
mainscale.Pointers.Add(circularPointer);
CircularPointer circularPointer1 = new CircularPointer();
circularPointer1.PointerType = PointerType.RangePointer;
circularPointer1.RangePointerStroke = new
SolidColorBrush(Color.FromArgb(0xff, 0x39, 0xb2, 0xc6));
circularPointer1.EnableAnimation = true;
circularPointer1.AnimationDuration = 1000;
circularPointer1.Value = 100;
mainscale.Pointers.Add(circularPointer1);
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
```

### Circular pointer dragging

Symbol pointer and needle pointer can be dragged over the scale value by setting the [EnableDragging](#) property as true. [Pointers](#) can be moved to the respective position.

### XML

```
<gauge:SfCircularGauge >
  <gauge:SfCircularGauge.Scales >
    <gauge:CircularScale LabelPosition="Custom" Grid.Row="1"
      RimStroke="LightGray"
      RadiusFactor="1" ShowTicks="False"
      RimStrokeThickness="30"
      StartValue="0" EndValue="100" Interval="10"
```

```

LabelOffset="0.75" LabelStroke="Black" FontSize="15">
<gauge:CircularScale.Ranges>
<gauge:CircularRange StrokeThickness="30" StartValue="0"
x:Name="range" EndValue="25"
Stroke="DeepSkyBlue"/>
</gauge:CircularScale.Ranges>
<gauge:CircularScale.Pointers>
<gauge:CircularPointer PointerType="SymbolPointer"
Symbol="InvertedTriangle"
SymbolPointerHeight="18" SymbolPointerWidth="18"
ValueChanged="CircularPointer_ValueChanged"
SymbolPointerStroke="DarkBlue" Value="25"
EnableAnimation="False" EnableDragging="True"/>
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

**C#**

```

public partial class PointerDragging : Window
{
    CircularRange range = new CircularRange();
    public PointerDragging()
    {
        InitializeComponent();
        SfCircularGauge circularGauge = new SfCircularGauge();
        CircularScale scale = new CircularScale();
        scale.RimStroke = new SolidColorBrush(Colors.LightGray);
        scale.RimStrokeThickness = 30;
        scale.RadiusFactor = 1;
        scale.ShowTicks = false;
        scale.StartValue = 0;
        scale.EndValue = 100;
        scale.Interval = 10;
        scale.LabelOffset = 0.75;
        scale.LabelStroke = new SolidColorBrush(Colors.Black);
        scale.FontSize = 15;
        scale.LabelPosition = LabelPosition.Custom;
        scale.RadiusFactor = 1;
        scale.ShowTicks = false ;
        CircularPointer circularPointer = new CircularPointer();
        circularPointer.PointerType = PointerType.SymbolPointer;
        circularPointer.Symbol = Symbol.InvertedTriangle;
        circularPointer.Value = 25;
        circularPointer.SymbolPointerStroke = new SolidColorBrush(Colors.DarkBlue);
        circularPointer.SymbolPointerHeight = 18;
        circularPointer.EnableAnimation = false;
        circularPointer.EnableDragging = true;
        circularPointer.SymbolPointerWidth = 18;
        circularPointer.ValueChanged += CircularPointer_ValueChanged;
        range.StrokeThickness = 30;
        range.StartValue = 0;
        range.EndValue = 25;
        range.Stroke = new SolidColorBrush(Colors.DeepSkyBlue);
        scale.Ranges.Add(range);
    }
}

```



```

scale.Pointers.Add(circularPointer);
circularGauge.Scales.Add(scale);
this.Content = circularGauge;
}
private void CircularPointer_ValueChanged(object sender,
ValueChangedEventArgs e)
{
    range.EndValue = e.Value;
}
}

```



## Events

### *Pointer position changed*

When the pointer position changes, this event is raised. Event arguments contain the following properties:

[PointerValue](#) - Denotes the pointer value.

[PointerValuePosition](#) - Denotes the position of the pointer value.

[RangePointerStartPosition](#) - Denotes the position of the range start value.

[RangeStartValue](#) - Denotes the range start value.

## XML

```

<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer EnableDragging="True"
          PointerPositionChanged="CircularPointer_PointerPositionChanged"
          Value="70" RangeStart="40"

```

```

PointerType="RangePointer" />
</gauge:CircularScale.Pointers>
</gauge:CircularScale>
</gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

**C#**

```

SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
circularPointer.EnableDragging = true;
circularPointer.Value = 70;
circularPointer.RangeStart = 40;
circularPointer.PointerType = PointerType.RangePointer;
circularPointer.PointerPositionChanged +=
CircularPointer_PointerPositionChanged;
mainscale.Pointers.Add(circularPointer);
sfCircularGauge.Scales.Add(mainscale);
this.Content = sfCircularGauge;
private void CircularPointer_PointerPositionChanged(object sender,
PointerPosition pointerPosition)
{
    var pointerValue = pointerPosition.PointerValue;
    var pointerValuePos = pointerPosition.PointerValuePosition;
    var rangePointerPos = pointerPosition.RangePointerStartPosition;
    var rangeStartValue = pointerPosition.RangeStartValue;
}

```

*Value change started*

Called when the user starts updating a new value of pointer by initiating the dragging. While dragging the pointer, other events ([ValueChanging](#), [ValueChanged](#) and [ValueChangeCompleted](#)) will be followed after this event.

The [ValueChangeStarted](#) event contains the following arguments.

**Value** : This will be the pointers' last value before the changes began.

**XML**

```

<gauge:CircularScale.Pointers>
<gauge:CircularPointer EnableDragging="True"
ValueChangeStarted="CircularPointer_ValueChangeStarted"
Value="20" Symbol="InvertedTriangle"
PointerType="SymbolPointer" />
</gauge:CircularScale.Pointers>

```

**C#**

```

SfCircularGauge circularGauge = new SfCircularGauge();
CircularScale scale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
circularPointer.EnableDragging = true;
circularPointer.Value = 20;
circularPointer.Symbol = Symbol.InvertedTriangle;

```

```

circularPointer.PointerType = PointerType.SymbolPointer;
circularPointer.ValueChangeStarted += CircularPointer_ValueChangeStarted;
scale.Pointers.Add(circularPointer);
circularGauge.Scales.Add(scale);
this.Content = circularGauge;
private void CircularPointer_ValueChangeStarted(object sender,
ValueChangedEventArgs e)
{
}

```

### *Value changing event*

Called during a drag when the user is updating before a new value for the pointer by dragging. The [ValueChangingEventArgs](#) contains [OldValue](#), [NewValue](#), and [Cancel](#) properties.

[OldValue](#): Contains pointer old value.

[NewValue](#): Contains pointer new value.

[Cancel](#): To restrict the update of the current drag pointer value, set `ValueChangingArgs.Cancel` is true.

### XML

```

<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer EnableDragging="True"
ValueChanging="CircularPointer_ValueChanging"
Value="20" Symbol="InvertedTriangle"
PointerType="SymbolPointer" />
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>

```

### C#

```

public partial class PointerDragging : Window
{
    public PointerDragging()
    {
        InitializeComponent();
        SfCircularGauge circularGauge = new SfCircularGauge();
        CircularScale scale = new CircularScale();
        CircularPointer circularPointer = new CircularPointer();
        circularPointer.EnableDragging = true;
        circularPointer.Value = 20;
        circularPointer.Symbol = Symbol.InvertedTriangle;
        circularPointer.PointerType = PointerType.SymbolPointer;
        circularPointer.ValueChanging += CircularPointer_ValueChanging;
        scale.Pointers.Add(circularPointer);
        circularGauge.Scales.Add(scale);
        this.Content = circularGauge;
    }
    private void CircularPointer_ValueChanging(object sender,
ValueChangingEventArgs e)

```

```
{
    if (e.NewValue > 30 && e.NewValue < 70)
    {
        e.Cancel = true;
    }
}
```

#### *Value changed event*

Called during a drag when the user is updating a new value for the pointer by dragging. The [ValueChangedEventArgs](#) contains [Value](#) property.

[Value](#) : Contains the drag pointer value.

#### **XML**

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer ValueChanged="CircularPointer_ValueChanged" />
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### **C#**

```
public partial class PointerDragging : Window
{
    public PointerDragging()
    {
        InitializeComponent();
        SfCircularGauge circularGauge = new SfCircularGauge();
        CircularScale scale = new CircularScale();
        CircularPointer circularPointer = new CircularPointer();
        circularPointer.ValueChanged += CircularPointer_ValueChanged;
        scale.Pointers.Add(circularPointer);
        circularGauge.Scales.Add(scale);
        this.Content = circularGauge;
    }
    private void CircularPointer_ValueChanged(object sender,
        ValueChangedEventArgs e)
    {
        var value = e.Value;
    }
}
```

#### *Value change completed*

Called after a new value has been updated by terminating the dragging of the pointer. While dragging the pointer, other events (ValueChangeStarted, ValueChanging and ValueChanged) will be called prior to the [ValueChangeCompleted](#) event.

This event will notify the completion of dragging with a new value being updated.

The [ValueChangeCompleted](#) event contains the following arguments.

[Value](#) : After dragging, this value will be the new updated pointer value.

#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer EnableDragging="True"
          ValueChangeCompleted="CircularPointer_ValueChangeCompleted"
          Value="20" Symbol="InvertedTriangle"
          PointerType="SymbolPointer" />
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge circularGauge = new SfCircularGauge();
CircularScale scale = new CircularScale();
CircularPointer circularPointer = new CircularPointer();
circularPointer.EnableDragging = true;
circularPointer.Value = 20;
circularPointer.Symbol = Symbol.InvertedTriangle;
circularPointer.PointerType = PointerType.SymbolPointer;
circularPointer.ValueChangeCompleted +=
CircularPointer_ValueChangeCompleted;
scale.Pointers.Add(circularPointer);
circularGauge.Scales.Add(scale);
this.Content = circularGauge;
```

See also

[How to create a WPF SfCircularGauge range pointer with rounded corner edge](#)

[How to customize a needle pointer knob in WPF SfCircularGauge](#)

[How to drag the \(symbol & needle\) pointers in the WPF SfCircularGauge](#)

#### Annotations in WPF Radial Gauge (SfCircularGauge)

SfCircularGauge supports annotations, which allows you to mark the specific area of interest in circular gauge. You can place custom views as annotations. The text and images can also be added by using [Annotations](#) property.

##### Setting a view annotation

When the annotation allows you to place the custom elements, a gauge can be initialized to the element, and this can be used to place the annotation in another gauge. The Following properties are used to customize the Annotations:

- [Angle](#) : Used to place the view at the given angle.
- [Offset](#) : Used to move the view from the center to edge of the circular gauge. The value should be ranges from 0 to 1.

- [ViewMargin](#) : Used to customize the Annotation x and y position by using the pixel values.

The following code is used to create the Annotations.

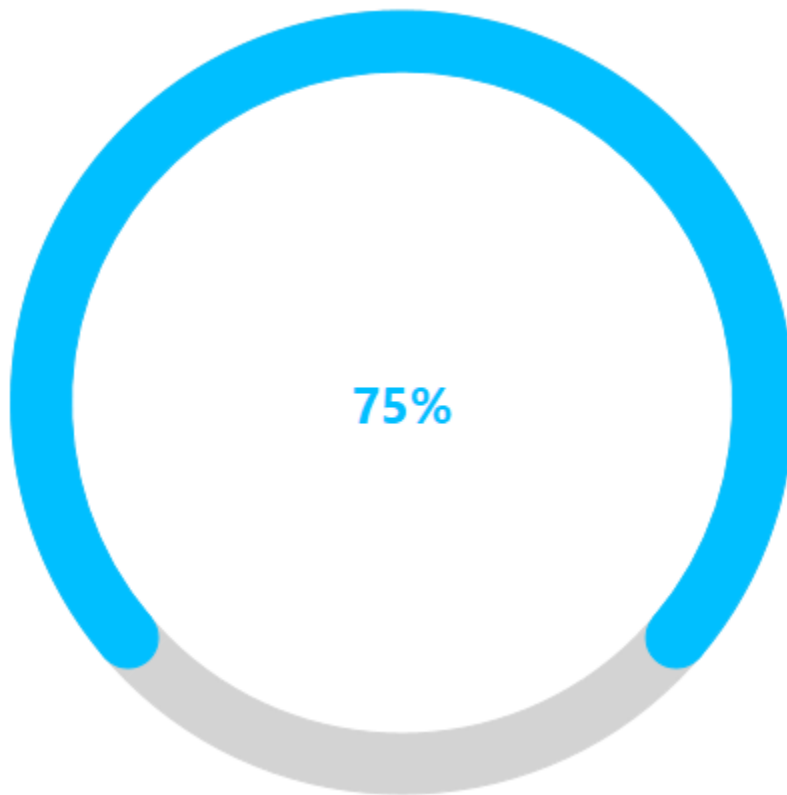
#### XML

```
<gauge:SfCircularGauge>
  <gauge:SfCircularGauge.Annotations>
    <gauge:GaugeAnnotation Offset="0" Angle="270">
      <TextBlock Text="75%" FontSize="25"
        Foreground="DeepSkyBlue" FontWeight="Bold"
        HorizontalAlignment="Center" VerticalAlignment="Center"/>
    </gauge:GaugeAnnotation>
  </gauge:SfCircularGauge.Annotations>
  <gauge:SfCircularGauge.Scales>
    <gauge:CircularScale x:Name="scale" SweepAngle="360" RimStroke="LightGray"
      LabelStroke="Transparent"
      RimStrokeThickness="30" RangePointerPosition="Custom">
      <gauge:CircularScale.MajorTickSettings>
        <gauge:MajorTickSetting Stroke="Transparent"/>
      </gauge:CircularScale.MajorTickSettings>
      <gauge:CircularScale.MinorTickSettings>
        <gauge:MinorTickSetting Stroke="Transparent"/>
      </gauge:CircularScale.MinorTickSettings>
      <gauge:CircularScale.Pointers>
        <gauge:CircularPointer PointerType="RangePointer"
          RangePointerStrokeThickness="30"
          RangeCap="Both"
          RangePointerStroke="DeepSkyBlue" Value="75"/>
      </gauge:CircularScale.Pointers>
    </gauge:CircularScale>
  </gauge:SfCircularGauge.Scales>
</gauge:SfCircularGauge>
```

#### C#

```
SfCircularGauge sfCircularGauge = new SfCircularGauge();
CircularScale mainscale = new CircularScale();
mainscale.RangePointerPosition = RangePointerPosition.Custom;
mainscale.SweepAngle = 360;
mainscale.RimStroke = new SolidColorBrush(Colors.LightGray);
mainscale.RimStrokeThickness = 30;
mainscale.MinorTickSettings.Stroke = new
SolidColorBrush(Colors.Transparent);
mainscale.MajorTickSettings.Stroke = new
SolidColorBrush(Colors.Transparent);
mainscale.LabelStroke = new SolidColorBrush(Colors.Transparent);
TextBlock annotationText = new TextBlock();
annotationText.Text = "75%";
annotationText.FontWeight = FontWeights.Bold;
annotationText.FontSize = 25;
annotationText.Foreground = new SolidColorBrush(Colors.DeepSkyBlue);
annotationText.HorizontalAlignment = HorizontalAlignment.Center;
annotationText.VerticalAlignment = VerticalAlignment.Center;
GaugeAnnotation gaugeAnnotation = new GaugeAnnotation();
gaugeAnnotation.Content = annotationText;
```

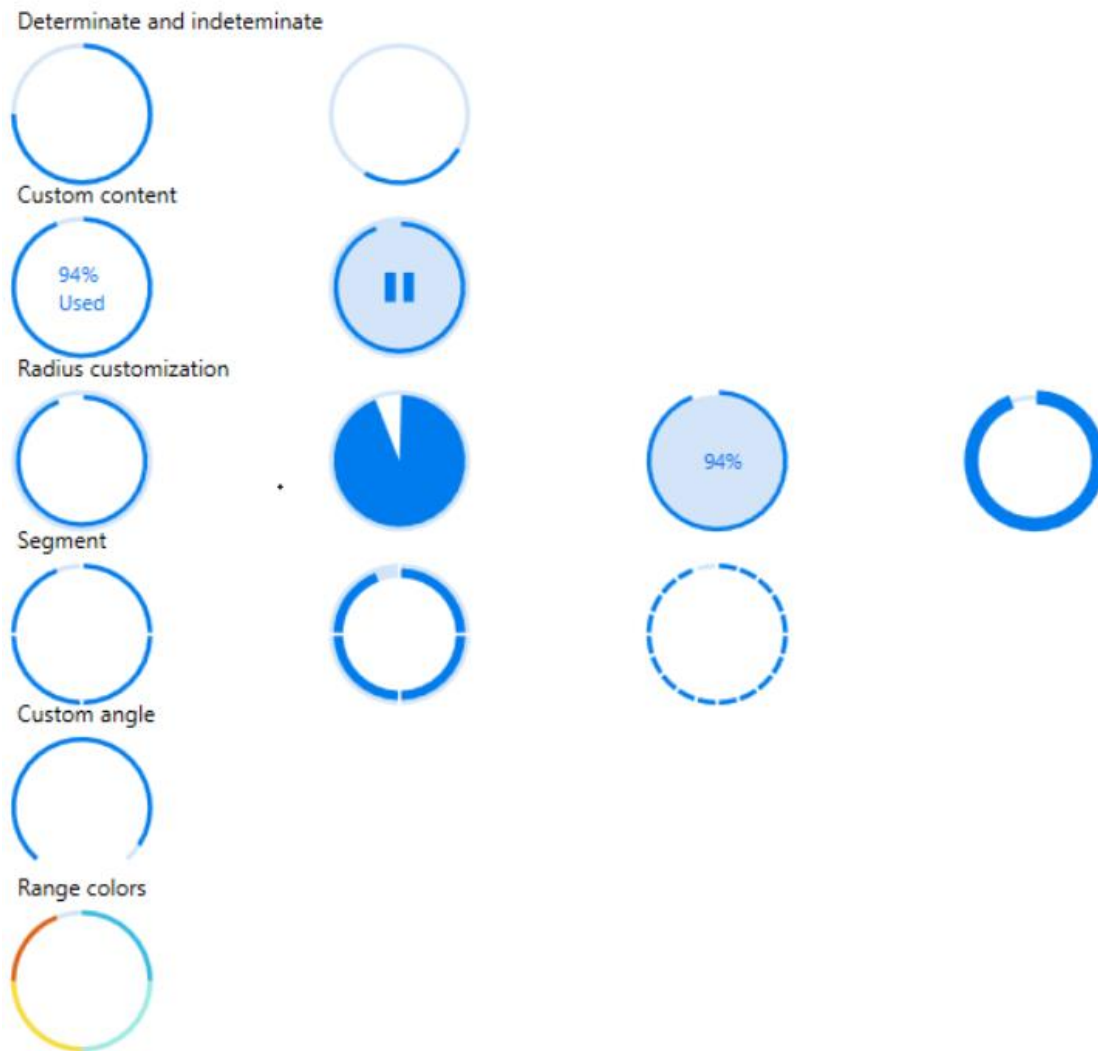
```
gaugeAnnotation.Angle = 270;  
gaugeAnnotation.Offset = 0;  
CircularGaugeAnnotationCollection annotations = new  
CircularGaugeAnnotationCollection();  
annotations.Add(gaugeAnnotation);  
sfCircularGauge.Annotations = annotations;  
CircularPointer circularPointer = new CircularPointer();  
circularPointer.PointerType = PointerType.RangePointer;  
circularPointer.RangePointerStrokeThickness = 30;  
circularPointer.RangeCap = RangeCap.Both;  
circularPointer.RangePointerStroke = new  
SolidColorBrush(Colors.DeepSkyBlue);  
circularPointer.Value = 75;  
mainscale.Pointers.Add(circularPointer);  
sfCircularGauge.Scales.Add(mainscale);  
this.Content = sfCircularGauge;
```



## SfCircularProgressBar

### WPF Circular ProgressBar (SfCircularProgressBar) Overview

The SfCircularProgressBar control indicates the progress of an operation and let users know the remaining time for completion visualizes in circular fashion.



### Key features

- **Determinate and indeterminate:** Determinate shows specific quantity of progress that occurred and indeterminate shows a redundant animations of circular progress.
- **Custom content:** Custom content helps to show the controls progress through user-defined text.
- **Radius customization:** Radius customization helps to change both inner and outer radii of the Circular ProgressBar.
- **Segment:** Segment splits the ProgressBar into multiple segments and indicates the progress.
- **Custom angle:** Custom angle shows the progress drawn to an specific angle for customized appearance.
- **Ranges:** Specifies the start position and end position to visualize multiple ranges with different colors that are mapped to each range.

### Getting Started with WPF Circular ProgressBar (SfCircularProgressBar)

You can create a WPF application with the SfCircularProgressBar control using the following steps:



## Assembly deployment

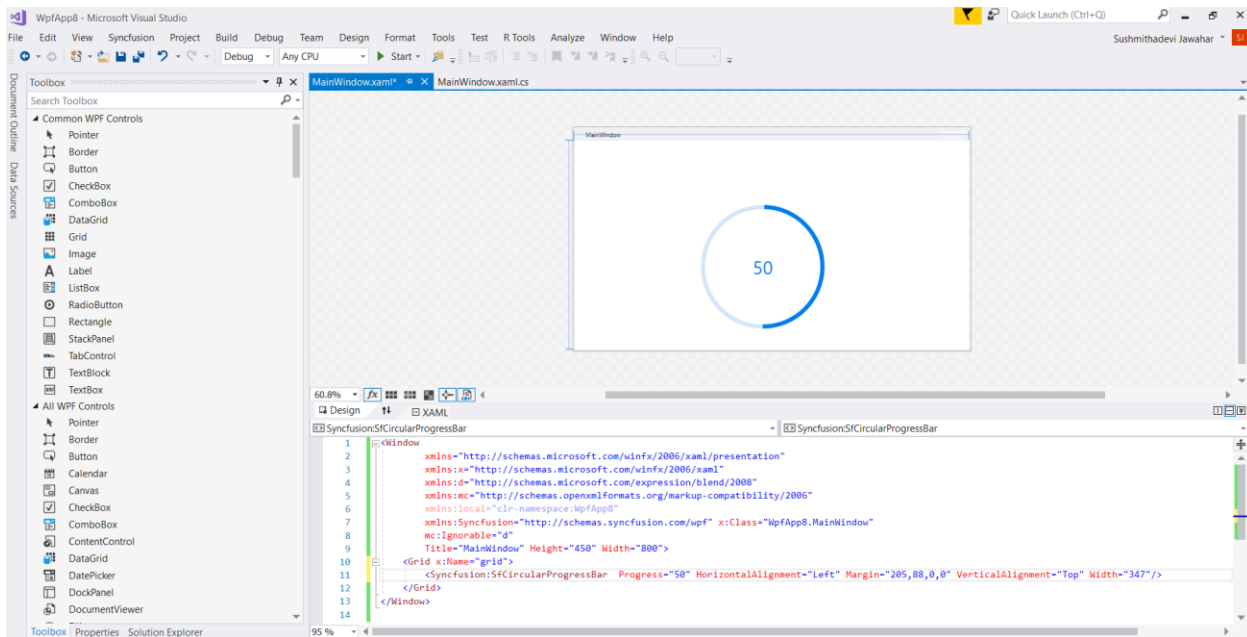
Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

You can find more details about installing the NuGet package in a WPF application in the following link: [How to install nuget packages](#)

## Adding control through designer

The SfCircularProgressBar control can be added to a WPF application by dragging it from the toolbox to a designer view. The following assembly reference will be added automatically:

- Syncfusion.SfProgressBar.WPF



## Adding control manually in XAML

To add control manually in XAML, follow the given steps:

1. Add the following required assembly references to the project: \* Syncfusion.SfProgressBar.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page.
3. Declare the SfCircularProgressBar control in the XAML page.

## XAML

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp4"
xmlns:Syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="WpfApp4.MainWindow"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
    <Grid x:Name="grid">
        <Syncfusion:SfCircularProgressBar Progress="50" Width="347"/>
    </Grid>
</Window>
```

```
</Syncfusion:SfCircularProgressBar>
</Grid>
</Window>
```

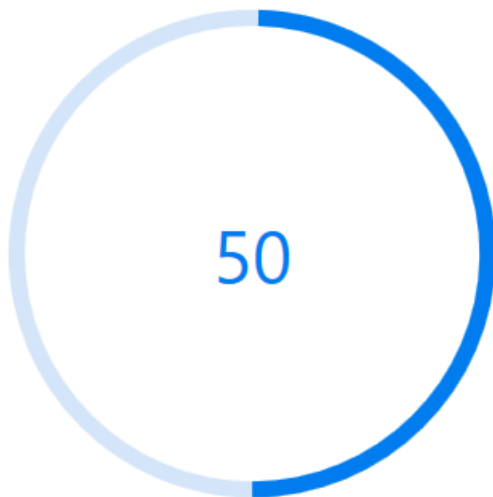
Adding control through code behind

To add control manually through code behind, follow the given steps:

1. Add the following required assembly references to the project: \* Syncfusion.SfProgressBar.WPF 2. Import the Circular ProgressBar namespace **using Syncfusion.UI.Xaml.ProgressBar;** 3. Create an Circular ProgressBar instance, and add it to the window.

#### C#

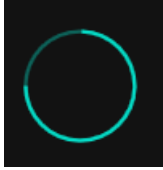
```
using Syncfusion.UI.Xaml.ProgressBar;
namespace SfProgressBar
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfCircularProgressBar circular = new SfCircularProgressBar();
            circular.Progress = 50;
            grid.Children.Add(circular);
        }
    }
}
```



#### Theme

Circular ProgressBar supports various built-in themes. Refer to the below links to apply themes for the Circular ProgressBar,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### States in WPF circular progressbar (SfCircularProgressBar)

States help to visualize the progress of a task in different modes. You can configure states of the circular progressbar control depending on its usage in following ways.

#### Determinate

Determinate is the default state. You can use it when the progress estimation is known.

[ShowProgressValue](#) property allows to indicate the progress of the operation taken place in numerical value.



#### Indeterminate

By enabling the [IsIndeterminate](#) property, the state of the progressbar can be changed to indeterminate when the progress cannot be estimated or when not being calculated.

#### XML

```
<Syncfusion:SfCircularProgressBar Progress="70" IsIndeterminate="True"/>
```

#### C#

```
SfCircularProgressBar Circular = new SfCircularProgressBar { Progress = 70, IsIndeterminate=true};  
grid.Children.Add(Circular);
```



### Buffer

Buffer is used as a secondary progress indicator when the primary task depends on the secondary task. This will allow users to visualize both the primary and secondary tasks' progress simultaneously. The `SecondaryProgress` property can be set to visualize the secondary progress, and a separate color for the secondary progress can be set by the `SecondaryProgressColor` property.

### XML

```
<Syncfusion:SfCircularProgressBar Progress="65" SecondaryProgress="75"/>
```

### C#

```
SfCircularProgressBar Circular = new SfCircularProgressBar { Progress = 65,  
SecondaryProgress = 75 };  
grid.Children.Add(Circular);
```



### Segment in WPF circular progressbar (SfCircularProgressBar)

Segmentation helps to divide the progressbar into multiple portions. To visualize the progress of multiple sequential tasks, split the progressbar into multiple segments by setting the [SegmentCount](#) property.

#### XML

```
<Syncfusion:SfCircularProgressBar Progress="70" SegmentCount="4" />
```

#### C#

```
SfCircularProgressBar Circular = new SfCircularProgressBar();  
Circular.Progress = 70;  
Circular.SegmentCount = 4;  
grid.Children.Add(Circular);
```



### Appearance in WPF circular progressbar (SfCircularProgressBar)

You can highly customize the appearance of the circular progressbar in the following ways.

#### Angle

The appearance of the circular progressbar can be customized to semi-circle, arc, etc. The start and end angles can be customized using the [StartAngle](#) and [EndAngle](#) properties. The following code sample demonstrates to change the appearance of the circular progressbar to semi-circle.

#### XML

```
<Syncfusion:SfCircularProgressBar Progress="75" StartAngle="180"  
EndAngle="360" />
```

#### C#

```
SfCircularProgressBar Circular = new SfCircularProgressBar { Progress = 70  
};  
Circular.StartAngle = 180;  
Circular.EndAngle = 360;  
grid.Children.Add(Circular);
```



### Radius/Thickness

You can customize the radius or thickness of the circular progressbar based on its usage. The following properties are used to change the look and appearance of the circular progressbar:

- [IndicatorOuterRadius](#): Defines the outer radius of the progress indicator.
- [IndicatorInnerRadius](#): Defines the inner radius of the progress indicator.
- [TrackOuterRadius](#): Defines the outer radius of the track indicator.
- [TrackInnerRadius](#): Defines the inner radius of the track indicator.

The below code samples are provided to showcase various customizations of the circular progressbar.

### Track progress outside

The main progressbar shows the current status of the work while track progressbar is aligned outside.

### XML

```
<Syncfusion:SfCircularProgressBar Progress="80" IndicatorOuterRadius="0.7"
IndicatorInnerRadius="0.65" ShowProgressValue="True" />
```

### C#

```
SfCircularProgressBar Circular = new SfCircularProgressBar { Progress = 80
};
Circular.IndicatorInnerRadius = 0.65;
Circular.IndicatorOuterRadius = 0.7;
Circular.ShowProgressValue=true;
grid.Children.Add(Circular);
```



#### *Filled progress indicator*

The main progressbar shows the current status of the work by filling the inside area while track progressbar is aligned outside.

#### **XML**

```
<Syncfusion:SfCircularProgressBar Progress="80" IndicatorOuterRadius="0.7"
IndicatorInnerRadius="0" ShowProgressValue="False"/>
```

#### **C#**

```
SfCircularProgressBar Circular = new SfCircularProgressBar { Progress = 80
};
Circular.IndicatorInnerRadius = 0;
Circular.IndicatorOuterRadius = 0.7;
Circular.ShowProgressValue=false;
grid.Children.Add(Circular);
```



#### *Track progress inside*

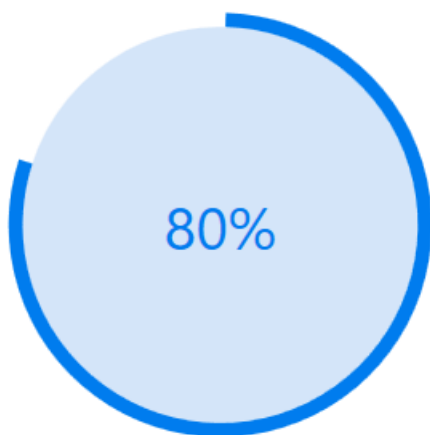
The main progressbar shows the current status of the work while track progressbar is aligned inside.

#### **XML**

```
<Syncfusion:SfCircularProgressBar x:Name="CircularProgressBar"
Progress="80" TrackOuterRadius="0.7" TrackInnerRadius="0" >
<Syncfusion:SfCircularProgressBar.ProgressContent>
<StackPanel>
<TextBlock Text="{Binding Progress,StringFormat={}{0}%"
TextAlignment="Center" DataContext="CircularProgressBar">
</TextBlock>
</StackPanel>
</Syncfusion:SfCircularProgressBar.ProgressContent>
</Syncfusion:SfCircularProgressBar>
```

### C#

```
SfCircularProgressBar Circular = new SfCircularProgressBar { Progress = 80
};
Circular.TrackOuterRadius = 0.7;
Circular.TrackInnerRadius = 0;
grid.Children.Add(Circular);
```



#### Thin track progress style

The main progressbar shows the current status of the work with relative thickness.

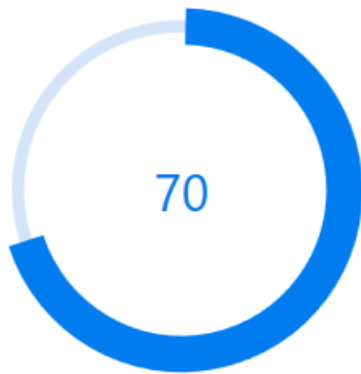
### XML

```
<Syncfusion:SfCircularProgressBar Progress="70" IndicatorOuterRadius="0.75"
IndicatorInnerRadius="0.6" TrackOuterRadius="0.7" TrackInnerRadius="0.65"
ShowProgressValue="True" >
```

### C#

```
SfCircularProgressBar Circular = new SfCircularProgressBar { Progress = 70
};
Circular.IndicatorInnerRadius = 0.6;
Circular.IndicatorOuterRadius = 0.75;
Circular.TrackOuterRadius = 0.7;
Circular.TrackInnerRadius = 0.65;
grid.Children.Add(Circular);
```





### Color Customization

You can customize the color of the circular progressbar's progress color and track color. The following properties are used to customize the color in the progressbar:

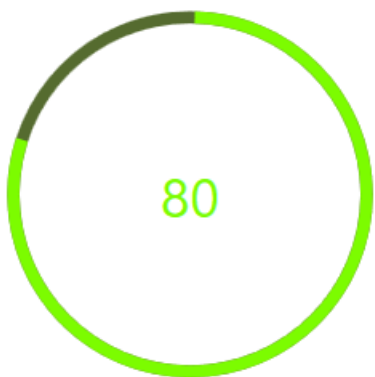
- [ProgressColor](#): Represents the color of the progress indicator.
- [TrackColor](#): Represents the color of the track indicator.

### XML

```
<Syncfusion:SfCircularProgressBar Progress="70" ProgressColor="LawnGreen"
TrackColor="DarkOliveGreen"/>
```

### C#

```
SfCircularProgressBar circular = new SfCircularProgressBar();
circular.Progress = 80;
circular.ProgressColor = new SolidColorBrush(Colors.LawnGreen);
circular.TrackColor = new SolidColorBrush(Colors.DarkOliveGreen);
grid.Children.Add(circular);
```



### Range Colors

You can visualize the multiple ranges with different solid colors that are mapped to each range to enhance the readability of progress.

The solid colors can be mapped to the specific ranges using the [RangeColor] (<https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.ProgressBar.RangeColor.html>).

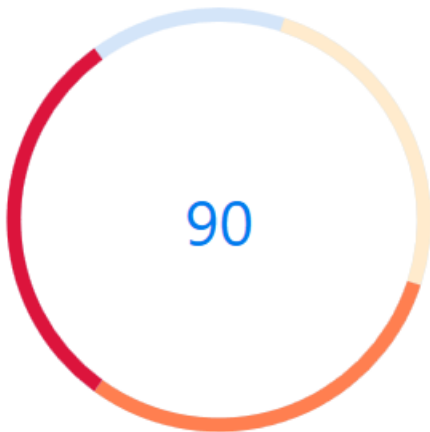
- [Color](#): Represents the color to the specified range.
- [Start](#): Represents the start range of the color.
- [End](#): Represents the end range of the color.

**XML**

```
<Syncfusion:SfCircularProgressBar Progress="80" >
<Syncfusion:SfCircularProgressBar.RangeColors>
<Syncfusion:RangeColorCollection>
<Syncfusion:RangeColor Color="BlanchedAlmond" Start=5" End="30"/>
<Syncfusion:RangeColor Color="Coral" Start="30" End="60"/>
<Syncfusion:RangeColor Color="Crimson" Start="60" End="100"/>
</Syncfusion:RangeColorCollection>
</Syncfusion:SfCircularProgressBar.RangeColors>
</Syncfusion:SfCircularProgressBar>
```

**C#**

```
SfCircularProgressBar circular = new SfCircularProgressBar();
RangeColorCollection rangeColors = new RangeColorCollection();
rangeColors.Add(new RangeColor() { Color = Colors.BlanchedAlmond, Start = 5,
End = 30 });
rangeColors.Add(new RangeColor() { Color = Colors.Coral, Start = 30, End =
60 });
rangeColors.Add(new RangeColor() { Color = Colors.Crimson, Start = 60, End =
100 });
circular.Progress = 90;
circular.RangeColors = rangeColors;
grid.Children.Add(circular);
```

**Gradient**

Gradient shows change in intensity of the colors during the progress. [IsGradient](#) property in [RangeColor](#) class is used to set gradient effect to the colors applied to the progressbar.

**XML**

```
<Syncfusion:SfCircularProgressBar Progress="80" >
<Syncfusion:SfCircularProgressBar.RangeColors>
<Syncfusion:RangeColorCollection>
```

```
<Syncfusion:RangeColor IsGradient="True" Color="SkyBlue" Start="10"
End="30"/>
<Syncfusion:RangeColor IsGradient="True" Color="DeepSkyBlue" Start="30"
End="60"/>
<Syncfusion:RangeColor IsGradient="True" Color="Blue" Start="60" End="100"/>
</Syncfusion:RangeColorCollection>
</Syncfusion:SfCircularProgressBar.RangeColors>
</Syncfusion:SfCircularProgressBar>
```

## C#

```
SfCircularProgressBar circular = new SfCircularProgressBar();
RangeColorCollection rangeColors = new RangeColorCollection();
rangeColors.Add(new RangeColor() { IsGradient=true, Color = Colors.SkyBlue,
Start = 10, End = 30 });
rangeColors.Add(new RangeColor() { IsGradient=true, Color =
Colors.DeepSkyBlue, Start = 30, End = 60 });
rangeColors.Add(new RangeColor() { IsGradient=true, Color = Colors.Blue,
Start = 60, End = 100 });
circular.RangeColors = rangeColors;
circular.Progress = 80;
grid.Children.Add(circular);
```



## Corner radius

The [IndicatorCornerRadius](#) property is used to frame the rounded edges in the circular progress bar as shown in the following code sample.

**Note:** The proper IndicatorCornerRadius value can be set using the formula  $\text{IndicatorOuterRadius} * 10 = \text{IndicatorCornerRadius}$ .

## XML

```
<Grid Name="grid">
<Syncfusion:SfCircularProgressBar
Width="150"
Height="150"
Margin="5"
EndAngle="410"
IndicatorCornerRadius="5"
```

```
Progress="50"  
ShowProgressValue="False"  
StartAngle="130" />  
</Grid>
```

### C#

```
SfCircularProgressBar circular = new SfCircularProgressBar();  
circular.Width = 150;  
circular.Height = 150;  
circular.IndicatorCornerRadius = 5;  
circular.Progress = 50;  
circular.ShowProgressValue = false;  
circular.StartAngle = 130;  
circular.EndAngle = 410;  
grid.Children.Add(circular);
```



### AnimationDuration

You can customize the duration for completing one animation cycle and it applies when the `IsIndeterminate` is true. The default value is `3000ms`.

### XML

```
<Grid Name="grid">  
<Syncfusion:SfCircularProgressBar  
Width="150"  
Height="150"  
AnimationDuration="00:00:01"  
IsIndeterminate="True"  
Progress="50"  
ShowProgressValue="False" />  
</Grid>
```

## C#

```
SfCircularProgressBar circular = new SfCircularProgressBar();  
circular.Width = 150;  
circular.Height = 150;  
circular.AnimationDuration = new TimeSpan(0, 0, 1);  
circular.IsIndeterminate = true;  
circular.Progress = 50;  
circular.ShowProgressValue = false;  
grid.Children.Add(circular);
```



## AnimationEasing

You can customize the easing function to apply for the linear and circular progress bar animation and it applies when the `IsIndeterminate` is true.

## XML

```
<Grid Name="grid">  
  <Syncfusion:SfCircularProgressBar  
    Width="150"  
    Height="150"  
    IsIndeterminate="True"  
    Progress="50"  
    ShowProgressValue="False">  
    <Syncfusion:SfCircularProgressBar.AnimationEasing>  
      <BounceEase  
        Bounces="20"  
        Bounciness="5"  
        EasingMode="EaseOut" />  
    </Syncfusion:SfCircularProgressBar.AnimationEasing>  
  </Syncfusion:SfCircularProgressBar>  
</Grid>
```

## C#

```
SfCircularProgressBar circular = new SfCircularProgressBar();  
circular.Width = 150;  
circular.Height = 150;  
circular.IsIndeterminate = true;  
circular.Progress = 50;  
circular.ShowProgressValue = false;
```

```

BounceEase bounceEase = new BounceEase();
bounceEase.Bounces = 20;
bounceEase.Bounciness = 5;
bounceEase.EasingMode = EasingMode.EaseOut;
circular.AnimationEasing = bounceEase;
grid.Children.Add(circular);

```



### Custom Content in WPF circular progressbar (SfCircularProgressBar)

In the circular progress bar, you can add any custom content to the center of the control using [ProgressContent](#) property.

For example, you can include add, play or pause button to control the progress. You can also add an image that indicates the actual task in progress or add custom text that conveys how far the task is completed.

The following code sample demonstrates to add custom text that conveys how far the resource is used.

#### XML

```

<Syncfusion:SfCircularProgressBar x:Name="CustomContentProgressBarLevel"
Progress="70">
  <Syncfusion:SfCircularProgressBar.ProgressContent>
    <StackPanel>
      <TextBlock Text="{Binding Progress,StringFormat={}{0}%"
TextAlignment="Center" DataContext="{x:Reference
CustomContentProgressBarLevel}" />
      <TextBlock Text="Used" />
    </StackPanel>
  </Syncfusion:SfCircularProgressBar.ProgressContent>
</Syncfusion:SfCircularProgressBar>

```

#### C#

```

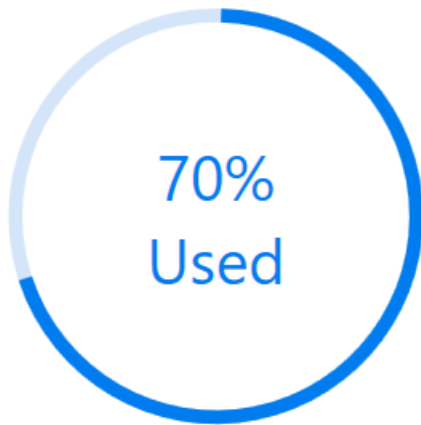
SfCircularProgressBar circular = new SfCircularProgressBar();
circular.Progress = 70;
StackPanel stack = new StackPanel();
TextBlock textblock = new TextBlock();
textblock.DataContext = circular;
Binding binding = new Binding
{
    Source = circular,

```

```

Path = new PropertyPath("Progress"),
StringFormat = "{0}%"
};
textblock.SetBinding(TextBlock.TextProperty, binding);
TextBlock textblock2 = new TextBlock();
textblock2.Text = "used";
stack.Children.Add(textblock);
stack.Children.Add(textblock2);
circular.ProgressContent = stack;
grid.Children.Add(circular);

```



The following code sample demonstrates to include play and pause button which can be controlled at runtime.

#### XML

```

<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Syncfusion:SfCircularProgressBar x:Name="VideoPlayerProgressBar"
Progress="{Binding Path= PlayPauseProgress, Mode=TwoWay}"
TrackInnerRadius="0" IndicatorOuterRadius="0.7" IndicatorInnerRadius="0.65"
>
<Syncfusion:SfCircularProgressBar.ProgressContent>
<Grid>
<Button Grid.Row="0" Grid.Column="0" Visibility="Hidden"
x:Name="PlayButton" Click="PlayButton_Clicked" Background="#00FFFFFF"
Width="30" Height="30" >
<Image x:Name="image1" Source="Resources\SfProgressPlay.png"/>
</Button>
<Button Grid.Row="0" Grid.Column="0" x:Name="PauseButton"
Click="PauseButton_Clicked" Background="#00FFFFFF" Width="30" Height="30" >
<Image x:Name="image" Source="Resources\SfProgressPause.png"/>
</Button>
</Grid>
</Syncfusion:SfCircularProgressBar.ProgressContent>
</Syncfusion:SfCircularProgressBar>

```

**C#**

```

public partial class MainWindow : Window
{
    public DispatcherTimer SfCircularBarPlayPauseTimer;
    public MainWindow()
    {
        InitializeComponent();
    }
    /// <summary>
    /// Method represent when pause button clicked.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void PauseButton_Clicked(object sender, RoutedEventArgs e)
    {
        PlayButton.Visibility = Visibility.Visible;
        PauseButton.Visibility = Visibility.Hidden;
        (this.DataContext as ViewModel).SfCircularBarPlayPauseTimer.Stop();
    }
    /// <summary>
    /// Method represent when play button clicked.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void PlayButton_Clicked(object sender, RoutedEventArgs e)
    {
        PlayButton.Visibility = Visibility.Hidden;
        PauseButton.Visibility = Visibility.Visible;
        (this.DataContext as ViewModel).SfCircularBarPlayPauseTimer.Start()
    }
}

public class ViewModel : INotifyPropertyChanged
{
    public DispatcherTimer SfCircularBarPlayPauseTimer;
    #region Cons
    /// <summary>
    /// Initializes a new instance of the time class.
    /// </summary>
    public ViewModel()
    {
        SfCircularBarPlayPauseTimer = new DispatcherTimer();
        SfCircularBarPlayPauseTimer.Tick += SfCircularBarPlayPauseTimer_Tick;
        SfCircularBarPlayPauseTimer.Interval = new TimeSpan(0, 0, 0, 0, 30);
        SfCircularBarPlayPauseTimer.Start();
    }
    #endregion
    #region Method
    /// <summary>
    /// Represents to increase the <see cref="PlayPauseProgress"/> value based
    time interval.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void SfCircularBarPlayPauseTimer_Tick(object sender, EventArgs e)
    {
        if (PlayPauseProgress < 100)

```



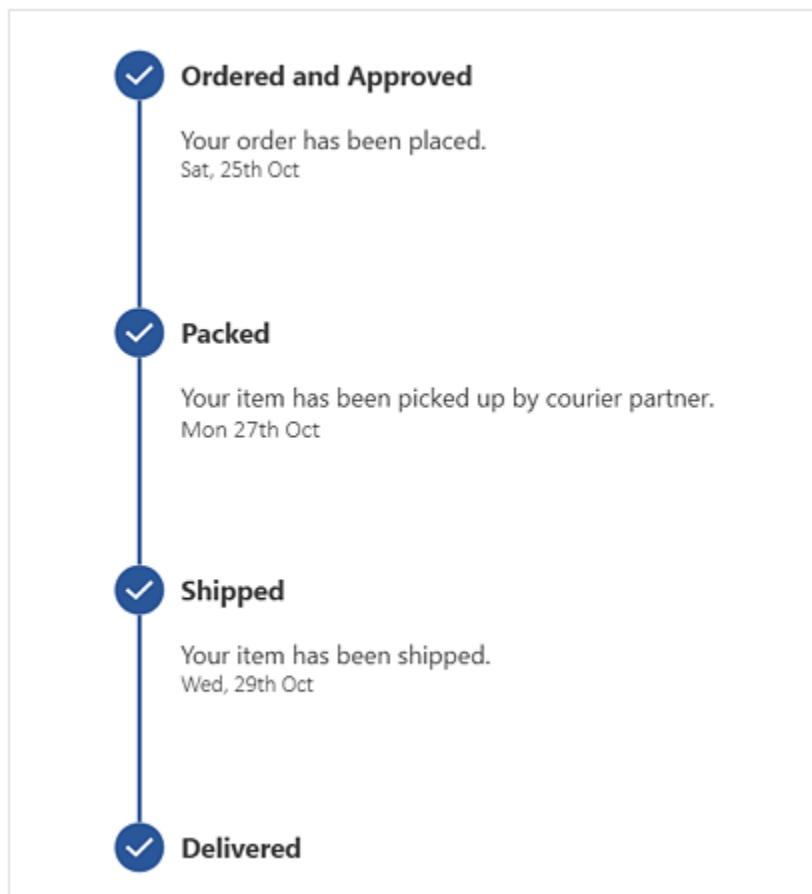
```
{
    PlayPauseProgress += 1;
}
else
{
    PlayPauseProgress = 0;
}
}
#endregion
#region Properties
/// <summary>
/// 
/// </summary>
private int playPauseProgress;
/// <summary>
/// Gets or sets the CircularProgressBar <see cref="PlayPauseProgress"/>
value.
/// </summary>
public int PlayPauseProgress
{
    get
    {
        return playPauseProgress;
    }
    set
    {
        playPauseProgress = value;
        RaisePropertyChanged("PlayPauseProgress");
    }
}
/// <summary>
/// Represents the event trigger while change the property.
/// </summary>
public event PropertyChangedEventHandler PropertyChanged;
/// <summary>
/// Method represents the name of the property that changed.
/// </summary>
/// <param name="name"></param>
private void RaisePropertyChanged(string name)
{
    if (PropertyChanged != null)
    {
        PropertyChanged.Invoke(this, new PropertyChangedEventArgs(name));
    }
}
#endregion
}
```



## SfStepProgressBar

### WPF Step ProgressBar (SfStepProgressBar) Overview

The SfStepProgressBar control is used to show the progress of a multiple-step process, such as new user registration or package status tracking. You can customize its appearance by changing the step shape, progress bar color, step template, and content template.



### Key features

- Visualize the step progress markers with different shapes, such as square and circle.
- Supports active, inactive, and indeterminate statuses to show progress.

- Visualize the progress of a multiple-step process in the horizontal or vertical orientation.
- Customize the progress bar styles, markers, and contents using the templates.

## Getting Started with WPF Step ProgressBar (SfStepProgressBar)

You can create a WPF application with the SfStepProgressBar control using the following steps:

### Assembly deployment

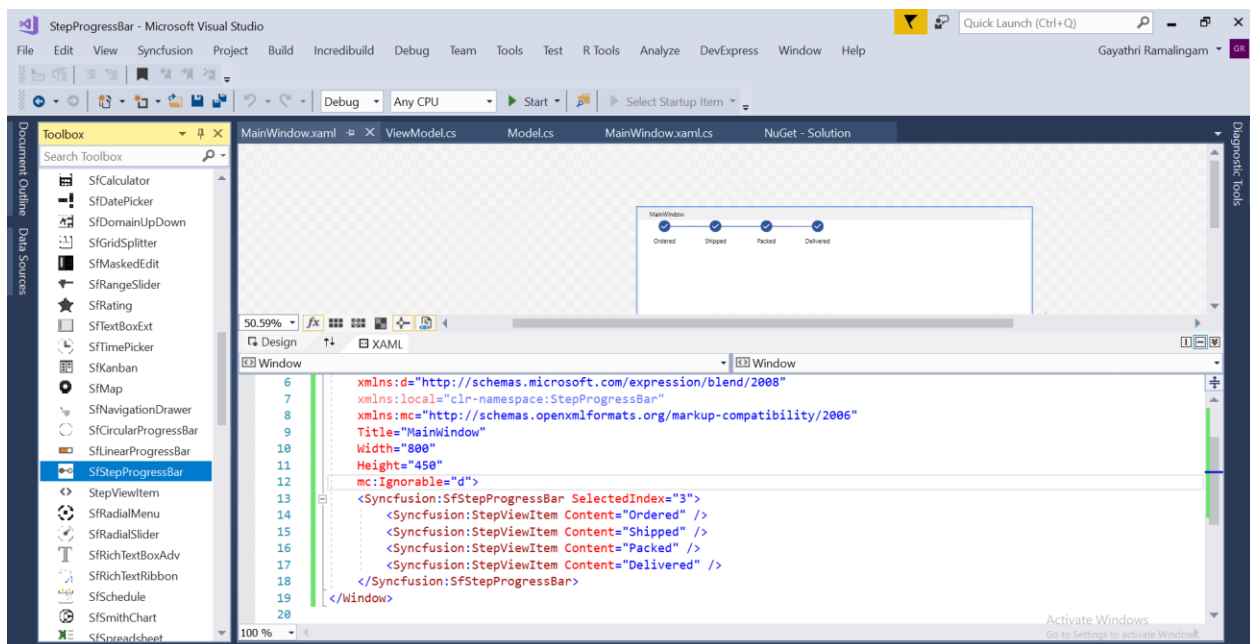
Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

You can find more details about installing the NuGet package in a WPF application in the following link: [How to install NuGet packages](#)

### Adding control through designer

The SfStepProgressBar control can be added to a WPF application by dragging it from the toolbox to a designer view. The following assembly reference will be added automatically:

- Syncfusion.SfProgressBar.WPF



### Adding control manually in XAML

To add control manually in the XAML, follow the given steps:

1. Add the following required assembly reference to the project: \* Syncfusion.SfProgressBar.WPF
2. Import the Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page.
3. Declare the SfStepProgressBar control in the XAML page.

### Select item using Selected Index

You can change the index of the last active (selected) item where items before this index will move to the active status by using the [SelectedIndex](#) property.

### XML

```
<Grid x:Name="grid">
```

```
<syncfusion:SfStepProgressBar SelectedIndex="3">
<syncfusion:StepViewItem Content="Ordered" />
<syncfusion:StepViewItem Content="Shipped" />
<syncfusion:StepViewItem Content="Packed" />
<syncfusion:StepViewItem Content="Delivered" />
</syncfusion:SfStepProgressBar>
</Grid>
```

**C#**

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        SfStepProgressBar stepProgressBar = new SfStepProgressBar();
        StepViewItem orderedStepViewItem = new StepViewItem();
        StepViewItem shippedStepViewItem = new StepViewItem();
        StepViewItem packedStepViewItem = new StepViewItem();
        StepViewItem deliveredStepViewItem = new StepViewItem();
        orderedStepViewItem.Content = "Ordered";
        shippedStepViewItem.Content = "Shipped";
        packedStepViewItem.Content = "Packed";
        deliveredStepViewItem.Content = "Delivered";
        stepProgressBar.Items.Add(orderedStepViewItem);
        stepProgressBar.Items.Add(shippedStepViewItem);
        stepProgressBar.Items.Add(packedStepViewItem);
        stepProgressBar.Items.Add(deliveredStepViewItem);
        stepProgressBar.SelectedIndex = 3;
        grid.Children.Add(stepProgressBar);
    }
}
```

**XML**

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp4"
xmlns:Syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="StepProgressBar.MainWindow"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid x:Name="grid">
<Syncfusion:SfStepProgressBar SelectedIndex="3">
<Syncfusion:StepViewItem Content="Ordered" />
<Syncfusion:StepViewItem Content="Shipped" />
<Syncfusion:StepViewItem Content="Packed" />
```

```
<Syncfusion:StepViewItem Content="Delivered" />
</Syncfusion:SfStepProgressBar>
</Grid>
</Window>
```

### Adding control through code behind

To add control manually using the code behind, follow the given steps:

1. Add the following required assembly reference to the project: \* Syncfusion.SfProgressBar.WPF 2. Import the ProgressBar namespace

**using Syncfusion.UI.Xaml.ProgressBar;**

3. Create a Step Progressbar instance, and add it to the window.



### C#

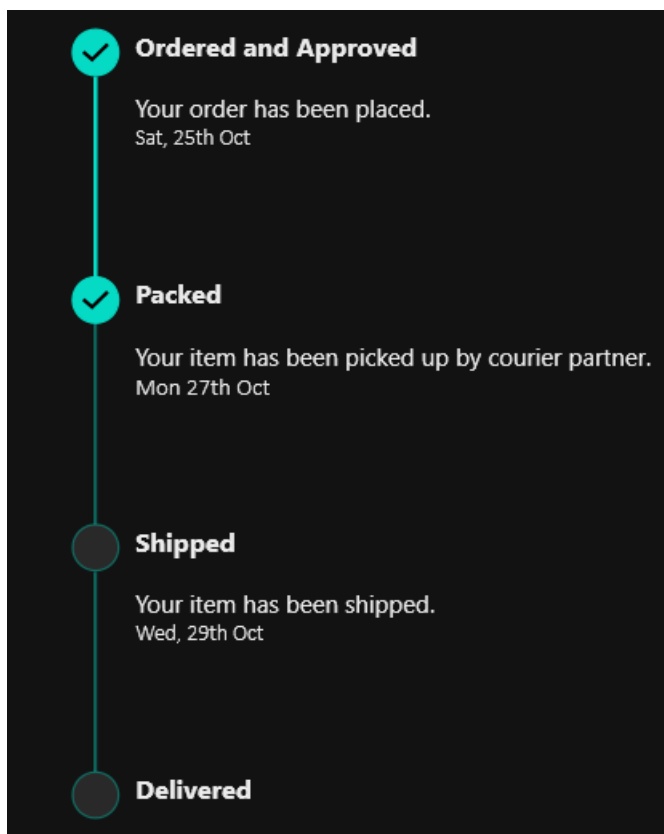
```
using Syncfusion.UI.Xaml.ProgressBar;
namespace SfProgressBar
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfStepProgressBar stepProgressBar = new SfStepProgressBar();
            StepViewItem orderedStepViewItem = new StepViewItem();
            StepViewItem shippedStepViewItem = new StepViewItem();
            StepViewItem packedStepViewItem = new StepViewItem();
            StepViewItem deliveredStepViewItem = new StepViewItem();
            orderedStepViewItem.Content = "Ordered";
            shippedStepViewItem.Content = "Shipped";
            packedStepViewItem.Content = "Packed";
            deliveredStepViewItem.Content = "Delivered";
            stepProgressBar.Items.Add(orderedStepViewItem);
            stepProgressBar.Items.Add(shippedStepViewItem);
            stepProgressBar.Items.Add(packedStepViewItem);
            stepProgressBar.Items.Add(deliveredStepViewItem);
            stepProgressBar.SelectedIndex = 3;
            grid.Children.Add(stepProgressBar);
        }
    }
}
```

Download demo from [GitHub](#).

### Theme

SfStepProgressBar supports various built-in themes. Refer to the below links to apply themes for the SfStepProgressBar,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Data Binding in WPF Step ProgressBar (SfStepProgressBar)

You can add a [StepViewItem](#) using the data binding in the WPF [SfStepProgressBar](#).

#### Data binding to Objects

The [SfStepProgressBar](#) can bound to an external source to auto-create [StepViewItem](#) and display the data using the [ItemsSource](#) property.

**Note:** To bind the [ItemsSource](#) to [SfStepProgressBar](#), you need to have a collection with a data object which holds the step view item details.

Here, the `StepItem` class defined with [Content](#) and its properties, and the `ViewModel` class has the [ItemsSource](#) property of type `ObservableCollection<StepItem>`.

#### C#

```
// Model.cs
/// <summary>
/// Represents the model.
/// </summary>
public class StepItem
```

```

{
    /// <summary>
    /// Gets or sets the text for step view item.
    /// </summary>
    public string ModelText { get; set; }
    /// <summary>
    /// Gets or sets the space between text and step view item.
    /// </summary>
    public double TitleSpace { get; set; }
}
//ViewModel.cs
/// <summary>
/// Represents the view model class.
/// </summary>
public class ViewModel : INotifyPropertyChanged
{
    /// <summary>
    /// Represents the step view items.
    /// </summary>
    private ObservableCollection<StepItem> m_stepViewItems;
    /// <summary>
    /// Represents the property changed event.
    /// </summary>
    public event PropertyChangedEventHandler PropertyChanged;
    /// <summary>
    /// Gets or sets the step view items.
    /// </summary>
    public ObservableCollection<StepItem> StepViewItems
    {
        get
        {
            return m_stepViewItems;
        }
        set
        {
            m_stepViewItems = value;
            OnPropertyChanged(new PropertyChangedEventArgs("StepViewItems"));
        }
    }
    /// <summary>
    /// Triggers the on property changed event.
    /// </summary>
    /// <param name="e"></param>
    public void OnPropertyChanged(PropertyChangedEventArgs e)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, e);
    }
    /// <summary>
    /// Initialize the instance of <see cref="ViewModel"/> class.
    /// </summary>
    public ViewModel()
    {
        StepViewItems = new ObservableCollection<StepItem>();
        PopulateData();
    }
    /// <summary>

```

```
/// Populates the data.
/// </summary>
private void PopulateData()
{
    //Adding the step view items into the collection
    StepItem orderedStepViewItem = new StepItem()
    {
        ModelText = "Ordered",
        TitleSpace = 8
    };
    StepItem shippedStepViewItem = new StepItem()
    {
        ModelText = "Shipped",
        TitleSpace = 8
    };
    StepItem packedStepViewItem = new StepItem()
    {
        ModelText = "Packed",
        TitleSpace = 8
    };
    StepItem deliveredStepViewItem = new StepItem()
    {
        ModelText = "Delivered",
        TitleSpace = 8
    };
    StepViewItems.Add(orderedStepViewItem);
    StepViewItems.Add(shippedStepViewItem);
    StepViewItems.Add(packedStepViewItem);
    StepViewItems.Add(deliveredStepViewItem);
}
}
```

## XML

```
<Grid Name="grid">
<syncfusion:SfStepProgressBar
x:Name="stepperControlName"
Margin="40"
ItemsSource="{Binding StepViewItems}"
Orientation="Horizontal"
SelectedIndex="2">
<syncfusion:SfStepProgressBar.ItemContainerStyle>
<Style TargetType="syncfusion:StepViewItem">
<Setter Property="Content" Value="{Binding ModelText}" />
<Setter Property="TextSpacing" Value="{Binding TitleSpace}" />
</Style>
</syncfusion:SfStepProgressBar.ItemContainerStyle>
<syncfusion:SfStepProgressBar.DataContext>
<local:ViewModel />
</syncfusion:SfStepProgressBar.DataContext>
</syncfusion:SfStepProgressBar>
</Grid>
```





Download demo from [GitHub](#).

### Data-Binding with XML

An XML file can also be used as the [ItemsSource](#) for the Step Progress Bar control. The following example shows this.

Create an XML file with the following information and name it Data.xml.

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<StepItems SelectedIndex="3">
  <Step Name="Ordered"/>
  <Step Name="Shipped"/>
  <Step Name="Packed"/>
  <Step Name="Delivered"/>
</StepItems>
```

The ItemsSource property for the Step ProgressBar control.

#### XML

```
<Window.Resources>
  <XmlDataProvider x:Key="xmlSource" Source="Data.xml" XPath="StepItems" />
</Window.Resources>
<syncfusion:SfStepProgressBar x:Name="stepperControlName"
  ItemsSource="{Binding Source={StaticResource xmlSource}, XPath=Step}"
  SelectedIndex="{Binding Source={StaticResource xmlSource},
  XPath=@SelectedIndex}">
  <syncfusion:SfStepProgressBar.ItemContainerStyle>
    <Style TargetType="syncfusion:StepViewItem">
      <Setter Property="Content" Value="{Binding XPath=@Name}" />
    </Style>
  </syncfusion:SfStepProgressBar.ItemContainerStyle>
</syncfusion:SfStepProgressBar>
```

This will create the following Step ProgressBar control.



Download demo from [GitHub](#).

### Selected Item in WPF Step ProgressBar (SfStepProgressBar)

You can customize the status of the StepView item in the following ways.

#### SelectedItemStatus

You can change the status of last active (selected) step view item by using the [SelectedItemStatus](#) property. The default value of this property is [Inactive](#).

**XML**

```
<Grid x:Name="grid">
<syncfusion:SfStepProgressBar SelectedIndex="2"
SelectedItemStatus="Indeterminate">
<syncfusion:StepViewItem Content="Ordered" />
<syncfusion:StepViewItem Content="Shipped" />
<syncfusion:StepViewItem Content="Packed" />
<syncfusion:StepViewItem Content="Delivered" />
</syncfusion:SfStepProgressBar>
</Grid>
```

**C#**

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        SfStepProgressBar stepProgressBar = new SfStepProgressBar();
        StepViewItem orderedStepViewItem = new StepViewItem();
        StepViewItem shippedStepViewItem = new StepViewItem();
        StepViewItem packedStepViewItem = new StepViewItem();
        StepViewItem deliveredStepViewItem = new StepViewItem();
        orderedStepViewItem.Content = "Ordered";
        shippedStepViewItem.Content = "Shipped";
        packedStepViewItem.Content = "Packed";
        deliveredStepViewItem.Content = "Delivered";
        stepProgressBar.Items.Add(orderedStepViewItem);
        stepProgressBar.Items.Add(shippedStepViewItem);
        stepProgressBar.Items.Add(packedStepViewItem);
        stepProgressBar.Items.Add(deliveredStepViewItem);
        stepProgressBar.SelectedIndex = 2;
        stepProgressBar.SelectedItemStatus = StepStatus.Indeterminate;
        grid.Children.Add(stepProgressBar);
    }
}
```

**SelectedItemProgress**

You can change the progress value of the last active (selected) step view item by using the [SelectedItemProgress](#) property. The default value of this property is **100**.

**XML**

```
<Grid x:Name="grid">
<syncfusion:SfStepProgressBar SelectedIndex="2" SelectedItemProgress="50"
SelectedItemStatus="Indeterminate">
<syncfusion:StepViewItem Content="Ordered" />
<syncfusion:StepViewItem Content="Shipped" />
<syncfusion:StepViewItem Content="Packed" />
```

```
<syncfusion:StepViewItem Content="Delivered" />
</syncfusion:SfStepProgressBar>
</Grid>
```

**C#**

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        SfStepProgressBar stepProgressBar = new SfStepProgressBar();
        StepViewItem orderedStepViewItem = new StepViewItem();
        StepViewItem shippedStepViewItem = new StepViewItem();
        StepViewItem packedStepViewItem = new StepViewItem();
        StepViewItem deliveredStepViewItem = new StepViewItem();
        orderedStepViewItem.Content = "Ordered";
        shippedStepViewItem.Content = "Shipped";
        packedStepViewItem.Content = "Packed";
        deliveredStepViewItem.Content = "Delivered";
        stepProgressBar.Items.Add(orderedStepViewItem);
        stepProgressBar.Items.Add(shippedStepViewItem);
        stepProgressBar.Items.Add(packedStepViewItem);
        stepProgressBar.Items.Add(deliveredStepViewItem);
        stepProgressBar.SelectedIndex = 2;
        stepProgressBar.SelectedItemStatus = StepStatus.Indeterminate;
        stepProgressBar.SelectedItemProgress = 50;
        grid.Children.Add(stepProgressBar);
    }
}
```

**AnimationDuration**

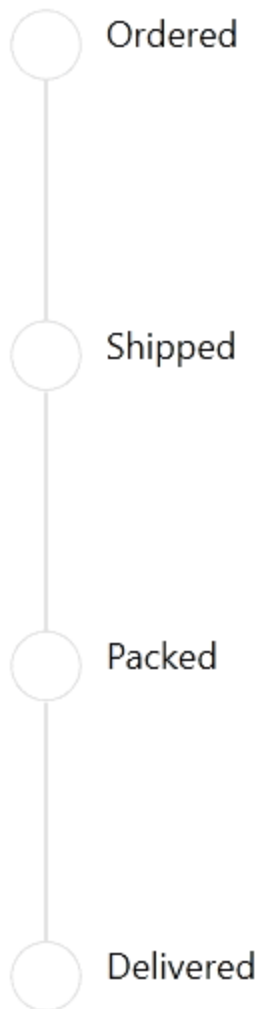
You can change the the duration for completing the animation status of step view item by using the [AnimationDuration](#) property. The default value of this property is **500ms**.

**XML**

```
<Grid x:Name="grid">
<syncfusion:SfStepProgressBar SelectedIndex="3" AnimationDuration="0:0:1"
Orientation="Vertical">
<syncfusion:StepViewItem Content="Ordered" />
<syncfusion:StepViewItem Content="Shipped" />
<syncfusion:StepViewItem Content="Packed" />
<syncfusion:StepViewItem Content="Delivered" />
</syncfusion:SfStepProgressBar>
</Grid>
```

**C#**

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        SfStepProgressBar stepProgressBar = new SfStepProgressBar();
        StepViewItem orderedStepViewItem = new StepViewItem();
        StepViewItem shippedStepViewItem = new StepViewItem();
        StepViewItem packedStepViewItem = new StepViewItem();
        StepViewItem deliveredStepViewItem = new StepViewItem();
        orderedStepViewItem.Content = "Ordered";
        shippedStepViewItem.Content = "Shipped";
        packedStepViewItem.Content = "Packed";
        deliveredStepViewItem.Content = "Delivered";
        stepProgressBar.Items.Add(orderedStepViewItem);
        stepProgressBar.Items.Add(shippedStepViewItem);
        stepProgressBar.Items.Add(packedStepViewItem);
        stepProgressBar.Items.Add(deliveredStepViewItem);
        stepProgressBar.SelectedIndex = 3;
        stepProgressBar.Orientation = Orientation.Vertical;
        stepProgressBar.AnimationDuration = new TimeSpan(0, 0, 1);
        grid.Children.Add(stepProgressBar);
    }
}
```



### Appearance in WPF Step ProgressBar (SfStepProgressBar)

You can highly customize the appearance of the Step progress bar in the following ways.

#### Step Shape

You can change the shape of a step marker by using the [MarkerShapeType](#) property. The default value of this property is [Circle](#).

#### XML

```
<Grid x:Name="grid">
  <syncfusion:SfStepProgressBar MarkerShapeType="Square" SelectedIndex="3">
    <syncfusion:StepViewItem Content="Ordered" />
    <syncfusion:StepViewItem Content="Shipped" />
    <syncfusion:StepViewItem Content="Packed" />
    <syncfusion:StepViewItem Content="Delivered" />
  </syncfusion:SfStepProgressBar>
</Grid>
```

#### C#

```
public partial class MainWindow : Window
```

```

{
public MainWindow()
{
InitializeComponent();
SfStepProgressBar stepProgressBar = new SfStepProgressBar();
StepViewItem orderedStepViewItem = new StepViewItem();
StepViewItem shippedStepViewItem = new StepViewItem();
StepViewItem packedStepViewItem = new StepViewItem();
StepViewItem deliveredStepViewItem = new StepViewItem();
orderedStepViewItem.Content = "Ordered";
shippedStepViewItem.Content = "Shipped";
packedStepViewItem.Content = "Packed";
deliveredStepViewItem.Content = "Delivered";
stepProgressBar.Items.Add(orderedStepViewItem);
stepProgressBar.Items.Add(shippedStepViewItem);
stepProgressBar.Items.Add(packedStepViewItem);
stepProgressBar.Items.Add(deliveredStepViewItem);
stepProgressBar.SelectedIndex = 3;
stepProgressBar.MarkerShapeType = MarkerShapeType.Square;
grid.Children.Add(stepProgressBar);
}
}

```



### Orientation

You can change the orientation of step progressbar by using the [Orientation](#) property. The default value of this property is [Horizontal](#).

### XML

```

<Grid x:Name="grid">
<syncfusion:SfStepProgressBar Orientation="Vertical" SelectedIndex="3">
<syncfusion:StepViewItem Content="Ordered" />
<syncfusion:StepViewItem Content="Shipped" />
<syncfusion:StepViewItem Content="Packed" />
<syncfusion:StepViewItem Content="Delivered" />
</syncfusion:SfStepProgressBar>
</Grid>

```

### C#

```

public partial class MainWindow : Window
{
public MainWindow()
{
InitializeComponent();
SfStepProgressBar stepProgressBar = new SfStepProgressBar();
StepViewItem orderedStepViewItem = new StepViewItem();
StepViewItem shippedStepViewItem = new StepViewItem();
StepViewItem packedStepViewItem = new StepViewItem();
StepViewItem deliveredStepViewItem = new StepViewItem();

```

```
orderedStepViewItem.Content = "Ordered";  
shippedStepViewItem.Content = "Shipped";  
packedStepViewItem.Content = "Packed";  
deliveredStepViewItem.Content = "Delivered";  
stepProgressBar.Items.Add(orderedStepViewItem);  
stepProgressBar.Items.Add(shippedStepViewItem);  
stepProgressBar.Items.Add(packedStepViewItem);  
stepProgressBar.Items.Add(deliveredStepViewItem);  
stepProgressBar.SelectedIndex = 3;  
stepProgressBar.Orientation = Orientation.Vertical;  
grid.Children.Add(stepProgressBar);  
}  
}
```



### Connector Customization

You can customize the color and thickness of the Step progressbar using the following property.

- [ActiveConnectorColor](#): Represents the color of the connector for active state.
- [ConnectorColor](#): Represents the color of the connector for inactive state.

- [ConnectorThickness](#): Represents the thickness of connector line that connecting neighboring step view items.

**XML**

```
<syncfusion:SfStepProgressBar SelectedIndex="2" ActiveConnectorColor="Coral"
ConnectorColor="Crimson" ConnectorThickness="4">
  <syncfusion:StepViewItem Content="Ordered" />
  <syncfusion:StepViewItem Content="Shipped" />
  <syncfusion:StepViewItem Content="Packed" />
  <syncfusion:StepViewItem Content="Delivered" />
</syncfusion:SfStepProgressBar>
```

**C#**

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        SfStepProgressBar stepProgressBar = new SfStepProgressBar();
        StepViewItem orderedStepViewItem = new StepViewItem();
        StepViewItem shippedStepViewItem = new StepViewItem();
        StepViewItem packedStepViewItem = new StepViewItem();
        StepViewItem deliveredStepViewItem = new StepViewItem();
        orderedStepViewItem.Content = "Ordered";
        shippedStepViewItem.Content = "Shipped";
        packedStepViewItem.Content = "Packed";
        deliveredStepViewItem.Content = "Delivered";
        stepProgressBar.Items.Add(orderedStepViewItem);
        stepProgressBar.Items.Add(shippedStepViewItem);
        stepProgressBar.Items.Add(packedStepViewItem);
        stepProgressBar.Items.Add(deliveredStepViewItem);
        stepProgressBar.SelectedIndex = 3;
        stepProgressBar.ActiveConnectorColor = new SolidColorBrush(Colors.Coral);
        stepProgressBar.ConnectorColor = new SolidColorBrush(Colors.Crimson);
        stepProgressBar.ConnectorThickness = 4;
        grid.Children.Add(stepProgressBar);
    }
}
```

**MarkerClicked Event**

You can get the [StepViewItem](#) values when the marker of step view item is clicked. The following example shows this.

**XML**

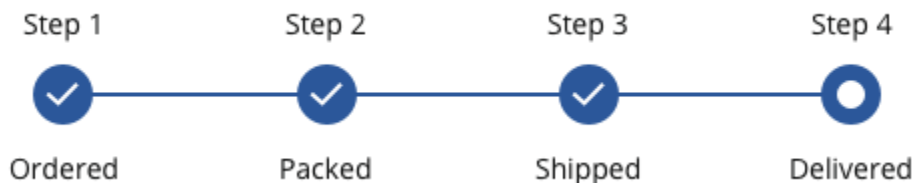
```
<Syncfusion:SfStepProgressBar
SelectedIndex="2"
MarkerClicked="SfStepProgressBar_MarkerClicked">
```



```
<Syncfusion:StepViewItem Content="Ordered" />
<Syncfusion:StepViewItem Content="Packed" />
<Syncfusion:StepViewItem Content="Shipped" />
<Syncfusion:StepViewItem Content="Delivered" />
</Syncfusion:SfStepProgressBar>
```

**C#**

```
private void SfStepProgressBar_MarkerClicked(object sender,
MarkerClickedEventArgs e)
{
    ItemsControl itemsControl =
    ItemsControl.ItemsControlFromItemContainer(e.StepViewItem);
    int index =
    itemsControl.ItemContainerGenerator.IndexFromContainer(e.StepViewItem);
    (sender as SfStepProgressBar).SelectedIndex = index;
}
```



## Customizing Data Templates in WPF Step ProgressBar (SfStepProgressBar)

Data templates can be customized for the step markers and step content. The next sections explain how to customize the data templates.

## Item Template

You can customize the content of [StepViewItem](#) by using the `ItemTemplate` property. The following example shows how to customize the step view item's content with `DataTemplate`.

**XML**

```
<syncfusion:SfStepProgressBar ItemsSource="{Binding StepViewItems}"
SelectedIndex="2">
<syncfusion:SfStepProgressBar.ItemTemplate>
<DataTemplate>
<Grid>
<Path
Width="25"
Height="25"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M9 16.2L4.8 121-1.4 1.4L9 19 21 71-1.4-1.4L9 16.2z"
Fill="#2B579A"
Stroke="#2B579A" />
</Grid>
</DataTemplate>
</syncfusion:SfStepProgressBar.ItemTemplate>
</syncfusion:SfStepProgressBar>
```

Implementing the above code will create the following Step ProgressBar control.



### Item Template Selector

Using the `ItemTemplateSelector`, you can use the different templates for step content depends on the step view item status. The following example shows this.

Use this template selector to choose a template for the Step ProgressBar control.

### XML

```
<Window.Resources>
<DataTemplate x:Key="ActiveContentTemplate">
<Grid>
<Path
Width="25"
Height="25"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M9 16.2L4.8 12l-1.4 1.4L9 19 21 7l-1.4-1.4L9 16.2z"
Fill="#2B579A"
Stroke="#2B579A" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="IndeterminateContentTemplate">
<Grid>
<Path
Width="25"
Height="25"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M12 4V1L8 5l4 4V6c3.31 0 6 2.69 6 6 0 1.01-.25 1.97-.7 2.81l1.46
1.46C19.54 15.03 20 13.57 20 12c0-4.42-3.58-8-8-8zm0 14c-3.31 0-6-2.69-6-6
0-1.01.25-1.97.7-2.81L5.24 7.74C4.46 8.97 4 10.43 4 12c0 4.42 3.58 8 8 8v3l4-
4-4-4v3z"
Fill="#2B579A"
Stroke="#2B579A" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="InactiveContentTemplate">
<Grid>
<Path
Width="25"
Height="25"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M19 6.41L17.59 5 12 10.59 6.41 5 5 6.41 10.59 12 5 17.59 6.41 19 12
13.41 17.59 19 19 17.59 13.41 12z"
Fill="#D2D2D2"
Stroke="#D2D2D2" />
</Grid>
</DataTemplate>
<local:StepViewItemTemplateSelector x:Key="stepViewItemTemplateSelector" />
</Window.Resources>
<syncfusion:SfStepProgressBar
```

```
x:Name="stepperControlName"
ItemsSource="{Binding StepViewItems}"
SelectedIndex="2"
ItemTemplateSelector="{StaticResource stepViewItemTemplateSelector}">
</syncfusion:SfStepProgressBar>
```

**C#**

```
/// <summary>
/// Represents the step view item template selector class.
/// </summary>
public class StepViewItemTemplateSelector : DataTemplateSelector
{
    /// <summary>
    /// Selects the template based on the step view item status.
    /// </summary>
    /// <param name="item">step view item.</param>
    /// <param name="container">step progress bar.</param>
    /// <returns></returns>
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
        StepViewItem stepViewItem = item as StepViewItem;
        if (stepViewItem != null)
        {
            if (stepViewItem.Status == StepStatus.Indeterminate)
            {
                return (item as StepViewItem).FindResource("IndeterminateContentTemplate")
                    as DataTemplate;
            }
            else if (stepViewItem.Status == StepStatus.Active)
            {
                return (item as StepViewItem).FindResource("ActiveContentTemplate") as
                    DataTemplate;
            }
            else
            {
                return (item as StepViewItem).FindResource("InactiveContentTemplate") as
                    DataTemplate;
            }
        }
        return null;
    }
}
```

This will generate the following Step ProgressBar control.



Download demo from [GitHub](#).

Marker Template Selector

With the

[MarkerTemplateSelector](https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.ProgressBar.SfStepP

progressBar.html#SyncfusionUIXamlProgressBarSfStepProgressBar\_MarkerTemplateSelector property), you can use the different templates for step marker depends on the step view item status. The following example shows this.

Use this marker template selector to choose a template for the Step ProgressBar control.

#### XML

```
<Window.Resources>
<DataTemplate x:Key="IndeterminateTemplate">
<Grid>
<Ellipse
Width="35"
Height="35"
Fill="#00ccb1"
Stroke="#00ccb1" />
<Path
Width="12"
Height="12"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M8,0 C12.411011,0 16,3.5890198 16,8 16,12.411011 12.411011,16 8,16
3.5889893,16 0,12.411011 0,8 0,3.5890198 3.5889893,0 8,0 z"
Fill="White"
Stretch="Fill" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="ActiveTemplate">
<Grid>
<Ellipse
Width="35"
Height="35"
Fill="#00ccb1"
Stroke="#00ccb1" />
<Path
Width="16"
Height="11"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M15.288992,0 L15.997,0.70702839 5.7260096,11.000999 0,5.8859658
0.66601563,5.1399964 5.6870084,9.6239898 z"
Fill="White"
Stretch="Fill"
Stroke="White" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="InactiveTemplate">
<Grid>
<Ellipse
Width="35"
Height="35"
Fill="FFFFFF"
Stroke="D2D2D2" />
</Grid>
</DataTemplate>
<local:StepViewItemMarkerTemplateSelector
x:Key="stepViewItemMarkerTemplateSelector" />
```

```
</Window.Resources>
<syncfusion:SfStepProgressBar
x:Name="stepperControlName"
Margin="40"
ItemsSource="{Binding StepViewItems}"
MarkerTemplateSelector="{StaticResource stepViewItemMarkerTemplateSelector}"
SelectedItemStatus="Indeterminate"
ActiveConnectorColor="#00ccb1"
SelectedIndex="2" >
<syncfusion:SfStepProgressBar.ItemContainerStyle>
<Style TargetType="syncfusion:StepViewItem">
<Setter Property="MarkerWidth" Value="35"/>
<Setter Property="MarkerHeight" Value="35"/>
</Style>
</syncfusion:SfStepProgressBar.ItemContainerStyle>
</syncfusion:SfStepProgressBar>
```

## C#

```
/// <summary>
/// Represents the step view item template selector class.
/// </summary>
public class StepViewItemMarkerTemplateSelector : DataTemplateSelector
{
    /// <summary>
    /// Selects the template based on the step view item status.
    /// </summary>
    /// <param name="item">step view item.</param>
    /// <param name="container">step progress bar.</param>
    /// <returns></returns>
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
        StepViewItem stepViewItem = item as StepViewItem;
        if (stepViewItem != null)
        {
            if (stepViewItem.Status == StepStatus.Indeterminate)
                return (item as StepViewItem).FindResource("IndeterminateTemplate") as DataTemplate;
            else if (stepViewItem.Status == StepStatus.Active)
                return (item as StepViewItem).FindResource("ActiveTemplate") as DataTemplate;
            else
                return (item as StepViewItem).FindResource("InactiveTemplate") as DataTemplate;
        }
        return null;
    }
}
```

This will populate the Step ProgressBar control.



Download demo from [GitHub](#).

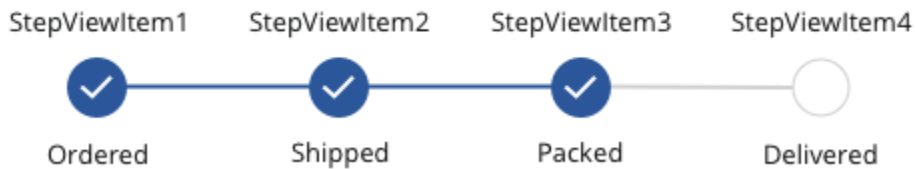
### Secondary content in Step ProgressBar

You can get or set the data template used to display the secondary content of [StepViewItem](#) by using the [SecondaryContentTemplate](#) property. The following example shows how to customize the step view item's secondary content with DataTemplate.

#### XML

```
<syncfusion:SfStepProgressBar
    ItemsSource="{Binding StepViewItems}"
    Orientation="Horizontal"
    SelectedIndex="2">
    <syncfusion:SfStepProgressBar.ItemContainerStyle>
    <Style TargetType="syncfusion:StepViewItem">
    <Setter Property="Content" Value="{Binding Text}" />
    </Style>
    </syncfusion:SfStepProgressBar.ItemContainerStyle>
    <syncfusion:SfStepProgressBar.SecondaryContentTemplate>
    <DataTemplate>
    <TextBlock
        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        Text="{Binding SecondaryText}"
        TextWrapping="Wrap" />
    </DataTemplate>
    </syncfusion:SfStepProgressBar.SecondaryContentTemplate>
    <syncfusion:SfStepProgressBar.DataContext>
    <local:ViewModel />
    </syncfusion:SfStepProgressBar.DataContext>
</syncfusion:SfStepProgressBar>
```

Implementing the above code will create the following Step ProgressBar control.



### Secondary Content Template Selector

Using the [SecondaryContentTemplateSelector](#) property, you can use the different templates for the step secondary content depends on the step view item status. The following example shows this.

Use this template selector to choose a template for the Step ProgressBar control.

#### XML

```
<Window.Resources>
<DataTemplate x:Key="ActiveContentTemplate">
```

```

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Path
Width="25"
Height="25"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M9 16.2L4.8 12l-1.4 1.4L9 19 21 7l-1.4-1.4L9 16.2z"
Fill="#2B579A"
Stroke="#2B579A" />
<TextBlock
Grid.Row="1"
Width="10"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Text="{Binding XPath=@PrimaryText}"
TextWrapping="Wrap" />
</Grid>
</DataTemplate>
<DataTemplate x:Key="IndeterminateContentTemplate">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Path
Width="25"
Height="25"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M12 4V1L8 5l4 4V6c3.31 0 6 2.69 6 6 0 1.01-.25 1.97-.7 2.8l1.46
1.46C19.54 15.03 20 13.57 20 12c0-4.42-3.58-8-8-8zm0 14c-3.31 0-6-2.69-6-6
0-1.01.25-1.97.7-2.8L5.24 7.74C4.46 8.97 4 10.43 4 12c0 4.42 3.58 8 8 8v3l4-
4-4-4v3z"
Fill="#2B579A"
Stroke="#2B579A" />
<TextBlock
Grid.Row="1"
Width="10"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Text="{Binding XPath=@PrimaryText}"
TextWrapping="Wrap" />
</Grid>
</DataTemplate>
<local:StepViewItemTemplateSelector x:Key="stepViewItemTemplateSelector" />
<XmlDataProvider x:Key="xmlSource" Source="Data.xml" XPath="StepItems" />
</Window.Resources>
<syncfusion:SfStepProgressBar
ItemsSource="{Binding Source={StaticResource xmlSource}, XPath=Step}"
SecondaryContentTemplateSelector="{StaticResource
stepViewItemTemplateSelector}"
SelectedIndex="{Binding Source={StaticResource xmlSource},
XPath=@SelectedIndex}"

```

```

SelectedItemStatus="Indeterminate">
<syncfusion:SfStepProgressBar.ItemContainerStyle>
<Style TargetType="syncfusion:StepViewItem">
<Setter Property="Content" Value="{Binding XPath=@Name}" />
</Style>
</syncfusion:SfStepProgressBar.ItemContainerStyle>
</syncfusion:SfStepProgressBar>

```

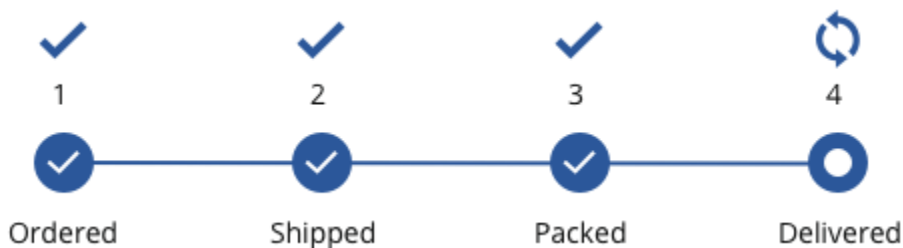
## C#

```

/// <summary>
/// Represents the step view item template selector class.
/// </summary>
public class StepViewItemTemplateSelector : DataTemplateSelector
{
    /// <summary>
    /// Selects the template based on the step view item status.
    /// </summary>
    /// <param name="item">step view item.</param>
    /// <param name="container">step progress bar.</param>
    /// <returns></returns>
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
        StepViewItem stepViewItem = item as StepViewItem;
        if (stepViewItem != null)
        {
            if (stepViewItem.Status == StepStatus.Indeterminate)
            {
                return (item as StepViewItem).FindResource("IndeterminateContentTemplate")
                as DataTemplate;
            }
            else
            {
                return (item as StepViewItem).FindResource("ActiveContentTemplate") as
                DataTemplate;
            }
        }
        return null;
    }
}

```

This will generate the following Step ProgressBar control.



## Secondary Content Template in Step View Item

You can get or set the data template used to display the secondary content of the [StepViewItem](#) by using the [SecondaryContentTemplate](#) property. The following example shows how to customize the step view item's secondary content with the DataTemplate.



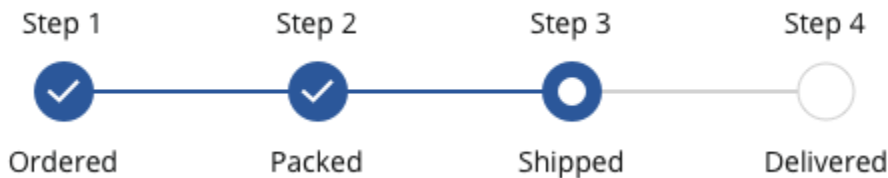
**XML**

```

<Window.Resources>
<DataTemplate x:Key="FirstStepSecondaryContentTemplate">
<TextBlock VerticalAlignment="Center" HorizontalAlignment="Center"
Text="Step 1"/>
</DataTemplate>
<DataTemplate x:Key="SecondStepSecondaryContentTemplate">
<TextBlock VerticalAlignment="Center" HorizontalAlignment="Center"
Text="Step 2"/>
</DataTemplate>
<DataTemplate x:Key="ThirdStepSecondaryContentTemplate">
<TextBlock VerticalAlignment="Center" HorizontalAlignment="Center"
Text="Step 3"/>
</DataTemplate>
<DataTemplate x:Key="FourthStepSecondaryContentTemplate">
<TextBlock VerticalAlignment="Center" HorizontalAlignment="Center"
Text="Step 4"/>
</DataTemplate>
</Window.Resources>
<Syncfusion:SfStepProgressBar SelectedIndex="2"
SelectedItemStatus="Indeterminate">
<Syncfusion:StepViewItem Content="Ordered"
SecondaryContentTemplate="{StaticResource
FirstStepSecondaryContentTemplate}" />
<Syncfusion:StepViewItem Content="Packed"
SecondaryContentTemplate="{StaticResource
SecondStepSecondaryContentTemplate}" />
<Syncfusion:StepViewItem Content="Shipped"
SecondaryContentTemplate="{StaticResource
ThirdStepSecondaryContentTemplate}" />
<Syncfusion:StepViewItem Content="Delivered"
SecondaryContentTemplate="{StaticResource
FourthStepSecondaryContentTemplate}" />
</Syncfusion:SfStepProgressBar>

```

Implementing the above code will create the following Step ProgressBar control.



### Layouts in WPF Step ProgressBar (SfStepProgressBar)

You can customize the layout of the Step progress bar in the following ways.

#### ItemsStretch

Represents how the item size is increased to fill the unused space. The default value of [ItemsStretch](#) property is [None](#).

#### Types:

- None

- Fill
- Auto

### None

The step view items retain their its natural size. You can change the spacing between the adjacent step view items by using the [ItemSpacing](#) property. The default value of this property is **80**.

**Note:** The single step size is calculated using the ItemSpacing and MarkerWidth or MarkerHeight based on the Orientation. The ItemSpacing applies between the current step view item and the previous step view item.

### XML

```
<Grid x:Name="grid">
<syncfusion:SfStepProgressBar SelectedIndex="2" ItemSpacing="150">
<syncfusion:StepViewItem Content="Ordered" />
<syncfusion:StepViewItem Content="Shipped" />
<syncfusion:StepViewItem Content="Packed" />
<syncfusion:StepViewItem Content="Delivered" />
</syncfusion:SfStepProgressBar>
</Grid>
```

### C#

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        SfStepProgressBar stepProgressBar = new SfStepProgressBar();
        StepViewItem orderedStepViewItem = new StepViewItem();
        StepViewItem shippedStepViewItem = new StepViewItem();
        StepViewItem packedStepViewItem = new StepViewItem();
        StepViewItem deliveredStepViewItem = new StepViewItem();
        orderedStepViewItem.Content = "Ordered";
        shippedStepViewItem.Content = "Shipped";
        packedStepViewItem.Content = "Packed";
        deliveredStepViewItem.Content = "Delivered";
        stepProgressBar.Items.Add(orderedStepViewItem);
        stepProgressBar.Items.Add(shippedStepViewItem);
        stepProgressBar.Items.Add(packedStepViewItem);
        stepProgressBar.Items.Add(deliveredStepViewItem);
        stepProgressBar.SelectedIndex = 2;
        stepProgressBar.ItemSpacing = 150;
        grid.Children.Add(stepProgressBar);
    }
}
```



### Fill

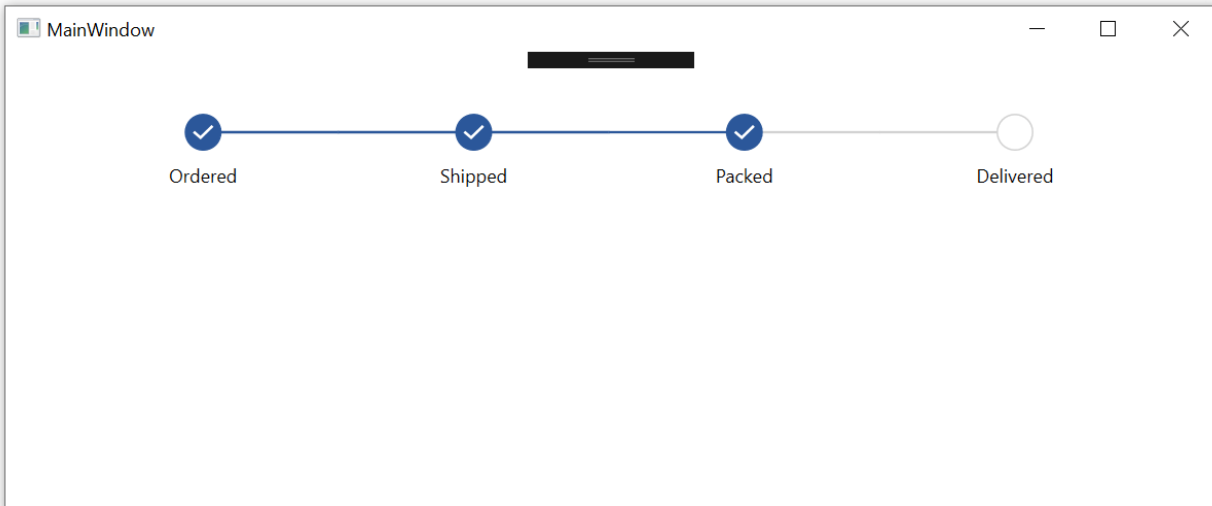
The step view items are sized to fill the available space.

### XML

```
<Grid x:Name="grid">
<syncfusion:SfStepProgressBar SelectedIndex="2" ItemsStretch="Fill">
<syncfusion:StepViewItem Content="Ordered" />
<syncfusion:StepViewItem Content="Shipped" />
<syncfusion:StepViewItem Content="Packed" />
<syncfusion:StepViewItem Content="Delivered" />
</syncfusion:SfStepProgressBar>
</Grid>
```

### C#

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        SfStepProgressBar stepProgressBar = new SfStepProgressBar();
        StepViewItem orderedStepViewItem = new StepViewItem();
        StepViewItem shippedStepViewItem = new StepViewItem();
        StepViewItem packedStepViewItem = new StepViewItem();
        StepViewItem deliveredStepViewItem = new StepViewItem();
        orderedStepViewItem.Content = "Ordered";
        shippedStepViewItem.Content = "Shipped";
        packedStepViewItem.Content = "Packed";
        deliveredStepViewItem.Content = "Delivered";
        stepProgressBar.Items.Add(orderedStepViewItem);
        stepProgressBar.Items.Add(shippedStepViewItem);
        stepProgressBar.Items.Add(packedStepViewItem);
        stepProgressBar.Items.Add(deliveredStepViewItem);
        stepProgressBar.SelectedIndex = 2;
        stepProgressBar.ItemsStretch = ItemsStretch.Fill;
        grid.Children.Add(stepProgressBar);
    }
}
```



### Auto

The step view item size is determined by the size of the content and secondary content. It is applicable only when the [Orientation](#) property is [Vertical](#). If the size of the content and secondary content is less than the [MarkerHeight](#), then the [MinimumItemSpacing](#) is used.

### XML

```
<Window.Resources>
<DataTemplate x:Key="FirstStepContentTemplate">
<TextBlock Width="100" Text="The SfStepProgressBar control is used to show
the progress of a multiple-step process." TextWrapping="Wrap">
<LineBreak/>
</TextBlock>
</DataTemplate>
<DataTemplate x:Key="SecondStepContentTemplate">
<TextBlock Width="100" Text="Visualize the step progress markers with
different shapes." TextWrapping="Wrap">
<LineBreak/>
</TextBlock>
</DataTemplate>
<DataTemplate x:Key="SecondStepSecondaryContentTemplate">
<TextBlock Width="100" Text="Step 2" Margin="120,0,0,0"
TextWrapping="Wrap"/>
</DataTemplate>
<DataTemplate x:Key="ThirdStepContentTemplate">
<TextBlock Width="100" Text="Supports active, inactive, and indeterminate
statuses to show progress." TextWrapping="Wrap">
<LineBreak/>
</TextBlock>
</DataTemplate>
<DataTemplate x:Key="FourthStepContentTemplate">
<TextBlock Width="100" Text="Customize the progress bar styles, markers, and
contents using the templates." TextWrapping="Wrap">
<LineBreak/>
</TextBlock>
</DataTemplate>
<DataTemplate x:Key="FourthStepSecondaryContentTemplate">
```

```

<TextBlock Width="100" Text="Step 4" Margin="120,0,0,0"
TextWrapping="Wrap"/>
</DataTemplate>
</Window.Resources>
<Grid Name="grid">
<syncfusion:SfStepProgressBar SelectedIndex="2" Margin="20"
Orientation="Vertical" ItemsStretch="Auto">
<syncfusion:StepViewItem Content="Ordered"
SecondaryContentTemplate="{StaticResource FirstStepContentTemplate}"/>
<syncfusion:StepViewItem ContentTemplate="{StaticResource
SecondStepContentTemplate}" SecondaryContentTemplate="{StaticResource
SecondStepSecondaryContentTemplate}"/>
<syncfusion:StepViewItem Content="Packed"
SecondaryContentTemplate="{StaticResource ThirdStepContentTemplate}"/>
<syncfusion:StepViewItem ContentTemplate="{StaticResource
FourthStepContentTemplate}" SecondaryContentTemplate="{StaticResource
FourthStepSecondaryContentTemplate}"/>
</syncfusion:SfStepProgressBar>
</Grid>

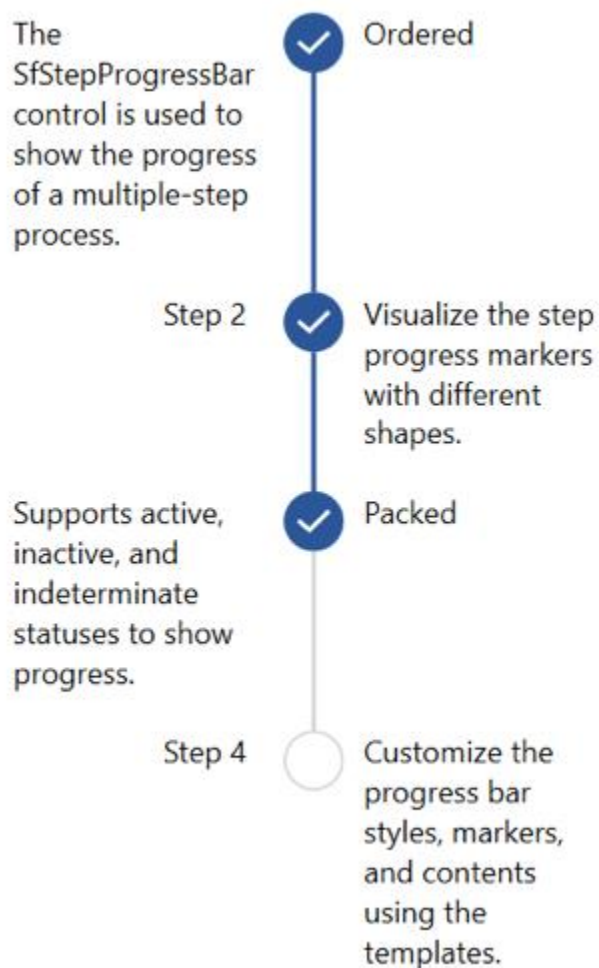
```

**C#**

```

public partial class MainWindow : Window
{
public MainWindow()
{
InitializeComponent();
SfStepProgressBar stepProgressBar = new SfStepProgressBar();
stepProgressBar.Orientation = Orientation.Vertical;
StepViewItem orderedStepViewItem = new StepViewItem();
StepViewItem shippedStepViewItem = new StepViewItem();
StepViewItem packedStepViewItem = new StepViewItem();
StepViewItem deliveredStepViewItem = new StepViewItem();
orderedStepViewItem.Content = "Ordered";
shippedStepViewItem.ContentTemplate =
FindResource("SecondStepContentTemplate") as DataTemplate;
packedStepViewItem.Content = "Packed";
deliveredStepViewItem.ContentTemplate =
FindResource("FourthStepContentTemplate") as DataTemplate;
orderedStepViewItem.SecondaryContentTemplate =
FindResource("FirstStepContentTemplate") as DataTemplate;
shippedStepViewItem.SecondaryContentTemplate =
FindResource("SecondStepSecondaryContentTemplate") as DataTemplate;
packedStepViewItem.SecondaryContentTemplate =
FindResource("ThirdStepContentTemplate") as DataTemplate;
deliveredStepViewItem.SecondaryContentTemplate =
FindResource("FourthStepSecondaryContentTemplate") as DataTemplate;
stepProgressBar.Items.Add(orderedStepViewItem);
stepProgressBar.Items.Add(shippedStepViewItem);
stepProgressBar.Items.Add(packedStepViewItem);
stepProgressBar.Items.Add(deliveredStepViewItem);
stepProgressBar.SelectedIndex = 2;
stepProgressBar.ItemsStretch = ItemsStretch.Fill;
grid.Children.Add(stepProgressBar);
}
}

```



### MinimumItemSpacing

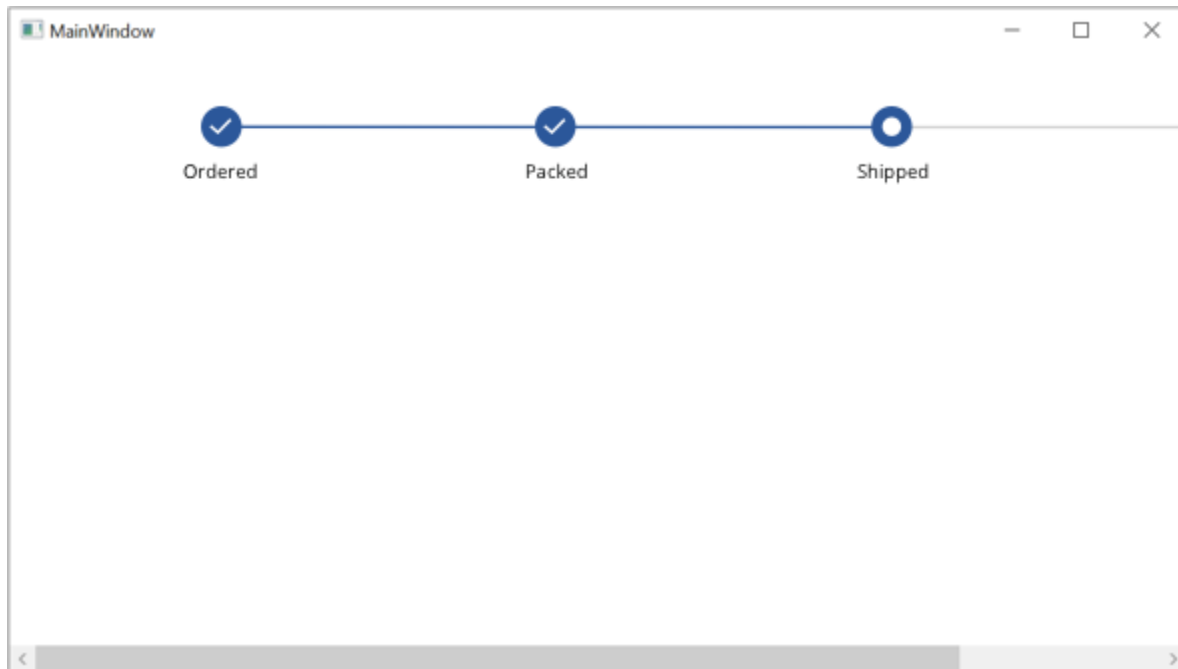
You can customize the value that indicates the minimum space between the step view items when the [ItemStretch](#) is [Fill](#). The default value of this property is 40. The following example shows how to customize the step view item's minimum space.

### XML

```
<ScrollViewer HorizontalScrollBarVisibility="Auto"
VerticalScrollBarVisibility="Auto">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>
    <Syncfusion:SfStepProgressBar
      ItemsStretch="Fill"
      MinimumItemSpacing="220"
      SelectedIndex="2"
      SelectedItemStatus="Indeterminate">
      <Syncfusion:StepViewItem Content="Ordered" />
      <Syncfusion:StepViewItem Content="Packed" />
      <Syncfusion:StepViewItem Content="Shipped" />
    </Syncfusion:SfStepProgressBar>
  </Grid>
</ScrollViewer>
```

```
<Syncfusion:StepViewItem Content="Delivered" />
</Syncfusion:SfStepProgressBar>
</Grid>
</ScrollViewer>
```

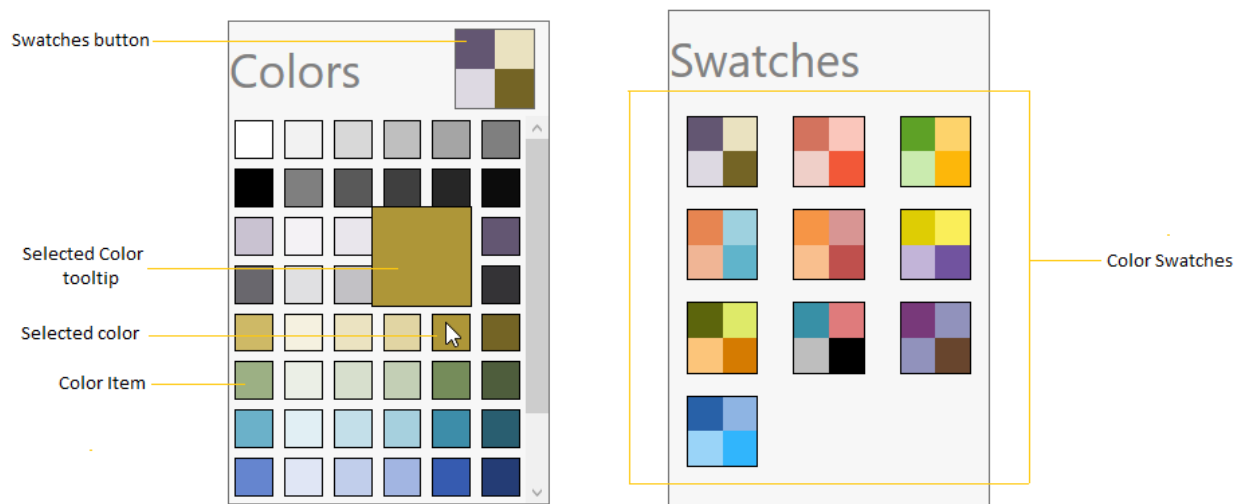
Implementing the above code will create the following Step ProgressBar control.



## SfColorPalette

### WPF Color Palette (SfColorPalette) Overview

The various elements of the [sfColorPalette](#) are illustrated in the following screenshot.



- The [Swatch button](#) is used to switch between available color packages.
- In Swatch button different [Color Swatches](#) are available to allow the user to select colors.

- The [Color item](#) displays the various color items available in current swatch.
- Color ToolTip shows preview of the hovered color item.

### Key features

Color Swatches – You can choose different color items from the different swatches colors.

SelectedColor – Returns the currently selected color.

Binding support – Selected color can be bound with any object.

### Getting Started with WPF Color Palette (SfColorPalette)

This section explains how to create a WPF [SfColorPalette](#) and explains about its structure and features.

### Assembly deployment

Refer to the [Control Dependencies](#) section to get the list of assemblies or **NuGet** package that needs to be added as a reference to use the control in any application.

Refer to this [documentation](#) to find more details about installing nuget packages in a WPF application.

### Creating Application with SfColorPalette control

In this walk through, user will create a WPF application that contains [SfColorPalette](#) control.

1. Creating project
2. Adding control via designer.
3. Adding control manually in XAML.
4. Adding control manually in C#.

### Creating project

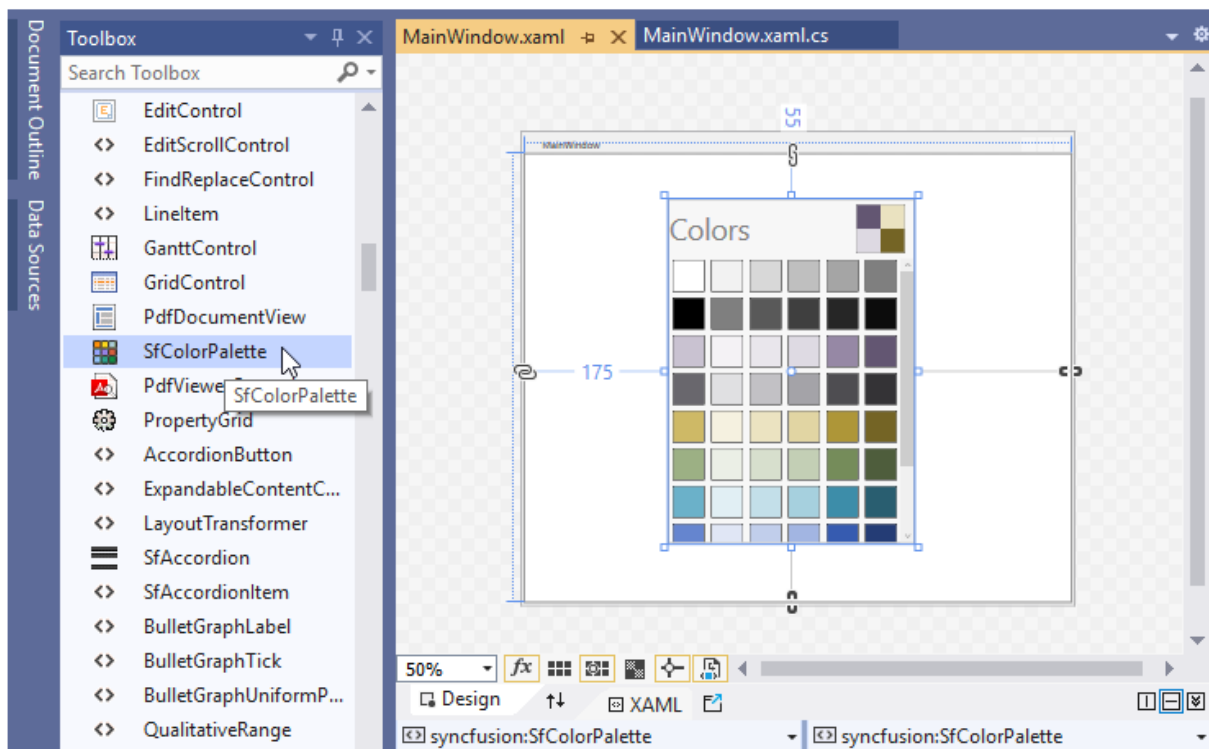
Below section provides detailed information to create new project in Visual Studio to display [SfColorPalette](#).

### Adding control via designer

The [SfColorPalette](#) control can be added to the application by dragging it from Toolbox and dropping it in designer. The required assembly will be added automatically.

- Syncfusion.SfColorPalette.WPF
- Syncfusion.SfShared.WPF





#### Adding control manually in XAML

In order to add [SfColorPalette](#) control manually in XAML, do the below steps,

1. Add the below required assembly references to the project,
  - Syncfusion.SfColorPalette.Wpf
  - Syncfusion.SfShared.Wpf
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> or SfColorPalette namespace [Syncfusion.Windows.Controls.Media](#) in XAML page.
3. Declare [SfColorPalette](#) control in XAML page.

#### XML

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:Check_UG"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="Check_UG.MainWindow"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<syncfusion:SfColorPalette HorizontalAlignment="Left" Margin="90,50,0,0"
VerticalAlignment="Top" Height="206" Width="239"/>
</Grid>
</Window>
```



#### *Adding control manually in C#*

In order to add [SfColorPalette](#) control manually in C#, do the below steps,

1. Add the below required assembly references to the project,
  - Syncfusion.SfColorPalette.Wpf
  - Syncfusion.SfShared.Wpf
2. Import SfColorPalette namespace **Syncfusion.Windows.Controls.Media** .

#### **C#**

```
using Syncfusion.Windows.Controls.Media;
```

3. Create [sfColorPalette](#) control instance and add it to the Page.

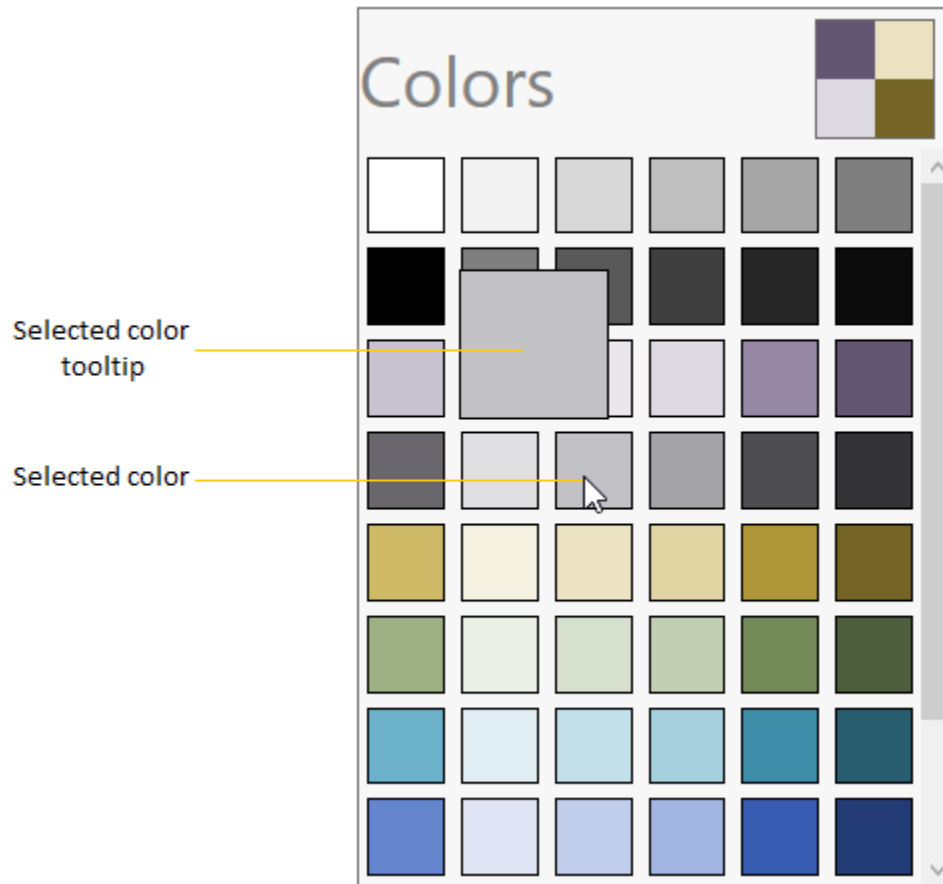
#### **C#**

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        SfColorPalette colorPalette = new SfColorPalette();
        colorPalette.Height = 300;
        colorPalette.Width = 200;
        this.Content = colorPalette;
    }
}
```

**Note:** [View Sample in GitHub](#)

### Select a Color

You can select any color by clicking the respective color item in the [SfColorPalette](#). You can get selected color from the [SelectedColor](#) property. If you select any color, then the selected color will be displayed as tooltip.



**Note:** [View Sample in GitHub](#)

### Binding a selected color

You can bind the selected color of [SfColorPalette](#) to any objects by using the [SelectedColor](#) property.

Here, the `SelectedColor` of the `SfColorPalette` bind with the `Rectangle.Fill` property with color to brush converter.

### C#

```
//ColorToBrushConverter.cs
public class ColorToSolidColorBrushValueConverter : IValueConverter {
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture) {
        if (null == value)
            return null;
        Color color = (Color)value;
        return new SolidColorBrush(color);
    }
}
```

```
public object ConvertBack(object value, Type targetType, object parameter,
    CultureInfo culture) {
    return true;
}
```

**XML**

```
<Grid>
<Grid.Resources>
<!--Color to brush converter-->
<local:ColorToSolidColorBrushValueConverter
x:Key="ColorToSolidColorBrush_ValueConverter"/>
</Grid.Resources>
<StackPanel>
<TextBlock Text="SelectedColor"/>
<Rectangle Fill="{Binding ElementName=SfColorPalette ,
    Path= SelectedColor,
    Converter={StaticResource ColorToSolidColorBrush_ValueConverter}}"/>
<syncfusion:SfColorPalette x:Name="SfColorPalette" />
</StackPanel>
</Grid>
```

**SelectedColor**

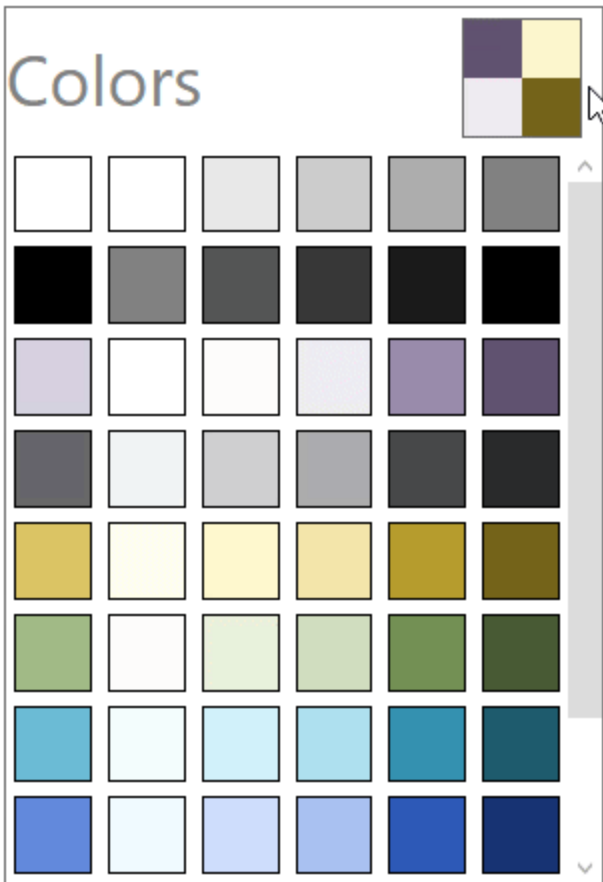
**Note:** [View Sample in GitHub](#)

Navigate to the list of swatches

You can navigate and select a different colors from the different swatches by clicking the Swatches button which is placed on the right top corner of the [SfColorPalette](#) control.

*List of swatches*





---

**Note:** [View Sample in GitHub.](#)

---

### Theme

ColorPalette supports various built-in themes. Refer to the below links to apply themes for the ColorPalette,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



---

**Note:** [View Sample in GitHub.](#)

---

### Appearance in WPF Color Palette (SfColorPalette)

This section explains different UI customization options available in [ColorPalette](#) control.

#### Setting the Foreground

You can change the foreground color for `ColorPalette` by setting the `Foreground` property. The default color value of `Foreground` property is `Gray`.

#### XML

```
<syncfusion:SfColorPalette Foreground="Red"
Name="sfColorPalette"/>
```

#### C#

```
SfColorPalette sfColorPalette = new SfColorPalette();
sfColorPalette.Foreground = Brushes.Red;
```



**Note:** View [Sample](#) in GitHub

### Setting the Background

You can change the background color for ColorPalette by setting the Background property. The default color value of Background property is Snow.

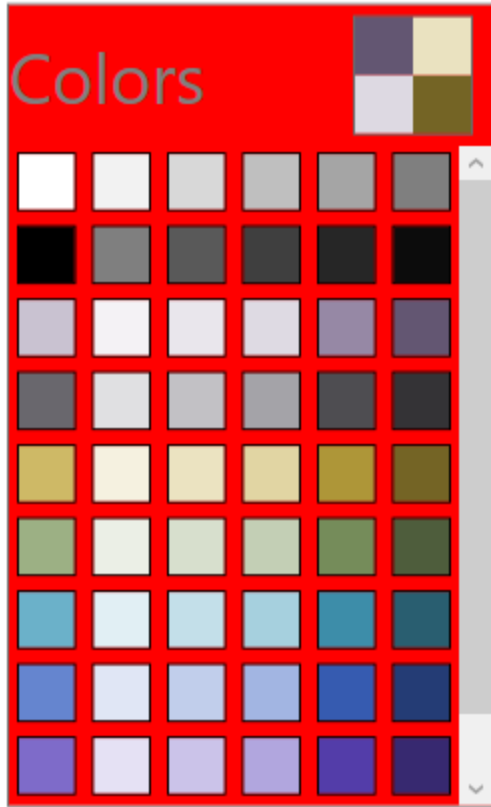
### XML

```
<syncfusion:SfColorPalette Background="Red"
Name="sfColorPalette"/>
```

### C#

```
SfColorPalette sfColorPalette = new SfColorPalette();
sfColorPalette.Background = Brushes.Red;
```





**Note:** View [Sample](#) in GitHub

#### Change flow direction

You can change the flow direction of the `ColorPalette` layout from right to left by setting the `FlowDirection` property value as `RightToLeft`. The Default value of `FlowDirection` property is `LeftToRight`.

#### XML

```
<syncfusion:SfColorPalette FlowDirection="RightToLeft"
Name="sfColorPalette"/>
```

#### C#

```
SfColorPalette sfColorPalette = new SfColorPalette();
sfColorPalette.FlowDirection = FlowDirection.RightToLeft;
```



---

**Note:** View [Sample](#) in GitHub

---

### Theme

ColorPalette supports various built-in themes. Refer to the below links to apply themes for the ColorPalette,

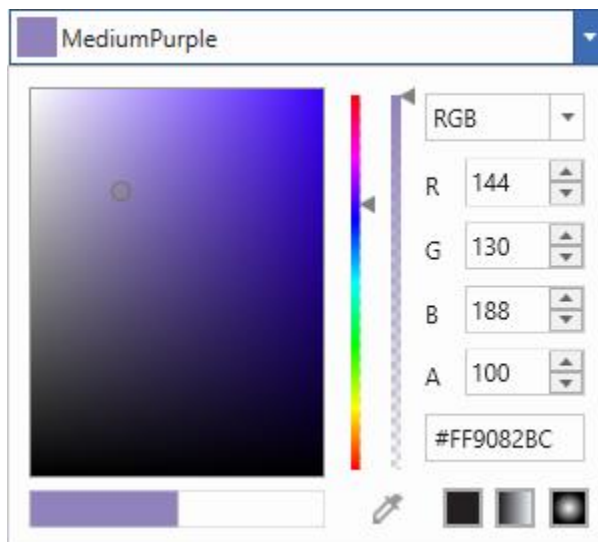
- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



## ColorPicker

### WPF color picker (ColorPicker) Overview

WPF [ColorPicker](#) is an user interface to select and adjust color values. This supports various color specifications like RGB (Red Green Blue), HSV (Hue Saturation Value), and Hex codes.



This section also discusses the WPF [ColorEdit](#) control, which is similar to the [ColorPicker](#).

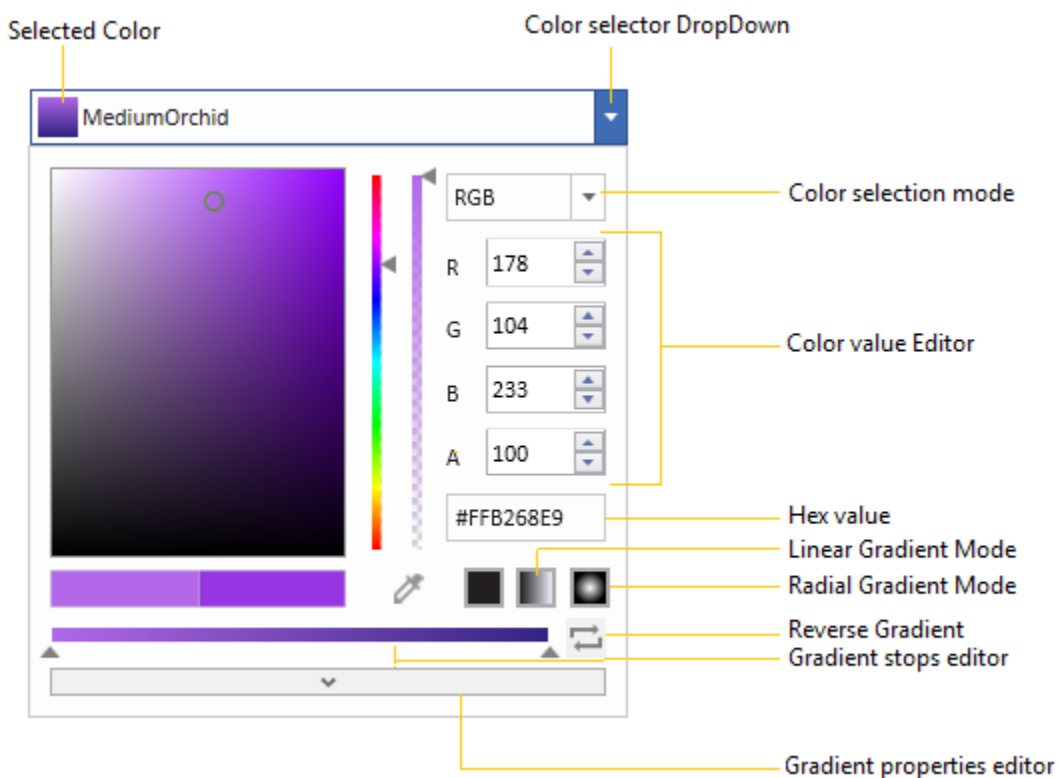
## Features

- **Color Editing** - Drag the handle to pick your favorite color in the picker region or set the color values manually with input control (RGB or HSV or Hex codes).
- A slider displaying the **Hue spectrum**, allowing for selection of a hue value.
- **Gradient Editor** - Sliders capable of altering the stops of a linear or radial brush.
- **EyeDropper** - Drag the eyedropper to anywhere in your application to pick the color of a specific location or even a point.
- Supports **ToolTip** to show the selected color while dragging the picker in **ColorEdit** control.
- Supports **rich set of Themes** to alter the look and feel of the control according to the application needs.
- Comes with build-in color palette drop down for ease color selection.

## Getting Started with WPF color picker (ColorPicker)

This section explains how to create a WPF [ColorPicker](#) and explains about its structure and features.

### Structure of ColorPicker



### Assembly deployment

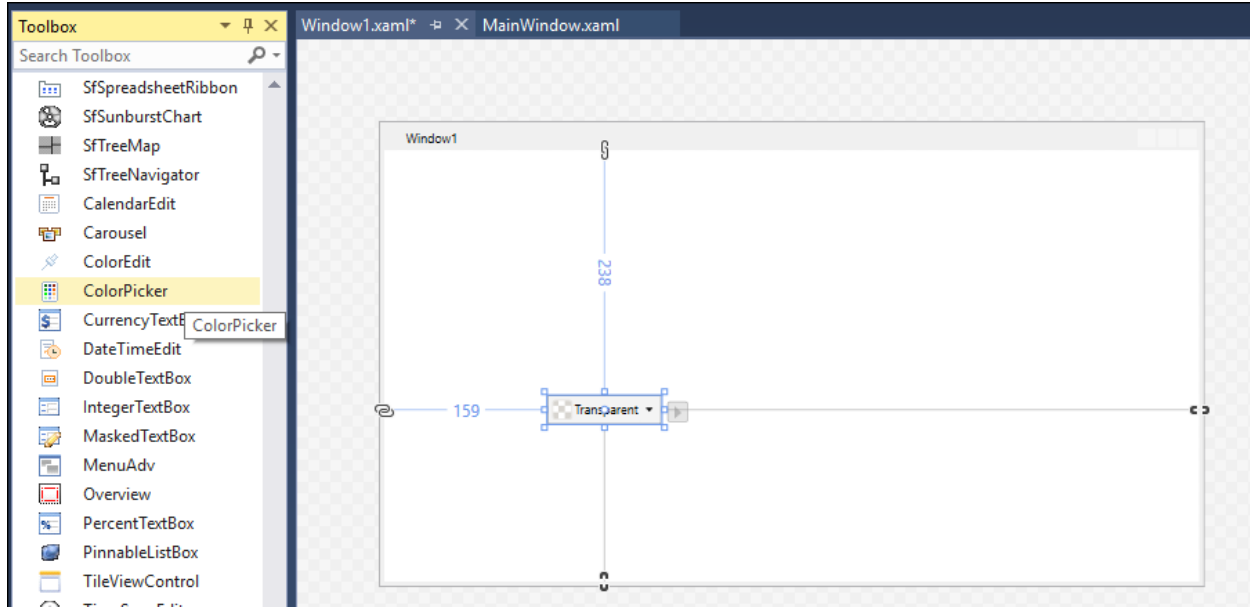
Refer to the [Control Dependencies](#) section to get the list of assemblies or **NuGet** package that needs to be added as a reference to use the control in any application.

Refer to this [documentation](#) to find more details about installing nuget packages in a WPF application.

### Adding WPF ColorPicker via designer

**ColorPicker** can be added to an application by dragging it from the toolbox to a designer view. The following dependent assemblies will be added automatically:

- Syncfusion.Shared.WPF



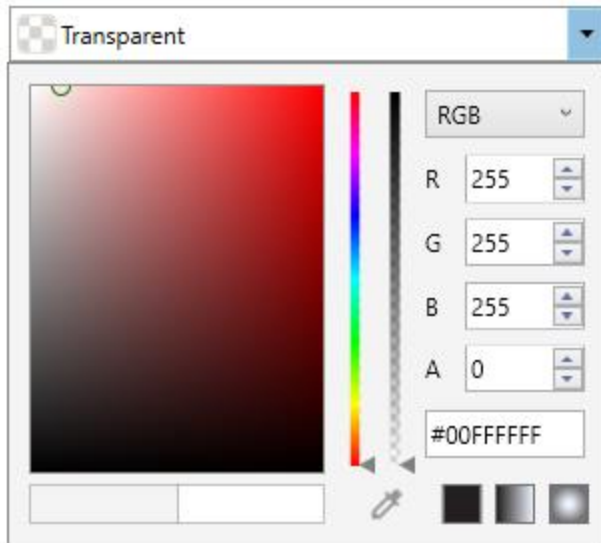
### Adding WPF ColorPicker via XAML

To add the **ColorPicker** manually in XAML, follow these steps:

- 1) Create a new WPF project in Visual Studio.
- 2) Add the following required assembly reference to the project: \* Syncfusion.Shared.WPF
- 3) Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf>, and declare the **ColorPicker** in WPF XAML page.

### XML

```
<Window x:Class="ColorPicker_sample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:ColorPicker_sample"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid Name="grid">
<syncfusion:ColorPicker Name="colorPicker" Height="100" Width="280"/>
</Grid>
```



#### Adding WPF ColorPicker via C#

To add the **ColorPicker** manually in C#, follow these steps:

1) Create a new WPF application via Visual Studio. 2) Add the following required assembly references to the project: \* Syncfusion.Shared.WPF 3) Include the required namespace.

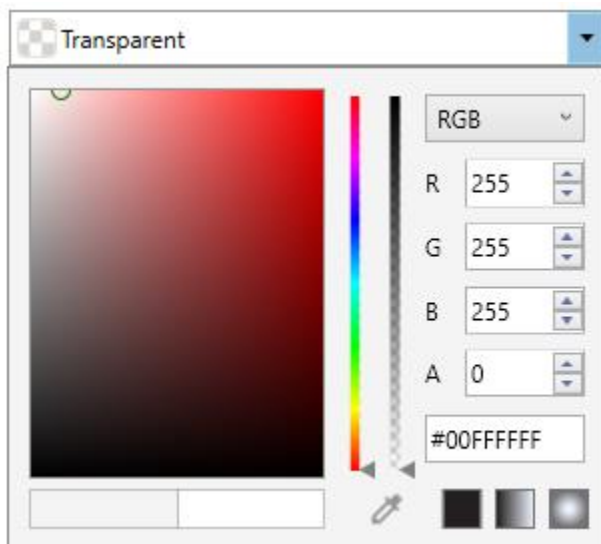
#### C#

```
using Syncfusion.Windows.Shared;
```

4) Create an instance of **ColorPicker**, and add it to the window.

#### C#

```
ColorPicker colorPicker = new ColorPicker();  
colorPicker.Width = 300;  
colorPicker.Height=100;
```



### Select a Color

We can select a solid color or gradient color from a `ColorPicker` using the [Color](#) and [Brush](#) properties.

#### Select Solid Color

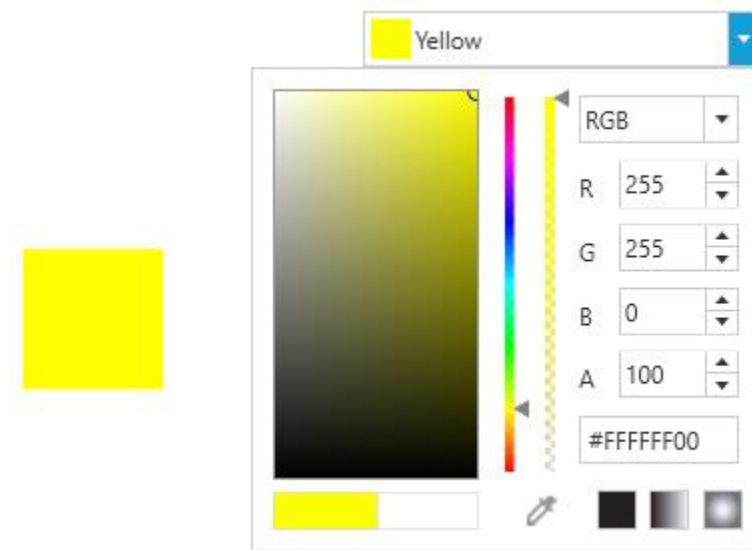
We can select the solid color by using the `Color` property.

#### XML

```
<syncfusion:ColorPicker x:Name="colorPicker"
    Color="Yellow"/>
```

#### C#

```
ColorPicker colorPicker = new ColorPicker();
colorPicker.Color = Colors.Yellow;
```



#### Select a Gradient Color

We can select a linear or radial gradient color which holds the multiple colors from the `ColorPicker`.

#### Linear Gradient #####

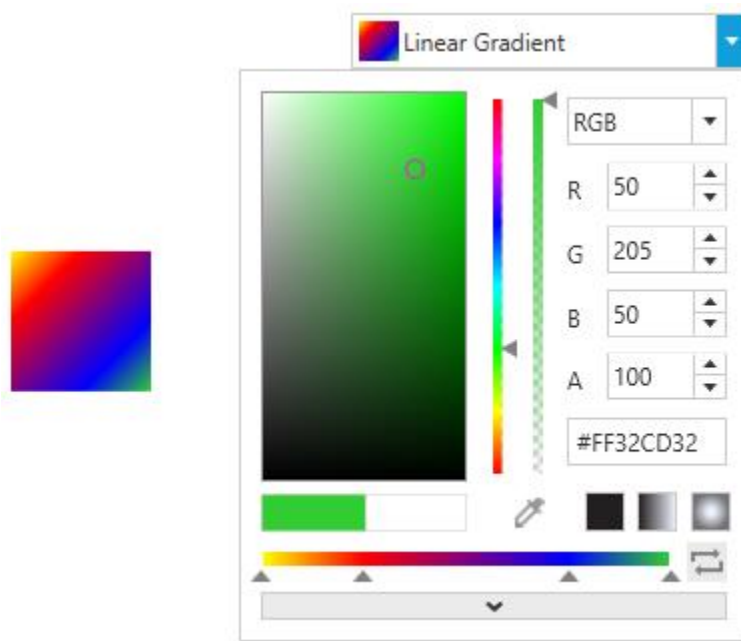
Linear Gradient color can be selected by the multiple colors and their location along the gradient axis using the `GradientStops` objects and [StartPoint](#) and [EndPoint](#) properties. Based on the `StartPoint` and `EndPoint`, the selected colors will be combined in linear manner.

#### XML

```
<syncfusion:ColorPicker x:Name="colorPicker" Width="200">
    <syncfusion:ColorPicker.Brush>
        <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
            <GradientStop Color="Yellow" Offset="0.0" />
            <GradientStop Color="Red" Offset="0.25" />
            <GradientStop Color="Blue" Offset="0.75" />
            <GradientStop Color="LimeGreen" Offset="1.0" />
        </LinearGradientBrush>
    </syncfusion:ColorPicker.Brush>
</syncfusion:ColorPicker>
```

**C#**

```
//Creating the linear gradient brush
LinearGradientBrush linearGradient = new LinearGradientBrush();
linearGradient.StartPoint = new Point(0, 0);
linearGradient.EndPoint = new Point(1, 1);
linearGradient.GradientStops.Add(new GradientStop(Colors.Yellow, 0.0));
linearGradient.GradientStops.Add(new GradientStop(Colors.Red, 0.25));
linearGradient.GradientStops.Add(new GradientStop(Colors.Blue, 0.75));
linearGradient.GradientStops.Add(new GradientStop(Colors.LimeGreen, 1.0));
//Assigning a linear gradient brush to ColorPicker
ColorPicker colorPicker= new ColorPicker();
colorPicker.Brush = linearGradient;
```

**Radial Gradient #####**

Radial Gradient color is similar to Linear Gradient color, except for the axis defined by the circle. Based on the [GradientOrigin](#), [Center](#) and [RadiusPoint](#) properties values, the selected gradient colors are combined in a circle manner.

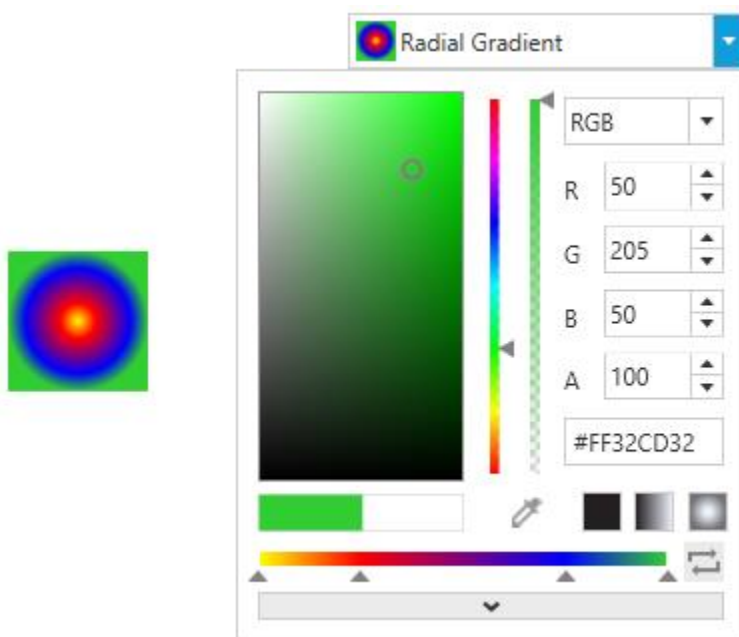
**XML**

```
<syncfusion:ColorPicker x:Name="colorPicker" Width="200">
  <syncfusion:ColorPicker.Brush>
    <RadialGradientBrush GradientOrigin="0.5,0.5" Center="0.5,0.5" RadiusX="0.5"
      RadiusY="0.5">
      <GradientStop Color="Yellow" Offset="0" />
      <GradientStop Color="Red" Offset="0.25" />
      <GradientStop Color="Blue" Offset="0.75" />
      <GradientStop Color="LimeGreen" Offset="1" />
    </RadialGradientBrush>
  </syncfusion:ColorPicker.Brush>
</syncfusion:ColorPicker>
```



**C#**

```
//Creating a  
RadialGradientBrush radialGradient = new RadialGradientBrush();  
radialGradient.GradientOrigin = new Point(0.5, 0.5);  
radialGradient.Center = new Point(0.5, 0.5);  
radialGradient.RadiusX = 0.5;  
radialGradient.RadiusY = 0.5;  
radialGradient.GradientStops.Add(new GradientStop(Colors.Yellow, 0.0));  
radialGradient.GradientStops.Add(new GradientStop(Colors.Red, 0.25));  
radialGradient.GradientStops.Add(new GradientStop(Colors.Blue, 0.75));  
radialGradient.GradientStops.Add(new GradientStop(Colors.LimeGreen, 1.0));  
colorPicker.Brush = radialGradient;
```

*Change Selected Color at runtime*

**ColorPicker** consist of bunch of input components to select color and edit its properties at runtime.



Color and Brush changed notification

Selected Color and Brush changed in [ColorPicker](#) can be examined using [SelectedBrushChanged](#) and [ColorChanged](#) events.

#### XML

```
<syncfusion:ColorPicker ColorChanged="ColorPicker_ColorChanged"
SelectedBrushChanged="ColorPicker_SelectedBrushChanged"
Name="colorPicker"/>
```

#### C#

```
ColorPicker colorPicker = new ColorPicker();
colorPicker.SelectedBrushChanged += ColorPicker_SelectedBrushChanged;
colorPicker.ColorChanged += ColorPicker_ColorChanged;
```

#### C#

```
//Invoked when the selected color is changed
private void ColorPicker_ColorChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
// Enter your code here
}
//Invoked when the selected brush is changed
private void ColorPicker_SelectedBrushChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
// Enter your code here
}
```

### Change opacity of the color

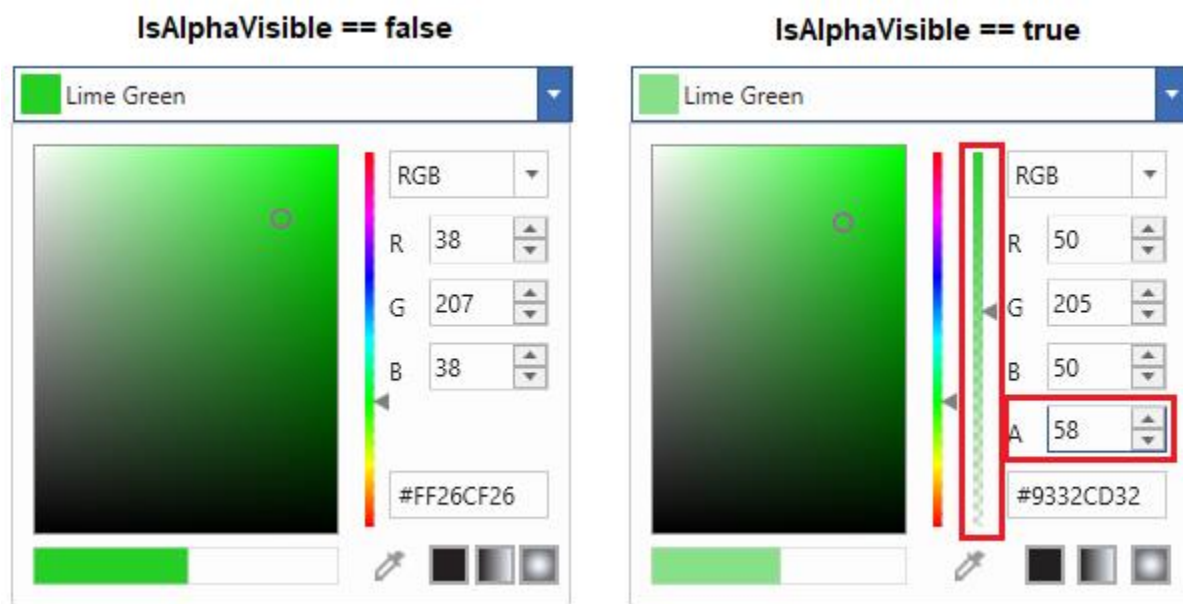
We can change the opacity of the selected color by using the A-Alpha value editor or delicate slider in the ColorPicker. We can hide the A-Alpha value editor and delicate slider by using the [IsAlphaVisible](#) property value as `false`. The default value of the `IsAlphaVisible` property is `true`.

#### XML

```
<syncfusion:ColorPicker IsAlphaVisible="False" x:Name="colorPicker" />
```

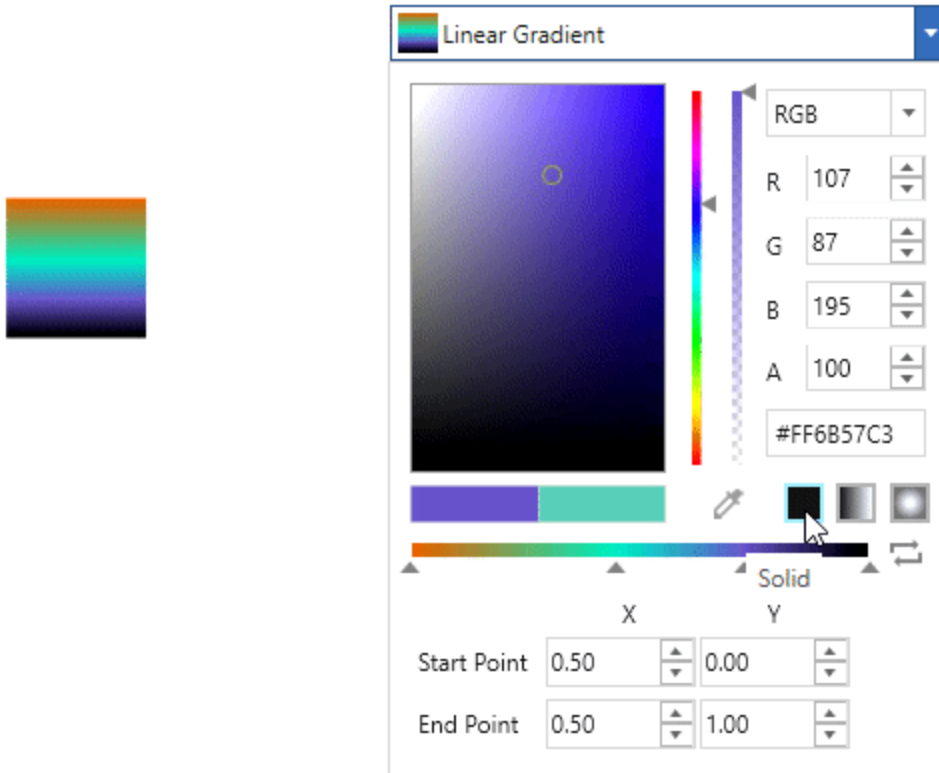
#### C#

```
ColorPicker colorPicker = new ColorPicker();  
colorPicker.IsAlphaVisible = false;
```



### Switch between Solid, Linear and Gradient brush mode

We can change the color selection mode directly by clicking on the corresponding Solid, Linear or Gradient brush mode buttons which are placed in the bottom right corner of the ColorPicker.



#### *Restrict the brush mode from Solid to Gradient*

We can restrict color selection mode switching at runtime by setting the [EnableSolidToGradientSwitch](#) property value as `false`. It will hide the Solid, Linear and Gradient brush switch buttons.

#### **XML**

```
<syncfusion:ColorPicker x:Name="colorPicker"
    EnableSolidToGradientSwitch="false"/>
```

#### **C#**

```
ColorPicker colorPicker = new ColorPicker ();
colorPicker.EnableSolidToGradientSwitch = false;
```

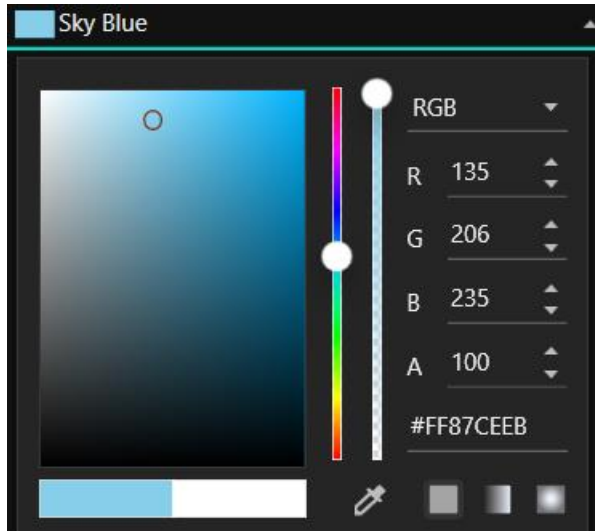


Click [here](#) to download the sample that showcases the **ColorPicker** overall features.

### Theme

ColorPicker supports various built-in themes. Refer to the below links to apply themes for the ColorPicker,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Select solid color in WPF color picker (ColorPicker)

This section explains how to select a solid color from different color models, how to modify their individual properties and also gives brief information about eye dropper, standard colors.

#### *What is solid color?*

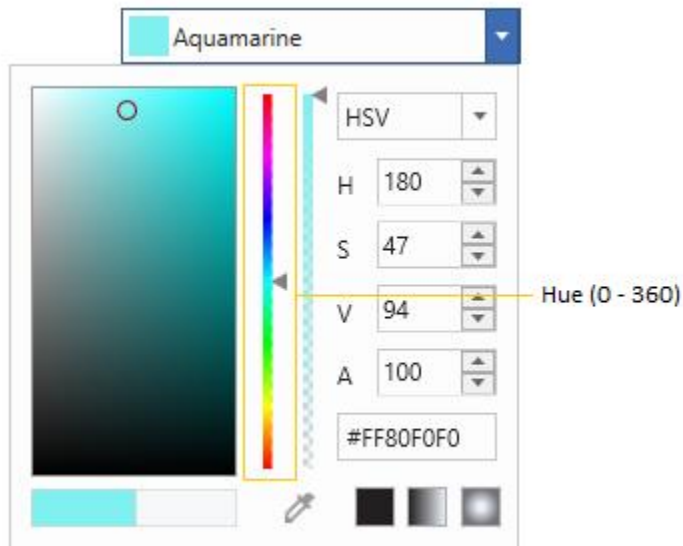
Solid color comprises a single color with its alpha, red, blue and green channels or use one of the predefined color provided by the **Colors** class.

### How to select your solid color

Choosing solid color from HSV(Hue, saturation and value) explained below.

#### Hue

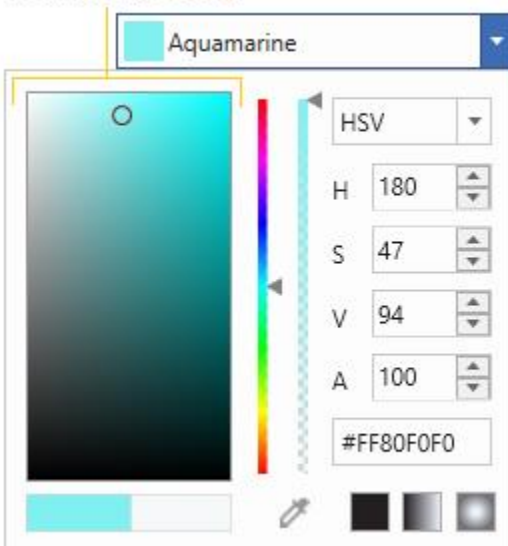
Hue is the color portion of the model, expressed as a number between 0 and 360 degrees, with all colors falling within a certain range. In [ColorPicker](#), the Hue value can be modified using the slider or H-Hue value editor.



#### Saturation

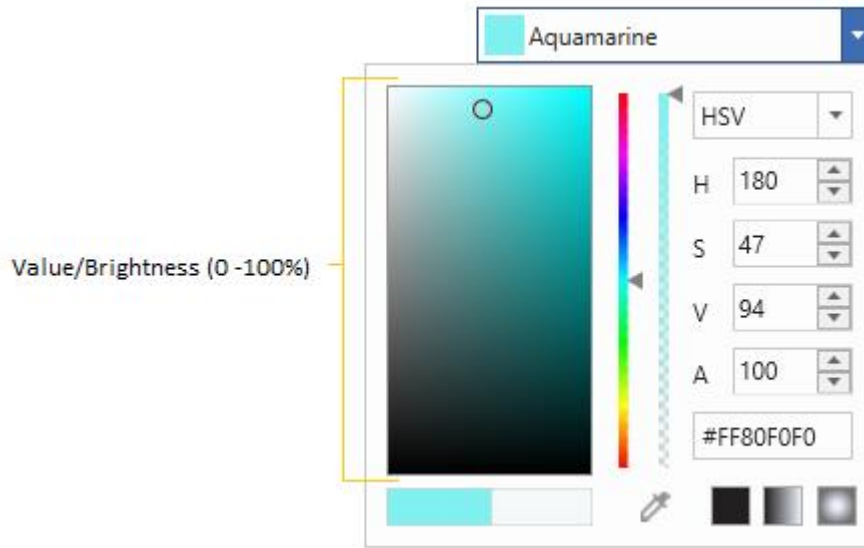
Saturation describes the amount of gray in a particular color, from 0 to 100 percent. The Saturation value can be modified using the slider or S-Saturation value editor.

Saturation (0 - 100%)



### Value/Brightness

Value works in conjunction with saturation and describes the brightness or intensity of the color, from 0-100 percent, where 0 is completely black, and 100 is the brightest and reveals the most color. The Value/Brightness value can be modified using the slider or V-Value value editor.



### Select RGB and HSV color

ColorPicker controls can be displayed in two different modes. They are HSV and RGB modes. The VisualizationStyle property is used to switch between these modes. By default, the RGB mode is enabled.

### RGB

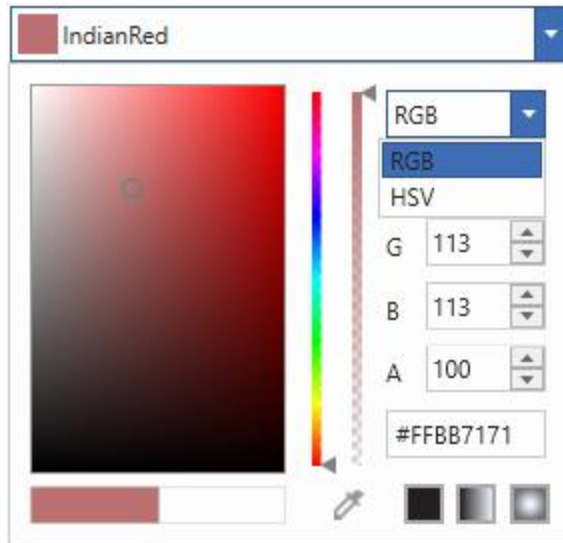
We can pick a color in RGB (Red, green, and blue) color format by setting the value of the VisualizationStyle property as ColorSelectionMode.RGB. Color formats can be switched from HSV to RGB at runtime, using built-in color model ComboBox.

### XML

```
<syncfusion:ColorPicker VisualizationStyle="RGB" Name="colorpicker"/>
```

### C#

```
ColorPicker colorPicker = new ColorPicker();  
colorPicker.VisualizationStyle = ColorSelectionMode.RGB;  
this.Content = colorPicker;
```



### HSV

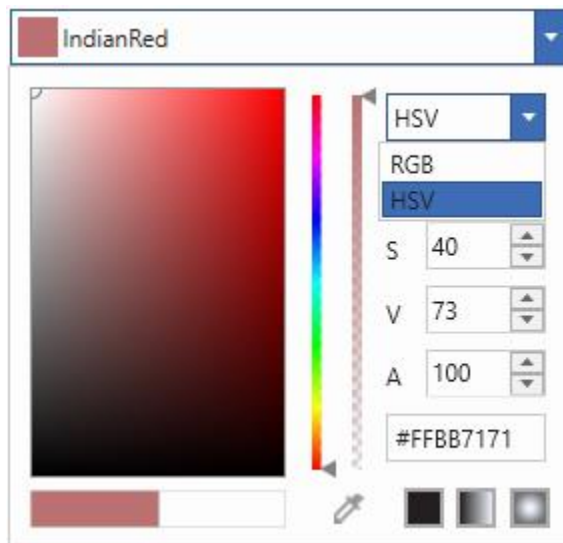
We can pick a color in HSV (Hue, Saturation, and Value/Brightness) color format by setting the value of the `VisualizationStyle` property as [ColorSelectionMode.HSV](#). Color formats can be switched from RGB to HSV at runtime, using built-in color model `ComboBox`.

### XML

```
<syncfusion:ColorPicker VisualizationStyle="HSV" Name="colorpicker"/>
```

### C#

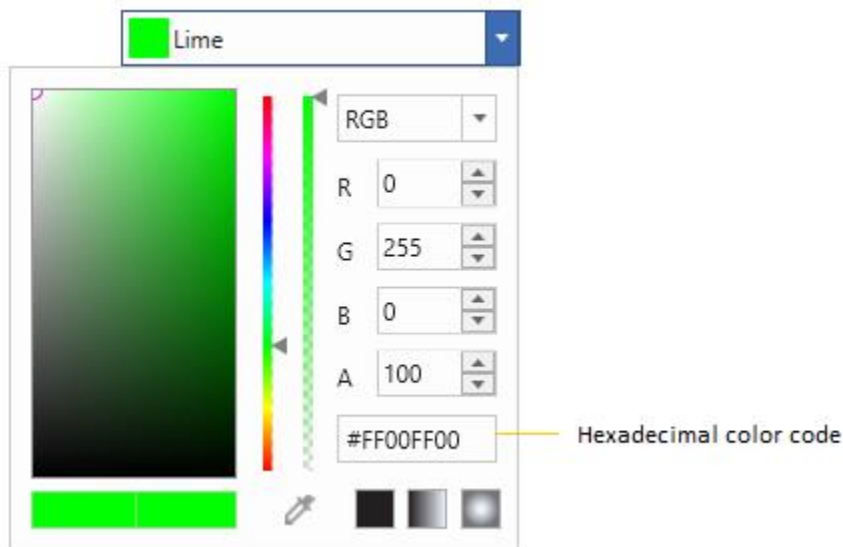
```
ColorPicker colorPicker = new ColorPicker();  
colorPicker.VisualizationStyle = ColorSelectionMode.HSV;  
this.Content = colorPicker;
```





### Get solid color using Hexadecimal code

Hexadecimal color values are also supported in `ColorPicker`, the built-in `TextBox` helps with color selection and editing. Based on the hexadecimal values in the `TextBox`, the color will be picked.



### Pick a color from anywhere (Eye Dropper)

`ColorPicker` consists of an `eye-dropper` which can be dragged across the anywhere on the screen and picks the color where it is currently hovering above, along with the associated hexadecimal (HEX) color value.



### Select a standard color

`ColorPicker` has built-in color `ComboBox` to select standard color easily. By default, the standard color `ComboBox` is not shown in the `ColorPicker`. If we want to use the standard color `ComboBox`, use the [IsColorPaletteVisible](#) property value as `true`. The default value of `IsColorPaletteVisible` property is `false`.

### XML

```
<syncfusion:ColorPicker x:Name="colorPicker" IsColorPaletteVisible="True"/>
```

**C#**

```
ColorPicker colorPicker = new ColorPicker();  
colorPicker.IsColorPaletteVisible = true;
```



---

Solid color changed notification

Selected color changed in [ColorPicker](#) can be examined using [ColorChanged](#) event.

**XML**

```
<syncfusion:ColorPicker ColorChanged="ColorPicker_ColorChanged"
Name="colorPicker"/>
```

**C#**

```
ColorPicker colorPicker = new ColorPicker();
colorPicker.ColorChanged += ColorPicker_ColorChanged;
```

**C#**

```
//Invoked when the selected color is changed
private void ColorPicker_ColorChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
// Enter your code here
}
```

Get color name from color property

**ColorPicker** come along with method which returns the nearest names of **Color** property, this can be obtained by **SuchColor** method. We can get similar four color names of the **Color** property by passing the index value from 0 to 3 in the **SuchColor** method.

**XML**

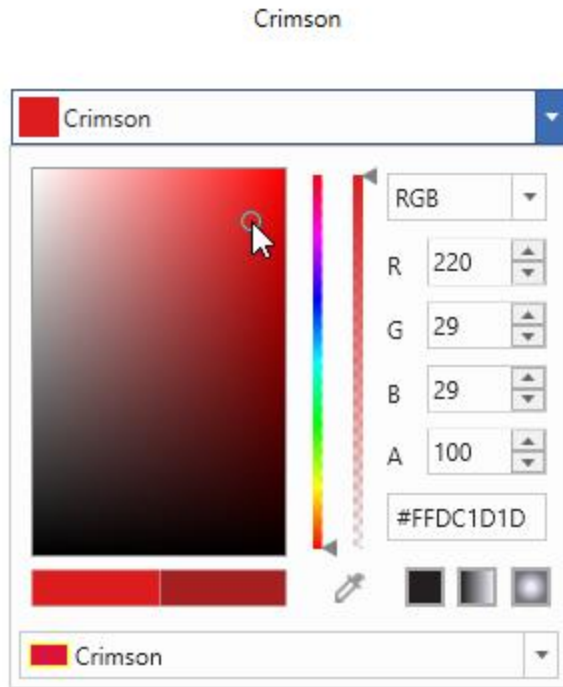
```
<TextBlock Name= "textBlock" Width="200" Height="30"/>
<syncfusion:ColorPicker Name="colorPicker"
SelectedBrushChanged="ColorPicker_SelectedBrushChanged"/>
```

**C#**

```
ColorPicker colorPicker= new ColorPicker();
colorPicker.SelectedBrushChanged += ColorPicker_SelectedBrushChanged;
```

**C#**

```
private void ColorPicker_SelectedBrushChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
textBlock.Text =
Syncfusion.Windows.Shared.ColorEdit.SuchColor(colorPicker.Color)[0];
}
```



We can select a gradient colors which is explained in the [Select gradient color](#) page.

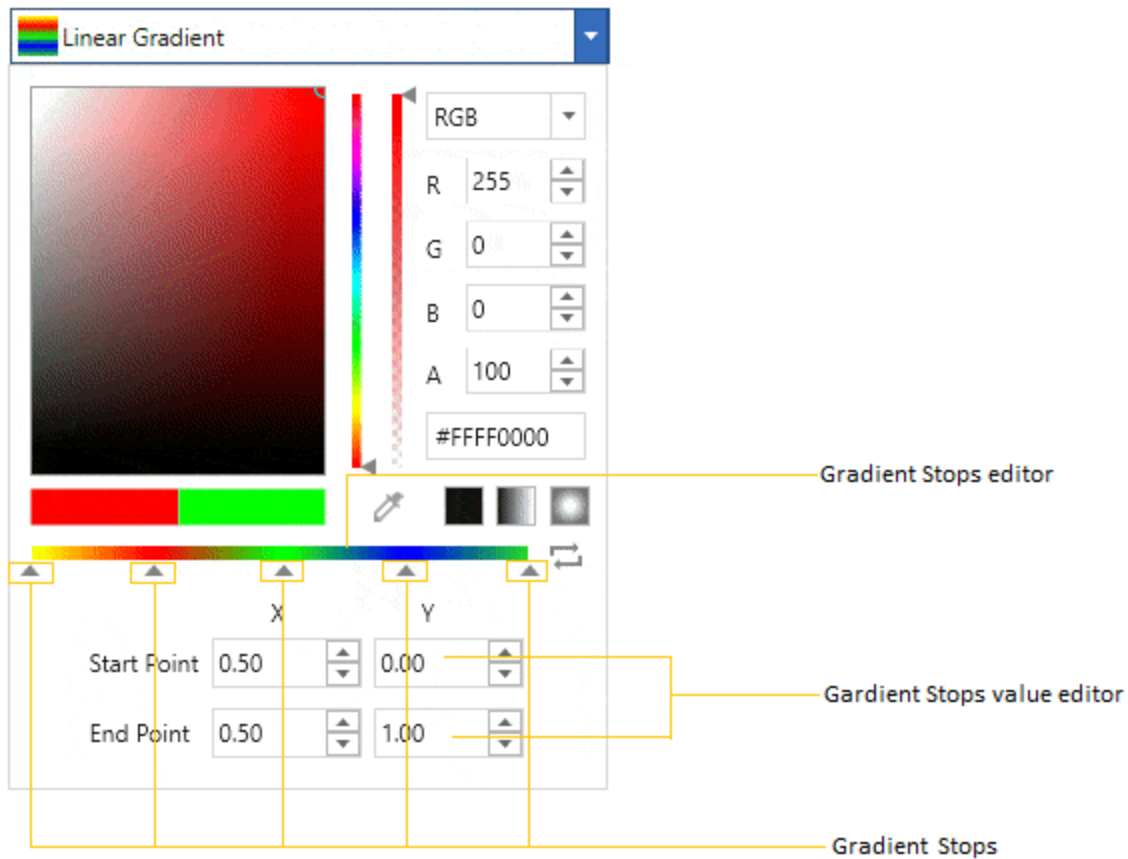
Click [here](#) to download the sample that showcases how to select a solid color from the **ColorPicker**.

### [Select gradient color in WPF color picker \(ColorPicker\)](#)

This section gives a brief note on how to create gradient color, modify their colors and modify their properties.

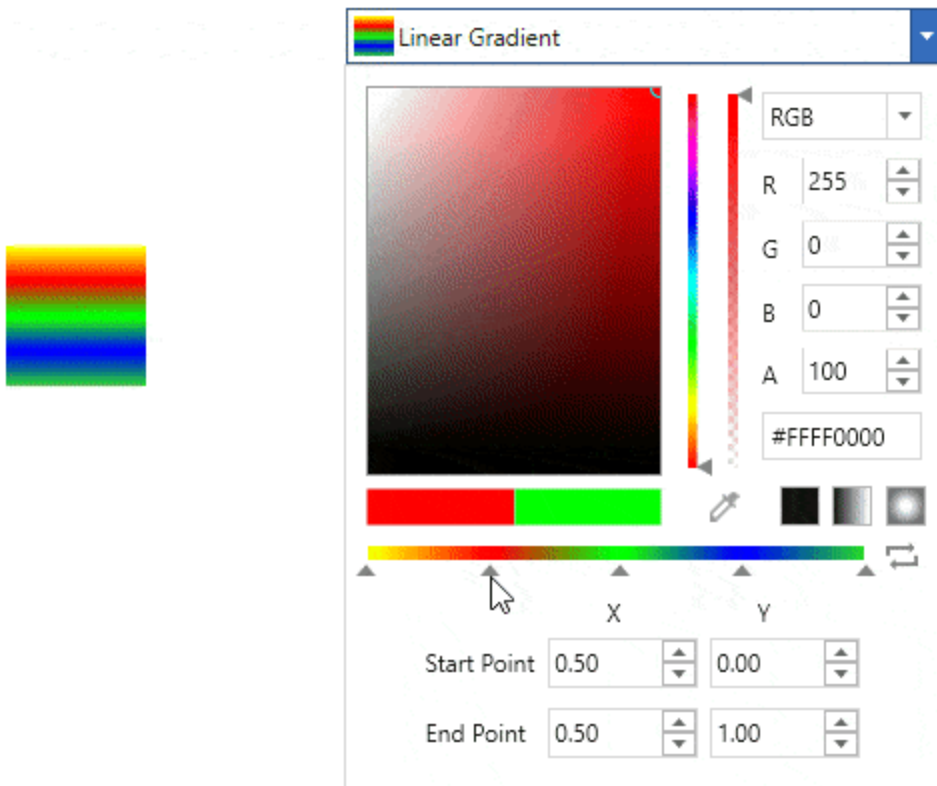
#### *What is a gradient color?*

A gradient color paints an area with multiple colors that blend into each other along an axis. [ColorPicker](#) now comes with Gradient tools which returns a brush of type Linear and Radial gradient colors. The offsets can be added or dropped dynamically and its position can be changed to produce different color combinations.



### Create Gradient colors using GradientStops Editor

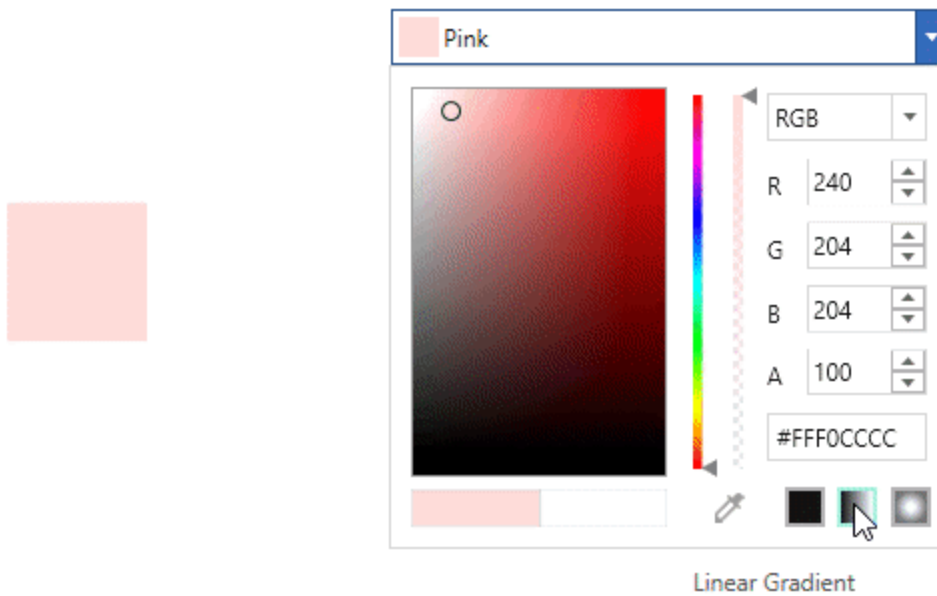
We can add a multiple color combination for the gradient color using the `GradientStopsEditor`. We can add new gradient stops, change the offset and change the color of the gradient stops at run-time. The created gradient stops are combined together provides a gradient color.



#### *Add or Remove GradientStops*

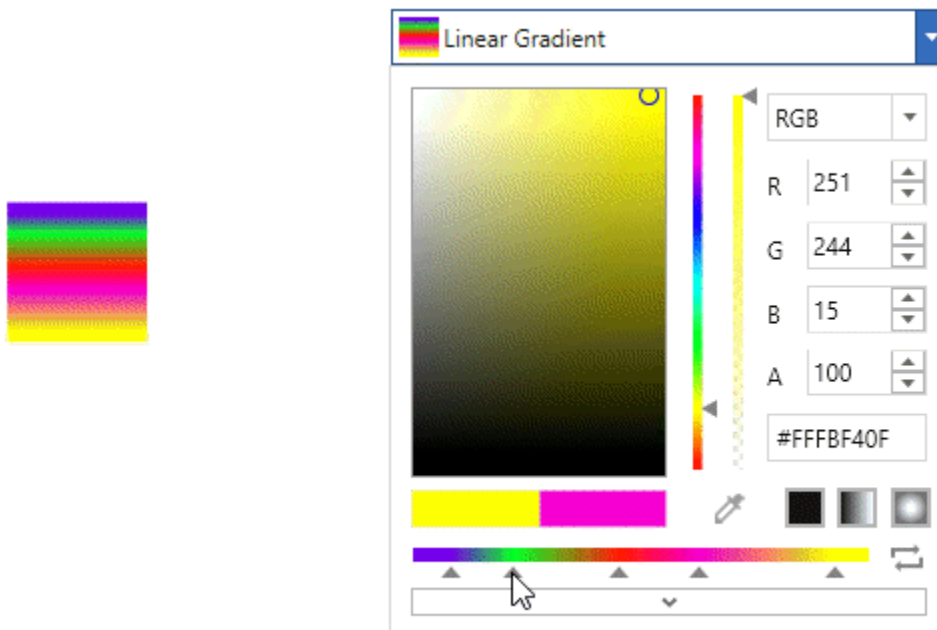
We can add a more colors for the gradient color by using the gradient stops. Gradient stops can be added to existing gradient by clicking on the `GradientStopEditor`.

To remove a gradient stops, select the gradient stop which want to be remove and press `Delete` key or mouse drag it away, so that it will removed from the `GradientStopsEditor`.



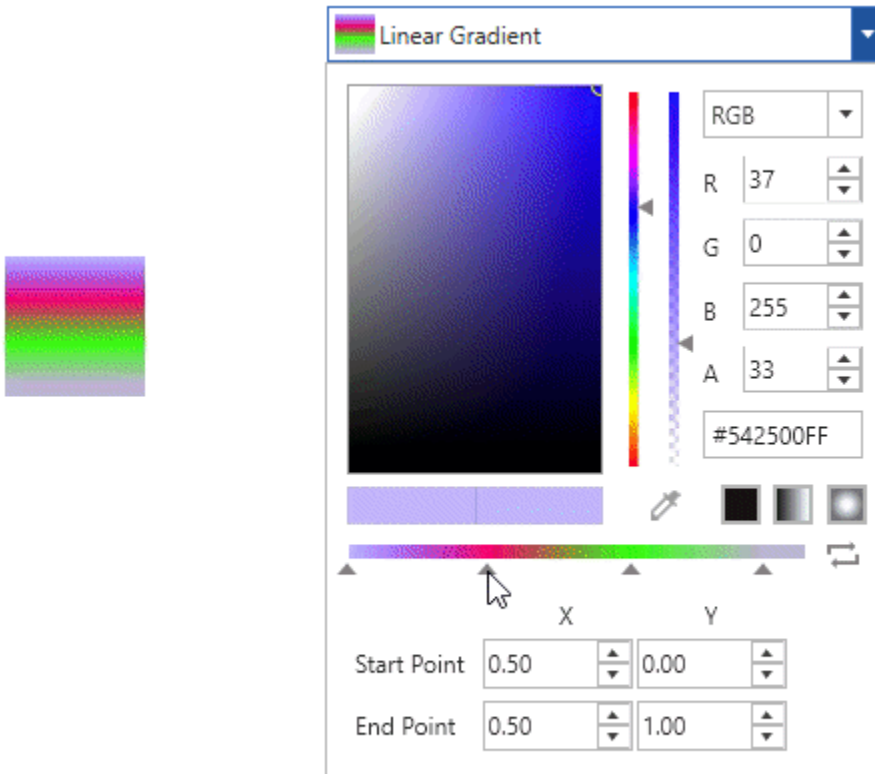
#### Rearrange GradientStops

We can re-arrange the color combination of the gradient color by adjusting the gradient stops. Gradient stops positions can be altered just by dragging it along the `GradientStopEditor`. The Gradient color will be generated on the basis of the order of the gradient stops arranged.



### Change GradientStops Colors

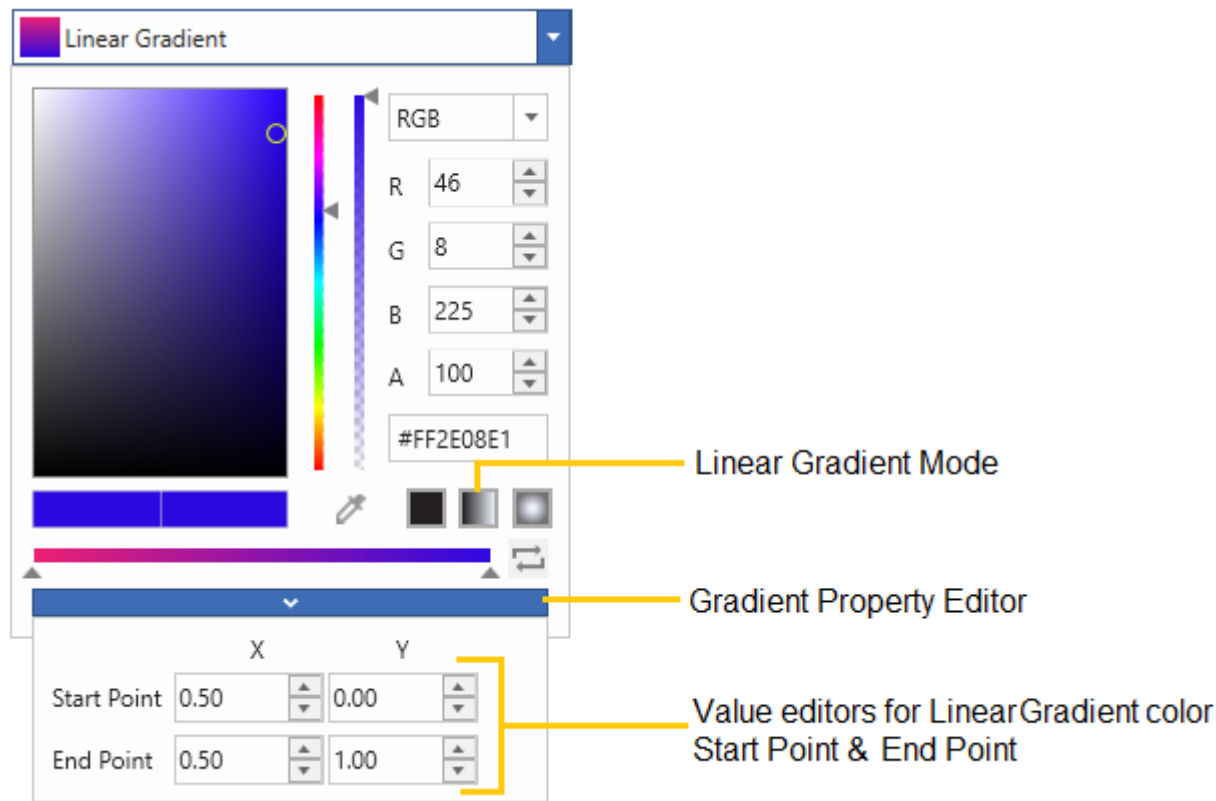
We can change the colors for created gradient color by changing the color of gradient stops. Color of a gradient stops changed by selecting that particular gradient stop and change the color from the color picker.



### Create Linear Gradient colors

We can create linear gradient color by programmatically using the `LinearGradientBrush` with its `LinearGradientBrush.GradientStops`, `StartPoint` and `EndPoint` properties or can create and change it at runtime by using `GradientStopsEditor` and `StartPoint`, `EndPoint` input options available in the `GradientPropertyEditor`. By default, the linear gradient colors are combined horizontally by start and end points. The default value of `StartPoint` is (0.5, 0) and `EndPoint` is (0.5, 1).





Properties	Description
<a href="#">Startpoint</a>	Indicates the Start point of LinearGradientBrush.
<a href="#">Endpoint</a>	Indicates the End point of LinearGradientBrush.

C#

```

class ViewModel {
public Brush LinearGradientBrush {get; set;}
public ViewModel() {
//Creating LinearGradient brush
LinearGradientBrush LinearBrush = new LinearGradientBrush();
LinearBrush.StartPoint = new Point(0, 1);
LinearBrush.EndPoint = new Point(1, 1);
LinearBrush.GradientStops.Add(new GradientStop(Colors.Yellow, 0.0));
LinearBrush.GradientStops.Add(new GradientStop(Colors.Red, 0.25));
LinearBrush.GradientStops.Add(new GradientStop(Colors.Blue, 0.75));
LinearBrush.GradientStops.Add(new GradientStop(Colors.LimeGreen, 1.0));
// Assigning a created linear gradient brush to `LinearGradientBrush`
property
LinearGradientBrush = LinearBrush;
}
}

```

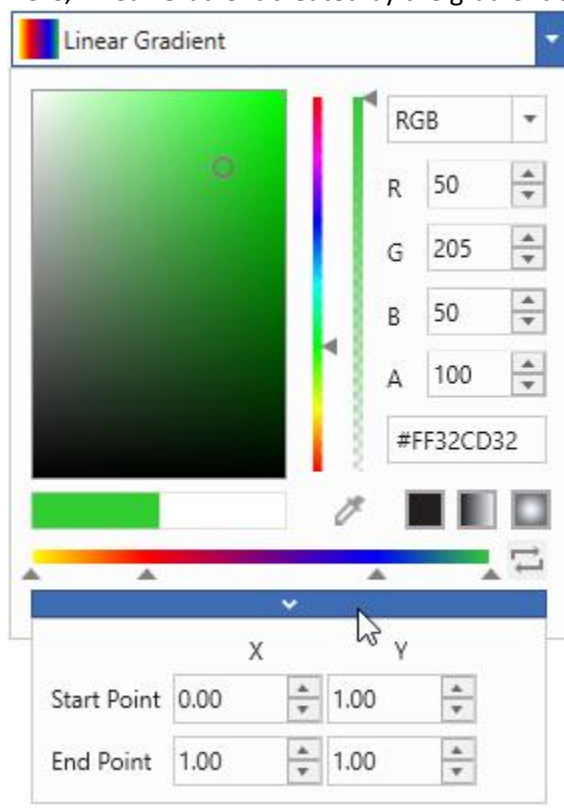
XML

```
<syncfusion:ColorPicker Brush="{Binding LinearGradientBrush}"  
Name="colorPicker">  
  <syncfusion:ColorPicker.DataContext>  
  <local:ViewModel></local:ViewModel>  
  </syncfusion:ColorPicker.DataContext>  
</syncfusion:ColorPicker>
```

## C#

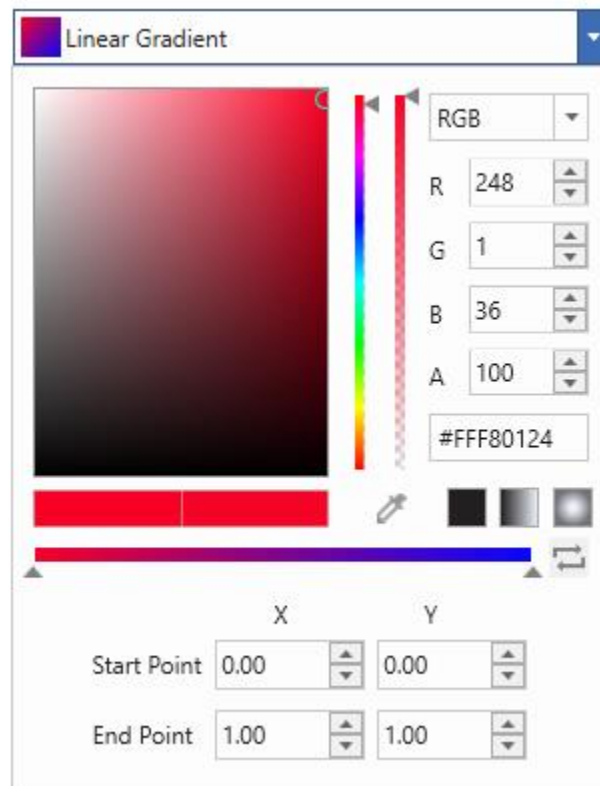
```
ColorPicker colorPicker = new ColorPicker();  
colorPicker.DataContext = new ViewModel();  
colorPicker.SetBinding(ColorPicker.BrushProperty, new  
Binding("LinearGradientBrush"));
```

Here, Linear Gradient created by the gradient colors and their location along the gradient axis using the



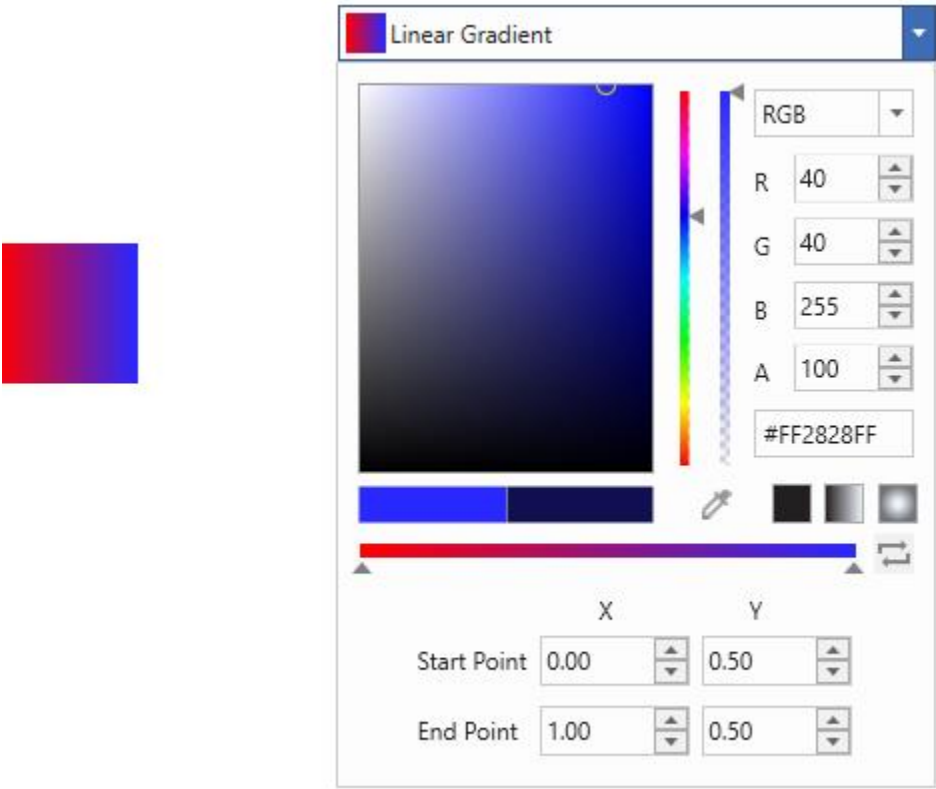
GradientStop objects.

![[ColorPicker with Horizontal Linear Gradient Editor](ColorPicker-with-Gradient-Support



*images/DefaultLinearGradient.png)*

Diagonal Linear Gradient (StartPoint(0,0), EndPoint(1,1))  
![ColorPicker Linear Gradient Editor](ColorPicker-with-Gradient-Support)



images/LinearGradientExample\_1.png)

Vertical Linear Gradient (StartPoint(0, 0.5), EndPoint(1, 0.5))  
![ColorPicker Linear Gradient Editor](ColorPicker-with-Gradient-Support)  
images/LinearGradientExample\_2.png)

Create Radial Gradient colors

We can create radial gradient colors by programmatically using the `RadialGradientBrush` with its `RadialGradientBrush.GradientStops`, `GradientOrigin`, `Radius` and `Centre` properties. Radial gradient brush colors can be changed at runtime using the `GradientStopsEditor` and its `GradientOrigin`, `Centre` and `Radius` can be changed at runtime using the input options available in the `GradientPropertyEditor`.

Properties	Description
<a href="#">GradientOrigin</a>	Indicates the gradient origin of RadialGradientBrush.
<a href="#">CentrePoint</a>	Indicates the centre point of RadialGradientBrush.
<a href="#">RadiusX</a>	Indicates the X value in Radius of RadialGradientBrush.
<a href="#">RadiusY</a>	Indicates the Y value in Radius of RadialGradientBrush.

C#

```
class ViewModel {
public Brush RadialGradientBrush { get; set; }
public ViewModel() {
    //Creating Radial Gradient brush
    RadialGradientBrush radialBrush = new RadialGradientBrush();
    radialBrush.GradientOrigin = new Point(0.5, 0.5);
    radialBrush.Center = new Point(0.5, 0.5);
    radialBrush.RadiusX = 0.5;
    radialBrush.RadiusY = 0.5;
    radialBrush.GradientStops.Add(new GradientStop(Colors.Yellow, 0.0));
    radialBrush.GradientStops.Add(new GradientStop(Colors.Red, 0.25));
    radialBrush.GradientStops.Add(new GradientStop(Colors.Blue, 0.75));
    radialBrush.GradientStops.Add(new GradientStop(Colors.LimeGreen, 1.0));
    // Assigning a created radial gradient brush to `RadialGradientBrush` property
    RadialGradientBrush = radialBrush;
}
}
```

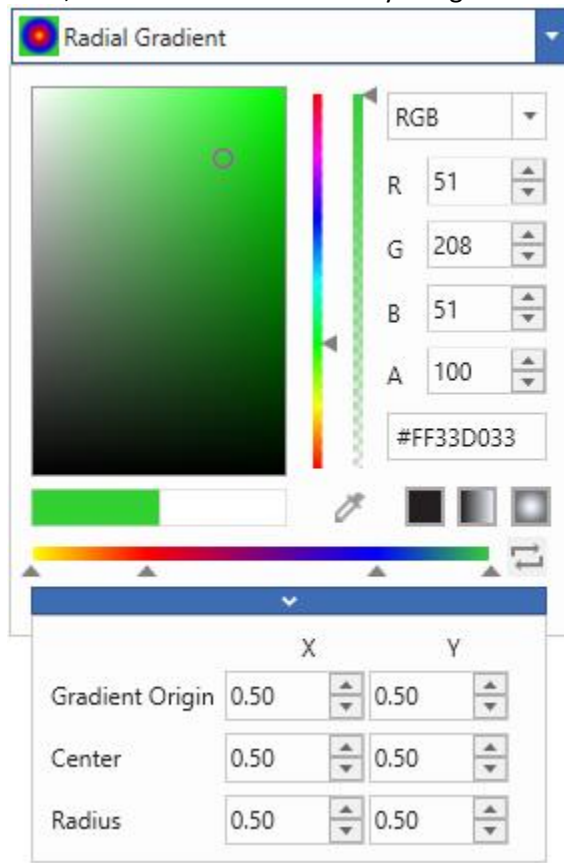
### XML

```
<syncfusion:ColorPicker Brush="{Binding RadialGradientBrush}"
Name="colorPicker">
<syncfusion:ColorPicker.DataContext>
<local:ViewModel></local:ViewModel>
</syncfusion:ColorPicker.DataContext>
</syncfusion:ColorPicker>
```

### C#

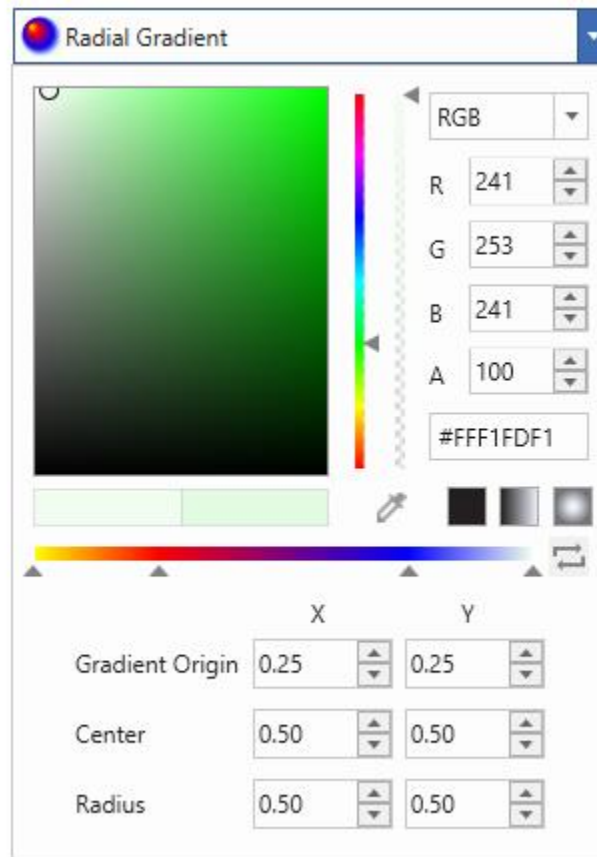
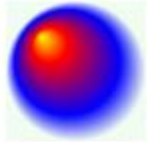
```
ColorPicker colorPicker = new ColorPicker();
colorPicker.DataContext = new ViewModel();
colorPicker.SetBinding(ColorPicker.BrushProperty, new
Binding("RadialGradientBrush"));
```

Here, Radial Gradient created by the gradient colors and their location along the gradient axis using the



GradientStop objects.

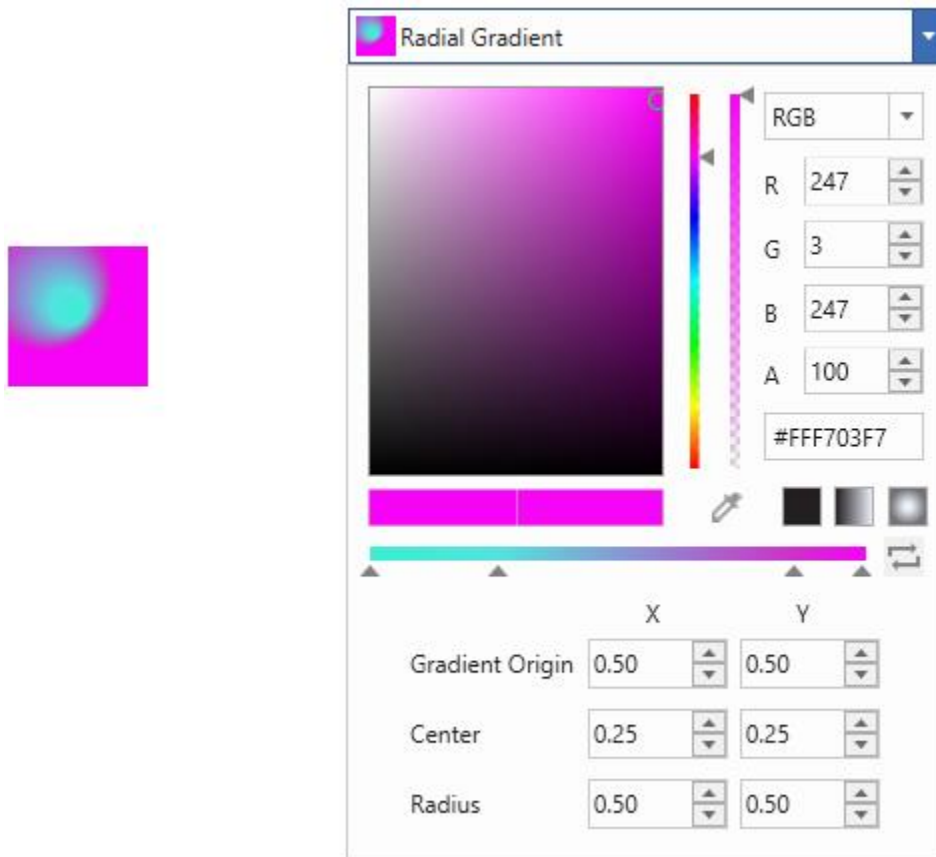
!ColorPicker with Radial Gradient Editor)(ColorPicker-with-Gradient-Support



*images/DefaultRadialGradient.png*)

Gradient Origin (0.25, 0.25)

![ColorPicker Radial Gradient Editor with Gradient Origin point](ColorPicker-with-Gradient-Support

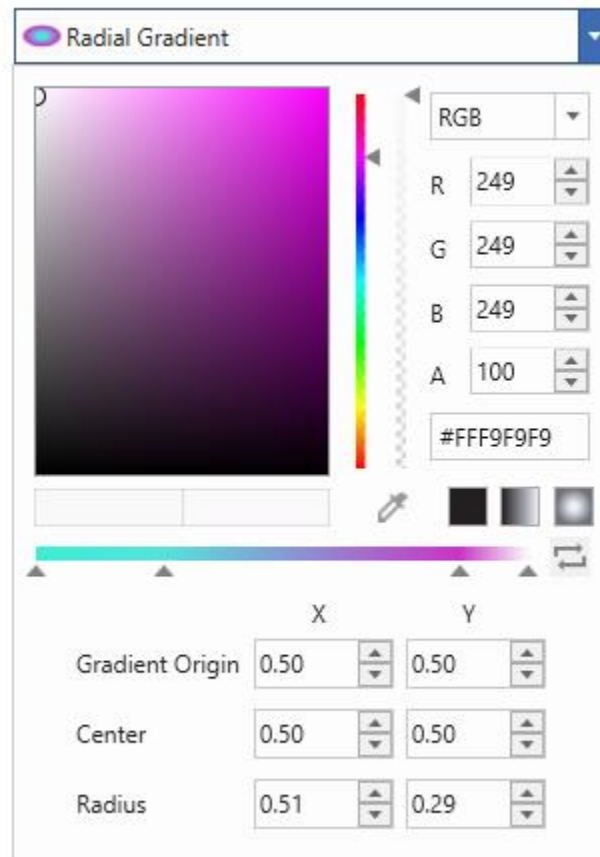


images/RadialGradientOrigin.png)



[Centre \(0.25, 0.25\)](#)

![[ColorPicker Radial Gradient Editor with Centre point]](ColorPicker-with-Gradient-Support



*images/RadialGradientCentre.png)*

[Radius \(0.25, 0.25\)](#)

![[ColorPicker Radial Gradient Editor with Radius]](ColorPicker-with-Gradient-Support*images/RadialGradientRadius.png)*

[Reverse the Gradient Colors](#)

[ColorPicker](#) comes with the reverse button which helps in changing the gradient colors upside down or in case of radial gradient inside out.



### Show selected gradient color name

By default, the selected Gradient mode name is displayed in **ColorPicker**. If we want to display the selected gradient color name instead of the Gradient mode name, use the [GradientBrushDisplayMode](#) value as **Extended**. The default value of **GradientBrushDisplayMode** property is **Default**.

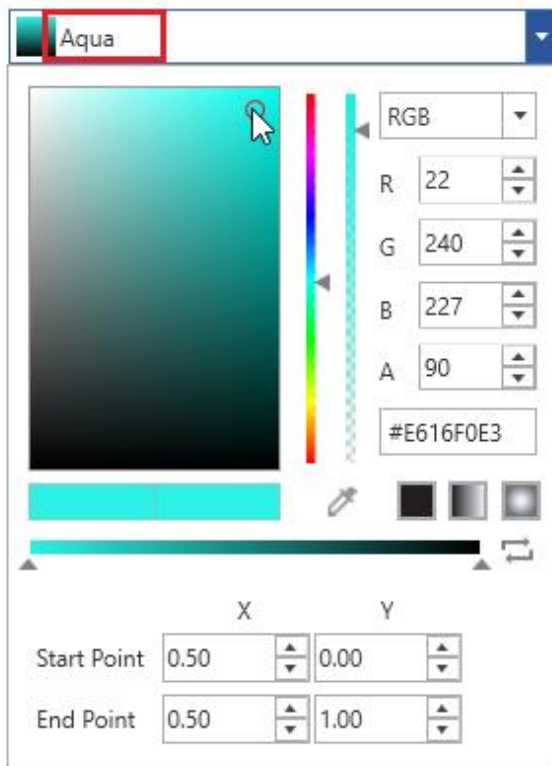
### XML

```
<Syncfusion:ColorPicker x:Name="colorPicker"
    GradientBrushDisplayMode="Extended"/>
```

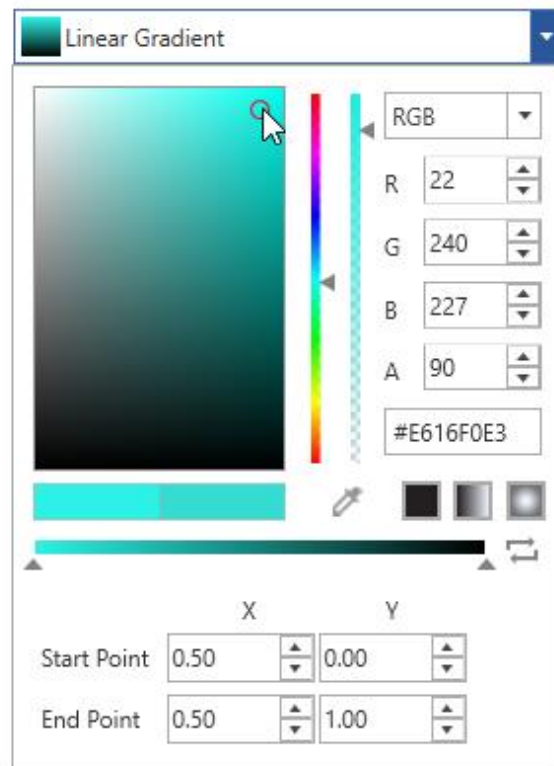
### C#

```
ColorPicker colorPicker = new ColorPicker();
colorPicker.GradientBrushDisplayMode = GradientBrushDisplayMode.Extended;
```

GradientBrushDisplayMode="Extended"



GradientBrushDisplayMode="Default"



### Show gradient color value editor

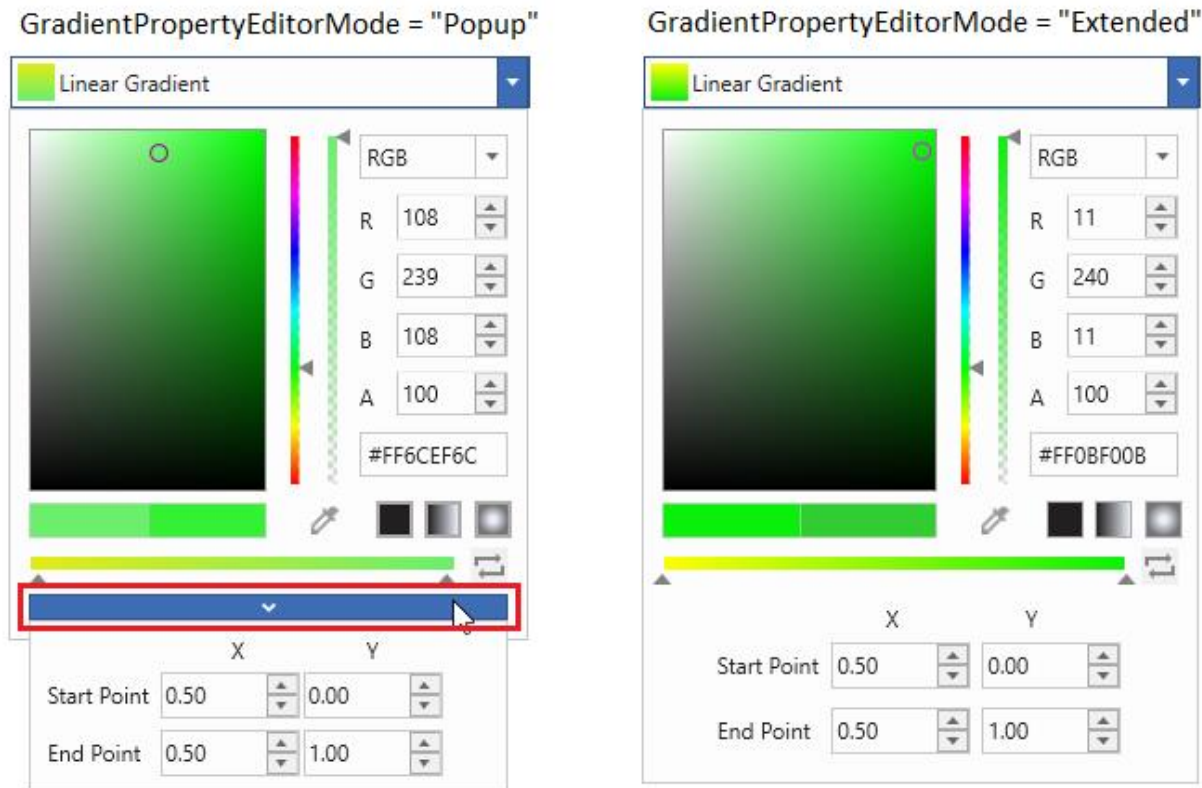
We can display the gradient property editor either in popup mode or in extended mode. By default, the gradient property editor is displayed in the extended mode. If we want display the gradient property editor only in a popup, use the [GradientPropertyEditorMode](#) property value as **PopUp**.

### XML

```
<Syncfusion:ColorPicker x:Name="colorPicker"
    GradientPropertyEditorMode="PopUp"/>
```

### C#

```
ColorPicker colorPicker = new ColorPicker();
colorPicker.GradientPropertyEditorMode = GradientPropertyEditorMode.Popup;
```



### Switch between Solid, Gradient mode

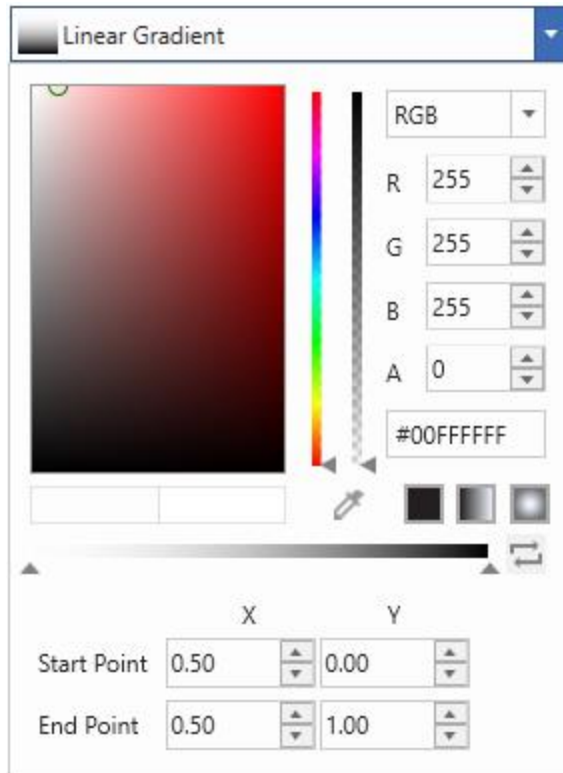
We can change the brush mode from solid to gradient or vice versa at runtime as well as programmatically. By default, the Solid brush mode is enabled. If we want Gradient brush mode, use the [BrushMode](#) property value as Gradient.

### XML

```
<Syncfusion: ColorPicker x:Name="colorPicker" BrushMode="Gradient"/>
```

### C#

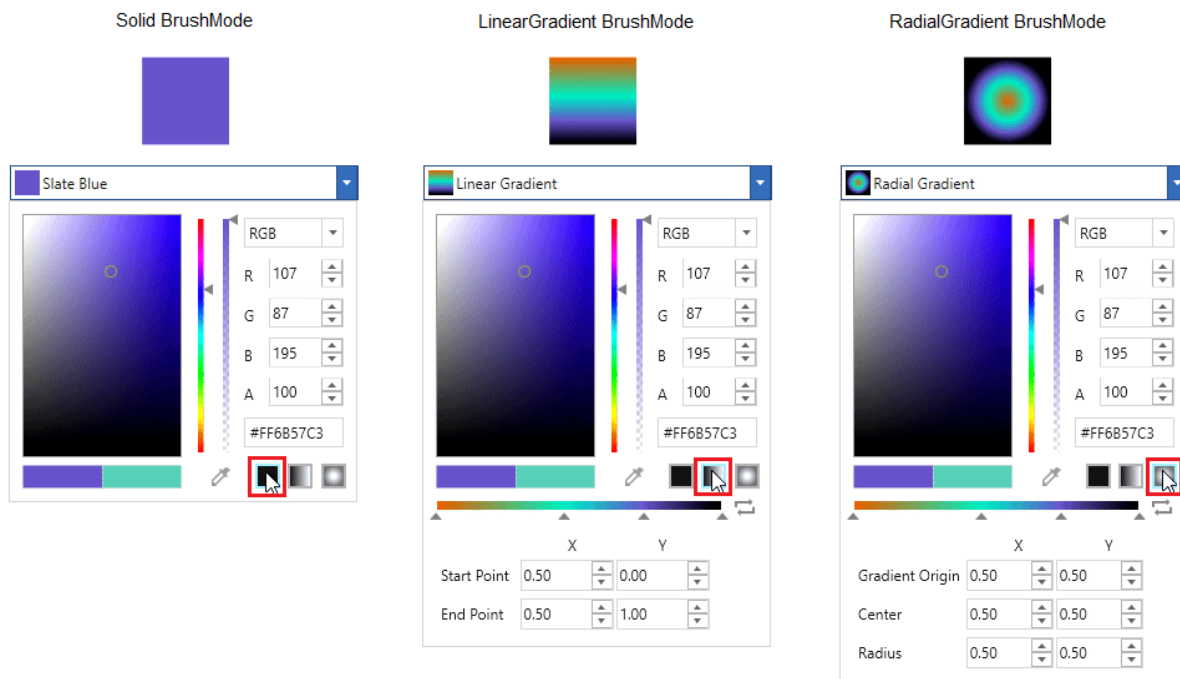
```
ColorPicker colorPicker = new ColorPicker();
colorPicker.BrushMode = BrushModes.Gradient;
```



Here, The ColorPicker is in Gradient brush mode.

*Switch between Solid, Linear and Gradient brush mode*

We can change the brush mode directly by clicking on the corresponding Solid, Linear or Gradient mode buttons which are placed in the bottom right corner of the ColorPicker.



Disable Switching between Solid, Linear and Gradient brush mode at runtime

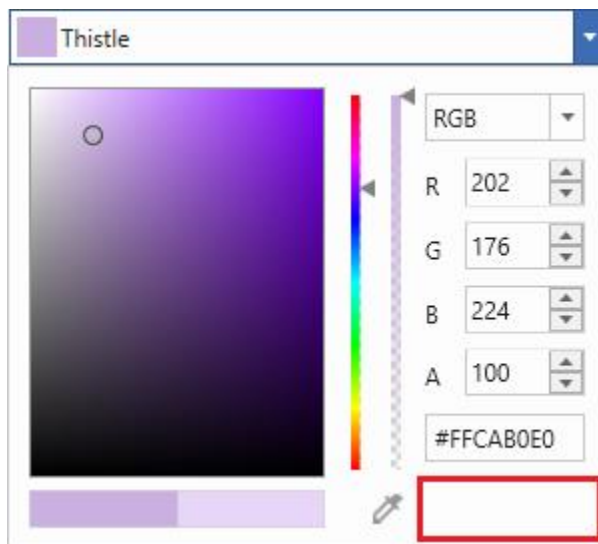
If we want to disable the Solid, Linear and Gradient brush mode transition at runtime, use the [EnableSolidToGradientSwitch](#) property value as false. It will hide the Solid, Linear and Gradient brush buttons. The Default value of [EnableSolidToGradientSwitch](#) property is true.

### XML

```
<syncfusion:ColorPicker x:Name="colorPicker"
    EnableSolidToGradientSwitch="false"/>
```

### C#

```
ColorPicker colorPicker = new ColorPicker ();
colorPicker.EnableSolidToGradientSwitch = false;
```



Gradient color changed notification

Selected gradient color changed in [ColorPicker](#) can be examined using [SelectedBrushChanged](#) event.

### XML

```
<syncfusion:ColorPicker
    SelectedBrushChanged="ColorPicker_SelectedBrushChanged"
    Name="colorPicker"/>
```

### C#

```
ColorPicker colorPicker = new ColorPicker();
colorPicker.SelectedBrushChanged += ColorPicker_SelectedBrushChanged;
```

### C#

```
//Invoked when the selected brush is changed
private void ColorPicker_SelectedBrushChanged(DependencyObject d,
    DependencyPropertyChangedEventArgs e)
{
    // Enter your code here
}
```

```
}

```

Click [here](#) to download the sample that showcases the Linear GradientBrush and its additional features.

Click [here](#) to download the sample that showcases the Radial GradientBrush and its additional features.

### Appearance in WPF color picker (ColorPicker)

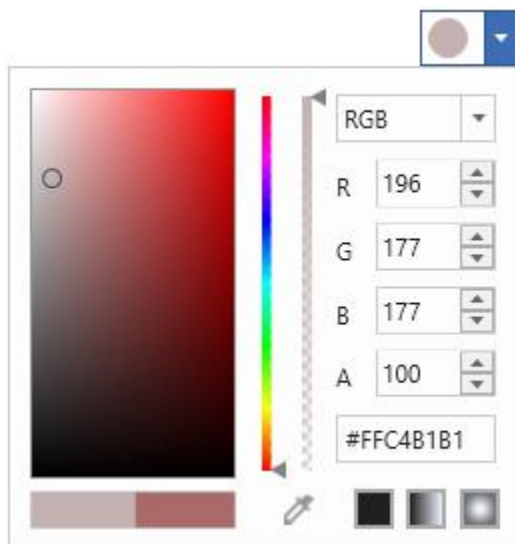
This section explains different UI customization, styling, theming options available in [ColorPicker](#) control.

#### Change Header Template

We can customize the header of the `ColorPicker` by using the [HeaderTemplate](#) property.

#### XML

```
<DataTemplate x:Key="CustomHeaderTemplate"
  DataType="syncfusion:ColorPicker">
  <StackPanel Orientation="Horizontal">
  <Ellipse Fill="{Binding Brush,
    RelativeSource={RelativeSource FindAncestor,
    AncestorType={x:Type syncfusion:ColorPicker}}}"
    Name="selectedColorEllipse"
    HorizontalAlignment="Left"
    Width="20" Height="20"
    Margin="2" />
  </StackPanel>
  </DataTemplate>
  <syncfusion:ColorPicker HeaderTemplate="{StaticResource
    CustomHeaderTemplate}"
    Name="colorPicker"
    Width="50" Height="30"/>
```



**Note:** [View Sample in GitHub](#)

### Change flow direction

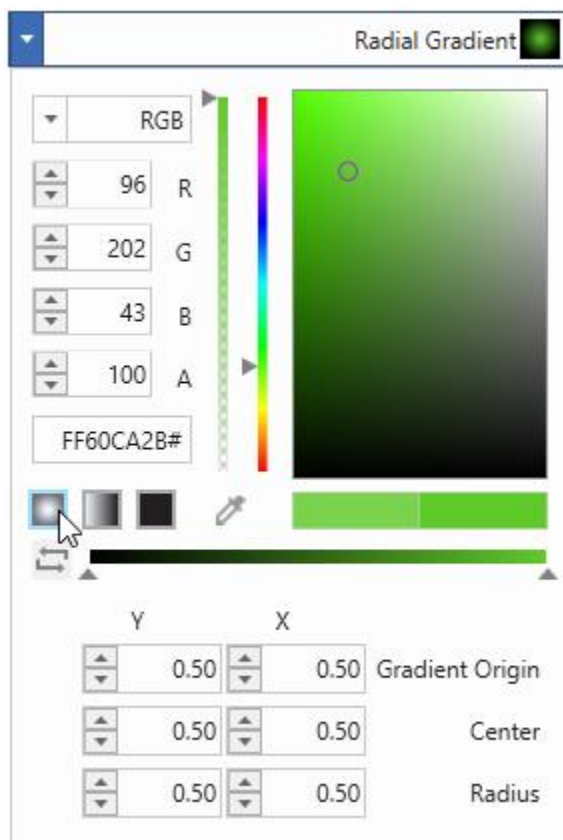
We can change the flow direction of the `ColorPicker` layout from right to left by setting the `FlowDirection` property value as `RightToLeft`. The Default value of `FlowDirection` property is `LeftToRight`.

#### XML

```
<syncfusion:ColorPicker FlowDirection="RightToLeft" Name="colorPicker"/>
```

#### C#

```
ColorPicker colorPicker= new ColorPicker();  
colorPicker.FlowDirection = FlowDirection.RightToLeft;
```



**Note:** [View Sample in GitHub](#)

### Setting ToolTip

ToolTip is used to show the information about the segment, when you mouse over on the segment. We can show information about the selected color name using tooltip when click and dragging the mouse on the color palette. Tooltip is enabled by default, you can disable it by setting `EnableToolTip` to `false`.

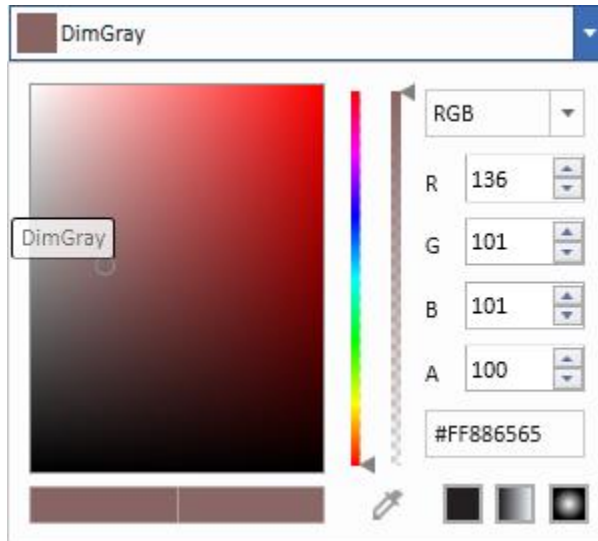
#### XML

```
<syncfusion:ColorPicker EnableToolTip="True" Name="colorPicker"/>
```

#### C#



```
ColorPicker colorPicker = new ColorPicker();  
colorPicker.EnableToolTip = true;
```

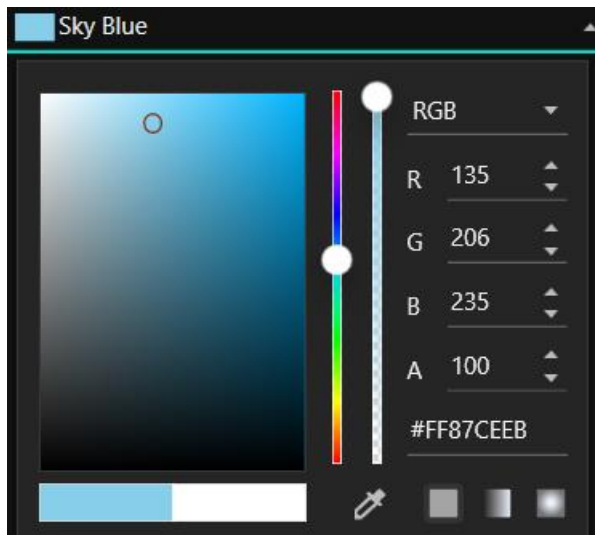


**Note:** [View Sample in GitHub](#)

### Theme

ColorPicker supports various built-in themes. Refer to the below links to apply themes for the ColorPicker,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



## ColorPickerPalette

### WPF Color Picker Palette Overview

The [ColorPickerPalette](#) control provides a rich visual interface for color selection. The structure of the control represents a palette which is displayed as a Drop-down with selected color highlighted at the

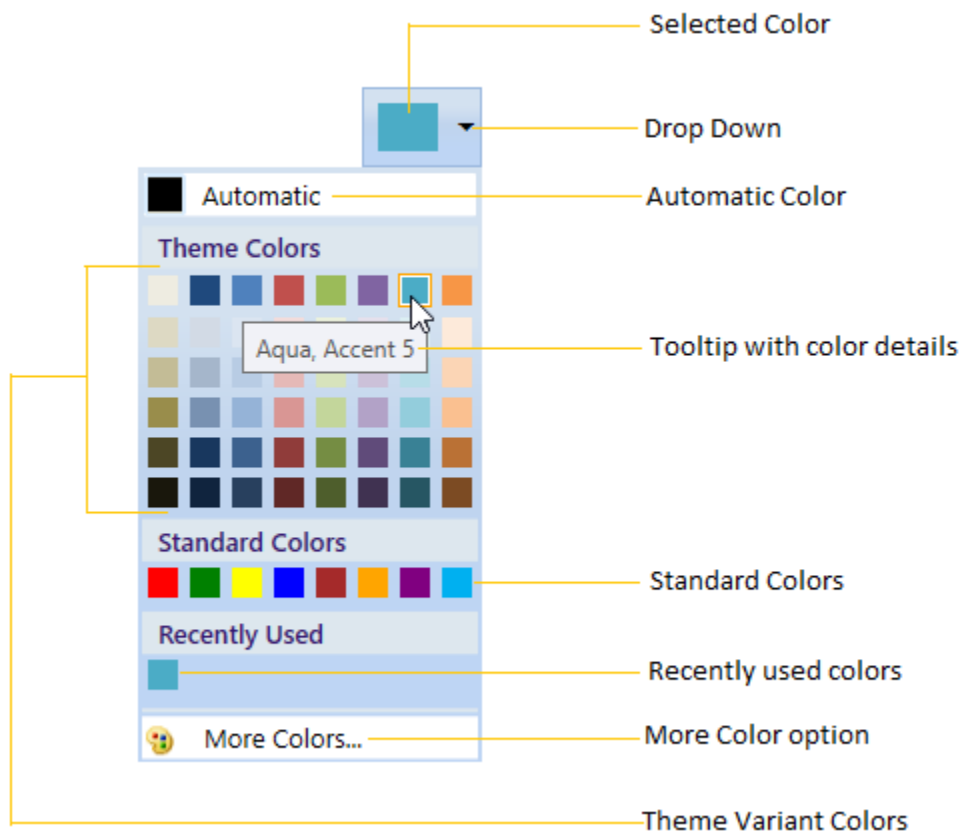
top. It provides standard colors and the various theme colors to choose. The control also has a ToolTip support which bears the name of the color. More color options are embedded with the control that provides you with a wide range of color options.



### Getting Started with WPF Color Picker Palette

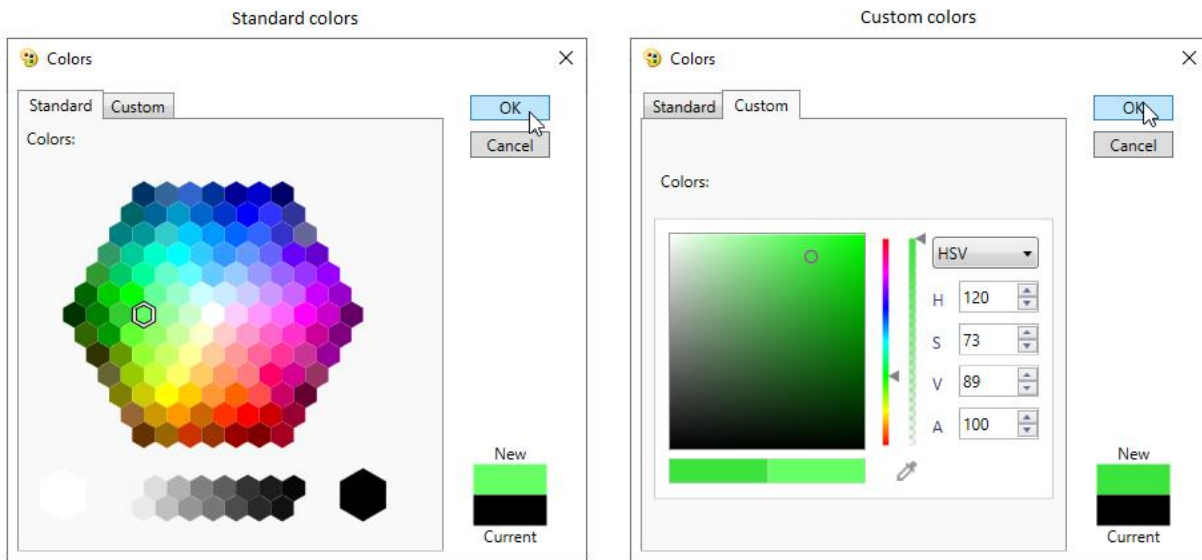
This section explains how to create a [WPF ColorPickerPalette](#) and explains about its structure and features.

## Control Structure



- The Selected Color represents the color that you select.
- The Drop-Down represents a button, ColorPickerPalette window will open when you click on it.
- The Automatic Color represents the Color, which can be set by you as default color.
- The ToolTip with Color Details represents the ToolTip, when the mouse hovers on the Color.
- The Standard Colors stores the standard colors like Red, Green, Blue and so on.
- The Recently User Colors stores the Colors that are recently selected.
- The More Colors Option provides wide range of color in addition to colors in the palette.
- The Theme Variant Colors represents the Theme colors with variant.

### More Color Window



### Assembly deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as reference to use the control in any application.

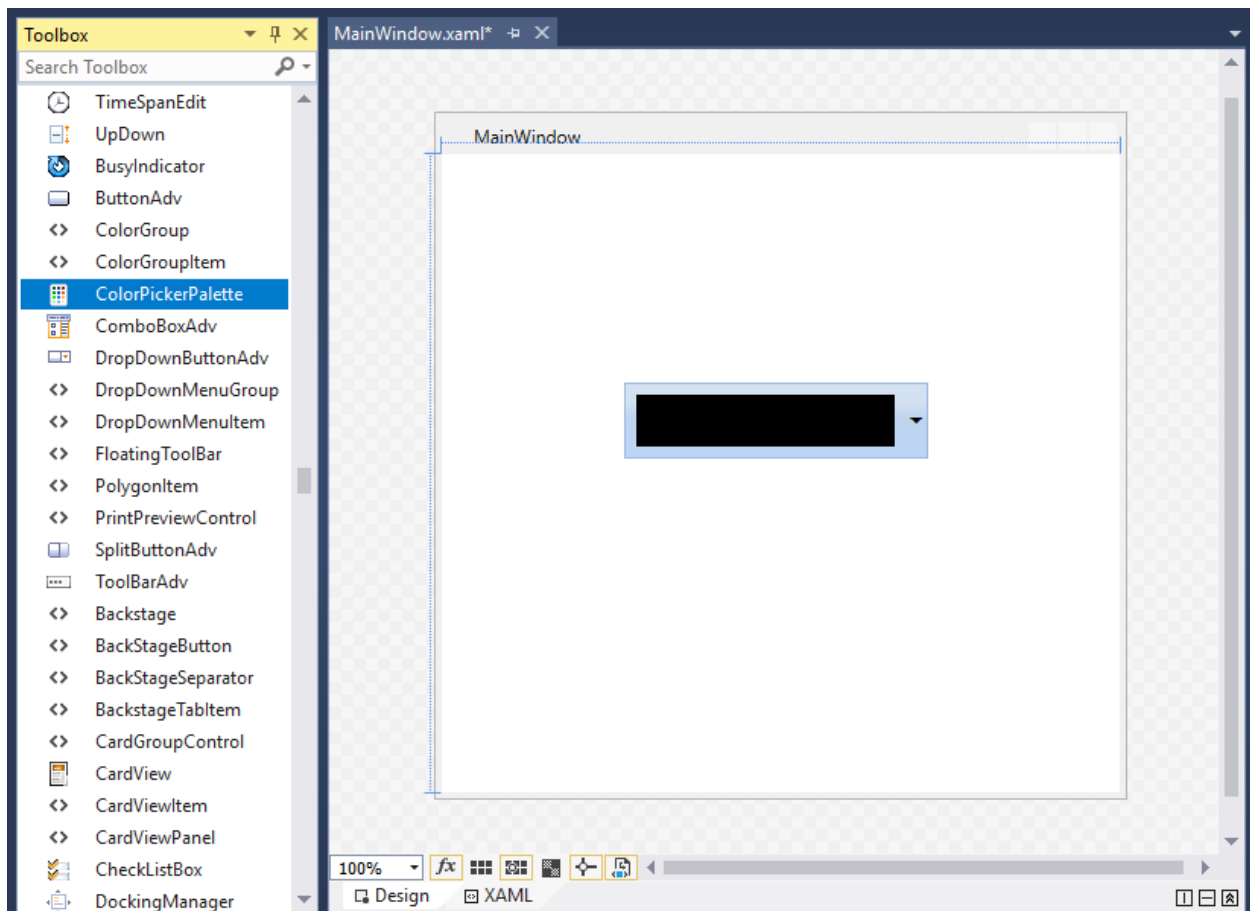
You can find more details about installing the NuGet package in a WPF application in the following link:

[How to install nuget packages](#)

### Adding WPF ColorPickerPalette via designer

You can add the WPF Color Picker Palette control to an application by dragging it from the toolbox to a view of the designer. The following dependent assembly will be added automatically.

- Syncfusion.Shared.WPF



### Adding WPF ColorPickerPalette via XAML

To add the `ColorPickerPalette` control manually in XAML, follow these steps:

1. Create a new WPF project in Visual Studio.
2. Add the following assembly references to the project,
  - o Syncfusion.Shared.WPF
3. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> and declare the `ColorPickerPalette` control in XAML page.

### XAML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="ColorPickerPaletteSample.MainWindow"
Title="ColorPickerPalette Sample" Height="350" Width="525">
<Grid>
<!--Adding ColorPickerPalette control -->
<syncfusion:ColorPickerPalette x:Name="colorPickerPalette"
Width="60"
Height="40" />
</Grid>
</Window>
```

### Adding WPF ColorPickerPalette via C\#

To add the `ColorPickerPalette` control manually in C#, follow these steps:

1. Create a new WPF application via Visual Studio.
2. Add the following assembly references to the project,
  - o Syncfusion.Shared.WPF
3. Include the required namespace and create an instance of `ColorPickerPalette` and add it to the window.
4. Declare the `ColorPickerPalette` control using C#.

### C#

```
using Syncfusion.Windows.Tools.Controls;  
public partial class MainWindow : Window {  
    public MainWindow() {  
        InitializeComponent();  
        //Creating an instance of ColorPickerPalette control  
        ColorPickerPalette colorPickerPalette = new ColorPickerPalette();  
        colorPickerPalette.Width = 60;  
        colorPickerPalette.Height = 40;  
        //Adding ColorPickerPalette as window content  
        this.Content = colorPickerPalette;  
    }  
}
```



### Accessing a Color programmatically

We can set or change the selected color of the `ColorPickerPalette` programmatically by setting the value for `Color` property. If we want to know the selected color name, use the `ColorName` property that holds the name of the selected color item. The default value of `Color` and `ColorName` property is `Black` and `Color`.

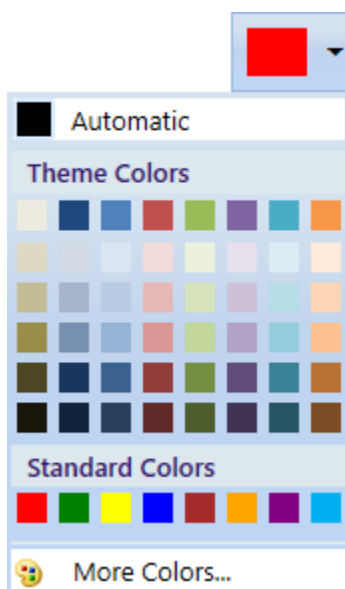
**Note:** You can also set or change the selected color brush by using the `SelectedBrush` property.

### XML

```
<syncfusion:ColorPickerPalette Color="Red"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```

### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.Color = Colors.Red;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```



Here, Red color is selected color in the ColorPickerPalette.

Select color from color palette

We can select a different colors from Theme Color and Standard Color panels. we can show or hide the variant colors of the base Theme Colors and Standard Colors by using the [GenerateThemeVariants](#) and [GenerateStandardVariants](#) properties value as true or false.

### XML

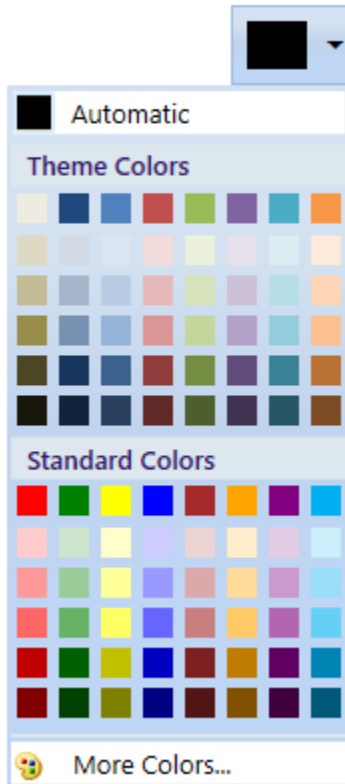
```
<syncfusion:ColorPickerPalette GenerateThemeVariants="True"
GenerateStandardVariants="True"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```

### C#

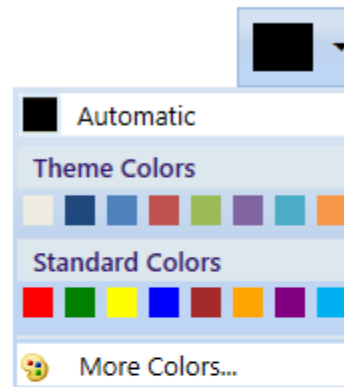
```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.GenerateThemeVariants = true;
```

```
colorPickerPalette.GenerateStandardVariants = true;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```

GenerateThemeVariants = "True"  
GenerateStandardVariants = "True"



GenerateThemeVariants = "False"  
GenerateStandardVariants = "False"



Add your own color in the palette

If we want to allow the user to select a color from own colors, add that color with its name using [CustomColor.ColorName](#) and [CustomColor.Color](#) into the [CustomColorsCollection](#) and set the [SetCustomColors](#) property value as true. The provided [CustomColor.ColorName](#) is shown in the tooltip while mouse hovering on the color item. We can change the custom color panel header text and its visibility by using the [CustomHeaderText](#) and [CustomHeaderVisibility](#) properties. The default value of [CustomHeaderText](#) is [CustomColors](#) and default value of [CustomHeaderVisibility](#) is [Visible](#).

### C#

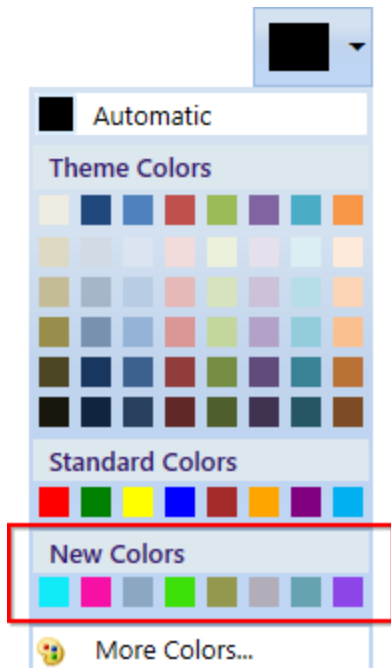
```
public class ViewModel : NotificationObject {
    private ObservableCollection<CustomColor> newColorCollection;
    public ObservableCollection<CustomColor> NewColorCollection {
        get {
            return newColorCollection;
        }
        set {
            newColorCollection = value;
            this.RaisePropertyChanged("NewColorCollection");
        }
    }
}
```



```
}  
public ViewModel() {  
    NewColorCollection = new ObservableCollection<CustomColor>();  
}  
}
```

## XML

```
<Window.Resources>  
    <local:ViewModel x:Key="viewModel">  
        <local:ViewModel.NewColorCollection>  
            <!-- Defining the color details -->  
            <syncfusion:CustomColor Color="#FF11EBF8" ColorName="Aqua" />  
            <syncfusion:CustomColor Color="#FFF80FA6" ColorName="Deep Pink" />  
            <syncfusion:CustomColor Color="#FF8BA7C2" ColorName="Dark Gray" />  
            <syncfusion:CustomColor Color="#F53CDF07" ColorName="Lime Green" />  
            <syncfusion:CustomColor Color="#C2929545" ColorName="Olive Drab" />  
            <syncfusion:CustomColor Color="#2E956145" ColorName="Sienna" />  
            <syncfusion:CustomColor Color="#78458E95" ColorName="Steel Blue" />  
            <syncfusion:CustomColor Color="#8B8220E4" ColorName="Blue Violet" />  
        </local:ViewModel.NewColorCollection>  
    </local:ViewModel>  
</Window.Resources>  
    <syncfusion:ColorPickerPalette CustomColorsCollection="{Binding  
NewColorCollection}"  
CustomHeaderText="New Colors"  
CustomHeaderVisibility="Visible"  
SetCustomColors="True"  
DataContext="{StaticResource viewModel}"  
Name="colorPickerPalette"  
Width="60"  
Height="40">  
</syncfusion:ColorPickerPalette>
```



Click [here](#) to download the sample that showcases how to add your own color items into the palette.

#### Recently used color items

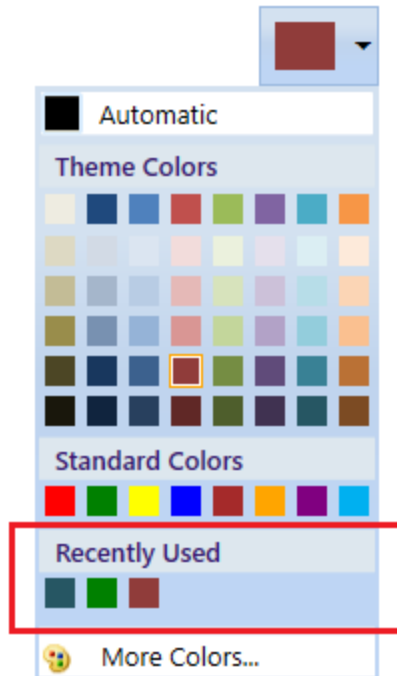
The recently selected color items are displayed in the [RecentlyUsedPanel](#). If we want to choose a color which are previously selected, use the RecentlyUsedPanel.

#### XML

```
<syncfusion:ColorPickerPalette RecentlyUsedPanelVisibility="Visible"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```

#### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.RecentlyUsedPanelVisibility = Visibility.Visible;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```



Choosing a color from MoreColor window

If we want more standard and custom colors to select, click the more color option and select the color from standard color tab or custom color tab and click the **Ok** button.

#### XML

```
<syncfusion:ColorPickerPalette MoreColorOptionVisibility="Visible"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```

#### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.MoreColorOptionVisibility = Visibility.Visible;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```



---

**Note:** We can show or hide all color panels. Refer the [Dealing with ColorPickerPalette](#) page that explains the panel visibility support.

---

#### Reset selected color

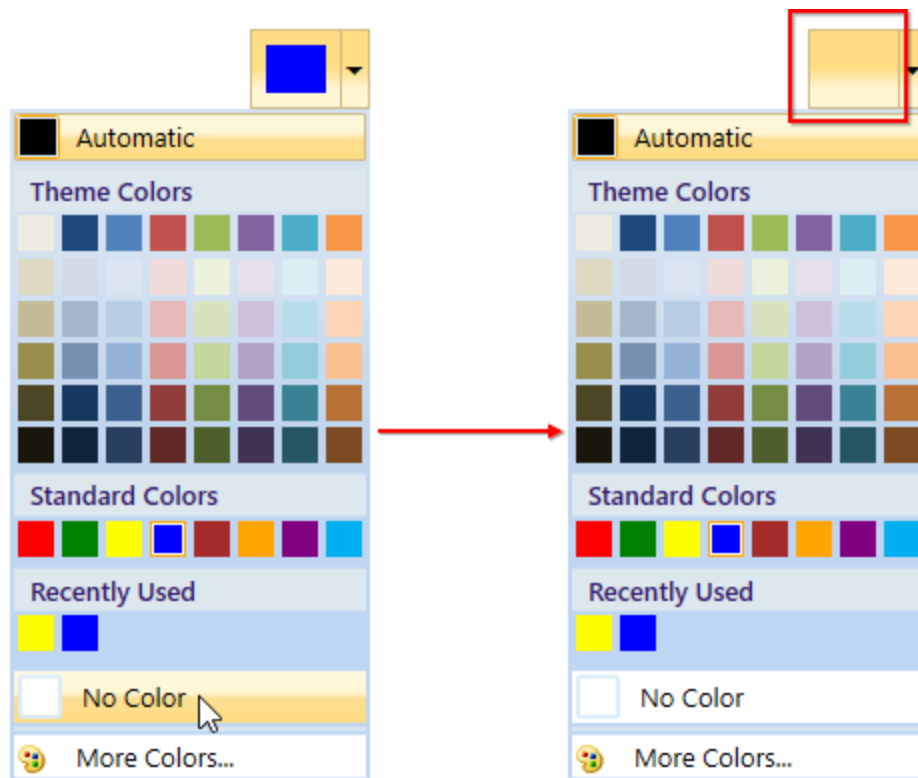
If you want to reset the selected color as Transparent color, click the No Color button. You will be display the No color button only by setting the [NoColorVisibility](#) property value as Visible. The default value of NoColorVisibility property is Collapsed.

#### XML

```
<syncfusion:ColorPickerPalette NoColorVisibility="Visible"
Name="colorPickerPalette"/>
```

#### C#

```
colorPickerPalette.NoColorVisibility = Visibility.Visible;
```



**Note:** [View Sample in GitHub](#)

Selected brush or color changed notification

The selected brush or color changed in `ColorPickerPalette` can be examined using `SelectedBrushChanged` event. The `SelectedBrushChangedEventArgs` contains the old and newly selected brush and its color values in the `OldBrush`, `NewBrush` and `OldColor`, `NewColor` properties.

### XML

```
<syncfusion:ColorPickerPalette
    SelectedBrushChanged="ColorPickerPalette_SelectedBrushChanged"
    Name="ColorPickerPalette"
    Width="60"
    Height="40">
</syncfusion:ColorPickerPalette>
```

### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.SelectedBrushChanged +=
    ColorPickerPalette_SelectedBrushChanged;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```

### C#

```
//Invoked when the selected color or brush is changed
private void ColorPickerPalette_SelectedBrushChanged(object sender,
    SelectedBrushChangedEventArgs e) {
```

```
//Old and newly selected brushes
var OldBrush = e.OldBrush ;
var newBrush = e.NewBrush;
//Old and newly selected colors
var oldColor = e.OldColor;
var newColor = e.NewColor;
}
```

Click [here](#) to download the sample that showcases different type of color items with its panel visibility customization.

### Localization support

Localization is the process of translating the application resources into different language for the specific cultures. You can localize the `ColorPickerPalette` control by adding resource file for each language.

**Note:** Refer [Localization](#) page to know more about how to provide a localization support for the `ColorPickerPalette`.

### Theme

`ColorPickerPalette` supports various built-in themes. Refer to the below links to apply themes for the `ColorPickerPalette`,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



## Working with ColorPickerPalette in WPF

This section explains the different types of colors available in the [ColorPickerPalette](#) and how to choose the colors and its panel customizations.

### Accessing a Color programmatically

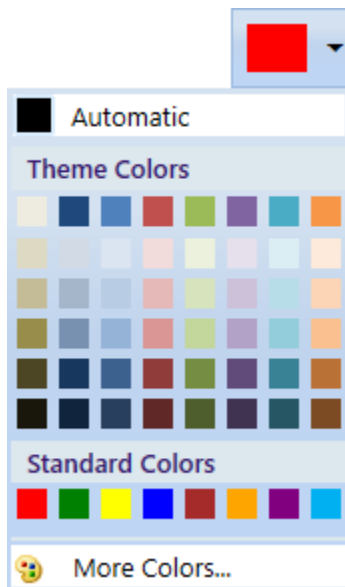
We can get or change the selected color of the `ColorPickerPalette` programmatically by setting the value to the [Color](#) property. If we want know the selected color name, use the [ColorName](#) property that holds the name of the selected color item. The default value of `Color` and `ColorName` property is `Black` and `Color`.

### XML

```
<syncfusion:ColorPickerPalette Color="Red"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```

### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.Color = Colors.Red;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```



Here, `Red` color is selected color in the `ColorPickerPalette`.

**Note:** [View Sample in GitHub](#)

### Accessing a color brush programmatically

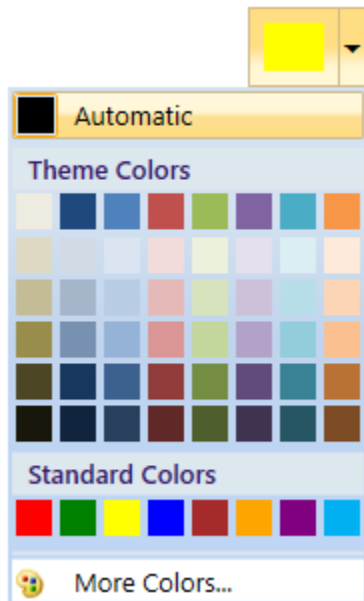
You can get or change the selected brush of the `ColorPickerPalette` programmatically by setting the value to the [SelectedBrush](#) property. The default value of `SelectedBrush` property is `Black`.

### XML

```
<syncfusion:ColorPickerPalette SelectedBrush="Yellow"
Name="colorPickerPalette"/>
```

**C#**

```
colorPickerPalette.SelectedBrush = Brushes.Yellow;
```



Here, Yellow color brush is selected in the ColorPickerPalette.

**Note:** [View Sample in GitHub](#)

**Setting automatic color**

If we want to change the default selected color on application launching, set the value for [AutomaticColor](#) property. If we changed the selected color, then we can easily make the default color as selected color by clicking the automatic color panel. We can hide the automatic color visibility by setting the [AutomaticColorVisibility](#) property value as `Collapsed`. The default value of `AutomaticColor` property is `Black` and the default value of `AutomaticColorVisibility` property is `Visible`.

**XML**

```
<syncfusion:ColorPickerPalette AutomaticColor="Green"
AutomaticColorVisibility="Visible"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```

**C#**

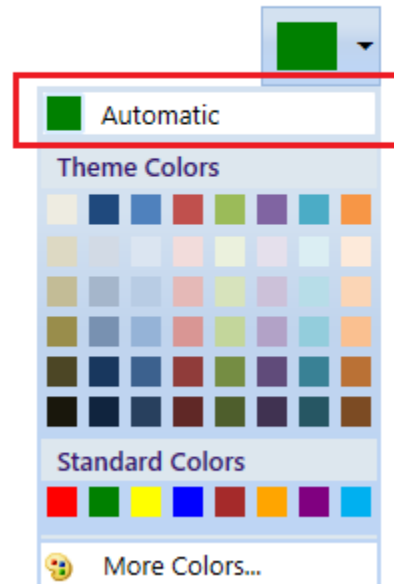
```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.AutomaticColor = Colors.Green;
colorPickerPalette.AutomaticColorVisibility= Visibility.Visible;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```



AutomaticColorVisibility="Collapsed"



AutomaticColorVisibility="Visible"



**Note:** [View Sample in GitHub](#)

Select transparent color programmatically

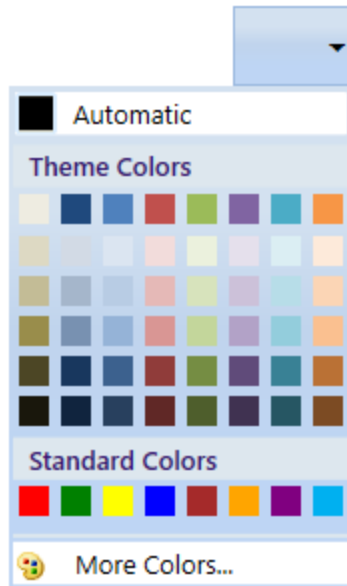
You can set a transparent color as selected color programmatically by setting the color code #00000000 or Colors.Transparent for Color property to indicate the null value.

#### XML

```
<syncfusion:ColorPickerPalette Color="Transparent"
Name="colorPickerPalette"/>
```

#### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.Color = Colors.Transparent;
```

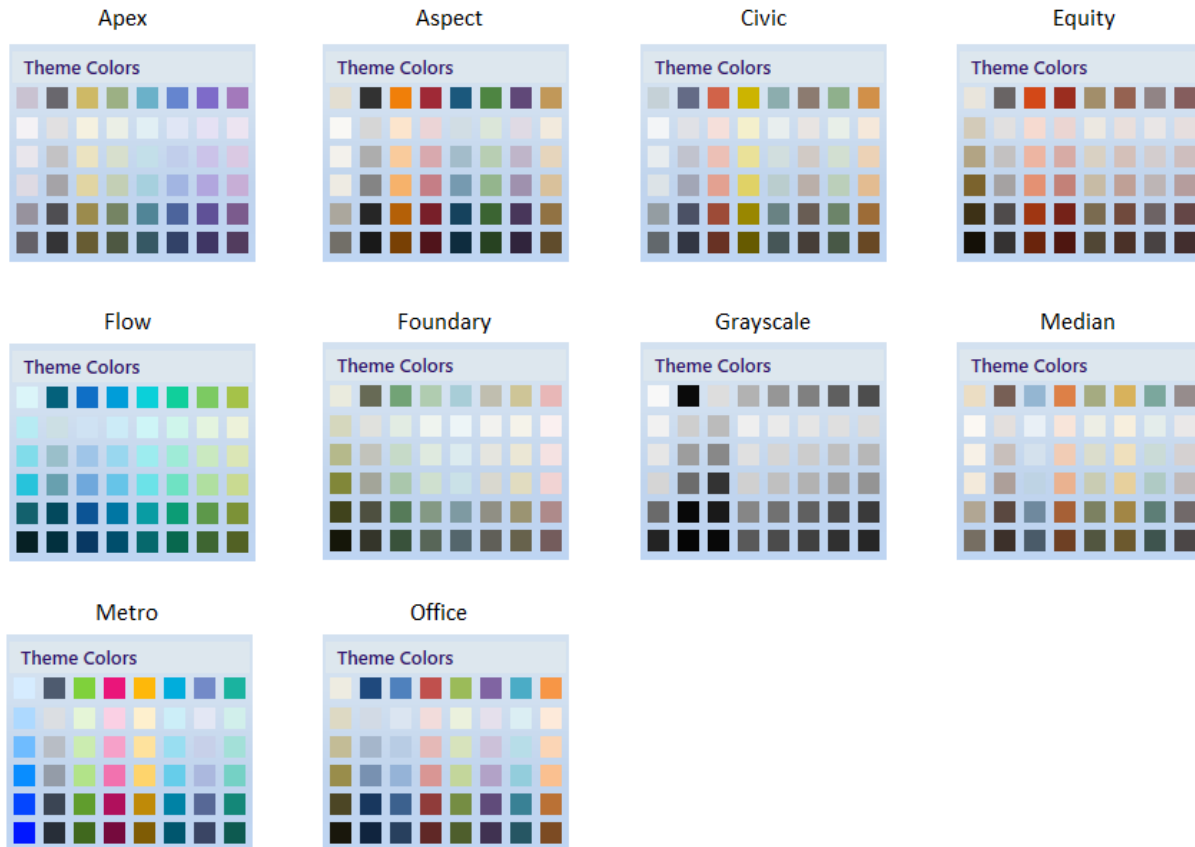


### Select a predefined colors

We can select a color from either various theme color items or standard color items. We can show or hide the color items panel visibilities.

#### Select a color from theme color items

We can select a various theme colors by setting the value for [Themes](#) property. Based on the [Themes](#) value, the respective base color items are displayed with its variants. If we want allow the user to select only base theme colors without its variants color, use the [GenerateThemeVariants](#) property as false. We can hide the theme color panel by setting the [ThemePanelVisibility](#) property value as Collapsed. The default value of [Themes](#) property is Office and default value of [ThemePanelVisibility](#) property is Visible.

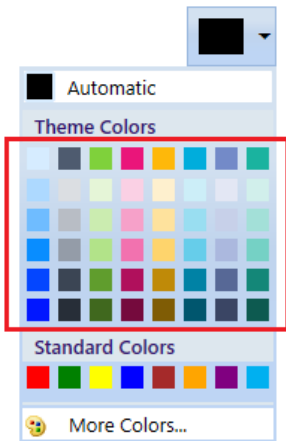


### XML

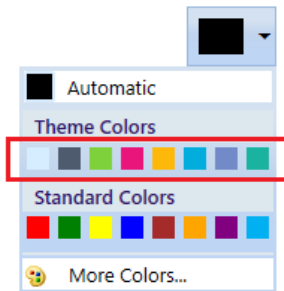
```
<syncfusion:ColorPickerPalette Themes="Metro"
GenerateThemeVariants="True"
ThemePanelVisibility="Visible"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```

### C#

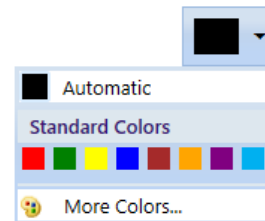
```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.Themes = PaletteTheme.Metro;
colorPickerPalette.GenerateThemeVariants = true;
colorPickerPalette.ThemePanelVisibility = Visibility.Visible;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```

Metro theme color items  
with its variant colors

GenerateThemeVariants="False"



ThemePanelVisibility = Collapsed



#### Select a color from standard color items

We can select a standard colors from the standard color panel. If we want allow the user to select standard colors with its variant colors, use the [GenerateStandardVariants](#) property as true. We can hide the standard color panel by setting the [StandardPanelVisibility](#) property value as Collapsed. The default value of [GenerateStandardVariants](#) property is false and default value of [StandardPanelVisibility](#) property is Visible.

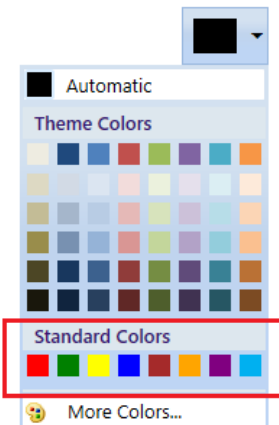
#### XML

```
<syncfusion:ColorPickerPalette GenerateStandardVariants="True"
StandardPanelVisibility="Visible"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```

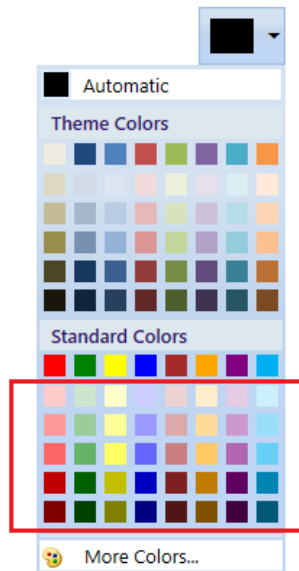
#### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.GenerateStandardVariants = true;
colorPickerPalette.StandardPanelVisibility = Visibility.Visible;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```

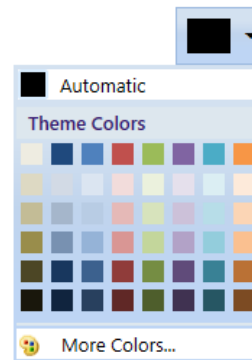
Default Standard colors



GenerateStandardVariants="True"



StandardPanelVisibility = "Collapsed"



Show white and black color variants

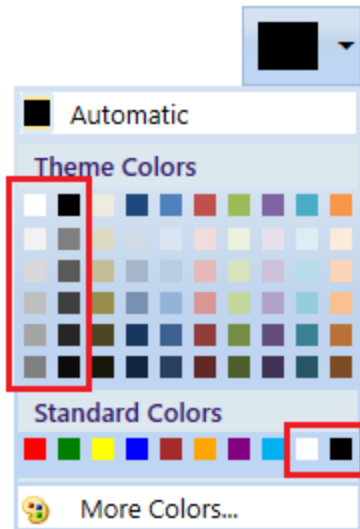
If we want to allow the user to select the theme color from white or black or both color variants, use the [BlackWhiteVisibility](#) property as `White` or `Black` or `Both`. The default value of `BlackWhiteVisibility` property is `None`.

#### XML

```
<syncfusion:ColorPickerPalette BlackWhiteVisibility="Both"
    Name="colorPickerPalette"
    Width="60"
    Height="40">
</syncfusion:ColorPickerPalette>
```

#### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.BlackWhiteVisibility = BlackWhiteVisible.Both;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```



Add your own colors in the palette

If we want to allow the user to select a color from own colors, add that color with its name using [CustomColor.ColorName](#) and [CustomColor.Color](#) into the [CustomColorsCollection](#) and set the [SetCustomColors](#) property value as `true`. The provided [CustomColor.ColorName](#) is shown in the tooltip while mouse hovering on the color item. We can change the custom color panel header text and its visibility by using the [CustomHeaderText](#) and [CustomHeaderVisibility](#) properties. The default value of [CustomHeaderText](#) is `CustomColors` and default value of [CustomHeaderVisibility](#) is `Visible`.

### C#

```
public class ViewModel : NotificationObject {
    private ObservableCollection<CustomColor> newColorCollection;
    public ObservableCollection<CustomColor> NewColorCollection {
        get {
            return newColorCollection;
        }
        set {
            newColorCollection = value;
            this.RaisePropertyChanged("NewColorCollection");
        }
    }
    public ViewModel() {
        NewColorCollection = new ObservableCollection<CustomColor>();
    }
}
```

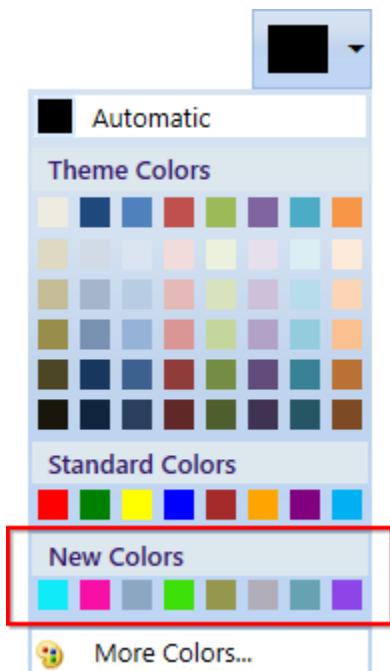
### XML

```
<Window.Resources>
<local:ViewModel x:Key="viewModel">
<local:ViewModel.NewColorCollection>
<!-- Defining the color details -->
<syncfusion:CustomColor Color="#FF11EBF8" ColorName="Aqua" />
<syncfusion:CustomColor Color="#FFF80FA6" ColorName="Deep Pink" />
<syncfusion:CustomColor Color="#FF8BA7C2" ColorName="Dark Gray" />
<syncfusion:CustomColor Color="#F53CDF07" ColorName="Lime Green" />
<syncfusion:CustomColor Color="#C2929545" ColorName="Olive Drab" />
```

```

<syncfusion:CustomColor Color="#2E956145" ColorName="Sienna" />
<syncfusion:CustomColor Color="#78458E95" ColorName="Steel Blue" />
<syncfusion:CustomColor Color="#8B8220E4" ColorName="Blue Violet" />
</local:ViewModel.NewColorCollection>
</local:ViewModel>
</Window.Resources>
<syncfusion:ColorPickerPalette CustomColorsCollection="{Binding
NewColorCollection}"
CustomHeaderText="New Colors"
CustomHeaderVisibility="Visible"
SetCustomColors="True"
DataContext="{StaticResource viewModel}"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>

```



Click [here](#) to download the sample that showcases how to add your own color items into the palette.

#### Recently used color items

The recently selected color items are displayed in the RecentlyUsedPanel. If we want to choose a color which are previously selected, use the RecentlyUsedPanel. We can hide the RecentlyUsedPanel by using the [RecentlyUsedPanelVisibility](#) property value as `Collapsed`. The default value of `RecentlyUsedPanelVisibility` property is `Visible`.

#### XML

```

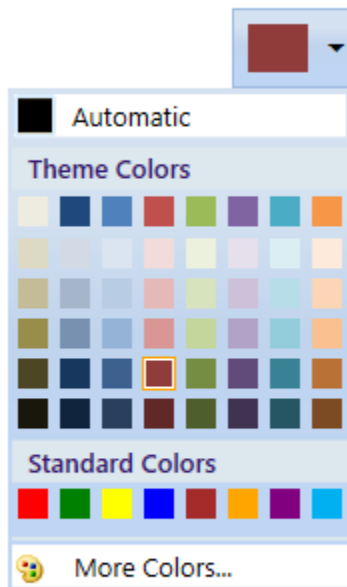
<syncfusion:ColorPickerPalette RecentlyUsedPanelVisibility="Visible"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>

```

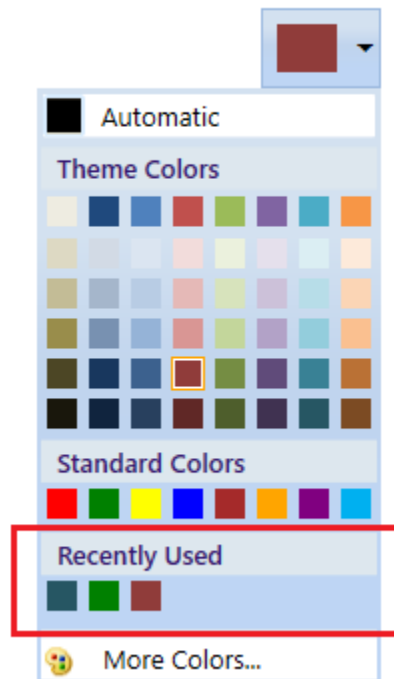
**C#**

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();  
colorPickerPalette.RecentlyUsedPanelVisibility = Visibility.Visible;  
colorPickerPalette.Width = 60;  
colorPickerPalette.Height = 40;
```

RecentlyUsedPanelVisibility = "Collapsed"



RecentlyUsedPanelVisibility = "Visible"

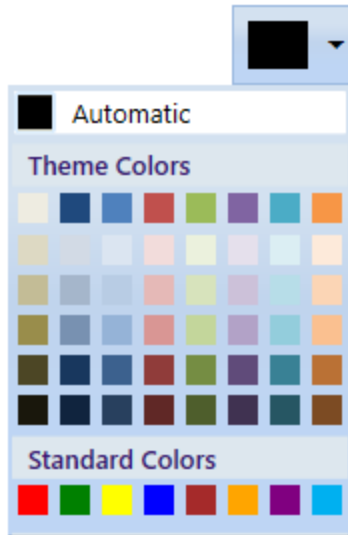


## Choosing a color from MoreColor window

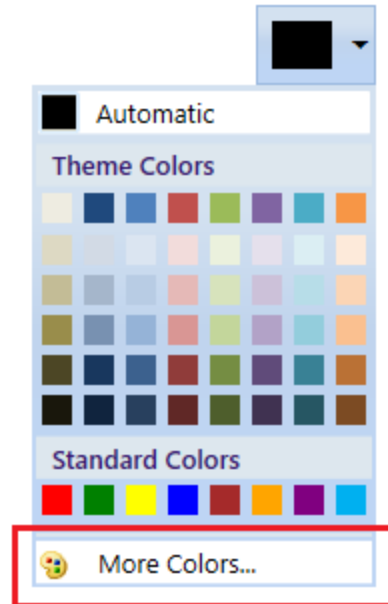
In addition to colors in Theme colors and Standard colors, MoreColor feature allows you to select wide range of color options. MoreColor feature includes two categories namely Standard Colors and Custom Colors. We can hide the visibility of the MoreColor Option by using the [MoreColorOptionVisibility](#) property value as `Collapsed`.



MoreColorOptionVisibility = "Collapsed"



MoreColorOptionVisibility = "Visible"



### Selecting more standard colors

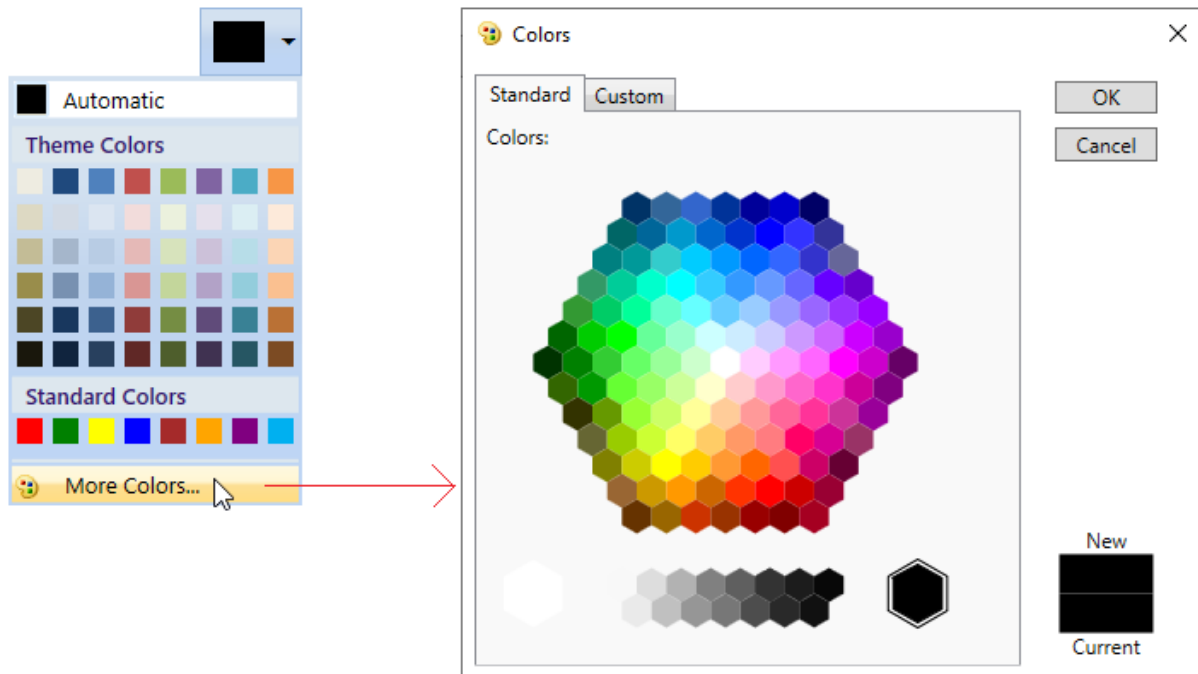
We can select color from 140 standard colors clustered in the shape of a Hexagon. If we want to hide the Standard color tab, use the [IsStandardTabVisible](#) property value as `Collapsed`. The color chosen from this cluster will also be added in the RecentlyUsedPanel.

### XML

```
<syncfusion:ColorPickerPalette IsStandardTabVisible="Visible"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```

### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.IsStandardTabVisible = Visibility.Visible;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```



### Selecting more custom colors

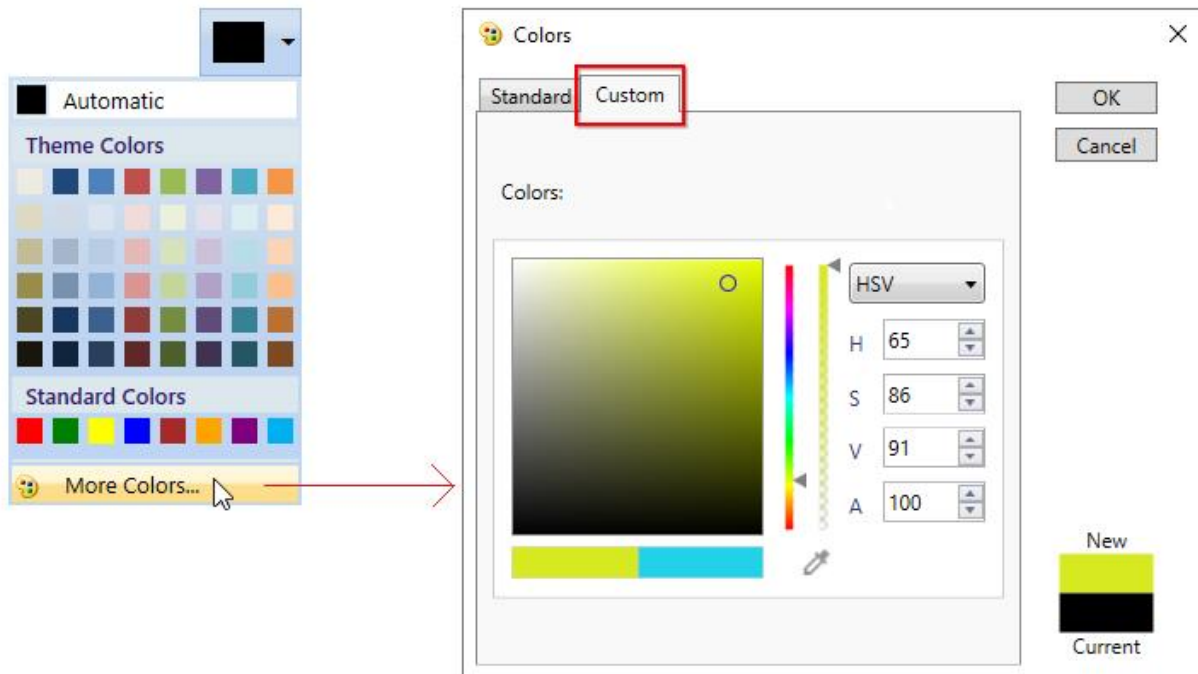
We can select any color and adjusting its saturation level by using the custom tab color picker . If we want to hide the custom color tab, use the [IsCustomTabVisible](#) property value as **Collapsed**. The color chosen from custom color picker will also be added in the RecentlyUsedPanel.

### XML

```
<syncfusion:ColorPickerPalette IsCustomTabVisible="Visible"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```

### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.IsCustomTabVisible = Visibility.Visible;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```



**Note:** If we set `IsCustomTabVisible` and `IsStandardTabVisible` property value as `false`, then `MoreColor` option automatically hides.

Clear the colour you picked with a transparent colour

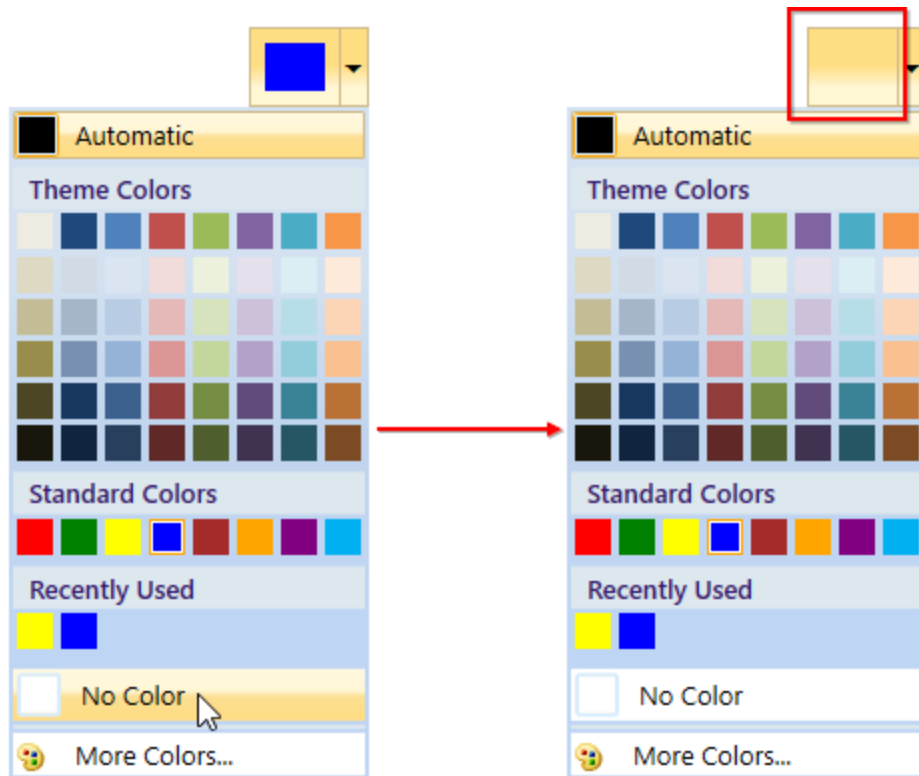
If you want to clear the selected color with a `Transparent` color, click the `No Color` button. You will be display the `No color` button only by setting the `NoColorVisibility` property value as `Visible`. The default value of `NoColorVisibility` property is `Collapsed`.

#### XML

```
<syncfusion:ColorPickerPalette NoColorVisibility="Visible"
    Name="colorPickerPalette"/>
```

#### C#

```
colorPickerPalette.NoColorVisibility = Visibility.Visible;
```



**Note:** [View Sample in GitHub](#)

Selected brush or color changed notification

The selected brush or color changed in `ColorPickerPalette` can be examined using `SelectedBrushChanged` event. The `SelectedBrushChangedEventArgs` contains the old and newly selected brush and its color values in the `OldBrush`, `NewBrush` and `OldColor`, `NewColor` properties. You can also get the selected brush and color changed notification by using the [SelectedCommand](#) property.

### XML

```
<syncfusion:ColorPickerPalette
    SelectedBrushChanged="ColorPickerPalette_SelectedBrushChanged"
    Name="ColorPickerPalette"
    Width="60"
    Height="40">
</syncfusion:ColorPickerPalette>
```

### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.SelectedBrushChanged +=
    ColorPickerPalette_SelectedBrushChanged;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```

### C#

```
//Invoked when the selected color or brush is changed
```

```
private void ColorPickerPalette_SelectedBrushChanged(object sender,
SelectedBrushChangedEventArgs e) {
    //Old and newly selected brushes
    var OldBrush = e.OldBrush ;
    var newBrush = e.NewBrush;
    //Old and newly selected colors
    var oldColor = e.OldColor;
    var newColor = e.NewColor;
}
```

Customize the header

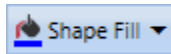
You can customize the appearance of the `ColorPickerPalette` header and can display the selected color name in the header by using the [HeaderTemplate](#) property.

**Note:** The `DataContext` of `HeaderTemplate` is `ColorPickerPalette`

### XML

```
<Window.Resources>
<DataTemplate x:Key="Custom_HeaderTemplate">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="auto"/>
<ColumnDefinition Width="auto"/>
</Grid.ColumnDefinitions>
<Grid x:Name="IconGrid"
Margin="2">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*/"/>
</Grid.RowDefinitions>
<Image x:Name="image"
Source="/images/fill.png"
Height="12"
Width="12"/>
<Border Name="color_border"
Grid.Row="1"
Height="3">
<Border.Background>
<SolidColorBrush Color="{Binding Color,
RelativeSource={RelativeSource FindAncestor,
AncestorLevel=1,
AncestorType={x:Type syncfusion:ColorPickerPalette}}},
UpdateSourceTrigger=PropertyChanged"/>
</Border.Background>
</Border>
</Grid>
<TextBlock Padding="1"
HorizontalAlignment="Left"
VerticalAlignment="Center"
TextAlignment="Center" Grid.Column="1"
Text="Shape Fill" FontSize="11"
Width="auto"/>
</Grid>
</DataTemplate>
</Window.Resources>
```

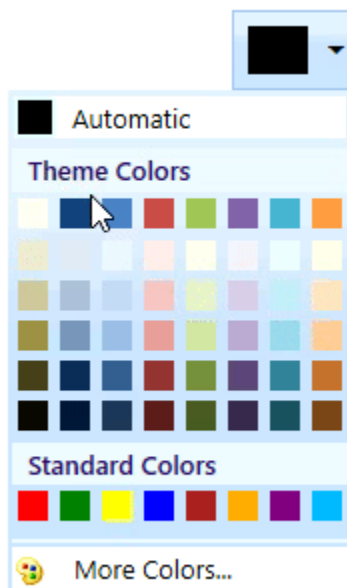
```
<Grid>
<syncfusion:ColorPickerPalette HeaderTemplate="{DynamicResource
Custom_HeaderTemplate}"
Name="ColorPickerPallette2"
Margin="10"
Mode="Split"/>
</Grid>
```



**Note:** View [Sample](#) in GitHub

### Tooltip support

Tooltip is used to show the information about the segment, when you mouse over on the segment. We can show information about the name of the color item using tooltip when hovering the mouse on the specific color item.



### Expanded mode

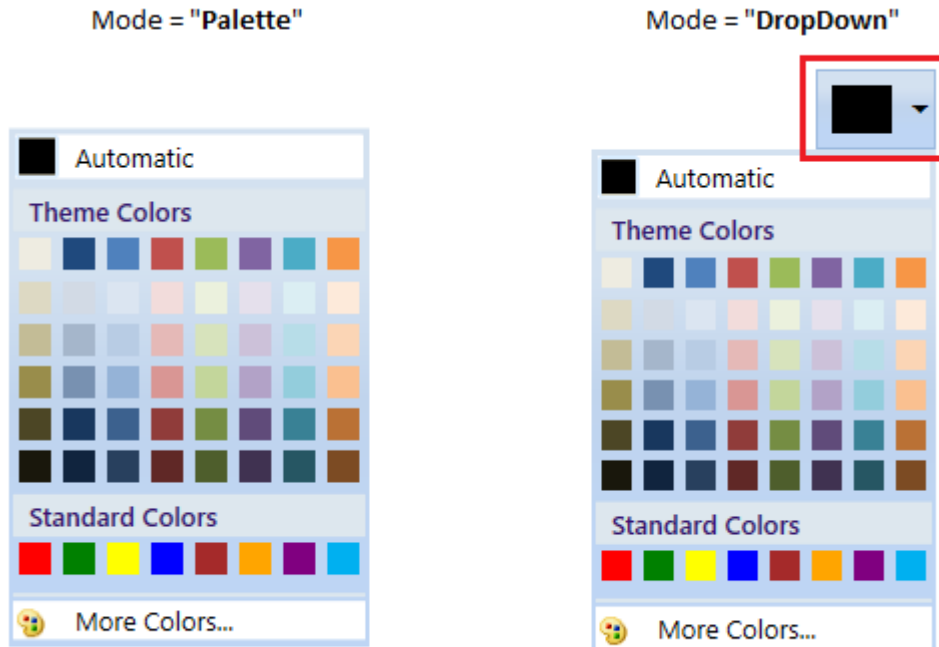
If you want to directly use the palette without drop down button, set the **Mode** property value as **Palette**.

### XML

```
<syncfusion:ColorPickerPalette Mode="Palette"
Name="colorPickerPalette"/>
```

### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.Mode = PickerMode.Palette;
```



**Note:** View [Sample](#) in GitHub

#### ColorPickerPalette as a command button

By default, ColorPickerPalette acts like a dropdown. It opening a color palette when clicking anywhere on the header. By setting the [Mode](#) property to `Split`, it acts like a button and dropdown as explained below.

1. When clicking on the dropdown arrow button, It acts like a dropdown.
2. When you click on the header area, it acts like a button and [SelectedCommand](#) will be triggered. Using this command, you can do some action like applying the selected color as background of selected text.

DropDown mode



Split mode



For example, if you want to apply a last selected color as a foreground to a TextEditor's selected text. You can direct click the button instead of opening the dropdown and selecting an already selected color again.

#### C#

```
//ViewModel.cs
public class ViewModel : NotificationObject
{
    private ICommand selectionChangedCommand;
    private ICommand loadedChangedCommand;
    private RichTextBox TextBox;
```

```

public ICommand SelectionChangedCommand {
    get {
        return selectionChangedCommand;
    }
}

public ICommand LoadedChangedCommand {
    get {
        return loadedChangedCommand;
    }
}

public void Loadedmethod(object param) {
    TextBox = param as RichTextBox;
}

public void PropertyChangedHandler(object param) {
    if (param != null && TextBox != null) {
        ColorSelectedCommandArgs groupItem = param as ColorSelectedCommandArgs;
        TextRange range = new TextRange(TextBox.Selection.Start,
            TextBox.Selection.End);
        range.ApplyPropertyValue(FlowDocument.ForegroundProperty, groupItem.Brush);
    }
}

public ViewModel() {
    selectionChangedCommand = new
        DelegateCommand<object>(PropertyChangedHandler);
    loadedChangedCommand = new DelegateCommand<object>(Loadedmethod);
}
}

```

**XML**

```

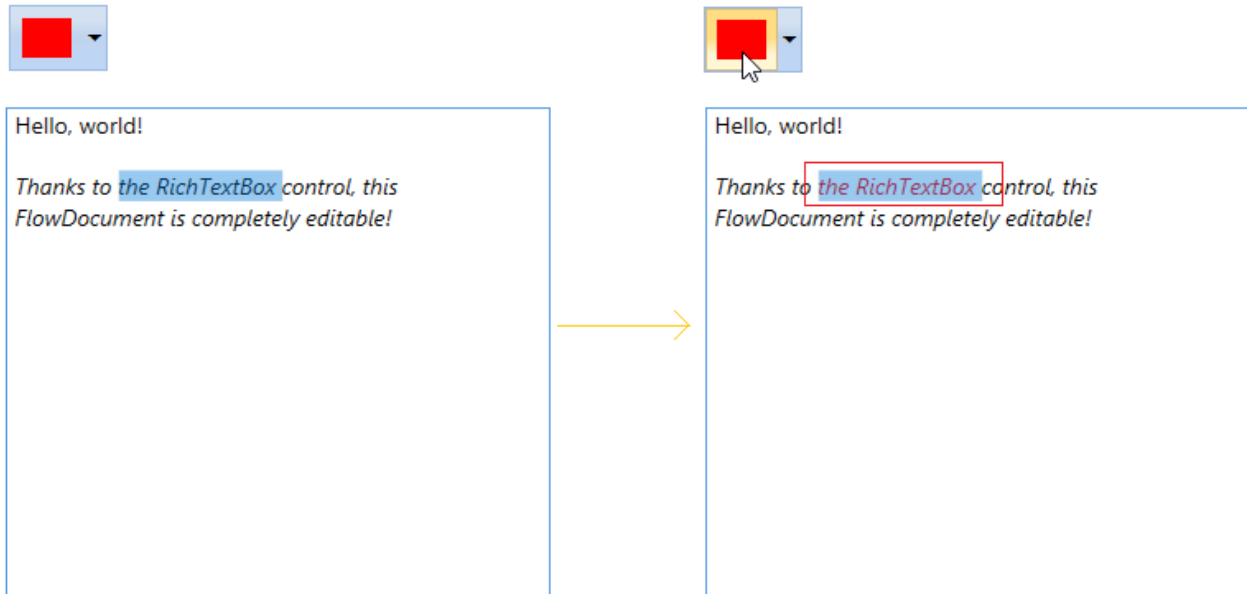
<syncfusion:ColorPickerPalette Name="colorpickerpalette"
Mode="Split"
SelectedCommand="{Binding SelectionChangedCommand}"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
<RichTextBox Name="richTextBox"
Height="297"
Width="331">
<i:Interaction.Triggers>
<i:EventTrigger EventName="Loaded">
<i:InvokeCommandAction Command="{Binding LoadedChangedCommand}"
CommandParameter="{Binding ElementName=richTextBox}"/>
</i:EventTrigger>
</i:Interaction.Triggers>
<FlowDocument>
<Paragraph FontSize="14">Hello, world!</Paragraph>
<Paragraph FontStyle="Italic"
TextAlignment="Left"
FontSize="14">Thanks to the RichTextBox control,
this FlowDocument is completely editable!</Paragraph>
</FlowDocument>
</RichTextBox>

```

**C#**



```
colorPickerPalette.Mode = PickerMode.Split;
```



**Note:** View [Sample](#) in GitHub

### Change color item size

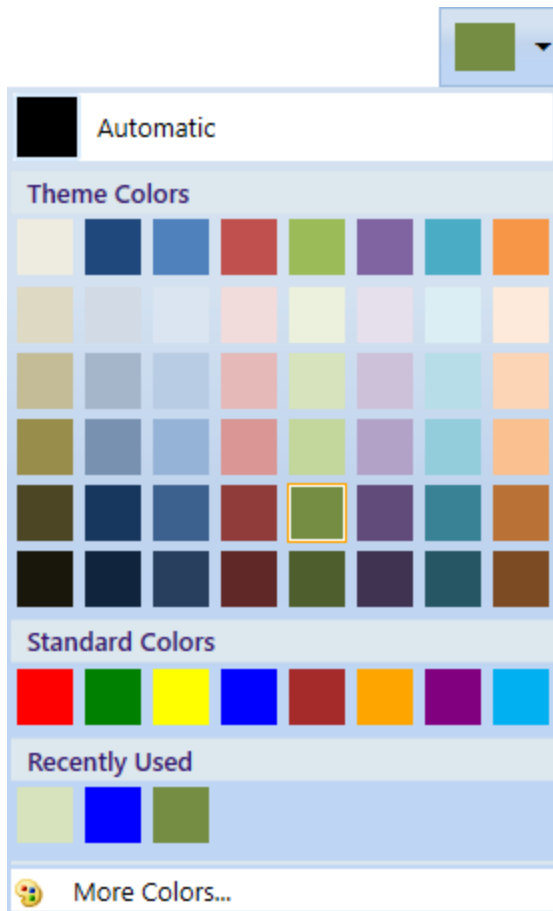
We can change each color item size by using the [BorderWidth](#) and [BorderHeight](#) properties. Based on the color items size, the color palette is resized. The default value of [BorderWidth](#) and [BorderHeight](#) properties is 17.

### XML

```
<syncfusion:ColorPickerPalette BorderWidth="30"  
BorderHeight="30"  
Name="colorPickerPalette"  
Width="60"  
Height="40">  
</syncfusion:ColorPickerPalette>
```

### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();  
colorPickerPalette.BorderWidth = 30;  
colorPickerPalette.BorderHeight = 30;  
colorPickerPalette.Width = 60;  
colorPickerPalette.Height = 40;
```



### Change color palette size

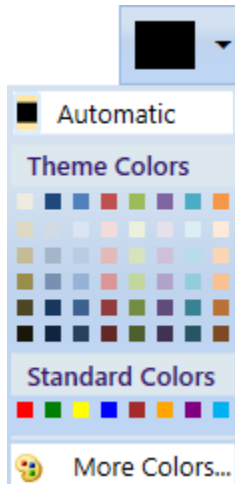
We can change the color palette pop size by using the [PopupWidth](#) and [PopupHeight](#) properties. Based on the popup color palette size, the color items are resized. The default value of `PopupWidth` and `PopupHeight` properties is 175 and 200.

### XML

```
<syncfusion:ColorPickerPalette PopupWidth="120"
PopupHeight="100"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```

### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.PopupWidth = 120;
colorPickerPalette.PopupHeight = 100;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```



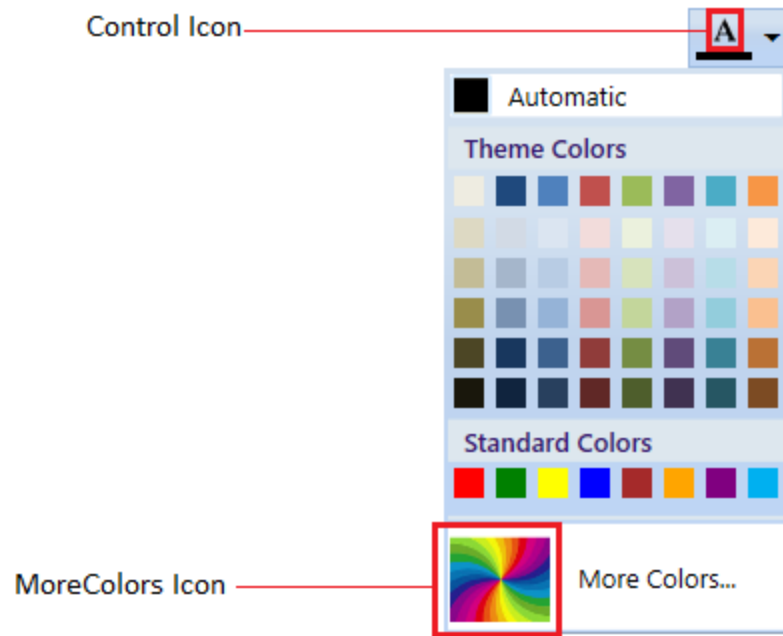
**Note:** If we use both `PopupWidth` & `PopupHeight` and `BorderWidth` & `BorderHeight`, then `BorderWidth` & `BorderHeight` properties have higher priority.

#### Change header and more color icons

We can set the icons for control header which is placed left to the DropDown button and more color panel header by using the [Icon](#) and [MoreColorsIcon](#) properties. We can change the icon size for the control icon and more color icon by using the [IconSize](#) and [MoreColorsIconSize](#) properties.

#### XML

```
<syncfusion:ColorPickerPalette Icon="/Label.png"
IconSize="18,18"
MoreColorsIcon="/MoreColor.png"
MoreColorsIconSize="50,50"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```



Click [here](#) to download the sample that showcases features and different type color items with its panel visibility customization.

#### Hide the drop down button

You can hide the dropdown button in the `ColorPickerPalette` by setting the dropdown button visibility as `Collapsed`. You can open a popup color palette by clicking the header of the `ColorPickerPalette`.

#### XML

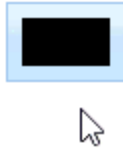
```
<syncfusion:ColorPickerPalette Loaded="ColorPickerPalette_Loaded"
Name="colorPickerPalette"/>
```

#### C#

```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.Loaded += ColorPickerPalette_Loaded;
```

#### C#

```
private void ColorPickerPalette_Loaded(object sender, RoutedEventArgs e) {
    var dropDown = (Border) (sender as
    ColorPickerPalette).Template.FindName("UpDownBorder", colorPickerPalette);
    if (dropDown != null) {
        dropDown.Visibility = Visibility.Collapsed;
    }
}
```



### Appearance in WPF Color Picker Palette

This section explains different UI customization, styling, theming options available in [ColorPickerPalette](#) control.

#### Change flow direction

We can change the flow direction of the `ColorPickerPalette` layout from right to left by setting the `FlowDirection` property value as `RightToLeft`. The Default value of `FlowDirection` property is `LeftToRight`.

#### XML

```
<syncfusion:ColorPickerPalette FlowDirection="RightToLeft"
Name="colorPickerPalette"
Width="60"
Height="40">
</syncfusion:ColorPickerPalette>
```

#### C#

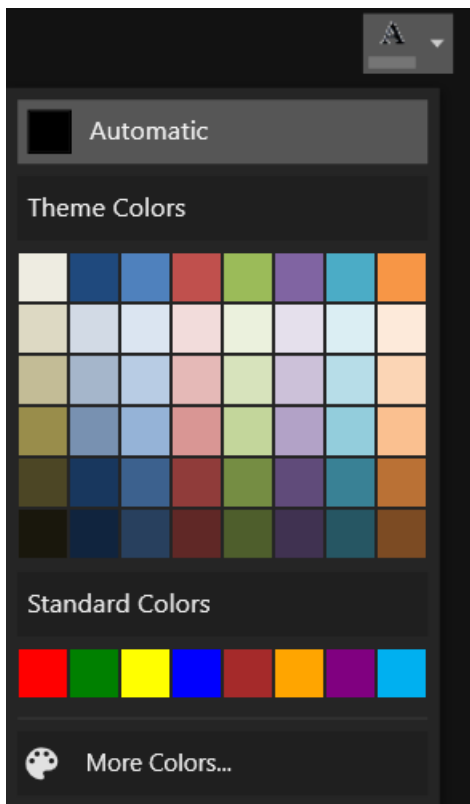
```
ColorPickerPalette colorPickerPalette = new ColorPickerPalette();
colorPickerPalette.FlowDirection = FlowDirection.RightToLeft;
colorPickerPalette.Width = 60;
colorPickerPalette.Height = 40;
```



### Theme

ColorPickerPalette supports various built-in themes. Refer to the below links to apply themes for the ColorPickerPalette,

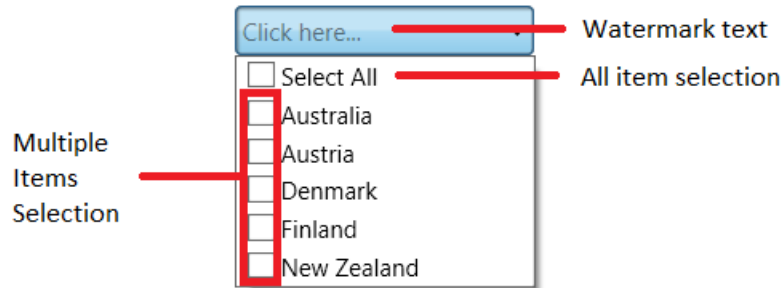
- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



## ComboBoxAdv

### WPF ComboBox (ComboBoxAdv) Overview

The [ComboBoxAdv](#) control allows the component that allows user to type a value or choose an option from a list of predefined options. It has several out of the box features such as data binding, multiselection, editing and more.



Key features are:

- [AllowMultiSelect](#) : Support to select multiple values from drop-down list.
- [DefaultText](#) : Helps to prompt user by providing additional hints about the data that should be entered into the text box.
- [SelectedValueDelimiter](#) : Allows customizing the delimiter string displayed between selected items during multiple selection.
- [EnableOkCancel](#) : Enables ok and cancel button at the bottom position in the dropdown box.
- [AllowSelectAll](#) : Adds **Select All** item in ComboBoxAdv dropdown and allows to select all items in it.

### Getting Started with WPF ComboBox (ComboBoxAdv)

This section provides a quick overview for working with the ComboBox ([ComboBoxAdv](#)).

**Note:** [View sample in GitHub](#).

#### Assembly deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the [ComboBoxAdv](#) control in any application.

You can find more details about installing the NuGet package in a WPF application in the following link:

[How to install nuget packages](#)

#### Creating Application with ComboBoxAdv control

In this walk through, user will create a WPF application that contains [ComboBox](#) control.

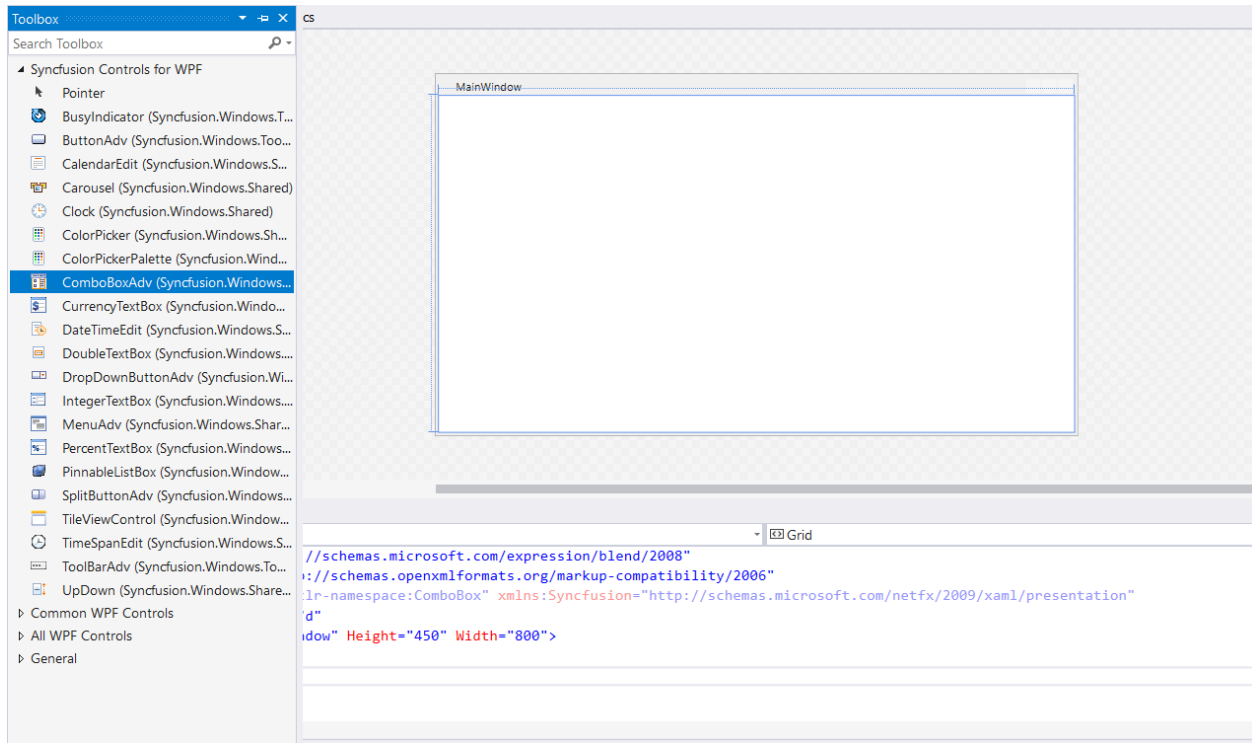
1. [Creating project](#)
2. [Adding control via designer](#)
3. [Adding control manually in XAML](#)
4. [Adding control manually in C#](#)
5. [Creating Data Model for sample application](#)
6. [Binding to Data](#)

## Creating project

Below section provides detailed information to create new project in Visual Studio to display [ComboBoxAdv](#).

## Adding control via designer

The [ComboBoxAdv](#) control can be added to the application by dragging it from Toolbox and dropping it in designer. The required [assemblies](#) will be added automatically.



## Adding control manually in XAML

In order to add [ComboBoxAdv](#) control manually in XAML, do the below steps,

1. Add the below required assembly references to the project,
  - o Syncfusion.Shared.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in XAML page or Syncfusion.Windows.Tools.Controls namespace.
3. Declare [ComboBoxAdv](#) in XAML page.

## XAML

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:GettingStartedComboBox"
    xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
    x:Class="GettingStartedComboBox.MainWindow"
    mc:Ignorable="d"
    Title="MainWindow" Height="450" Width="800">
    <Grid>
```



```
<syncfusion:ComboBoxAdv Height="30" Width="150"/>
</Grid>
</Window>
```

### Adding control manually in C#

In order to add [ComboBoxAdv](#) control manually in C#, do the below steps,

1. Add the below required assembly references to the project,
  - Syncfusion.Shared.WPF
2. Import ComboBoxAdv namespace **Syncfusion.Windows.Tools.Controls**.
3. Create ComboBoxAdv control instance and add it to the page.

### C#

```
using System.Windows;
using Syncfusion.Windows.Tools.Controls;
namespace ComboBox
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            ComboBoxAdv comboBoxAdv = new ComboBoxAdv();
            this.Content = comboBoxAdv;
            comboBoxAdv.Height = 30;
            comboBoxAdv.Width = 150;
            comboBoxAdv.DefaultText = "choose Items";
        }
    }
}
```

### Adding items in ComboBoxAdv

Items can be added in the [ComboBoxAdv](#) control by following ways.

1. Adding items by ComboBoxItemAdv.
2. Adding items by DataBinding.

### Add items using ComboBoxItemAdv

The items in [ComboBoxAdv](#) can be created by using [ComboBoxItemAdv](#) in XAML or C# code.

### XML

```
<syncfusion:ComboBoxAdv Height="30" Width="200"
    HorizontalAlignment="Center"
    VerticalAlignment="Center" >
    <syncfusion:ComboBoxItemAdv Content="Denmark" />
    <syncfusion:ComboBoxItemAdv Content="New Zealand" />
    <syncfusion:ComboBoxItemAdv Content="Canada" />
</syncfusion:ComboBoxAdv>
```

```
<syncfusion:ComboBoxItemAdv Content="Russia" />
<syncfusion:ComboBoxItemAdv Content="Japan" />
</syncfusion:ComboBoxAdv>
```

## C#

```
public MainWindow()
{
    InitializeComponent();
    ComboBoxAdv comboBoxAdv = new ComboBoxAdv() { Height=30,Width= 200
    ,HorizontalAlignment= HorizontalAlignment.Center, VerticalAlignment =
    VerticalAlignment.Center };
    ComboBoxItemAdv item1 = new ComboBoxItemAdv() { Content = "Denmark" };
    ComboBoxItemAdv item2 = new ComboBoxItemAdv() { Content = "New Zealand" };
    ComboBoxItemAdv item3 = new ComboBoxItemAdv() { Content = "Canada" };
    ComboBoxItemAdv item4 = new ComboBoxItemAdv() { Content = "Russia" };
    ComboBoxItemAdv item5 = new ComboBoxItemAdv() { Content = "Japan" };
    comboBoxAdv.Items.Add(item1);
    comboBoxAdv.Items.Add(item2);
    comboBoxAdv.Items.Add(item3);
    comboBoxAdv.Items.Add(item4);
    comboBoxAdv.Items.Add(item5);
    this.Content = comboBoxAdv;
}
```

## Adding items by DataBinding

The items in [ComboBoxAdv](#) can be added by data binding by following below procedure.

### *Creating Model and ViewModel data for DataBinding*

1. Create data object class named **PopulationInfo** and declare properties as shown below,

## C#

```
public class PopulationInfo
{
    private string continent;
    private double population;
    private string country;
    private double growth;
    public string Continent
    {
        get { return continent; }
        set { continent = value; }
    }
    public string Country
    {
        get { return country; }
        set { country = value; }
    }
    public double Growth
    {
        get { return growth; }
        set { growth = value; }
    }
}
```

```
public double Population
{
    get { return population; }
    set { population = value; }
}
```

2. Create a **ViewModel** class with several data objects in constructor.

## C#

```
public class PopulationViewModel
{
    public PopulationViewModel()
    {
        this.PopulationDetails = new ObservableCollection<PopulationInfo>();
        PopulationDetails.Add(new PopulationInfo() { Continent = "Asia", Country = "Indonesia", Growth = 3, Population = 237641326 });
        PopulationDetails.Add(new PopulationInfo() { Continent = "Asia", Country = "Russia", Growth = 2, Population = 152518015 });
        PopulationDetails.Add(new PopulationInfo() { Continent = "Asia", Country = "Malaysia", Growth = 1, Population = 29672000 });
        PopulationDetails.Add(new PopulationInfo() { Continent = "North America", Country = "United States", Growth = 4, Population = 315645000 });
        PopulationDetails.Add(new PopulationInfo() { Continent = "North America", Country = "Mexico", Growth = 2, Population = 112336538 });
        PopulationDetails.Add(new PopulationInfo() { Continent = "North America", Country = "Canada", Growth = 1, Population = 35056064 });
        PopulationDetails.Add(new PopulationInfo() { Continent = "South America", Country = "Colombia", Growth = 1, Population = 47000000 });
        PopulationDetails.Add(new PopulationInfo() { Continent = "South America", Country = "Brazil", Growth = 3, Population = 193946886 });
        PopulationDetails.Add(new PopulationInfo() { Continent = "Africa", Country = "Nigeria", Growth = 2, Population = 170901000 });
        PopulationDetails.Add(new PopulationInfo() { Continent = "Africa", Country = "Egypt", Growth = 1, Population = 83661000 });
        PopulationDetails.Add(new PopulationInfo() { Continent = "Europe", Country = "Germany", Growth = 1, Population = 81993000 });
        PopulationDetails.Add(new PopulationInfo() { Continent = "Europe", Country = "France", Growth = 1, Population = 65605000 });
        PopulationDetails.Add(new PopulationInfo() { Continent = "Europe", Country = "UK", Growth = 1, Population = 63181775 });
    }
    public ObservableCollection<PopulationInfo> PopulationDetails
    {
        get;
        set;
    }
}
```

### Binding to Data

To bind the [ComboBoxAdv](#) to data, bind the collection created in previous step to [ItemsSource](#) property in XAML by setting [PopulationViewModel](#) as [DataContext](#).

## XML

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:GettingStartedComboBox"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="GettingStartedComboBox.MainWindow"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<Grid.DataContext>
<local:PopulationViewModel/>
</Grid.DataContext>
<syncfusion:ComboBoxAdv x:Name="comboBoxAdv" Height="30" Width="200"
ItemsSource="{Binding PopulationDetails}"/>
</Grid>
</Window>
```

## C#

```
namespace ComboBox
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new PopulationViewModel();
        }
    }
}
```

### *Binding display member*

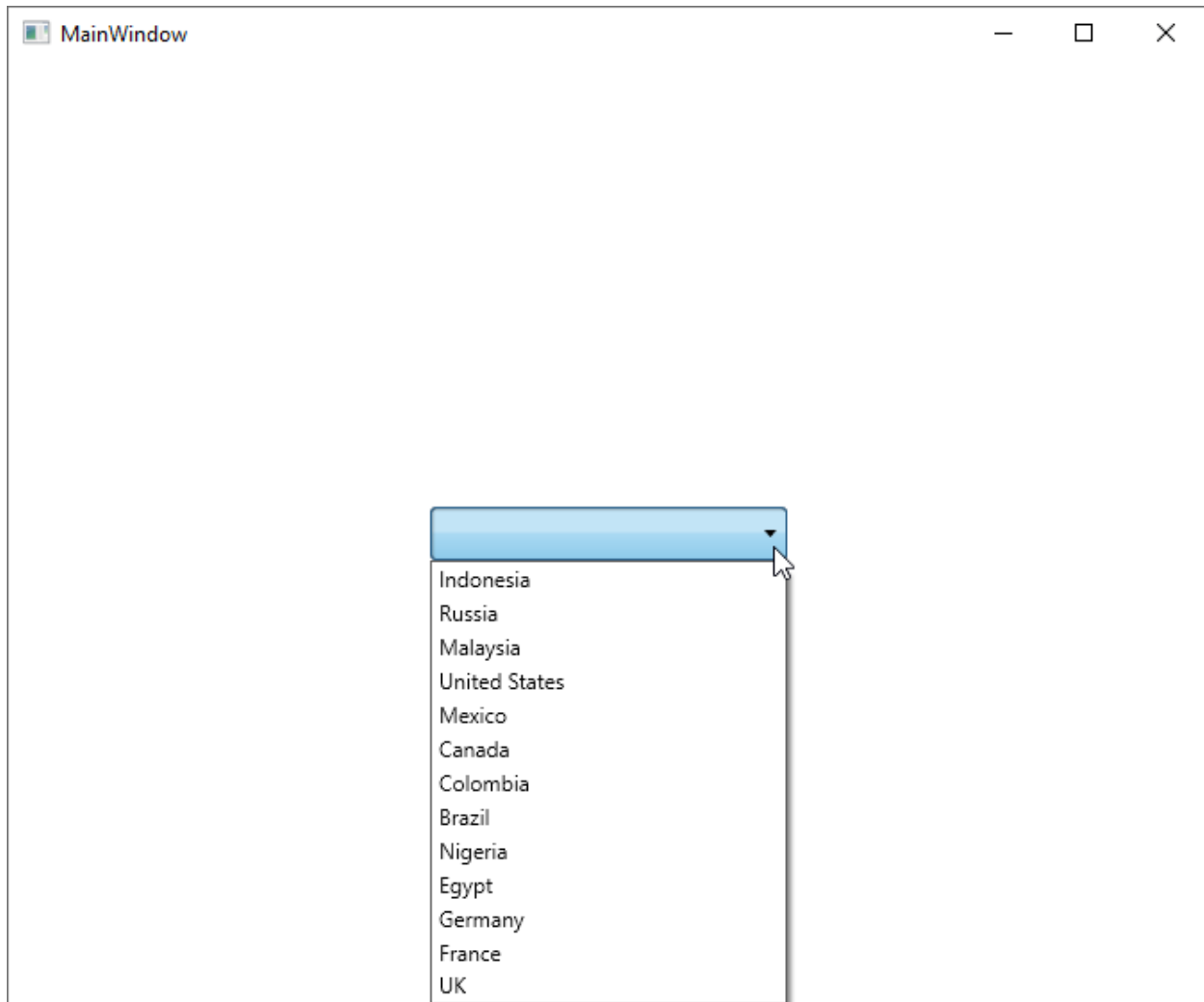
[DisplayMemberPath](#) denotes the path to a value on the data object for visual presentation of item to be displayed in combobox drop down list and displays the selected item in [ComboBoxAdv](#).

## XML

```
<Grid>
<Grid.DataContext>
<local:PopulationViewModel/>
</Grid.DataContext>
<syncfusion:ComboBoxAdv x:Name="comboBoxAdv" Height="30" Width="200"
ItemsSource="{Binding PopulationDetails}" DisplayMemberPath="Country"/>
</Grid>
```

## C#

```
// Initialize the display member path to comboboxadv.
this.comboBoxAdv.DisplayMemberPath = "Country";
```



**Note:** [View the sample in GitHub](#)

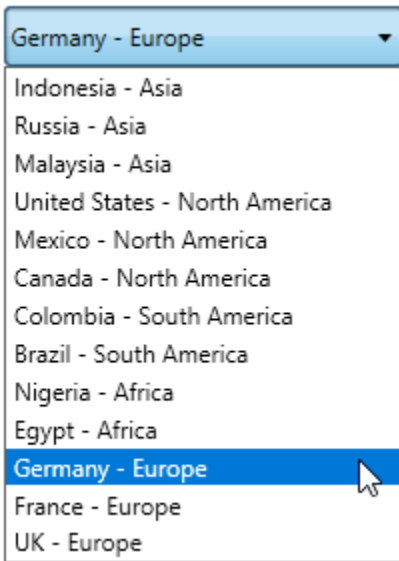
### Defining ItemTemplate

You can customize the visualization of data object using the [ItemTemplate](#).

### XML

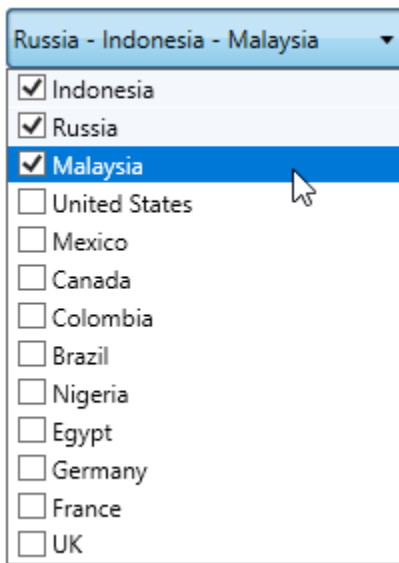
```
<Grid>
<Grid.DataContext>
<local:PopulationViewModel/>
</Grid.DataContext>
<syncfusion:ComboBoxAdv x:Name="comboBoxAdv" Height="30" Width="200"
ItemsSource="{Binding PopulationDetails}">
<syncfusion:ComboBoxAdv.ItemTemplate>
<DataTemplate>
<StackPanel Orientation="Horizontal">
<TextBlock Text="{Binding Country}"/>
<TextBlock Text=" - "/>
<TextBlock Text="{Binding Continent}"/>
</StackPanel>
</DataTemplate>
</syncfusion:ComboBoxAdv.ItemTemplate>
```

```
</syncfusion:ComboBoxAdv>
</Grid>
```



### Selection

[ComboBoxAdv](#) supports single and multiple selection of items. By default the selection of items in [ComboBoxAdv](#) is single selection. In order to select multiple items in [ComboBoxAdv](#), enable the [AllowMultiSelect](#) property and select those multiple items from the drop down list.



You can select the item or get the index of the selected item by using the [SelectedIndex](#) property. When an item is selected in [ComboBoxAdv](#), you can get their information using [SelectedItem](#) or [SelectedValue](#) property. For multiple selected items, use [SelectedItems](#) property. The selection of the items can be handled using [SelectionChanged](#) event.

### Editing

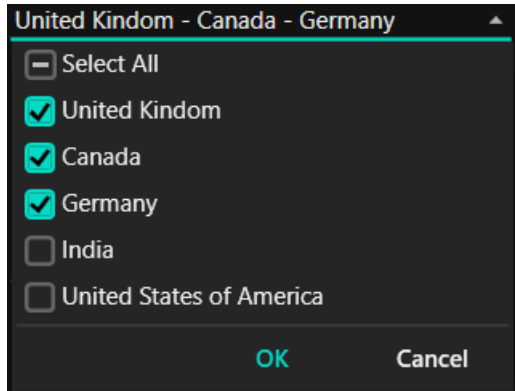
[IsEditable](#) property helps to edit the text in [ComboBoxAdv](#).



### Theme

ComboBoxAdv supports various built-in themes. Refer to the below links to apply themes for the ComboBoxAdv,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



See Also

[How to filter dropdown items in WPF editable ComboBoxAdv?](#)

[How to define maximum number of items to be shown in combobox dropdown?](#)

### Editable Support in WPF ComboBox (ComboBoxAdv)

#### IsEditable

It allows user to edit the text in the ComboBoxAdv.

Property	Description	Type	Data Type	Reference links
IsEditable	It is possible to edit the text of ComboBoxAdv.	Dependency Property	Boolean	NA

#### Adding IsEditable property to an application

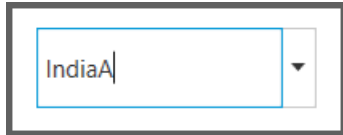
IsEditable property can be added directly to an application in the following way:

##### XML

```
<syncfusion:ComboBoxAdv IsEditable="true"></syncfusion:ComboBoxAdv>
```

##### C#

```
ComboBoxAdv comboBox = new ComboBoxAdv();
comboBox.IsEditable = true;
```



### Auto Complete Support in WPF ComboBox

You can able to find the expected item from the dropdown of the [ComboBoxAdv](#) and it can be enabled by using the [AutoCompleteMode](#) property. It can be used for both single and multiple selections on editable mode.

There are three different AutoComplete modes:

- **Suggest:** Suggestions are displayed in a drop-down menu.
- **None:** No suggestion is made.

Default value is *None*.

#### Suggest

By setting the [AutoCompleteMode](#) property to **Suggest**, a list of possible matches will be suggested and displayed in the drop-down list. The search text are included or start with the suggested items.

#### XML

```
<syncfusion:ComboBoxAdv AutoCompleteMode="Suggest"/>
```

#### C#

```
ComboBoxAdv comboBox = new ComboBoxAdv();
comboBox.AutoCompleteMode = AutoCompleteModes.Suggest;
```



**Note:** Suggest mode will be applicable only when the [ComboBoxAdv](#) is populated with [ItemSource](#) collection. When [AutoCompleteMode](#) is set to **Suggest** and [IsTextSearchEnabled](#) is set to **True** in editable mode, [AutoCompleteMode](#) will take precedence.

**Note:** [View sample in GitHub](#)

### MultiSelection Support in WPF ComboBox (ComboBoxAdv)

This section explains how to select the multiple items and select the items programmatically in the [WPF ComboBox](#) (ComboBoxAdv) control.



## Properties

Property	Description	Type	Data Type	Reference links
AllowMultiSelect	Multiple items can be selected.	Dependency Property	Boolean	NA
SelectedItems	It contains the selected items value	Dependency Property	ObservableCollection<object>	NA

## Adding multiple selections to an application

You can select the multiple items in the WPF ComboBox (ComboBoxAdv) control by setting the [AllowMultiSelect](#) property to `true`.

### XML

```
<syncfusion:ComboBoxAdv x:Name="comboBoxAdv" AllowMultiSelect="True">
</syncfusion:ComboBoxAdv>
```

### C#

```
ComboBoxAdv comboBoxAdv = new ComboBoxAdv();
comboBoxAdv.AllowMultiSelect = true;
```

## Selecting an item through programmatically

You can select the items programmatically by using the [SelectedItems](#) property. When `AllowMultiSelect` is set to `true`, the `SelectedItems` property exposes the items that are selected in the drop-down list.

In the below example, first two items from the Observable Collection bound to the `SelectedItems` property.

### Creating Model and ViewModel data for DataBinding

1. Create a data object class named **Country** and declare the property as follows.

### C#

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

2. Create a **ViewModel** class with `SelectedItems`, which are initialized with data objects in constructor.

### C#

```
public class ViewModel : INotifyPropertyChanged
```

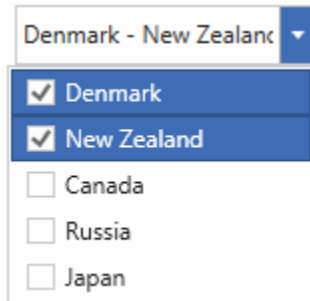
```
{
private ObservableCollection<object> _items;
private ObservableCollection<object> selectedItems;
public ObservableCollection<object> SelectedItems
{
get { return selectedItems; }
set
{
selectedItems = value;
RaisePropertyChanged("SelectedItems");
}
}
private ObservableCollection<Country> countries;
public ObservableCollection<Country> Countries
{
get { return countries; }
set
{
countries = value;
}
}
public ViewModel()
{
Countries = new ObservableCollection<Country>();
Countries.Add(new Country() { Name = "Denmark" });
Countries.Add(new Country() { Name = "New Zealand" });
Countries.Add(new Country() { Name = "Canada" });
Countries.Add(new Country() { Name = "Russia" });
Countries.Add(new Country() { Name = "Japan" });
_items = new ObservableCollection<object>();
for (int i = 0; i < 2; i++)
{
_items.Add(Countries[i]);
}
SelectedItems = new ObservableCollection<object>(_items);
}
public event PropertyChangedEventHandler PropertyChanged;
public void RaisePropertyChanged(string propertyName)
{
var property = PropertyChanged;
if (property != null)
property(this, new PropertyChangedEventArgs(propertyName));
}
}
```

3. To bind the `ComboBoxAdv` to data, bind the collection created in the previous step to the [ItemsSource](#) property in XAML by setting the `ViewModel` as `DataContext`.

#### XML

```
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid>
```

```
<syncfusion:ComboBoxAdv DisplayMemberPath="Name" SelectedItems="{Binding
SelectedItems, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
AllowMultiSelect="True" Name="comboboxadv" HorizontalAlignment="Center"
Height="30"
VerticalAlignment="Center" Width="150" ItemsSource="{Binding Products}"/>
</Grid>
```



**Note:** [View sample in GitHub](#)

Override selected items programmatically

You can override the selected items programmatically by overriding the [OnItemChecked](#) and [OnItemUnchecked](#) method.

### XML

```
<Window x:Class="ComboBoxExtDemo.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:ComboBoxExtDemo"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<local:ComboBoxExt
Width="250"
Height="24"
AllowMultiSelect="True"
AllowSelectAll="True"
EnableOKCancel="True"
DefaultText="Select continent..."
DisplayMemberPath="Name"
ItemsSource="{Binding Continent}" />
</Grid>
</Window>
```

### C#

```
public class ComboBoxExt : ComboBoxAdv
{
    protected override ObservableCollection<object> OnItemChecked(object
checkedItem, ObservableCollection<object> selectedItems)
    {
        var item = ((FrameworkElement)checkedItem).DataContext;
```

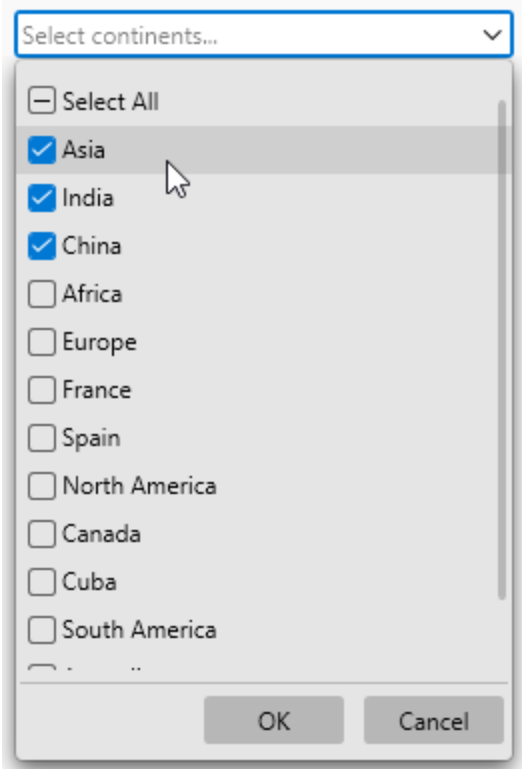
```
if (item == this.Items[0])
AddItem(selectedItems, new int[] { 1, 2 });
if (item == this.Items[4])
AddItem(selectedItems, new int[] { 5, 6 });
return base.OnItemChecked(item, selectedItems);
}

protected override ObservableCollection<object> OnItemUnchecked(object
uncheckedItem, ObservableCollection<object> selectedItems)
{
var item = ((FrameworkElement)uncheckedItem).DataContext;
if (item == this.Items[0])
RemoveItem(selectedItems, new int[] { 1, 2 });
if (item == this.Items[4])
RemoveItem(selectedItems, new int[] { 5, 6 });
return base.OnItemUnchecked(uncheckedItem, selectedItems);
}

public void AddItem(ObservableCollection<object> selectedItems, int[] index)
{
foreach (int i in index)
{
if (!selectedItems.Contains(this.Items[i]))
selectedItems.Add(this.Items[i]);
}
}

public void RemoveItem(ObservableCollection<object> selectedItems, int[]
index)
{
foreach (int i in index)
{
if (selectedItems.Contains(this.Items[i]))
selectedItems.Remove(this.Items[i]);
}
}
}
```

On selecting the Asia, then India and China will be automatically added into selected items.



---

**Note:** [View sample in GitHub](#)

---

### Multiselect edit using tokens

The selected items are now represented by a rounded-polygon shape with a close icon, which can be interacted with by pressing the close button. The [EnableToken](#) property determines whether the ComboBoxAdv's selected items should be displayed as tokens.

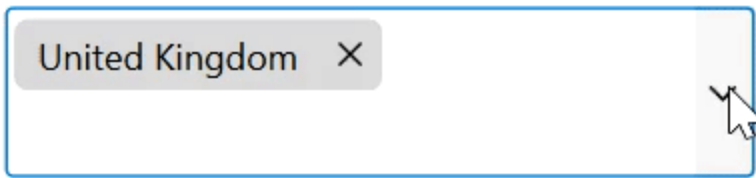
When an item is selected from the dropdown, it is added to the text area as a token. The appropriate item will be removed from the text box when you click the close icon.

### XML

```
<syncfusion:ComboBoxAdv
AllowMultiSelect="true"
IsEditable="true"
EnableToken="true">
</syncfusion:ComboBoxAdv>
```

### C#

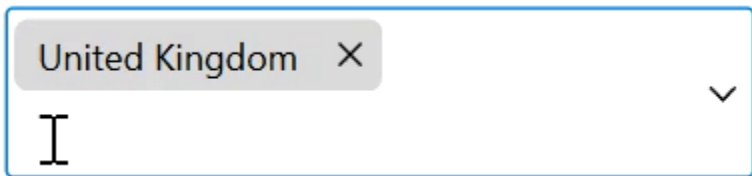
```
ComboBoxAdv comboBox = new ComboBoxAdv();
comboBox.AllowMultiSelect = true;
comboBox.IsEditable = true;
comboBox.EnableToken = true;
```



**Note:** Only the multiselection mode has token support. ComboBox's text area height will be increased or decreased automatically based on the placement of the selected items.

*Editing*

You can type any text in textbox, and it will be added as a token only if it matches the dropdown items.



*Keyboard access*

- Using the **Down Arrow**, **Up Arrow**, **Space**, **Enter** and **Tab** keys item can be selected from the combobox.
- Using the **Enter** and **Tab** keys, typed text will be validated and added as token if it is available in dropdown items.
- Using the **Backspace** key, the last positioned token will be removed from the text area.
- When the **Esc** key is pressed, the drop-down area will be closed if it has been opened already.

**Note:** [View sample in GitHub](#)

*Watermark Support in WPF ComboBox (ComboBoxAdv)*

It displays the default text in the ComboBoxAdv when none of the items is selected in the drop down list.

Property	Description	Type	Data Type	Reference links
DefaultText	It is possible to display the default text.	Dependency Property	String	NA

### Adding DefaultText property to an application

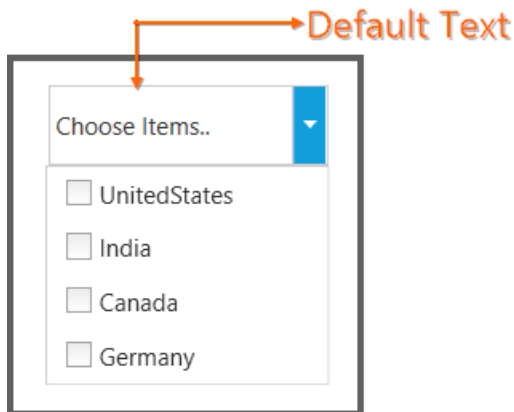
DefaultText property can be added directly to an application in the following way:

#### XML

```
<syncfusion:ComboBoxAdv DefaultText="..Choose  
Items.."></syncfusion:ComboBoxAdv>
```

#### C#

```
ComboBoxAdv comboBox = new ComboBoxAdv();  
comboBox.DefaultText = "..Choose Items..";
```



### Delimiter String Customization

A Delimiter string in a ComboBoxAdv is “A string that can be displayed between the selected items in the ComboBoxAdv”. We can customize this string by using the property called **SelectedValueDelimiter** in the ComboBoxAdv.

Property	Description	Type	Data Type	Reference links
SelectedValueDelimiter	The selected items can be separated by the given string.	Dependency Property	String	NA

### Adding delimiter string customization to an application

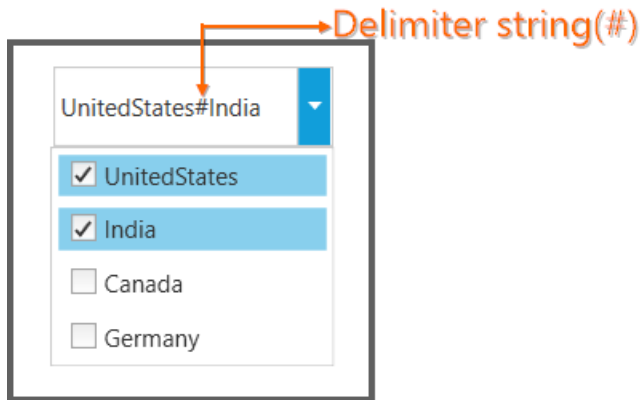
Delimiter string customization can be added directly to an application using the following code snippet:

#### XML

```
<syncfusion:ComboBoxAdv SelectedValueDelimiter="#"></syncfusion:ComboBoxAdv>
```

#### C#

```
ComboBoxAdv comboBox = new ComboBoxAdv();  
comboBox.SelectedValueDelimiter = "#";
```

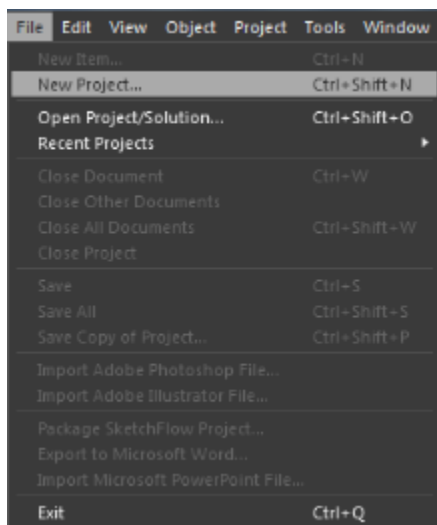


## Blendability in WPF ComboBox (ComboBoxAdv)

### *Creating the ComboBoxAdv control in expression blend*

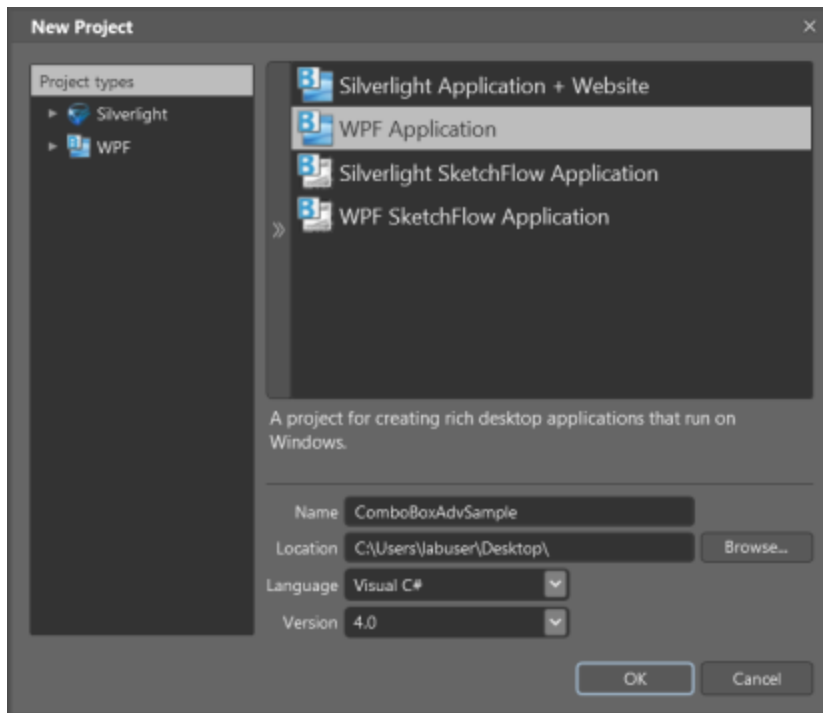
To create ComboBoxAdv instance in Expression Blend:

1. Open Expression Blend.
2. On the File menu, select New Project. The New Project dialog box opens.

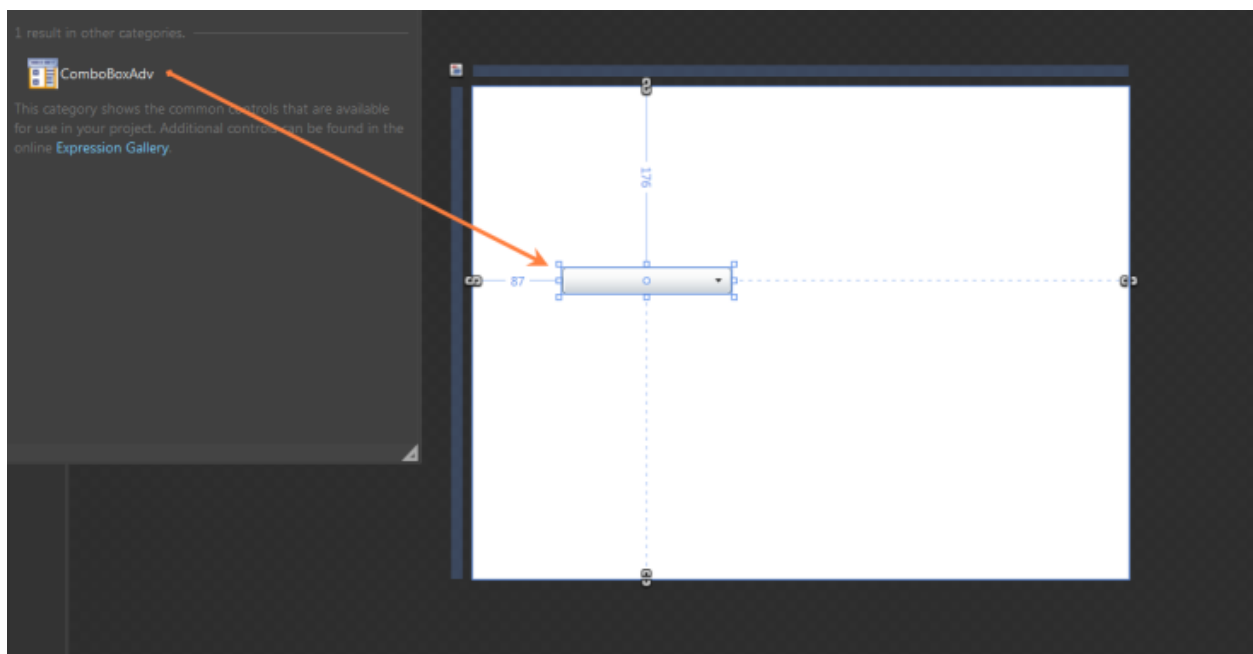


3. In the Project type's pane, select WPF, and then select WPF Application.
4. In the Name field, type the name of the project, and then click OK.





5. On the Window menu, select Assets. The Assets Library dialog box opens.
6. In the Search box, type ComboBoxAdv. The search results are displayed.
7. Drag the ComboBoxAdv control to Design view. An instance of the ComboBoxAdv control is created.



### Appearance

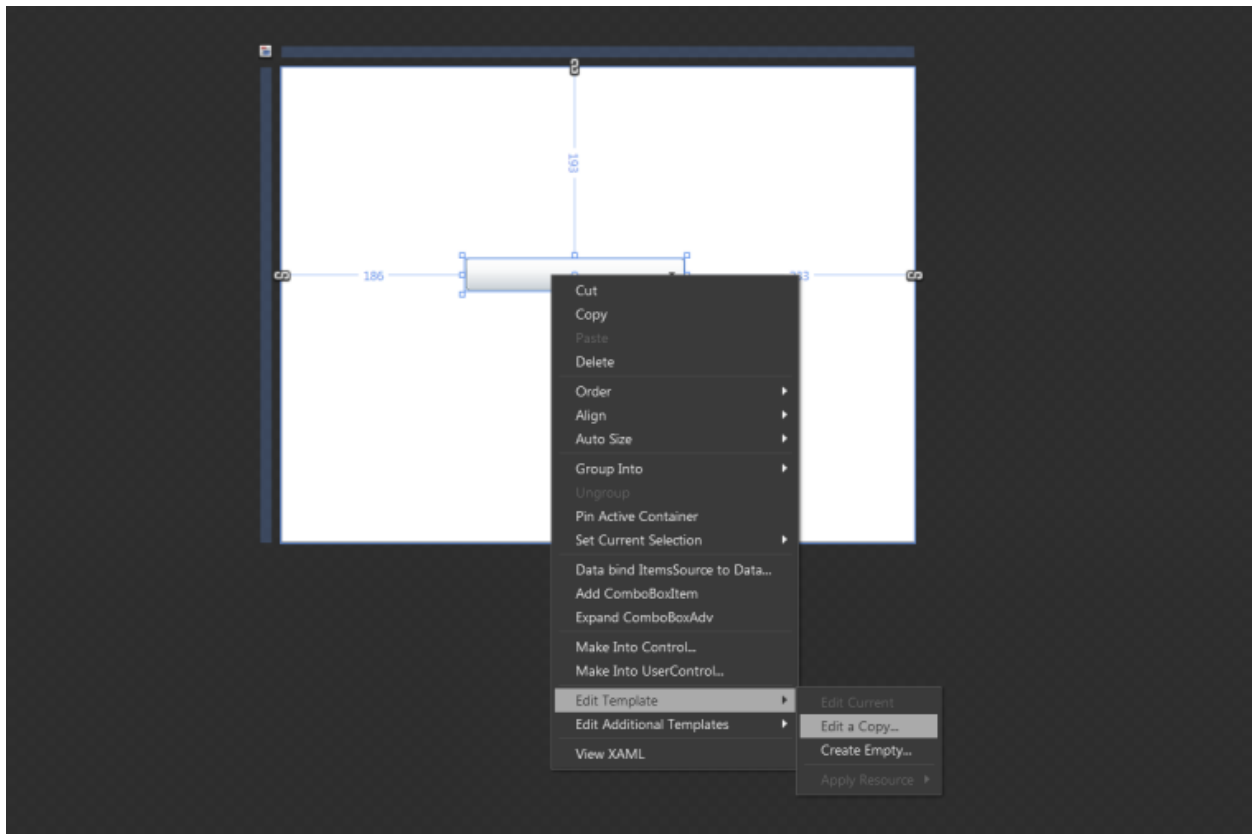
You can customize the appearance of the ComboBoxAdv control by editing the style of the control in Expression Blend or by using the following properties exposed by ComboBoxAdv control:

- Edit style in Expression Blend
- Customizing by using properties

### Blendability

You can edit the style of ComboBoxAdv by using Expression Blend. To edit the ComboBoxAdv control's style in Expression Blend:

1. Drag the ComboBoxAdv control to the Design view. The ComboBoxAdv control will appear as shown in the screen shot displayed below.
2. Right-click the ComboBoxAdv control, select Edit Template, and then select Edit a Copy.



## CurrencyTextBox

### WPF Currency TextBox Overview

The [CurrencyTextBox](#) control restricts text box input to only decimal values and displays it in the currency format with support for data binding, Watermark, Null Value, and culture support. It provides many customization options to enhance its appearance and to suit your applications.

#### Control structure



## Features

The core features of the `CurrencyTextBox` are as follows:

- Provides the ability to control the range of input values by using the `MinValue` and `MaxValue` properties.
- Provides different foreground brushes for positive, negative, and zero values.
- Provides data binding support.
- Provides built-in Visual Styles and themes.
- Provides Watermark support.
- Provides Number Format support.
- Provides Blendability support.
- Provides Null Value support.
- Provides culture support.

## Getting Started with WPF Currency TextBox

This section explains how to create a WPF `CurrencyTextBox` control and its features.

### Assembly deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

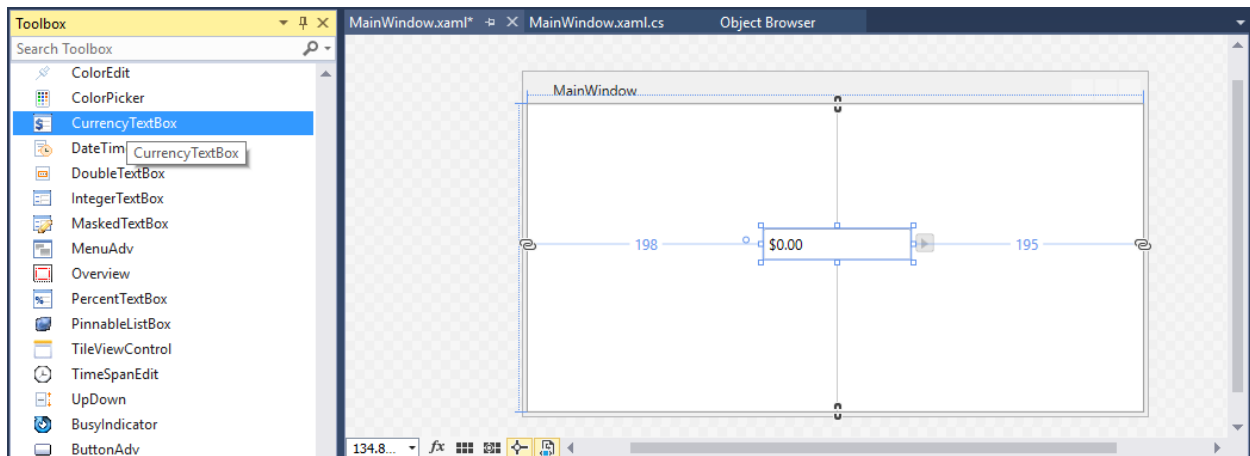
You can find more details about installing the NuGet package in a WPF application in the following link:

[How to install nuget packages](#)

### Adding WPF CurrencyTextBox via designer

You can add the `CurrencyTextBox` control to an application by dragging it from the toolbox to a view of the designer. The following dependent assembly will be added automatically:

- `Syncfusion.Shared.WPF`



### Adding WPF CurrencyTextBox via XAML

To add the `CurrencyTextBox` control manually in XAML, follow these steps:

1. Create a new WPF project in Visual Studio.
2. Add the **Syncfusion.Shared.WPF** assembly references to the project.

3. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> and declare the **CurrencyTextBox** control in XAML page.

### XML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="CurrencyTextBoxSample.MainWindow"
Title="CurrencyTextBox Sample" Height="350" Width="525">
<Grid>
<!--Adding CurrencyTextBox control -->
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="100" Height="25"
VerticalAlignment="Center" HorizontalAlignment="Center"/>
</Grid>
</Window>
```

### Adding WPF CurrencyTextBox via C\#

To add the CurrencyTextBox control manually in C#, follow these steps:

1. Create a new WPF application via Visual Studio.
2. Add the **Syncfusion.Shared.WPF** assembly references to the project.
3. Include the required namespace.

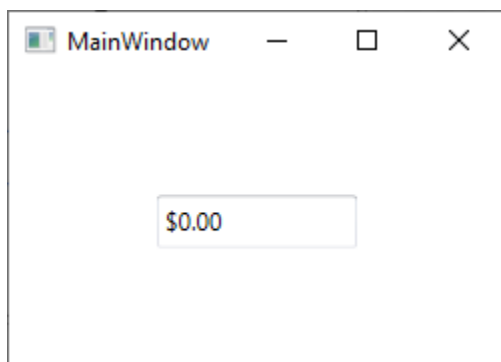
### C#

```
using Syncfusion.Windows.Shared;
```

4. Create an instance of CurrencyTextBox and add it to the window.

### C#

```
//Creating an instance of CurrencyTextBox control
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
// Setting height and width to CurrencyTextBox
currencyTextBox.Height = 25;
currencyTextBox.Width = 100;
//Adding CurrencyTextBox as window content
this.Content = currencyTextBox;
```



### Setting Value

The value of the `CurrencyTextBox` can be set by using the [Value](#) property.

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="100" Height="23" Value="100"/>
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.Width = 100;  
currencyTextBox.Height = 23;  
currencyTextBox.Value = 100;
```



**Note:** Do not use the [Text](#) property to set the value for the `CurrencyTextBox`. Use only the `Value` property.

### Binding Value

Data binding is the method of forming a connection between the application UI and business logic. Data binding can be unidirectional (source -> target or target <- source) or bidirectional (source <-> target). You can bind data to the `CurrencyTextBox` using the `Value` Property.

The following code snippets illustrate the value binding from one `CurrencyTextBox` to another.

#### XML

```
<StackPanel>  
<syncfusion:CurrencyTextBox x:Name="currencyTextBox1" Height="25" Width="100" Value="{Binding MyValue, UpdateSourceTrigger=PropertyChanged}"/>  
<syncfusion:CurrencyTextBox x:Name="currencyTextBox2" Width="100" Height="25" Value="{Binding MyValue, UpdateSourceTrigger=PropertyChanged}" />  
</StackPanel>
```

### ViewModel.cs

#### C#

```
class ViewModel : NotificationObject  
{  
    private double myValue;  
    public double MyValue  
    {  
        get  
        {  
            return myValue;  
        }  
        set  
        {  
            myValue = value;  
            RaisePropertyChanged("MyValue");  
        }  
    }  
}
```

```
}
```




### Value Changed Notification

The **CurrencyTextBox** control can notify the value changes through the [ValueChanged](#) event. You can get old value and new Value from **OldValue** and **NewValue** properties in **ValueChanged** event.

#### XML

```
<syncfusion:CurrencyTextBox ValueChanged="CurrencyTextBox_ValueChanged"/>
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.ValueChanged += new
PropertyChangedCallback(CurrencyTextBox_ValueChanged);
```

You can handle the event as follows:

#### C#

```
private void CurrencyTextBox_ValueChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    // Get old and new value
    var newValue = e.NewValue;
    var oldValue = e.OldValue;
}
```

### Min Max Value Restriction

The **Value** of **CurrencyTextBox** can be restricted within maximum and minimum limit. You can define the minimum and maximum values by setting the [MinValue](#) and [MaxValue](#) properties. It allows the user to enter the value between **MinValue** and **MaxValue**.

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="100" Height="25"
Value="455" MaxValue="999.99" MinValue="-999.99"/>
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 100;
currencyTextBox.Height = 25;
//Setting minimum value
currencyTextBox.MinValue = -999.99;
```

```
//Setting maximum value
currencyTextBox.MaxValue = 999.99;
currencyTextBox.Value = 455;
```



Step Interval to increase or decrease the value

The `CurrencyTextBox` control allows to increase or decrease the value by pressing up and down arrow keys in keyboard or mouse wheel over the control. The [ScrollInterval](#) property is used to specify the increment or decrement intervals. The default value of `ScrollInterval` is 1.

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="100" Height="25"
Value="8"
IsScrollingOnCircle="True" ScrollInterval="4"/>
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 100;
currencyTextBox.Height = 25;
currencyTextBox.MinValue = 0;
currencyTextBox.MaxValue = 100;
currencyTextBox.Value = 8;
currencyTextBox.IsScrollingOnCircle = true;
currencyTextBox.ScrollInterval = 4;
```



Formatting the value

You can customize the number format by using either the [NumberFormat](#) property or the [CurrencyGroupSeparator](#), [CurrencyGroupSizes](#), [CurrencyDecimalDigits](#) and [CurrencyDecimalSeparator](#), [CurrencySymbol](#), [CurrencyNegativePattern](#), and [CurrencyPositivePattern](#) properties of `CurrencyTextBox`.

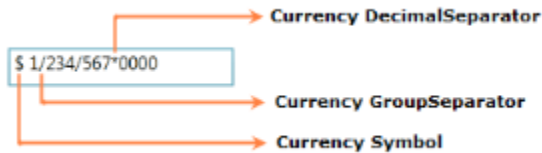
#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Height="25" Width="150"
Value="1234567">
  <syncfusion:CurrencyTextBox.NumberFormat >
    <numberformat:NumberFormatInfo CurrencyGroupSeparator="/"
CurrencyDecimalDigits="4" CurrencyDecimalSeparator="*"
CurrencySymbol="$"/>
  </syncfusion:CurrencyTextBox.NumberFormat>
</syncfusion:CurrencyTextBox>
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 100;
currencyTextBox.Height = 25;
currencyTextBox.Value = 1234567;
```

```
currencyTextBox.NumberFormat = new NumberFormatInfo()
{
    CurrencyGroupSeparator = "/",
    CurrencyDecimalDigits = 4,
    CurrencyDecimalSeparator = "*",
    CurrencySymbol = "$"
};
```



### Setting the Culture

The **CurrencyTextBox** provides support for globalization by using the [Culture](#) property. The **Culture** is used to format the decimal separator and group separator of the **CurrencyTextBox** value based on the respective culture.

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Height="25" Width="100"
    Culture="en-US" Value="1234567"/>
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 100;
currencyTextBox.Height = 25;
currencyTextBox.Value = 1234567;
currencyTextBox.Culture = new System.Globalization.CultureInfo("en-US");
```

**Note:** When you use both **NumberFormat** and **Culture**, the **NumberFormat** will have a higher priority.

### Theme

CurrencyTextBox supports various built-in themes. Refer to the below links to apply themes for the CurrencyTextBox,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

### Changing Currency Value in WPF Currency TextBox

The **CurrencyTextBox** allows the user to change the value using the [Value](#) property.

#### XML



```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Height="25"
Width="150" Value="10"/>
```

### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 150;
currencyTextBox.Height = 25;
currencyTextBox.Value = 10;
```



Data binding is the process of establishing a connection between the application UI and business logic. Data binding can be unidirectional (source -> target or target <- source) or bidirectional (source <-> target). By assigning a value to the `Value` property by binding, you can change the `CurrencyTextBox` value.

The following code snippets illustrate the value binding from one `CurrencyTextBox` to another.

### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox1" Value="{Binding
MyValue, UpdateSourceTrigger=PropertyChanged}" Height="25" Width="100"/>
<syncfusion:CurrencyTextBox x:Name="currencyTextBox2" Value="{Binding
MyValue, UpdateSourceTrigger=PropertyChanged}" Width="100" Height="25" />
```

ViewModel.cs

### C#

```
class ViewModel : NotificationObject
{
    private double myValue;
    public double MyValue
    {
        get
        {
            return myValue;
        }
        set
        {
            myValue = value;
            RaisePropertyChanged("MyValue");
        }
    }
}
```



### Change currency value by pasting the clipboard's text

By default, **CurrencyTextBox** simply replaces the whole value by copied value with the current number format. If you want to replace or insert the copied value on specific place, use the [PasteMode](#) property value as **Advanced**. The default value of **PasteMode** property is **Default**.

The following table explains the pasting behaviour in **Advanced** paste mode,

S.No	Action	Pasting behaviour in Advanced paste mode
1	When the whole value is selected	It simply replaces the whole value by copied value with the current number format.
2	When the cursor is at some position and the copied value does not contain a number decimal separator	It inserts the copied value into the current cursor position.
3	When the cursor is at some position and the copied value contains a number decimal separator	It won't perform pasting operation.
4	When the cursor is at some position and the control value is 0 or null	It simply replaces the whole value by copied value with the current number format.
5	When a part of the number is selected	If the selected value contains a number decimal separator, then copied value must contain number decimal separator. Otherwise, it won't perform pasting operation. If the selected text does not contain a number decimal separator, then copied value must not contain number decimal separator. Otherwise, it won't perform pasting operation.

### XML

```
<syncfusion:CurrencyTextBox PasteMode="Advanced"
Value="12345.67"
Name="currencyTextBox"/>
```

### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.PasteMode = PasteMode.Advanced;
currencyTextBox.Value = 12345.67;
```

Pasting "8.92"



### Show UpDown Button

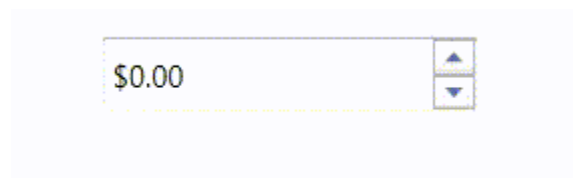
You can increment or decrement the currency value of `CurrencyTextBox` by setting the `ShowSpinButton` property value as `true`. Click `UpButton` to increment or `DownButton` to decrement the currency value. The default value of `ShowSpinButton` property is `false`.

#### XML

```
<syncfusion:CurrencyTextBox Height="30" Width="150" ShowSpinButton="True" />
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.ShowSpinButton = true;
```



### Value Changed Event

The `CurrencyTextBox` control can notify changes in value through the [ValueChanged](#) event. In `ValueChanged` event, you can get old value and new value from the `OldValue` and `NewValue` properties.

#### XML

```
<syncfusion:CurrencyTextBox ValueChanged="CurrencyTextBox_ValueChanged"/>
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.ValueChanged += new  
PropertyChangedCallback(CurrencyTextBox_ValueChanged);
```

You can handle the event as follows:

#### C#

```
private void CurrencyTextBox_ValueChanged(DependencyObject d,  
DependencyPropertyChangedEventArgs e)  
{  
    // Get old and new value  
    var newValue = e.NewValue;  
    var oldValue = e.OldValue;  
}
```

### Setting the Null value

By default, the `CurrencyTextBox` control will display zero value when the `Value` is set to `null`. You can use the [NullValue](#) and [UseNullOption](#) properties to show the null or any other value instead of zero.

The default value of the `NullValue` property is `null`, you can reset this to any other currency value. It will display only on setting the `UseNullOption` property is set to `true`.

**NullValue = Null**

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Height="25"
Width="100" UseNullOption="True" NullValue="{x:Null}"/>
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 100;
currencyTextBox.Height = 25;
currencyTextBox.NullValue = null;
currencyTextBox.UseNullOption = true;
```



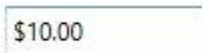
**NullValue = 10**

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Height="25"
Width="100" UseNullOption="True" NullValue="10"/>
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 100;
currencyTextBox.Height = 25;
currencyTextBox.NullValue = 10;
currencyTextBox.UseNullOption = true;
```



#### Setting Watermark Text

We can display certain information within the control by using the [WaterMarkText](#) property. `WaterMarkText` is shown when the [WatermarkTextIsVisible](#) property is `true` and the value is `null` or empty, the control is not in focus and the `UseNullOption` property is `true`.

#### *Setting the WatermarkText Foreground*

The `CurrencyTextBox` allows you to set the desired brush as a foreground for `WaterMarkText` using [WaterMarkTextForeground](#) property. The default color of `WaterMarkTextForeground` is `Black`.

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="100"
Height="25" UseNullOption="True" WatermarkText="Type here"
WatermarkTextIsVisible="True" WatermarkTextForeground="Red"/>
```

**C#**

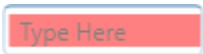
```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 100;
currencyTextBox.Height = 25;
currencyTextBox.UseNullOption = true;
currencyTextBox.WatermarkText = "Type Here";
currencyTextBox.WatermarkTextIsVisible = true;
currencyTextBox.WatermarkTextForeground = Brushes.Red;
```

*Setting Watermark Template*

You can customize the Visual appearance of the `WatermarkText` by using the [WatermarkTemplate](#) property.

**XML**

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="100" Height="25"
WatermarkText="Type Here" CornerRadius="3"
WatermarkTextIsVisible="True" WatermarkOpacity="0.5"
UseNullOption="True">
  <syncfusion:CurrencyTextBox.WatermarkTemplate >
    <DataTemplate>
      <Border Background="Red">
        <TextBlock Text="{Binding}" VerticalAlignment="Center" Margin="5,0,0,0"/>
      </Border>
    </DataTemplate>
  </syncfusion:CurrencyTextBox.WatermarkTemplate>
</syncfusion:CurrencyTextBox>
```



**Note:** The `UseNullOption` property must be enabled if you want to see `NullValue` or `WaterMarkText` in `CurrencyTextBox` control.

**Note:** If both `NullValue` and `WaterMarkText` are specified, you will only see `NullValue` but not `WaterMarkText`.

*Step Interval in WPF Currency TextBox*

The [CurrencyTextBox](#) control allows you to increase or decrease the value by pressing up-arrow and down-arrow keys in keyboard or mouse wheel over the control. The [ScrollInterval](#) property is used to specify the increment or decrement interval. The default value of `ScrollInterval` is 1.

For example, the `ScrollInterval` value is set to 4. So, that the `CurrencyTextBox` control [Value](#) increases or decreases by 4 while pressing Up arrow or Down arrow keys and Mouse wheel scrolling up or down.

*Change Value on Up, Down arrow key*

The `CurrencyTextBox` control allows you to increase or decrease the `Value` of `CurrencyTextBox` based on the `ScrollInterval` by pressing the up arrow and down arrow keys on the keyboard.


**XML**

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="150"
```

```
Height="25" Value ="34" ScrollInterval="3"/>
```

**C#**

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 150;
currencyTextBox.Height = 25;
currencyTextBox.Value = 34;
currencyTextBox.ScrollInterval = 3;
```


**Change Value on Mouse Wheel**

The **CurrencyTextBox** allows you to increase or decrease the **Value** based on the **ScrollInterval** by the Mouse scrolling over the control When the [IsScrollingOnCircle](#) property is **true**. The default value of **IsScrollingOnCircle** property is **true**.

**XML**

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="150" Height="25"
Value ="10"
IsScrollingOnCircle="True" ScrollInterval="2"/>
```

**C#**

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 150;
currencyTextBox.Height = 25;
currencyTextBox.Value = 10;
currencyTextBox.IsScrollingOnCircle = true;
currencyTextBox.ScrollInterval = 2;
```


**Change Value on Click and Drag**

The **CurrencyTextBox** allows you to increase or decrease the value based on the **ScrollInterval** by clicking and dragging the mouse when the [EnableExtendedScrolling](#) property is **true**. **CurrencyTextBox** value increases when the cursor moves to the right or the top of the screen and decreases when you click and drag the mouse to the left or the bottom of the screen. Before that, the control should be in an unfocused state.

**XML**

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="120" Height="25"
Value ="10"
ScrollInterval="5" EnableExtendedScrolling="True"/>
```

**C#**

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 120;
```

```
currencyTextBox.Height = 25;  
currencyTextBox.Value = 10;  
currencyTextBox.ScrollInterval = 5;  
currencyTextBox.EnableExtendedScrolling = true;
```



### Allow or restrict selection on focus

CurrencyTextBox allows you to automatically select text by setting [TextSelectionOnFocus](#) property to true and when the control got focus. If you want to restrict the selection on when control got focus, use the [TextSelectionOnFocus](#) property value as false. The default value of the [TextSelectionOnFocus](#) property is true.

### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox"  
TextSelectionOnFocus="False"/>
```

### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.TextSelectionOnFocus = true;
```

### TextSelectionOnFocus = "False"



### TextSelectionOnFocus = "True"



## Restriction or Validation in WPF Currency TextBox

This section explains how to validate or restrict the [CurrencyTextBox](#) control value.

### Restrict the value within minimum and maximum value

The [Value](#) of the [CurrencyTextBox](#) can be restricted within the maximum and minimum limits. Once the value has reached the maximum or minimum value, the value does not exceed the limit. We can change the maximum and minimum limits by using the [MinValue](#) property and [MaxValue](#) property.

You can choose when to validate the maximum and minimum limits while changing the values by using the [MinValidation](#) and [MaxValidation](#) properties.

- **OnKeyPress** — When setting the [MaxValidation](#) or [MinValidation](#) to [OnKeyPress](#), the value in the [CurrencyTextBox](#) will be validated shortly after pressing a key. So, it is not possible to provide any invalid input at all and the value does not exceed the maximum and minimum limits.

- **OnLostFocus** - When setting **MaxValidation** or **MinValidation** to **OnLostFocus**, the value in the **CurrencyTextBox** is validated, when the **CurrencyTextBox** loses the focus. That is, the **CurrencyTextBox** will accept any value, validation will only take place after the **CurrencyTextBox** has lost its keyboard focus. After validation, when the value of the **CurrencyTextBox** is greater than the **MaxValue** or less than the **MinValue**, the value will be automatically set to **MaxValue** or **MinValue**.
- **MaxValueOnExceedMaxDigit** - When you give input greater than specified maximum limit, **MaxValueOnExceedMaxDigit** property will decide either it should retain the old value or reset to maximum limit that is specified. For example, if **MaxValue** is set to 100 and you are trying to input 200. Value will be changed to 100 when **MaxValueOnExceedMaxDigit** is **true**. When **MaxValueOnExceedMaxDigit** is **false**, 20 will be retained and last entered 0 will be ignored.

---

**Note:** **MaxValueOnExceedMinDigit** property will be enabled only when the **MaxValidation** is set to **OnKeyPress**.

---

- **MinValueOnExceedMinDigit** - When you give input less than specified minimum limit, **MinValueOnExceedMinDigit** property will decide either it should retain the old value or reset to minimum limit that is specified. For example, if **MinValue** is set to 200 and the **Value** is 205 and you are trying to change the value to 20. Value will be changed to 200 when **MinValueOnExceedMinDigit** is **true**. When **MinValueOnExceedMinDigit** is **false**, Old value 205 will be retained.

---

**Note:** **MinValueOnExceedMinDigit** will be enabled only when the **MinValidation** is set to **OnKeyPress**.

---

### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="150"
MaxValue="100" MinValue="10"
MinValueOnExceedMinDigit="True" MaxValueOnExceedMaxDigit="True"
MinValidation="OnKeyPress" MaxValidation="OnLostFocus"/>
```

### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 150;
currencyTextBox.Height = 25;
currencyTextBox.MinValue = 10;
currencyTextBox.MaxValue = 100;
currencyTextBox.MinValidation = MinValidation.OnKeyPress;
currencyTextBox.MaxValidation = MaxValidation.OnLostFocus;
currencyTextBox.MinValueOnExceedMinDigit = true;
currencyTextBox.MaxValueOnExceedMaxDigit = true;
```

**MinValidation** is set to **OnKeyPress**, it cannot let to enter a value less than the **MinValue**. If try to enter a value less than the **MinValue**, then the **MinValue** will set to the **Value** property because **MinValueOnExceedMinDigit** is set to **true**.





MaxValidation is set to OnLostFocus, so the MaxValidation will be performed only in the lost focus.



### Restrict number of decimal digits

You can format the decimal digits in the [CurrencyTextBox](#) control using [CurrencyDecimalDigits](#) property. You can also restrict the decimal digits of the text within minimum and maximum limit in

[CurrencyTextBox](#) control using [MinimumCurrencyDecimalDigits](#) and [MaximumCurrencyDecimalDigits](#) properties. The default value of [MinimumCurrencyDecimalDigits](#), [MaximumCurrencyDecimalDigits](#) and [CurrencyDecimalDigits](#) properties is -1.

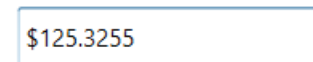
**Note:** If the value of [MinimumCurrencyDecimalDigits](#) property is greater than the value of [MaximumCurrencyDecimalDigits](#) property, the text of [CurrencyTextBox](#) will be updated based on value of [MinimumCurrencyDecimalDigits](#) property.

### XML

```
<syncfusion:CurrencyTextBox Value="125.32545" HorizontalAlignment="Center"
VerticalAlignment="Center"
MaximumCurrencyDecimalDigits="4"
MinimumCurrencyDecimalDigits="1" />
```

### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Value = 125.32545;
currencyTextBox.MaximumCurrencyDecimalDigits = 4;
currencyTextBox.MinimumCurrencyDecimalDigits = 1;
```



When the value of [MinimumCurrencyDecimalDigits](#), [MaximumCurrencyDecimalDigits](#) and [CurrencyDecimalDigits](#) properties are specified, [CurrencyDecimalDigits](#) property takes high precedence and updates the text of [CurrencyTextBox](#) property.


### XML

```
<syncfusion:CurrencyTextBox Value="125.32545" HorizontalAlignment="Center"
VerticalAlignment="Center"
MaximumCurrencyDecimalDigits="4"
MinimumCurrencyDecimalDigits="1"
CurrencyDecimalDigits="3"
/>
```

### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Value = 125.32545;
```

```
currencyTextBox.MaximumCurrencyDecimalDigits = 4;  
currencyTextBox.MinimumCurrencyDecimalDigits = 1;  
currencyTextBox.CurrencyDecimalDigits = 3;
```



### Read only mode

The CurrencyTextBox cannot allow the user input, edits when [IsReadOnly](#) property is sets to true. The user can still select text and display the cursor on the CurrencyTextBox by setting the [IsReadOnlyCaretVisible](#) property to true. However, value can be changed programmatically in readonly mode.

### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" IsReadOnly="True"  
Value="10" IsReadOnlyCaretVisible="True"/>
```

### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.Value = 10;  
currencyTextBox.IsReadOnly = true;  
currencyTextBox.IsReadOnlyCaretVisible = true;
```



## Culture and Formatting in WPF Currency TextBox

Value of CurrencyTextBox can be formatted in following ways:

- Culture
- NumberFormatInfo
- Dedicated properties (CurrencyGroupSeparator, CurrencyGroupSizes, CurrencyDecimalDigits, CurrencyDecimalSeparator)

### Culture based formatting

The [CurrencyTextBox](#) provides support for globalization by using the [Culture](#) property. The Culture property is used to format the decimal separator and group separator of the CurrencyTextBox value based on the respective culture.

### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Height="25" Width="150"  
Culture="fr-FR" Value="1234567"/>
```

### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.Width = 100;
```

```
currencyTextBox.Height = 25;
currencyTextBox.Value = 1234567;
currencyTextBox.Culture = new CultureInfo("fr-FR");
```

By default the US culture uses “,” as the **CurrencyGroupSeparator** "\$" as **CurrencySymbol** and "." as the **CurrencyDecimalSeparator** where as the France culture uses “Space” as the **CurrencyGroupSeparator**, "€" as **CurrencySymbol** and "," as the **CurrencyDecimalSeparator**.

### Default Culture

\$1,234,567.00

### France Culture

1 234 567,00 €

### NumberFormatInfo based formatting

The number formatting of **CurrencyTextBox** can be customized by setting [NumberFormat](#) property.

### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Height="25" Width="150"
Value="1234567">
<syncfusion:CurrencyTextBox.NumberFormat >
<numberformat:NumberFormatInfo CurrencyGroupSeparator="/"
CurrencyDecimalDigits="4"
CurrencyDecimalSeparator="*"
CurrencySymbol="$"/>
</syncfusion:CurrencyTextBox.NumberFormat>
</syncfusion:CurrencyTextBox>
```

### C#

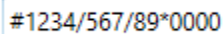
```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 150;
currencyTextBox.Height = 25;
currencyTextBox.Value = 1234567;
currencyTextBox.NumberFormat = new NumberFormatInfo()
{
    CurrencyGroupSeparator = "/",
    CurrencyDecimalDigits = 4,
    CurrencyDecimalSeparator = "*",
    CurrencySymbol = "$"
};
```



The following code illustrate how to set currency group size by using the **NumberFormat** property.

**C#**

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 150;
currencyTextBox.Height = 25;
currencyTextBox.Value = 123456789;
currencyTextBox.NumberFormat = new NumberFormatInfo()
{
    CurrencySymbol = "&",
    CurrencyDecimalDigits = 4,
    CurrencyGroupSeparator = "/",
    CurrencyDecimalSeparator = "*",
    // Adding the currency group size via NumberFormat property.
    CurrencyGroupSizes = new int[] { 2, 3, 4 }
};
```



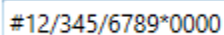
## Formatting with dedicated properties

The number formatting of `CurrencyTextBox` can also be customized by setting the [CurrencyGroupSeparator](#), [CurrencyGroupSizes](#), [CurrencyDecimalDigits](#) and [CurrencyDecimalSeparator](#), [CurrencySymbol](#), [CurrencyNegativePattern](#), and [CurrencyPositivePattern](#) properties of `CurrencyTextBox`. You can show the group separator by enable the [GroupSeperatorEnabled](#) property to `true`.

The following code illustrate how to format using the `CurrencyDecimalSeparator`, `CurrencyDecimalDigits`, `CurrencyGroupSeparator`, `CurrencyGroupSizes` property of the `CurrencyTextBox`.

**C#**

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 150;
currencyTextBox.Height = 25;
currencyTextBox.Value = 123456789;
currencyTextBox.CurrencySymbol = "#";
currencyTextBox.CurrencyDecimalDigits = 4;
currencyTextBox.GroupSeperatorEnabled = true;
currencyTextBox.CurrencyGroupSeparator = "/";
currencyTextBox.CurrencyDecimalSeparator = "*";
// Adding the currency group size via CurrencyGroupSizes property.
currencyTextBox.CurrencyGroupSizes = new Int32Collection() { 4, 3, 2 };
```



**Note:** When you use both the `NumberFormat` and the dedicated properties (`CurrencyGroupSeparator`, `CurrencySymbol`, `CurrencyDecimalDigits`, `CurrencyDecimalSeparator` and `CurrencyGroupSizes`) to format the value of `CurrencyTextBox`, the `CurrencyGroupSeparator` and `CurrencyGroupSizes` properties have higher priority.

**Note:** When you use both `NumberFormat` and `Culture`, the `NumberFormat` will have a higher priority.

*Positive Value Pattern*

You can use the [CurrencyPositivePattern](#) property to customize the location of the currency symbol and the positive currency values. In the table below, "\$" denotes the symbol of the currency, and "n" denotes the number.

**CurrencyPositivePattern table**

Value	Associated Pattern
0	\$n
1	n\$
2	\$ n
3	n \$

CurrencyPositivePattern = 0

\$12.34

CurrencyPositivePattern = 1

12.34\$

CurrencyPositivePattern = 2

\$ 12.34

CurrencyPositivePattern = 3

12.34 \$

**XML**

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Height="25" Width="150"
Value="1234" CurrencyPositivePattern="3"/>
```

**C#**

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 150;
currencyTextBox.Height = 25;
currencyTextBox.Value = 1234;
currencyTextBox.CurrencyPositivePattern = 3;
```

1,234.00 \$

*Negative Value Pattern*

You can use the [CurrencyNegativePattern](#) property to customize the location of the currency symbol and the negative currency values. In the table below, "\$" denotes the symbol of the currency, and "n" denotes the number.

**CurrencyNegativePattern table**

Value	Associated Pattern
0	(\$n)

1	-\$n
2	\$-n
3	\$n-
4	(n\$)
5	-n\$
6	n-\$
7	n\$-
8	-n \$
9	-\$ n
10	n \$-
11	\$ n-
12	\$ -n
13	n- \$
14	(\$ n)
15	(n \$)

CurrencyNegativePattern = 0

(\$12.34)

CurrencyNegativePattern = 1

-\$12.34

CurrencyNegativePattern = 2

\$-12.34

CurrencyNegativePattern = 3

\$12.34-

CurrencyNegativePattern = 4

(12.34\$)

CurrencyNegativePattern = 5

-12.34\$

CurrencyNegativePattern = 6

12.34-\$

CurrencyNegativePattern = 7

12.34\$-

CurrencyNegativePattern = 8

-12.34 \$

CurrencyNegativePattern = 9

-\$ 12.34

CurrencyNegativePattern = 10

12.34 \$-

CurrencyNegativePattern = 11

\$ 12.34-

CurrencyNegativePattern = 12

\$ -12.34

CurrencyNegativePattern = 13

12.34- \$

CurrencyNegativePattern = 14

(\$ 12.34)

CurrencyNegativePattern = 15

(12.34 \$)

## XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Height="25" Width="150"
Value="-1234" CurrencyNegativePattern="0"/>
```

## C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.Width = 150;
currencyTextBox.Height = 25;
currencyTextBox.Value = -1234;
currencyTextBox.CurrencyNegativePattern = 0;
```

(\$1,234.00)

## Appearance in WPF Currency TextBox

This section deals with the appearance of **CurrencyTextBox** control and contains the following topics.

### Setting the Foreground

The **CurrencyTextBox** control **Foreground** can be modified based on the value of the control. The following are the foreground for **CurrencyTextBox** control.

### Foreground for Positive Value

We can change a positive color for the value of `CurrencyTextBox` by setting the `PositiveForeground` property and it will be applied when the `Value` is positive. The default color of `PositiveForeground` is Black.

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Value="10" Width="100" Height="25" PositiveForeground="Blue" />
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.Width = 100;  
currencyTextBox.Height = 25;  
currencyTextBox.Value = 10;  
currencyTextBox.PositiveForeground = Brushes.Blue;
```



### Foreground for Negative Value

We can change a negative color for the value of `CurrencyTextBox` by setting the `NegativeForeground` property and it will be applied when the `ApplyNegativeForeground` property is `true` and the `Value` is negative. The default color of `NegativeForeground` is Red.

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Value="-10" Width="100" Height="25" NegativeForeground="SpringGreen" ApplyNegativeForeground="True" />
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.Width = 100;  
currencyTextBox.Height = 25;  
currencyTextBox.Value = -10;  
currencyTextBox.ApplyNegativeForeground = true;  
currencyTextBox.NegativeForeground = Brushes.SpringGreen;
```



### Foreground for Zero Value

We can change a zero color for the value of `CurrencyTextBox` by setting the `ZeroColor` property and it will be applied when the `ApplyZeroColor` property is `true` and the `Value` is zero.

The default color of `ZeroColor` is Green.

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Value="0" Width="100" Height="25" />
```



```
ApplyZeroColor="True" ZeroColor="DarkGoldenrod"/>
```

### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.Width = 100;  
currencyTextBox.Height = 25;  
currencyTextBox.Value = 0;  
currencyTextBox.ApplyZeroColor = true;  
currencyTextBox.ZeroColor = Brushes.DarkGoldenrod;
```



### Setting the Background

**CurrencyTextBox** allows different brushes to fill the control. The [Background](#) property can be used to modify the control background color. The default color of **Background** is **White**.

### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="100"  
Height="25" Value="80" Background="Cyan"/>
```

### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.Width = 100;  
currencyTextBox.Height = 25;  
currencyTextBox.Background = Brushes.Cyan;
```



### Setting the Corner Radius

**Corner Radius** indicates the degree to which the corners of the border can be rounded. To create curved borders for the **CurrencyTextBox**, use [CornerRadius](#) property. The default value of **CornerRadius** property is 1.

### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="100" Height="25"  
CornerRadius="5"/>
```

### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.Width = 100;  
currencyTextBox.Height = 25;  
currencyTextBox.CornerRadius = new CornerRadius(5);
```



### Apply Background for Selection

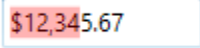
CurrencyTextBox allows different brushes to highlight the selected text by setting the [SelectionBrush](#) and [SelectionOpacity](#) properties. The SelectionOpacity property specifies the opacity of the SelectionBrush.

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="100" Height="25" SelectionBrush="Red" SelectionOpacity="0.5"/>
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.Width = 100;  
currencyTextBox.Height = 25;  
currencyTextBox.SelectionBrush = Brushes.Red;  
currencyTextBox.SelectionOpacity = 0.3;
```



### Align Value

CurrencyTextBox allows to display the value from right or center or left side by setting the [TextAlignment](#) property to Right or Left or Center. The Default value of TextAlignment is Left.

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="100" Height="25" TextAlignment="Center"/>
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.Width = 100;  
currencyTextBox.Height = 25;  
currencyTextBox.TextAlignment = TextAlignment.Center;
```



### Setting ToolTip

CurrencyTextBox provides support for ToolTip to display certain information when the mouse hovers on the CurrencyTextBox. You can customize the tooltip information by setting the [ToolTip](#) property.

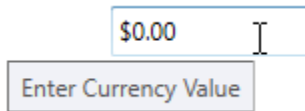
#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" Width="100" Height="25" ToolTip="Enter Currency Value"/>
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.Width = 100;
```

```
currencyTextBox.Height = 25;
currencyTextBox.ToolTip = "Enter Currency Value";
```



### Theme

CurrencyTextBox supports various built-in themes. Refer to the below links to apply themes for the CurrencyTextBox,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

### Range Adorner in WPF Currency TextBox

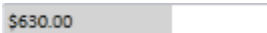
**Value** of [CurrencyTextBox](#) can be visually indicated like a progress bar using range-adorner feature, this feature is disabled by default. You can show the adorner over CurrencyTextBox control by setting [EnableRangeAdorner](#) property to **true**. Default value of [EnableRangeAdorner](#) is **false**. The adorner layer can be filled in the control area on the basis of the minimum and maximum values with considering the given value. Range Adorner is not displayed when a [MinValue](#) or [MaxValue](#) property is not set.

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" MinValue="0"
MaxValue="100" Value="630" EnableRangeAdorner="True" />
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();
currencyTextBox.MinValue = 0;
currencyTextBox.MaxValue = 1000;
currencyTextBox.Value = 630;
currencyTextBox.EnableRangeAdorner = true;
```



### Changing background of range-adorner

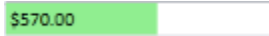
You can change the background color of the range adorner using [RangeAdornerBackground](#) property.

#### XML

```
<syncfusion:CurrencyTextBox x:Name="currencyTextBox" MinValue="0"
MaxValue="1000" Value="570" EnableRangeAdorner="True"
RangeAdornerBackground="LightGreen" />
```

#### C#

```
CurrencyTextBox currencyTextBox = new CurrencyTextBox();  
currencyTextBox.MinValue = 0;  
currencyTextBox.MaxValue = 100;  
currencyTextBox.Value = 570;  
currencyTextBox.EnableRangeAdorner = true;  
currencyTextBox.RangeAdornerBackground = Brushes.LightGreen;
```



## SfDataGrid

### WPF DataGrid (SfDataGrid) Overview

The SfDataGrid (DataGrid) control for WPF is used to display collection of data in rows and columns. The SfDataGrid control includes editing, exporting and data shaping features (sorting, grouping, filtering and etc) that allows the end users to easily manage the data.

Following are the key features of SfDataGrid control

- **Data binding** – Supports to bind different types of data sources.
- **Selection** – Support for row and also cell selection.
- **Editing** – Interactive support to edit with different column types.
- **Columns** – Support for various column types including unbound columns.
- **Sorting** – Interactive support to sort the data in SfDataGrid.
- **Grouping** – Interactive support to group the data in SfDataGrid.
- **Summaries** – Extensive support to show concise information about the individual data columns or groups of rows.
- **Filtering** – Interactive support for filtering data as like in Excel.
- **Data validation** – Support to validate the data on errors.
- **Data virtualization** – Support for different modes of data virtualization such as paging, incremental loading.
- **Master-Detail View** – Support to display relational data using hierarchies.
- **Printing and Exporting** – Support to print and also export the data to Excel, PDF.
- **Styling** – Extensive support for customizing styles of cells and rows in SfDataGrid.
- **Stacked Headers** – Extensive support to show multiple headers called stacked headers.
- **Unbound rows** – Support to display unbound rows.
- **Touch support** – Complete support for resizing, drag-drop column, sorting, filtering, grouping and etc in touch devices.

### Choose between different Grid's

Syncfusion WPF suite comes up with following different Grid's namely,

1. [GridControl](#)
2. [GridTreeControl](#) and [SfTreeGrid](#)
3. [GridDataControl](#) and [SfDataGrid](#)

### GridControl

[GridControl](#) is designed based on the cell-oriented architecture that does not make any assumptions about the structure of the data and provides support for Excel-like features. This control can be used

when the data is populated in on demand using virtualization or its internal storage. It is not possible to bind the data source directly.

#### *GridTreeControl and SfTreeGrid*

The [GridTreeControl](#) and [SfTreeGrid](#) control is a data-oriented control that displays self-relational data in a tree structure user interface like multicolumn treeview. The data can be loaded on demand. You can move items between parent nodes by using the built-in row drag-and-drop functionality. Its rich feature set includes editing with different column types, selection, and node selection with check boxes, sorting, and filtering.

#### *GridDataControl and SfDataGrid*

The [GridDataControl](#) and [SfDataGrid](#) controls are used to display collection of data in rows and columns. These controls support the data binding directly.

1. **SfDataGrid** - SfDataGrid is designed based on the WPF template-based architecture that provides support to customize the Grid easily and fully supports binding.
2. **GridDataControl** - GridDataControl is designed based on the cell-oriented architecture of the Windows Forms Grid control that provides more control over cells and supports for Excel-like features.

Both SfDataGrid and GridDataControl almost have the same set of features. But SfDataGrid control offers rich set of features over GridDataControl. When you want cell level customization and excel-like features you can use GridDataControl. When you want performance, customization features like styles & template features specific to WPF, you can use SfDataGrid control. Comparatively, the performance of SfDataGrid control is better than the GridDataControl.

---

**Note:** GridDataControl is marked as classic control. In future, new features & enhancements will be added only in SfDataGrid. It recommended to use SfDataGrid.

---

You can see the list of some of the specific API difference between [GridDataControl](#) and [SfDataGrid](#) as follows:

GridDataControl	SfDataGrid	Description
AutoPopulateColumns	AutoGenerateColumns	To Generate the Columns
AllowEdit	AllowEditing	To Edit the GridCells
AllowGroup	AllowGrouping	To Group the Column header
AllowResizeColumns	AllowResizingColumns	To Resize the columns
AllowDragColumns	AllowDraggingColumns	To Drag the columns
AllowSort	AllowSorting	To Sort the columns
ShowFilters	AllowFiltering	To Show the Filters in a Grid
VisibleColumn	Column	To Specify the GridColumns
AllowExcelLikeResizing	AllowResizingHiddenColumns	To Allow Resizing on the Hidden Columns

ShowHoveringBackground	AllowRowHoverHighlighting	To show the Hovers for Highlighting rows.
EnableTriStateSorting	AllowTristateSorting	To allow Tri State Sorting
ExpandGroupsWhenGrouped	AutoExpandGroups	To Allow Auto Expand groups
AutoPopulateRelations	AutoGenerateRelations	To set an item source for child Grid.
SortColumns	SortColumnDescriptions	The column sorting based on the column descriptions that are given in a particular column.
ShowSortNumber	ShowSortNumbers	To Show the Sorted Numbers when sorting.
HideColumnsWhenGrouped	ShowColumnsWhenGrouped	To Show the Columns when grouped.
HighlightSelectionBackground	RowSelectionBrush	To give a color for RowSelection.
ListBoxSelectionMode	SelectionMode	To Specify the Selection Mode for a selection.
DefaultHeaderRowHight	HeaderRowHeight	Specifies the Header Row and Height.
SummaryRows	GroupSummaryRows	To Group the Summary row based on the Summary given in that row.
FrozenRows	FrozenRowsCount	The number of rows is freeze from Top.
FooterRows	FooterRowsCount	The number of rows is freeze from bottom.
FrozenColumns	FrozenColumnCount	The number of columns is freeze from left.
FooterColumns	FooterColumnCount	The number of columns is freeze from right.

You can see the list of rich set of features in **SfDataGrid** over **GridDataControl** as follows:

Rich set of features in **SfDataGrid** over **GridDataControl**.

Feature	Description
AutoRowHeight	SfDataGrid enables fitting the height of the row based on its content on demand for all columns or certain columns by using <a href="#">AutoRowHeight</a> .
CellTemplate Support for all columns	SfDataGrid provides support for <a href="#">CellTemplate</a> . It is used to customize columns in display mode that present cells with DataTemplate.

IEditableObjectSupport	SfDataGrid supports to roll back the changes when you press <b>Esc</b> Key by implementing IEditableObject interface. For more information about IEditableObject refer <a href="#">IEditableObject</a> .
Incremental Loading	<a href="#">IncrementalLoading</a> allows you to load a subset of data to a SfDataGrid sequentially. It provides support for fast and fluid scrolling and loading a huge set of data.
Printing	The Printing feature in SfDataGrid is more flexible and customizable. It also provides a good performance when compared to GridDataControl. For more information about CustomPrinting, click <a href="#">here</a> . The PrintManager of SfDataGrid is designed to support different orientations, sizes, margins etc.
Exporting To Excel	SfDataGrid control provides support to Export data to Excel and returns an ExcelEngine that contains the exported workbook. SfDataGrid Exporting is faster than GridDataControl. It exports the content only to the excel sheet. It takes very less time to export the huge amounts of data. To know more about exporting in SfDataGrid, click <a href="#">here</a> .
Exporting To PDF	SfDataGrid control provides support for exporting the data into a PDF file. You can decide what are the contents is need to export in PDF file. You can export Grouping, Filtering, Summaries and DetailsView, StackedHeaders in to PDF file. To get more information about <a href="#">ExportToPdf</a> .
FilterPopupPerformance	When you have a large amount of data, it takes time to load the Filter popup. Though, you are having lots of data, you can get better performance while loading Filter popup, by setting CanGenerateUniqueltems to False. To know more about FilterPopupPerformance, click <a href="#">here</a> .
DataVirtualization	<a href="#">DataVirtualization</a> is a term that achieves Virtualization for the actual data objects that are bound to the DataGrid. For small collection of basic data objects, the memory consumption is not significant. However for large collections, the memory consumption becomes very significant.

### Getting Started with WPF DataGrid (SfDataGrid)

This section provides a quick overview for working with the [WPF DataGrid](#) (SfDataGrid) for WPF. Walk through the entire process of creating a real world of this control.

To get start quickly with WPF DataGrid, you can check on this video:

<style>#WPFDataGridVideoTutorial{width : 90% !important; height: 400px !important }</style>

<iframe id='WPFDataGridVideoTutorial'  
src='https://www.youtube.com/embed/1t4nXyDA9l0'></iframe>

### Assembly deployment

The following list of assemblies needs to be added as reference to use SfDataGrid control in any application,

Required assemblies	Description
Syncfusion.Data.WPF	Syncfusion.Data.WPF assembly contains fundamental and base classes for <a href="#">CollectionViewAdv</a> which is responsible for data processing operations handled in SfDataGrid.
Syncfusion.SfGrid.WPF	Syncfusion.SfGrid.WPF assembly contains classes that handles all UI operations of SfDataGrid. SfDataGrid control present Syncfusion.UI.Xaml.Grid namespace. This namespace also added in <a href="http://schemas.syncfusion.com/wpf">http://schemas.syncfusion.com/wpf</a> Syncfusion WPF schema.
Syncfusion.Shared.WPF	Syncfusion.Shared.WPF contains various editor controls (such as IntegerTextBox, DoubleTextBox and etc) which are used in SfDataGrid.

In order to use export to excel and export to PDF functionalities of SfDataGrid control, add the reference to following assemblies,

Optional Assemblies	Description
Syncfusion.SfGridConverter.WPF	Syncfusion.SfGridConverter.WPF contains static extension classes for exporting SfDataGrid to excel and PDF in Syncfusion.UI.Xaml.Grid.Converter namespace.
Syncfusion.XlsIO.Base	Syncfusion.XlsIO.Base contains fundamental and base classes for creating and manipulating excel files.
Syncfusion.Pdf.Base	Syncfusion.Pdf.Base contains fundamental and base classes for creating PDF.

### Creating simple application with SfDataGrid

In this walk through, you will create WPF application that contains SfDataGrid control.

1. [Creating project](#)
2. [Adding control via Designer](#)
3. [Adding control manually in XAML](#)
4. [Adding control manually in C#](#)
5. [Creating Data Model for application](#)
6. [Binding to Data](#)
7. [Defining Columns](#)
8. [Selection](#)
9. [Sorting, Grouping, and Filtering](#)
10. [Editing](#)

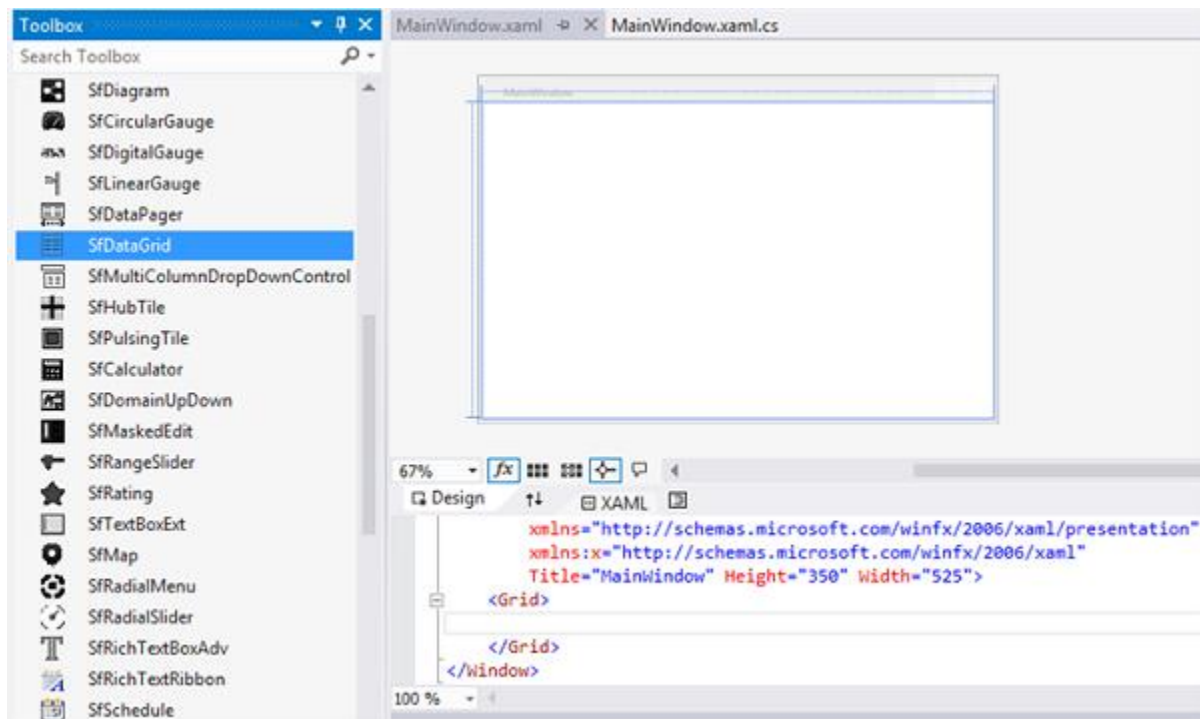
### Creating the project

Create new WPF Project in Visual Studio to display SfDataGrid with data objects.



*Adding control via Designer*

SfDataGrid control can be added to the application by dragging it from Toolbox and dropping it in Designer view. The required assembly references will be added automatically.

*Adding control manually in XAML*

In order to add control manually in XAML, do the below steps,

1. Add the below required assembly references to the project,
  - Syncfusion.Data.WPF
  - Syncfusion.SfGrid.WPF
  - Syncfusion.Shared.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> or SfDataGrid control namespace **Syncfusion.UI.Xaml.Grid** in XAML page.
3. Declare SfDataGrid control in XAML page.

**XML**

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="WpfApplication1.MainWindow"
Title="MainWindow" Height="350" Width="525">
<Grid>
<syncfusion:SfDataGrid x:Name="dataGrid"/>
</Grid>
</Window>
```

*Adding control manually in C#*

In order to add control manually in C#, do the below steps,

1. Add the below required assembly references to the project,
  - Syncfusion.Data.WPF
  - Syncfusion.SfGrid.WPF
  - Syncfusion.Shared.WPF
2. Import SfDataGrid namespace **Syncfusion.UI.Xaml.Grid** .
3. Create SfDataGrid control instance and add it to the Page.

**C#**

```
using Syncfusion.UI.Xaml.Grid;
namespace WpfApplication1
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SfDataGrid dataGrid = new SfDataGrid();
            Root_Grid.Children.Add(dataGrid);
        }
    }
}
```

*Creating Data Model for sample application*

SfDataGrid is a data-bound control. So before create binding to the control, you must create data model for Application.

1. Create data object class named **OrderInfo** and declare properties as shown below,

**C#**

```
public class OrderInfo
{
    int orderID;
    string customerId;
    string country;
    string customerName;
    string shippingCity;
    public int OrderID
    {
        get { return orderID; }
        set { orderID = value; }
    }
    public string CustomerID
    {
        get { return customerId; }
        set { customerId = value; }
    }
    public string CustomerName
    {
        get { return customerName; }
        set { customerName = value; }
    }
    public string Country
```

```
{
get { return country; }
set { country = value; }
}
public string ShipCity
{
get { return shippingCity; }
set { shippingCity = value; }
}
public OrderInfo(int orderId, string customerName, string country, string
customerId, string shipCity)
{
this.OrderID = orderId;
this.CustomerName = customerName;
this.Country = country;
this.CustomerID = customerId;
this.ShipCity = shipCity;
}
}
```

**Note:** If you want your data object (OrderInfo class) to automatically reflect property changes, then the object must implement **INotifyPropertyChanged** interface.

2. Create a **ViewModel** class with Orders property and Orders property is initialized with several data objects in constructor.

### C#

```
public class ViewModel
{
private ObservableCollection<OrderInfo> _orders;
public ObservableCollection<OrderInfo> Orders
{
get { return _orders; }
set { _orders = value; }
}
public ViewModel()
{
_orders = new ObservableCollection<OrderInfo>();
this.GenerateOrders();
}
private void GenerateOrders()
{
_orders.Add(new OrderInfo(1001, "Maria Anders", "Germany", "ALFKI",
"Berlin"));
_orders.Add(new OrderInfo(1002, "Ana Trujillo", "Mexico", "ANATR", "Mexico
D.F.));
_orders.Add(new OrderInfo(1003, "Antonio Moreno", "Mexico", "ANTON", "Mexico
D.F.));
_orders.Add(new OrderInfo(1004, "Thomas Hardy", "UK", "AROUT", "London"));
_orders.Add(new OrderInfo(1005, "Christina Berglund", "Sweden", "BERGS",
"Lula"));
_orders.Add(new OrderInfo(1006, "Hanna Moos", "Germany", "BLAUS",
"Mannheim"));
_orders.Add(new OrderInfo(1007, "Frederique Citeaux", "France", "BLONP",
"Strasbourg"));
}
```

```
_orders.Add(new OrderInfo(1008, "Martin Sommer", "Spain", "BOLID",  
"Madrid"));  
_orders.Add(new OrderInfo(1009, "Laurence Lebihan", "France", "BONAP",  
"Marseille"));  
_orders.Add(new OrderInfo(1010, "Elizabeth Lincoln", "Canada", "BOTTM",  
"Tsawassen"));  
}  
}
```

### *Binding to Data*

To bind the SfDataGrid to data, set the [SfDataGrid.ItemsSource](#) property to an IEnumerable implementation. Each row in SfDataGrid is bound to an object in data source and each column in SfDataGrid bound to a property in data object.

Bind the collection created in previous step to `SfDataGrid.ItemsSource` property in XAML by setting ViewModel as `DataContext`.

### **XAML**

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"  
x:Class="WpfApplication1.MainWindow"  
xmlns:local="clr-namespace:WpfApplication1"  
Title="MainWindow" Height="350" Width="525">  
  <Window.DataContext>  
    <local:ViewModel/>  
  </Window.DataContext>  
  <Grid x:Name="Root_Grid">  
    <syncfusion:SfDataGrid x:Name="dataGrid" ItemsSource="{Binding Orders}" />  
  </Grid>  
</Window>
```

### **C#**

```
ViewModel viewModel = new ViewModel();  
dataGrid.ItemsSource = viewModel.Orders;
```

Now, run the application and you can expect the see the below output,

OrderID	CustomerID	CustomerName	Country
1001	ALFKI	Maria Anders	Germany
1002	ANATR	Ana Trujilo	Mexico
1003	ANTON	Antonio Moreno	Mexico
1004	AROUT	Thomas Hardy	UK
1005	BERGS	Christina Berglund	Sweden
1006	BLAUS	Hanna Moos	Germany
1007	BLONP	Frédérique Citeaux	France
1008	BOLID	Martin Sommer	Spain
1009	BONAP	Laurence Lebihan	France
1010	BOTTM	Elizabeth Lincoln	Canada

### Defining Columns

By default, the SfDataGrid control generates the columns automatically when value assigned to [SfDataGrid.ItemsSource](#) property. The type of the column generated depends on the type of data in the column and the attribute of the property the column bound with.

The following table lists the column types and its constraints for auto column generation.

Generated Column Type	Data Type / Attribute
GridTextColumn	Property of type String and any other type apart from below specified cases.
GridNumericColumn	Property of type Int or Double.
GridCurrencyColumn	Property with Currency DataType attribute. [DataType(DataType.Currency)].
GridMaskColumn	Property with PhoneNumber DataType attribute. [DataType(DataType.PhoneNumber)].
GridDateTimeColumn	Property of type DateTime.
GridCheckBoxColumn	Property of type Bool.

When columns are auto-generated, you can handle the [SfDataGrid.AutoGeneratingColumn](#) event to customize or cancel the columns before they are added to the SfDataGrid.

You can prevent the automatic column generation by setting [SfDataGrid.AutoGenerateColumns](#) property to `false`. When [SfDataGrid.AutoGenerateColumns](#) property is `false`, you have to define the columns to be displayed as below,

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
AutoGenerateColumns="False">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="CustomerID"/>
<syncfusion:GridTextColumn MappingName="CustomerName"/>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

**C#**

```
SfDataGrid dataGrid = new SfDataGrid();
dataGrid.AutoGenerateColumns = false;
dataGrid.Columns.Add(new GridTextColumn() { MappingName = "CustomerID" });
dataGrid.Columns.Add(new GridTextColumn() { MappingName = "CustomerName" });
```

Below is the list of column types provided in SfDataGrid.

Column Type	Comments
GridTextColumn	Represents SfDataGrid column that hosts textual content in its cells.
GridNumericColumn	Represents SfDataGrid column that hosts <a href="#">DoubleTextBox</a> controls in its cells which is used to format and display Numeric values.
GridCurrencyColumn	Represents SfDataGrid column that hosts <a href="#">CurrencyTextBox</a> controls in its cells which is used to display numeric values with currency format.
GridPercentColumn	Represents SfDataGrid column that hosts <a href="#">PercentTextBox</a> controls in its cells which is used to display numeric values with percent format.
GridMaskColumn	Represents SfDataGrid column that hosts MaskedTextBox controls in its cells which is used to display textual content by applying Mask.
GridTimeSpanColumn	Represents SfDataGrid column that hosts <a href="#">TimeSpanEdit</a> controls in its cells which is used to display format and display TimeSpan values.
GridDateTimeColumn	Represents SfDataGrid column that hosts <a href="#">DateTimeEdit</a> controls in its cells which is used to display and format DateTime values.
GridComboBoxColumn	Represents SfDataGrid column that hosts ComboBox controls in its cells.
GridCheckBoxColumn	Represents SfDataGrid column that hosts CheckBox controls in its cells.
GridImageColumn	Represents SfDataGrid column that hosts Image controls in its cells.
GridHyperlinkColumn	Represents SfDataGrid column that hosts Hyperlink controls in its cells.
GridTemplateColumn	Represents SfDataGrid column that hosts template-specified content in its cells

GridUnboundColumn	Represents SfDataGrid column that hosts textual or template-specified content which are not actually bound with data object of row.
GridMultiColumnDropDownList	Represents SfDataGrid column that hosts <a href="#">SfMultiColumnDropDownControl</a> in its cells.

### Selection

By default, the entire row is selected when a user clicks a cell in a SfDataGrid. You can set the [SfDataGrid.SelectionMode](#) property to specify whether a user can select single row or cell, or multiple rows or cells. Set the [SfDataGrid.SelectionUnit](#) property to specify whether rows can be selected, or cells can be selected.

When [SelectionUnit](#) is **Row**, you can get information about the rows that are selected using [SfDataGrid.SelectedItem](#) and [SfDataGrid.SelectedItems](#) properties.

When [SfDataGrid.SelectionUnit](#) is **Cell**, you can get information about the cells that are selected by calling [SfDataGrid.GetSelectedCells](#) method.

You can handle the selection operations with the help of [SfDataGrid.SelectionChanging](#) and [SfDataGrid.SelectionChanged](#) events of SfDataGrid.

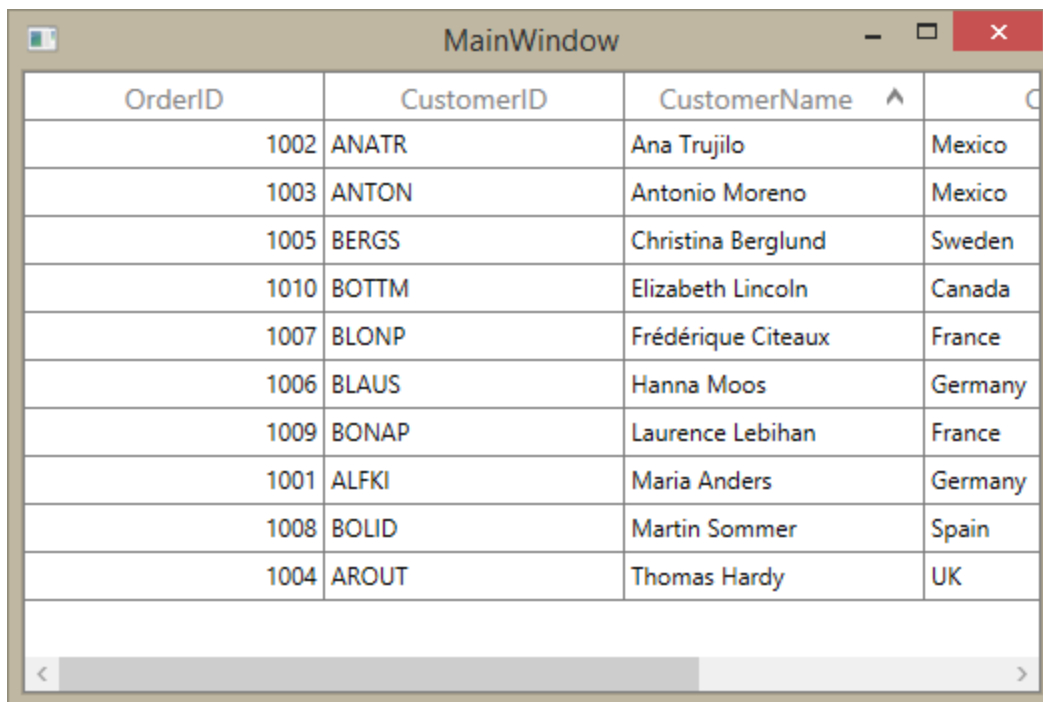
### Sorting, Grouping, and Filtering

#### Sorting

By default, you can sort columns in a SfDataGrid by clicking the column header. You can configure the sorting by setting [SfDataGrid.SortColumnDescriptions](#) property as below,

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}" >
<syncfusion:SfDataGrid.SortColumnDescriptions>
<syncfusion:SortColumnDescription ColumnName="CustomerName"/>
</syncfusion:SfDataGrid.SortColumnDescriptions>
</syncfusion:SfDataGrid>
```



OrderID	CustomerID	CustomerName ^	Country
1002	ANATR	Ana Trujilo	Mexico
1003	ANTON	Antonio Moreno	Mexico
1005	BERGS	Christina Berglund	Sweden
1010	BOTTM	Elizabeth Lincoln	Canada
1007	BLONP	Frédérique Citeaux	France
1006	BLAUS	Hanna Moos	Germany
1009	BONAP	Laurence Lebihan	France
1001	ALFKI	Maria Anders	Germany
1008	BOLID	Martin Sommer	Spain
1004	AROUT	Thomas Hardy	UK

You can customize sorting by handling the [SfDataGrid.SortColumnsChanging](#) and [SfDataGrid.SortColumnsChanged](#) events. To cancel the default sort, set the `Cancel` property to `true` in `SfDataGrid.SortColumnsChanging` event.

### C#

```
this.dataGrid.SortColumnsChanging += dataGrid_SortColumnsChanging;
void dataGrid_SortColumnsChanging(object sender,
GridSortColumnsChangingEventArgs e)
{
    if (e.AddedItems[0].ColumnName == "CustomerName")
        e.Cancel = true;
}
```

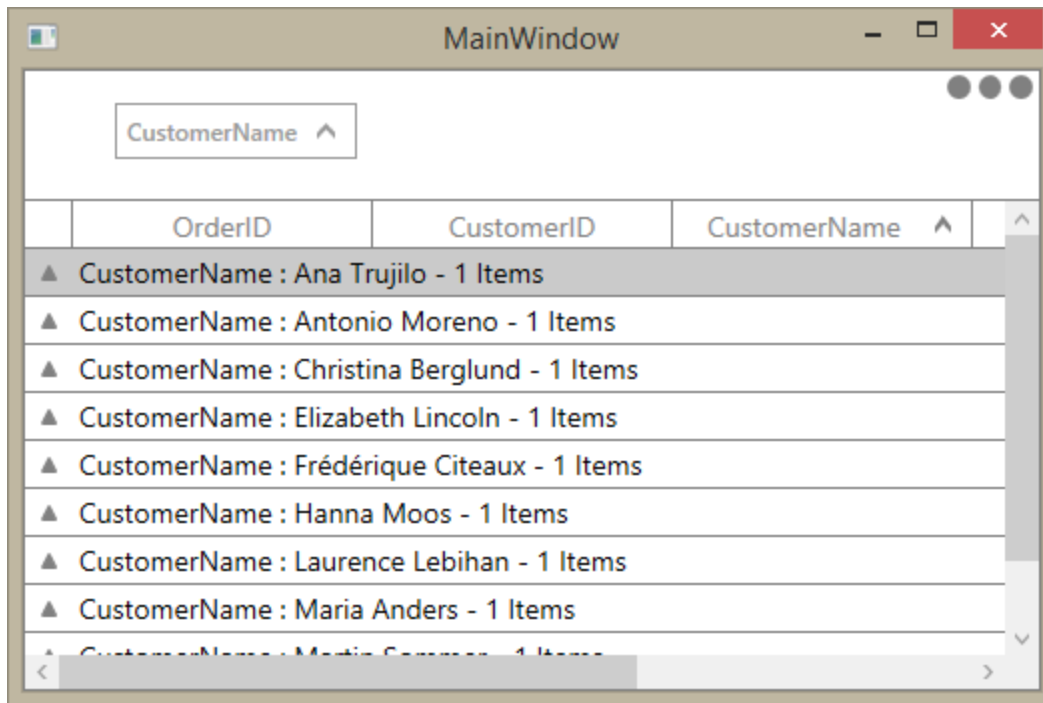
### Grouping

Grouping can be enabled by setting [SfDataGrid.ShowGroupDropArea](#) property, where you can group by dragging the column header and dropping it in the `GroupDropArea` over the column headers. You can configure the grouping by setting [SfDataGrid.GroupColumnDescriptions](#) property as below,

### XML

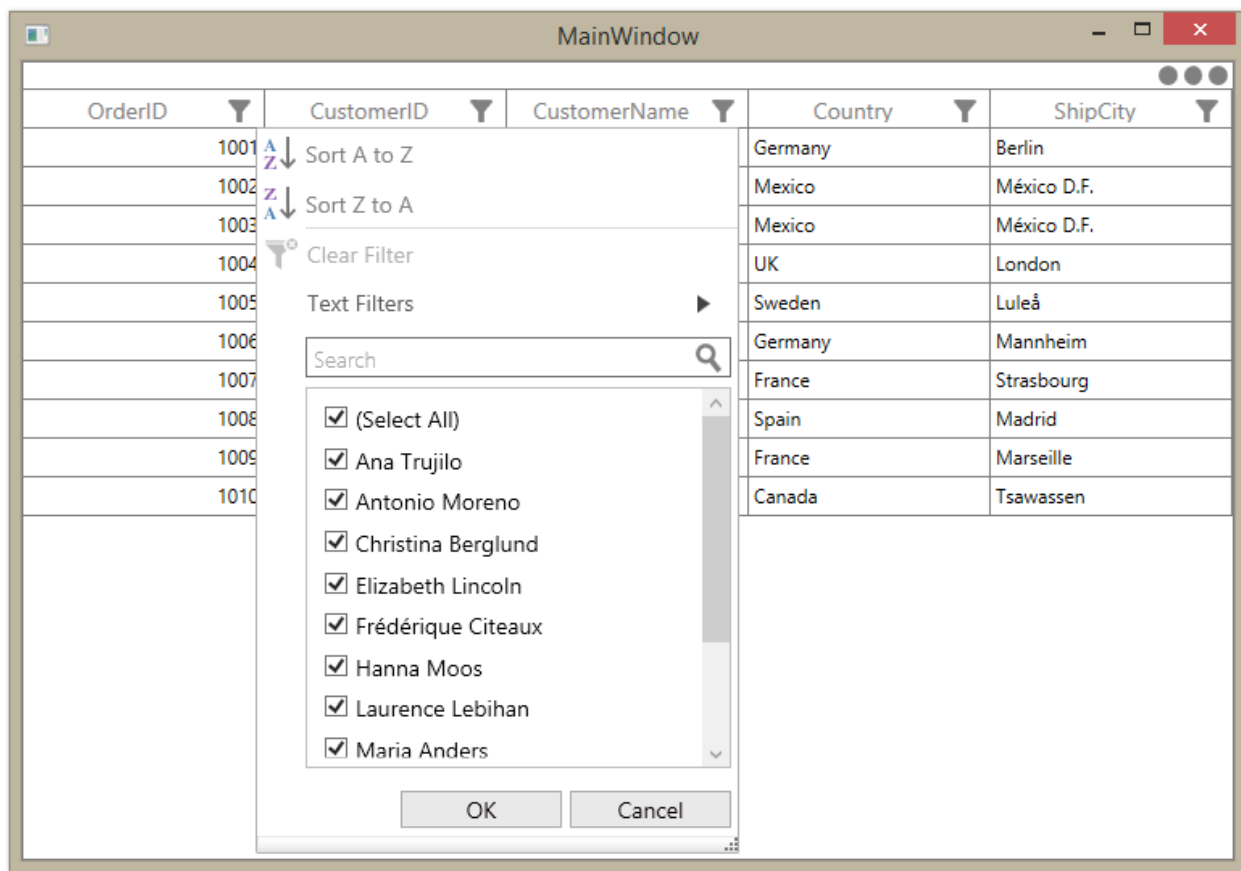
```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}" >
<syncfusion:SfDataGrid.GroupColumnDescriptions>
<syncfusion:GroupColumnDescription ColumnName="CustomerName"/>
</syncfusion:SfDataGrid.GroupColumnDescriptions>
</syncfusion:SfDataGrid>
```





### Filtering

Filtering can be enabled by setting [SfDataGrid.AllowFiltering](#) property to `true`, where you can open advanced filter UI by clicking the Filter icon in column header and filter the SfDataGrid. You can customize the filtering operations by handling [SfDataGrid.FilterChanging](#) and [SfDataGrid.FilterChanged](#) events.



### Editing

Editing can be enabled by setting [SfDataGrid.AllowEditing](#) property to `true`. Set [SfDataGrid.AllowDeleting](#) property to specify whether user can delete rows by pressing Delete key.

Set [SfDataGrid.AddNewRowPosition](#) property to enable additional row either Top or Bottom of SfDataGrid, where user can enter new items into the blank row. Adding new row adds an item to the [SfDataGrid.ItemsSource](#).

You can customize the editing operations by handling [SfDataGrid.CurrentCellBeginEdit](#) and [SfDataGrid.CurrentCellEndEdit](#) events.

### Theme

SfDataGrid supports various built-in themes. Refer to the below links to apply themes for the SfDataGrid,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

Order Details			Shipment Details		Delivery Status
Order ID	Customer ID	Customer Name	Ship Country	Shipping Date	
Ship Country : Argentina - 20 Items					
10004	11004	Franklin	Argentina	2/13/2014	<input checked="" type="checkbox"/>
10007	11007	Martin	Argentina	11/23/2012	<input type="checkbox"/>
10014	11014	George	Argentina	10/11/2017	<input checked="" type="checkbox"/>
10032	11032	Martin	Argentina	4/16/2016	<input checked="" type="checkbox"/>
10036	11036	Zachary	Argentina	2/9/2011	<input checked="" type="checkbox"/>
10037	11037	James	Argentina	6/22/2002	<input type="checkbox"/>
10038	11038	Thomas	Argentina	3/4/2006	<input checked="" type="checkbox"/>
10042	11042	Rutherford	Argentina	2/13/2008	<input checked="" type="checkbox"/>
10047	11047	James	Argentina	6/12/2006	<input type="checkbox"/>
10048	11048	Abraham	Argentina	4/21/2004	<input checked="" type="checkbox"/>
10050	11050	Grover	Argentina	7/15/2018	<input checked="" type="checkbox"/>
10051	11051	Zachary	Argentina	3/23/2017	<input type="checkbox"/>
10060	11060	Rutherford	Argentina	4/3/2007	<input checked="" type="checkbox"/>
Total Customers : 300					

## Data Binding in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) (SfDataGrid) control is designed to display the bounded data in a tabular format. The data binding can be achieved by assigning the data sources to [SfDataGrid.ItemsSource](#) property.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}" />
```

If the data source implements [INotifyCollectionChanged](#) interface, then SfDataGrid control will automatically refresh the UI when item is added, removed or while list cleared.

When you add, remove item in [ObservableCollection](#), SfDataGrid automatically refresh the UI as [ObservableCollection](#) implements [INotifyCollectionChanged](#). But when you do the same in [List](#), SfDataGrid will not refresh the UI automatically.

### Binding with IEnumerable

WPF DataGrid (SfDataGrid) control supports to bind any collection that implements the [IEnumerable](#) interface. All the data operations such as sorting, grouping, filtering, summaries are supported when you are binding collection derived from [IEnumerable](#).

### Binding with DataTable

WPF DataGrid (SfDataGrid) control supports to bind the [DataTable](#). SfDataGrid control automatically refresh the UI when you are binding DataTable as ItemsSource when rows are added, removed or cleared.

Below are the limitations when binding DataTable as ItemsSource to SfDataGrid.

- [GridUnboundColumn.Expression](#) is not supported. You can achieve the expression support when binding DataTable using [DataColumn](#) of DataTable by setting [DataColumn.Expression](#) property.
- [AddNewRow](#) is not support when filtering is enabled.
- Advanced Filtering does not support Case Sensitive filtering.
- [SfDataGrid.View.Filter](#) is not supported.
- Custom sorting is not supported.
- Filtering with [TimeSpan](#) values is not supported.
- [SfDataGrid.LiveDataUpdateMode](#) is not supported.
- Filtering with sub second components in [DateTime](#) values is not supported.

### *Apply DataView.RowFilter on initial loading*

When DataTable is bounded as SfDataGrid.ItemsSource, the [DataView.RowFilter](#) will not be applied to the data on initial loading by default. The [DataView.RowFilter](#) can be applied to the data on initial loading by enabling [CanUseViewFilter](#) property.

### *Binding with dynamic data object*

WPF DataGrid (SfDataGrid) control supports to bind [dynamic data object](#). Below are the limitations when you are binding dynamic data object,

1. SfDataGrid doesn't support [LiveDataUpdateMode](#) - [AllowDataShaping](#) and [AllowSummaryUpdate](#).
2. In WinRT, UI won't get refreshed when you are changing the property value. This is limitation in WinRT platform.

All the data operations (sorting, grouping, filtering and etc.) are supported when you are binding dynamic data object. If the data operations are not working as expected, set [SfDataGrid.IsDynamicItemsSource](#) property as [true](#) .

### *Binding Complex properties*

SfDataGrid control provides support to bind complex property to its columns. To bind the complex property to [GridColumn](#), set the complex property path to [MappingName](#).

### **XML**

```
<syncfusion:SfDataGrid AutoGenerateColumns="False" ItemsSource="{Binding Orders}">
  <syncfusion:SfDataGrid.Columns>
    <syncfusion:GridTextColumn MappingName="OrderID" />
    <syncfusion:GridTextColumn MappingName="Customer.CustomerID" />
    <syncfusion:GridTextColumn MappingName="ShipCity" />
  </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

All the data operations (sorting, grouping, filtering and etc.) are supported when you are binding complex property. If the data operations are not working as expected, set [GridColumn.UseBindingValue](#) as [true](#) to make it work.

### *Limitations when binding complex property*

- SfDataGrid doesn't support [LiveDataUpdateMode](#) - [AllowDataShaping](#) and [AllowSummaryUpdate](#).

### *Binding Indexer properties*

SfDataGrid control provides support to bind an indexer property to its columns. To bind an indexer property to [GridColumn](#), set the indexer property path to [MappingName](#).

### **XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid" ItemsSource="{Binding Students}"
  AutoGenerateColumns="False">
  <syncfusion:SfDataGrid.Columns>
    <syncfusion:GridTextColumn MappingName="RollNo" />
  </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

```
<syncfusion:GridTextColumn MappingName="Name"/>
<syncfusion:GridTextColumn MappingName="Marks[0]"/>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#

```
this.dataGrid.Columns.Add(new GridTextColumn() {MappingName="Marks[0]"});
```

All the data operations (sorting, grouping, filtering and etc.) are supported when you are binding indexer property. If the data operations are not working as expected, set [GridColumn.UseBindingValue](#) as `true` to make it work.

#### *Limitations when binding indexer property*

- SfDataGrid doesn't support [LiveDataUpdateMode](#) - `AllowDataShaping` and `AllowSummaryUpdate`.

#### Defining source data type

Based on type of data item bound to SfDataGrid, the data operations and column auto generation are carried out. You can specify the type of underlying data item explicitly for doing data operation by setting [SfDataGrid.SourceType](#) property.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid" ItemsSource="{Binding Orders}"
Source="{x:Type local:OrderInfo}"/>
```

### C#

```
dataGrid.SourceType = typeof(OrderInfo);
```

For example, when you set [SourceType](#) property, the columns are generated based on members of `SourceType` regardless of data items actual type. When your `ItemsSource` have different types derived from same type, you can set `SourceType` as base type for all different types.

#### Events

##### *ItemsSourceChanged*

[SfDataGrid.ItemsSourceChanged](#) event occurs when the data source is changed by using `ItemsSource` property.

This event receives two arguments namely `sender` that handles `SfDataGrid` and [GridItemsSourceChangedEventArgs](#) as objects.

The [GridItemsSourceChangedEventArgs](#) object contains the following properties:

- [OldItemsSource](#) - Gets the value of old data source
- [NewItemsSource](#) - Get the value of new data source

### View

WPF DataGrid has the View property of type [ICollectionViewAdv](#) interface that implements ICollectionView interface. View is responsible for maintain and manipulation data and other advanced operations like Sorting, Grouping, Filtering and etc.

When you bind Collection to **ItemsSource** property of SfDataGrid, then View will be created and maintains the operations on Data such as Grouping, Filtering, Sorting, Insert, Delete, and Modification.

Following are some important properties that can be used for various purposes.

**Note:** DataGrid creates different types of views derived from **ICollectionViewAdv** interface based on ItemsSource.

Property	Type	Description
Records	IRecordsList	Maintains the Records that are displayed in View when DataGrid is not Grouped.
TopLevelGroup	TopLevelGroup	Maintains the Group information when DataGrid is Grouped.
TopLevelGroup.DisplayElements	GroupDisplayElements	Maintains the Records and Group information that are displayed in View when DataGrid is Grouped.
Filter	Predicate<object>	Get or sets the method that determines the data is suitable to be displayed in View.
FilterPredicates	ObservableCollection<IFilterDefinition>	Maintains the FilterPredicates that are created while filtering using Filtering UI.
Groups	ReadOnlyObservableCollection<object>	Maintains the top-level group information. It returns null value when there are no groups.
GroupDescriptions	ObservableCollection<GroupDescription>	Maintains the GroupDescription collection information. It describes how the items in the collection are grouped in the view.
SortDescriptions	SortDescriptionCollection	Maintains the SortDescription collection information. It describes how the items in the collection are sort in the view.
SourceCollection	IEnumerable	Maintains the underlying source collection.
TableSummaryRows	ObservableCollection<ISummaryRow>	Maintains the TableSummaryRows collection information. To know more about TableSummaries <a href="#">Refer here</a>
SummaryRows	ObservableCollection<ISummaryRow>	Maintains the SummaryRows collection information. To know more about summaries {{ ' <a href="#">Refer here</a> '   markdownify }}
CaptionSummaryRows	ISummaryRow	Maintains the CaptionSummaryRow information.To know more about

		CaptionSummaries [Refer here](http://help.syncfusion.com/wpf/sfdatagrid/summaries)
--	--	--

The following events are associated with View.

#### *RecordPropertyChanged*

[RecordPropertyChanged](#) event is raised when the DataModel property value is changed, if the DataModel implements the INotifyPropertyChanged interface. The event receives with two arguments namely sender that handles the DataModel and [PropertyChangedEventArgs](#) as object.

[PropertyChangedEventArgs](#) has below property,

[PropertyName](#) – It denotes the PropertyName of the changed value.

#### *CollectionChanged*

[CollectionChanged](#) event is raised whenever that is some change in Records / DisplayElements collection. The event receives two arguments namely sender that handles View object and [NotifyCollectionChangedEventArgs](#) as object.

[NotifyCollectionChangedEventArgs](#) has below properties,

[Action](#) - It contains the current action. (i.e.) Add, Remove, Move, Replace, Reset.

[NewItems](#) - It contains the list of new items involved in the change.

[OldItems](#) - It contains the list of old items affected by the Action.

[NewStartingIndex](#) - It contains the index at which the change occurred.

[OldStartingIndex](#) - It contains the index at which the Action occurred.

#### *SourceCollectionChanged*

[SourceCollectionChanged](#) event is raised when you make changes in SourceCollection for example add or remove the collection. The event receives two arguments namely sender that handles GridQueryableCollectionViewWrapper object and [NotifyCollectionChangedEventArgs](#) as object.

[NotifyCollectionChangedEventArgs](#) has below properties,

[Action](#) - It contains the current action. (i.e.) Add, Remove, Move, Replace, Reset.

[NewItems](#) - It contains the list of new items involved in the change.

[OldItems](#) - It contains the list of old items affected by the Action.

[NewStartingIndex](#) - It contains the index at which the change occurred.

[OldStartingIndex](#) - It contains the index at which the Action occurred.

The following is the methods that are associated with View which can be used to defer refresh the view.

Method Name	Description
DeferRefresh	Enter the defer cycle so that you can perform all data operations in view and update once.

BeginInit & EndInit	When BeginInit method is called it suspends all the updates until EndInit method is called. You can perform all the updation with in these methods and update the view at once.
---------------------	---

---

**Note:** View has properties that already defined in SfDataGrid. It recommended setting those properties via SfDataGrid.

---

#### Binding data from WCF service

In this walkthrough, you will learn about how to create a WCF service and load it to SfDataGrid control.

Below are the topics,

1. Creating the WCF data service
2. Creating the WPF Client Application
3. Loading data from WCF Service

Reference:

<https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2015/data-tools/bind-wpf-controls-to-a-wcf-data-service?view=vs-2015>

#### Create the WCF data service

To create the WCF data service, follow the steps mentioned in the below MSDN link or the follow below steps,

<https://docs.microsoft.com/en-us/dotnet/framework/data/wcf/creating-the-data-service>.

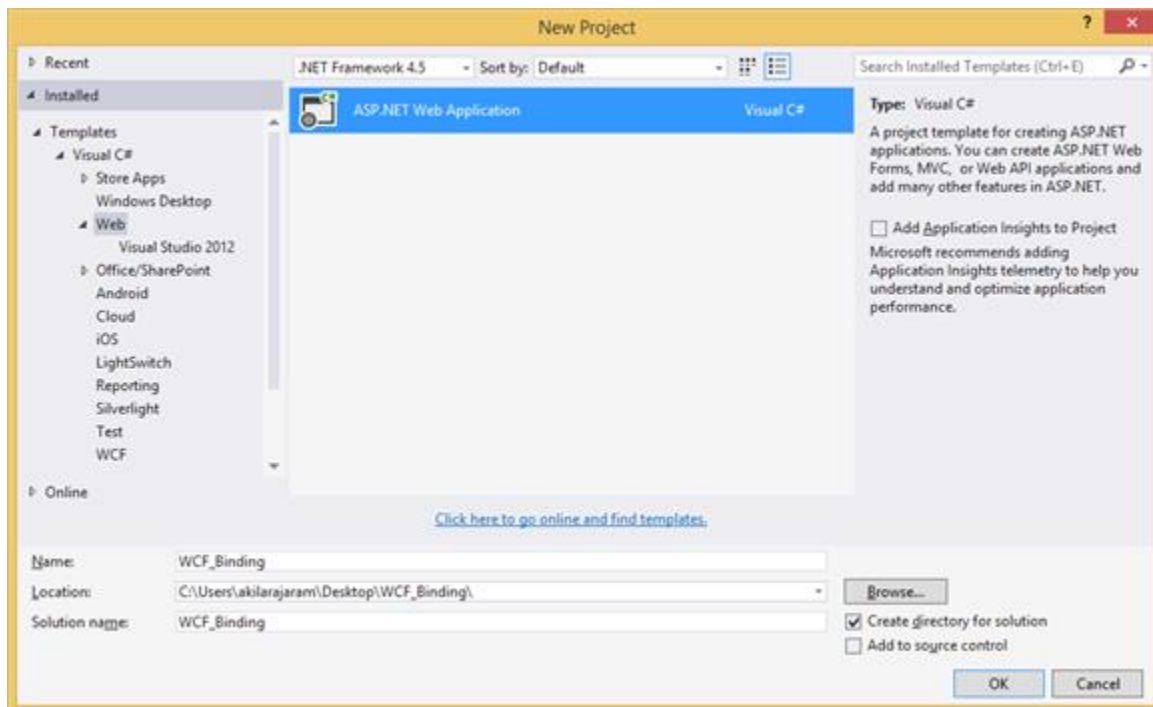
You can skip the below steps and directly add service reference to the client application, if you are having already running service.

#### Create an ASP.NET Web Application

To create an ASP.NET Web Application, follow the below steps,

1. In NewProject dialog box, select the **Web** template under **Visual C#** and then select **ASP.NET Web Application**.
2. Name the project file as **WCF\_Binding** and then click **OK** to create project.

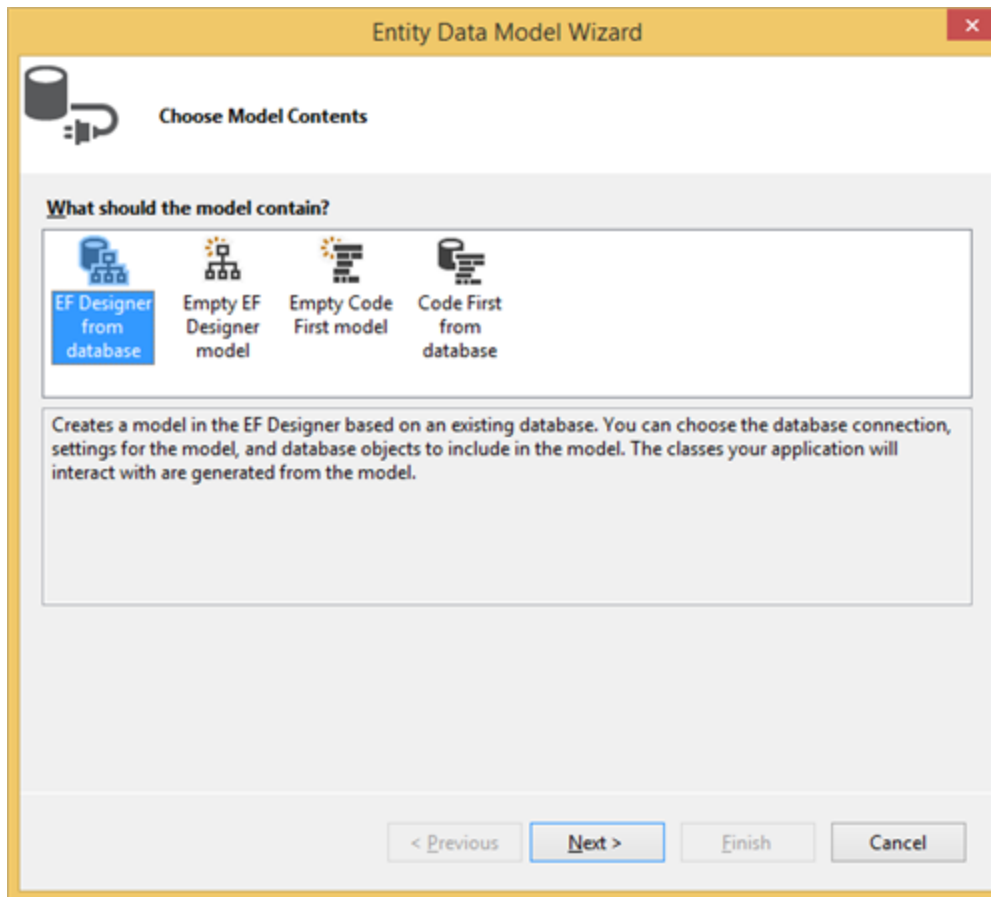




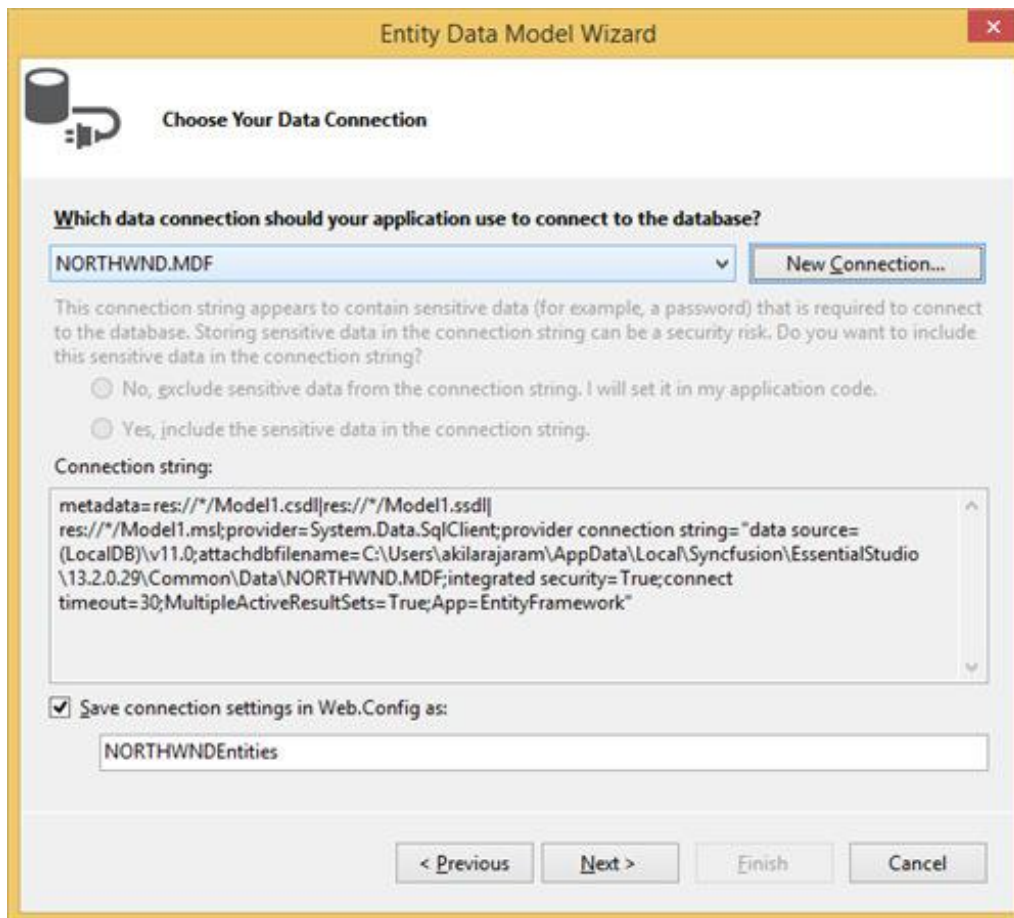
#### *Define data model using ADO.NET Entity Data Model*

Follow the below steps to define **ADO.NET Entity Data Model** in the web project created in the previous step,

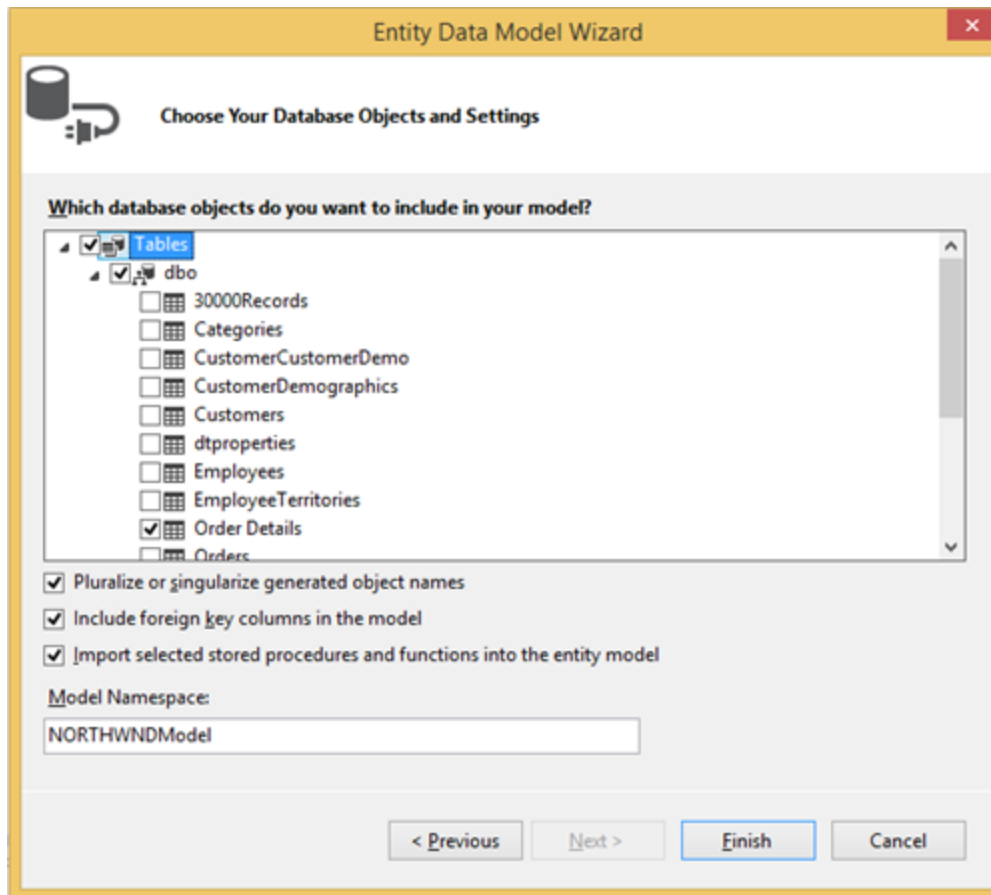
1. Right-click on your web project, select **Add** option and then click **New Item**.
2. In the **Add New Item** wizard, select **ADO.NET Entity Data Model** under **Data Template**.
3. Name the data model as "**Northwind**" and click **Add** button.
4. In the **Entity Data Model Wizard**, select **EF Designer from Database**, and then click **Next**.



5. **Choose Your Data Connection** page appears and select **Northwind** database available in the drop-down list (OR) select the **New Connection** button to configure a new data connection. For more information, you can refer: [How to: Create Connections to SQL Server Databases](#).



6. Click **Next** button and the **Choose Your Database Objects and settings** page appears.
7. Expand **Tables** node and select Order\_Details table.

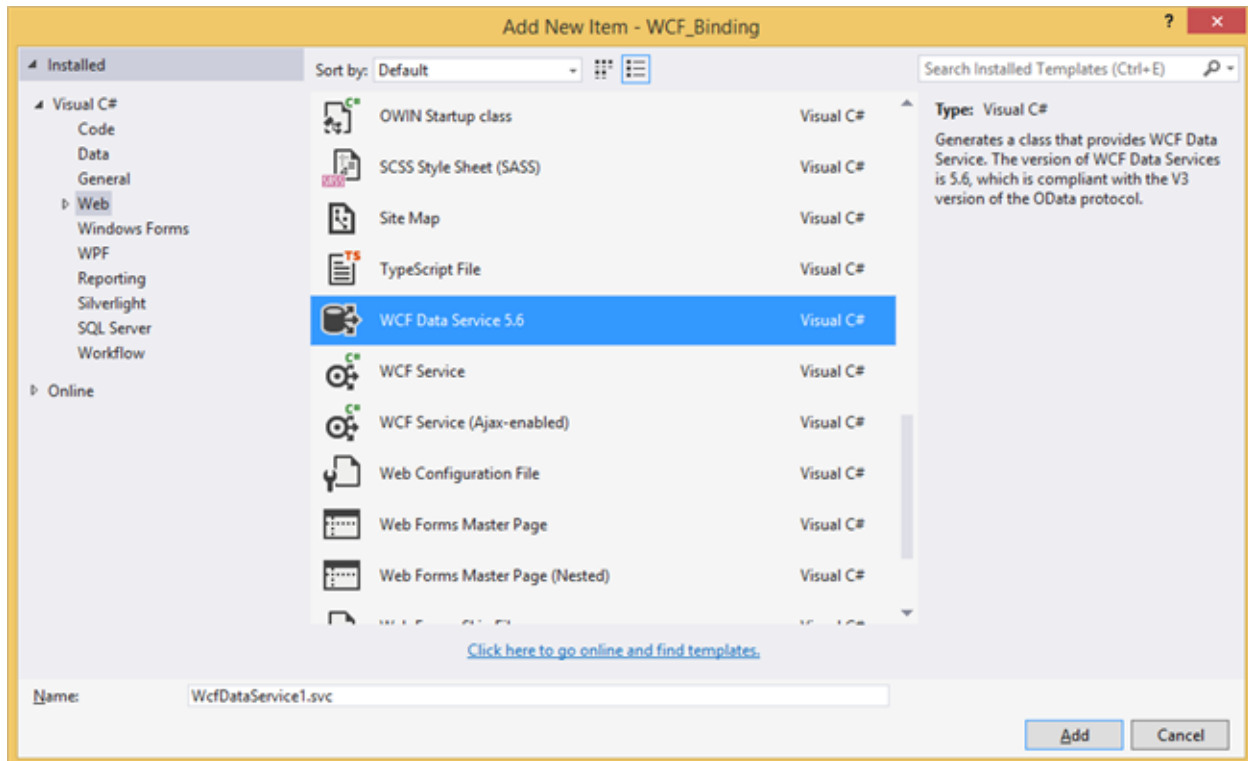


8. Now select **Finish** button to add the entity model for “Northwind.edmx” file to your web project.

#### *Add the WCF Data Service to web application*

To add the WCF Data Service to your web project created in first step,

1. Right-click on your web project, select **Add** option and then click **New Item**.
2. In **Add New Item Wizard**, Select **WCF Data Service 5.6** under **Web Template**.
3. In **Name** text box enter **WcfDataService1.svc** and then click **Add** button.



- Once service is added and configure the read and write access to resources in **InitializeService** function of DataService.

### C#

```
public class WcfDataService1 : DataService<NORTHWNDEntities>
{
    // This method is called only once to initialize service-wide policies.
    public static void InitializeService(DataServiceConfiguration config)
    {
        // TODO: set rules to indicate which entity sets and service operations are
        // visible, Updatable , etc.
        // Examples:
        config.SetEntitySetAccessRule("*", EntitySetRights.AllRead);
        // config.SetEntitySetAccessRule("MyEntitySet", EntitySetRights.AllRead);
        // config.SetServiceOperationAccessRule("MyServiceOperation",
        // ServiceOperationRights.All);
        //config.DataServiceBehavior.MaxProtocolVersion =
        DataServiceProtocolVersion.V3;
    }
}
```

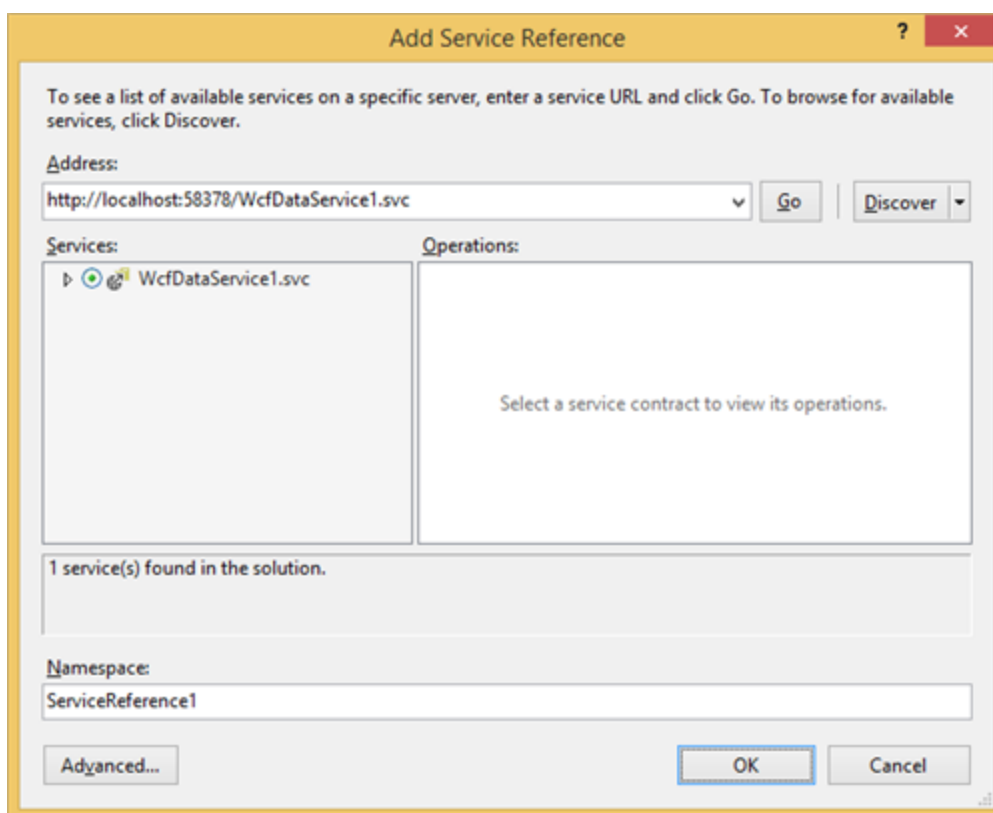
- On the menu bar, select Debug->Start without Debugging to execute the service. A browser window opens and the XML schema for the service is displayed. You can also get the localhost address for the service here.

### Creating the WPF Client Application

To create the WPF client application, add new WPF project in the same solution and name the project as NorthwindEditor. After creating the WPF application, you can add the service created in previous step to your WPF project.

To add service reference to client application,

1. Right-click on your WPF project and select **Add Service Reference** option to add the service created in previous step.
2. **Add Service Reference** wizard appears, then select **Discover** button which displays the list of available services in **Services** panel.
3. Select WcfDataService1.svc and click **OK** option to add the service reference to your WPF project.



### Loading data from WCF service

To load the data from WCF service to SfDataGrid,

1. Add SfDataGrid control and the required assemblies in your WPF application.
2. Wire the **SfDataGrid.Loaded** event and set the northwindEntities.Order\_Details table to **SfDataGrid.ItemsSource**.
3. Replace local host address as URI of your service. You can get the local host address from the shortcut menu of the "WcfDataService1.svc" file in Solution Explorer and select **View** in Browser. Internet Explorer opens and the XML schema for the service is displayed.
4. Copy the localhost address and replace it in your service URI.
5. Set the WPF application as StartUp project.

**C#**

```
this.dataGrid.Loaded += dataGrid_Loaded;
void dataGrid_Loaded(object sender, RoutedEventArgs e)
{
    NORTHWNDEntities northwindEntities = new NORTHWNDEntities(new
    Uri("http://localhost:58378/WcfDataService1.svc/"));
    this.dataGrid.ItemsSource = northwindEntities.Order_Details;
}
```

Now, run the application and you can see the SfDataGrid control loaded with data from WCF service.

### Binding data from ADO.NET Entity Framework

SfDataGrid control supports to bind data from ADO.NET Entity Framework. In this walk-through, you will learn about binding data from ADO.NET Entity Framework and save back the changes to the database. You can download the entire source code of this demo from [here](#).

To load the data from ADO.NET Entity Framework, you can refer the steps mentioned in below follow the below steps,

1. Creating WPF client Application.
2. Defining Data Model using Entity Framework 4.0
3. Loading data from Entity Framework.

### References :

[https://docs.microsoft.com/en-us/previous-versions/dd465159\(v=vs.120\)](https://docs.microsoft.com/en-us/previous-versions/dd465159(v=vs.120))

[https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/ee340709\(v=vs.100\)](https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/ee340709(v=vs.100))

<https://docs.microsoft.com/en-us/ef/ef6/modeling/designer/workflows/database-first>

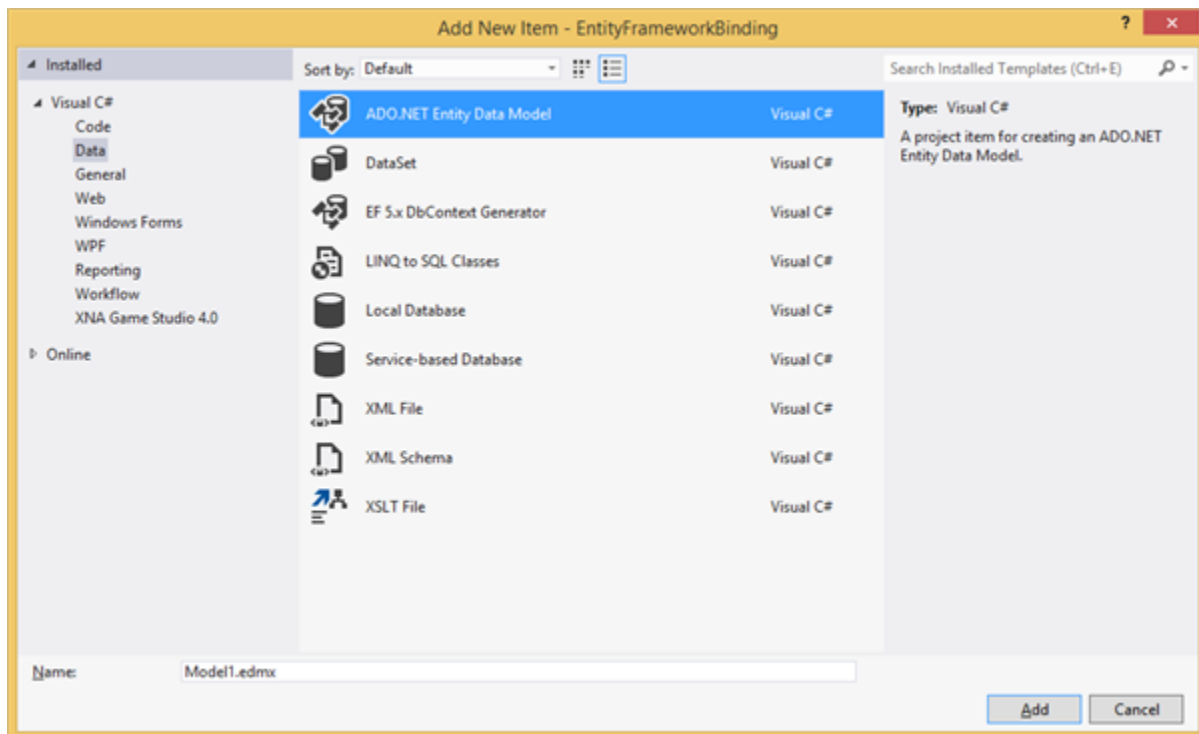
### *Creating WPF client Application*

To load data from Entity Framework, create a new WPF Application and add the SfDataGrid control to your application.

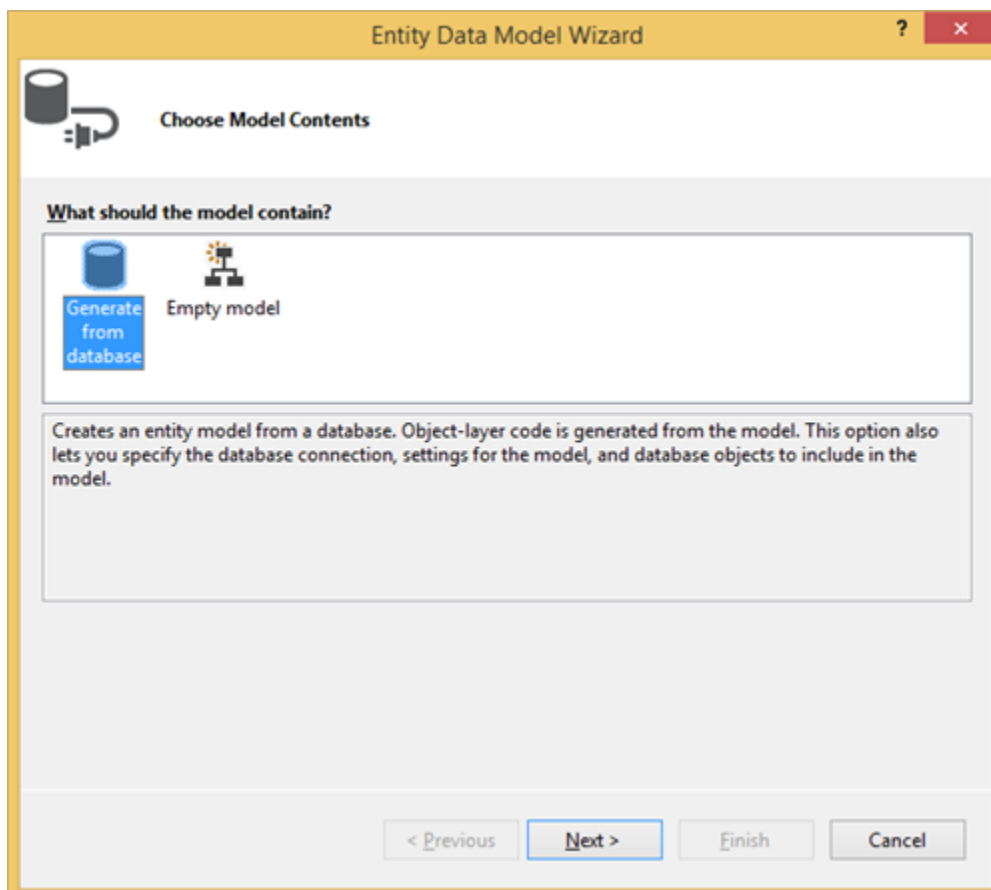
### *Defining Data Model using Entity Framework 4.0*

To create Data Model using Entity Framework in WPF application created in the previous step,

1. Right-click your WPF project, select **Add** option and then click **New Item**.
2. The **Add New Item** wizard appears, select “**ADO.NET Entity Data Model**” from the **Data** node.
3. Name the file as **Model1.edmx** and then select **Add** button.

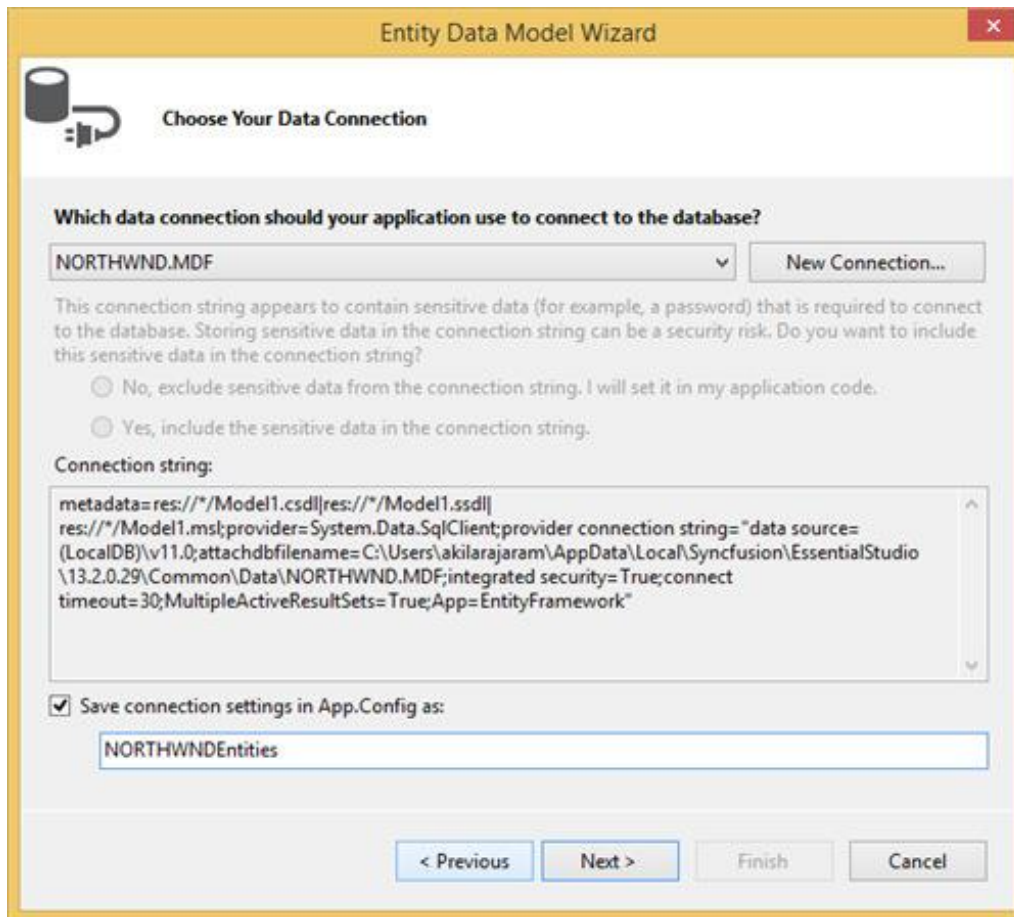


4. The Entity Data Model wizard appears. 5. In the **Choose Model Contents**, select **Generate from database** option and then click **Next**.

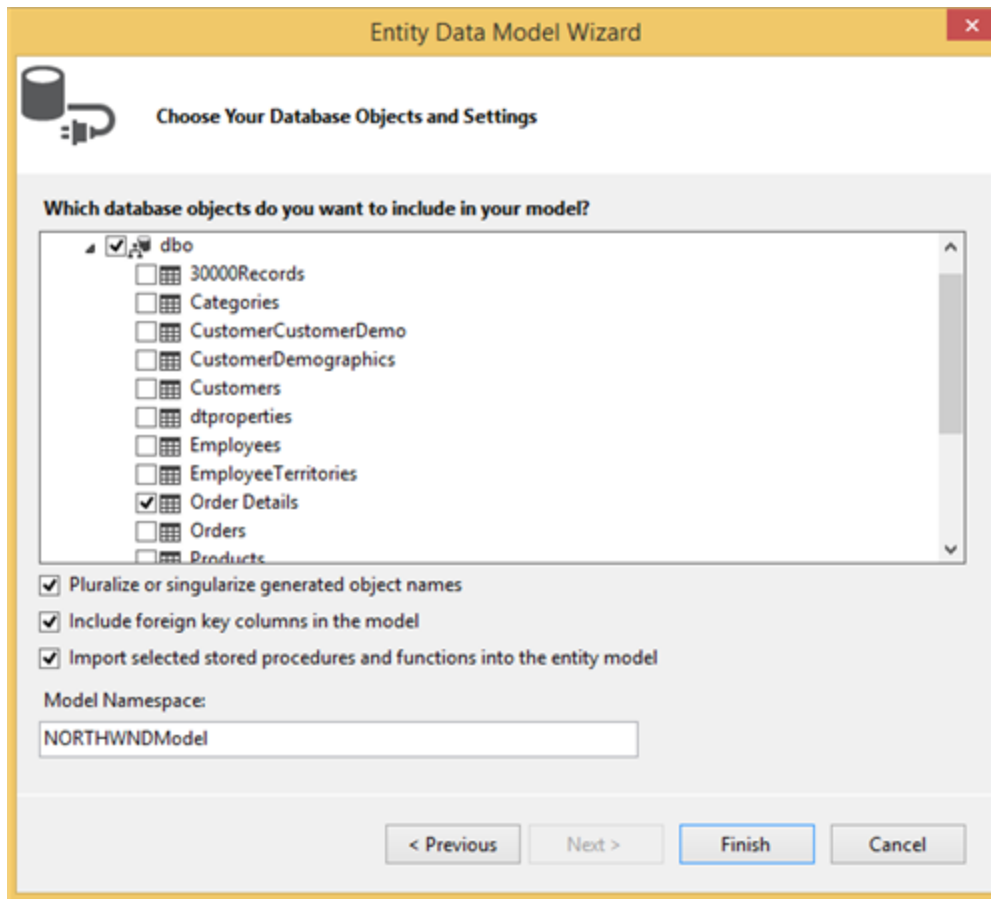




6. In the **Choose Your Data Connection**, select **Northwind** database from the drop-down list for data connection. To configure/modify connection, you can refer: [How to: Create Connections to SQL Server Databases](#).



7. Enable the **Save entity connection settings in App.config** check box and then click **Next**.
8. In the **Choose Your Database Objects and Settings**, expand the Tables node and select the **Order\_Details** table.



- Click **Finish** button to add the **EntityBindingFramework.edmx** file to your application. The **Entity diagram** for the **Order\_detail** table is opened.

#### *Loading data from Entity Framework data service*

To access data from the database created in previous step, create a ViewModel class with OrderDetails property and the OrderDetails is initialized by accessing the Order\_Details table from the database.

#### **C#**

```
public class ViewModel
{
    private List<Order_Detail> orderDetails;
    public List<Order_Detail> OrderDetails
    {
        get { return orderDetails; }
        set { orderDetails = value; }
    }
    public ViewModel()
    {
        NORTHWNDEntities northWind = new NORTHWNDEntities();
        OrderDetails = (List<Order_Detail>)(from data in northWind.Order_Details
        select data).ToList();
    }
}
```

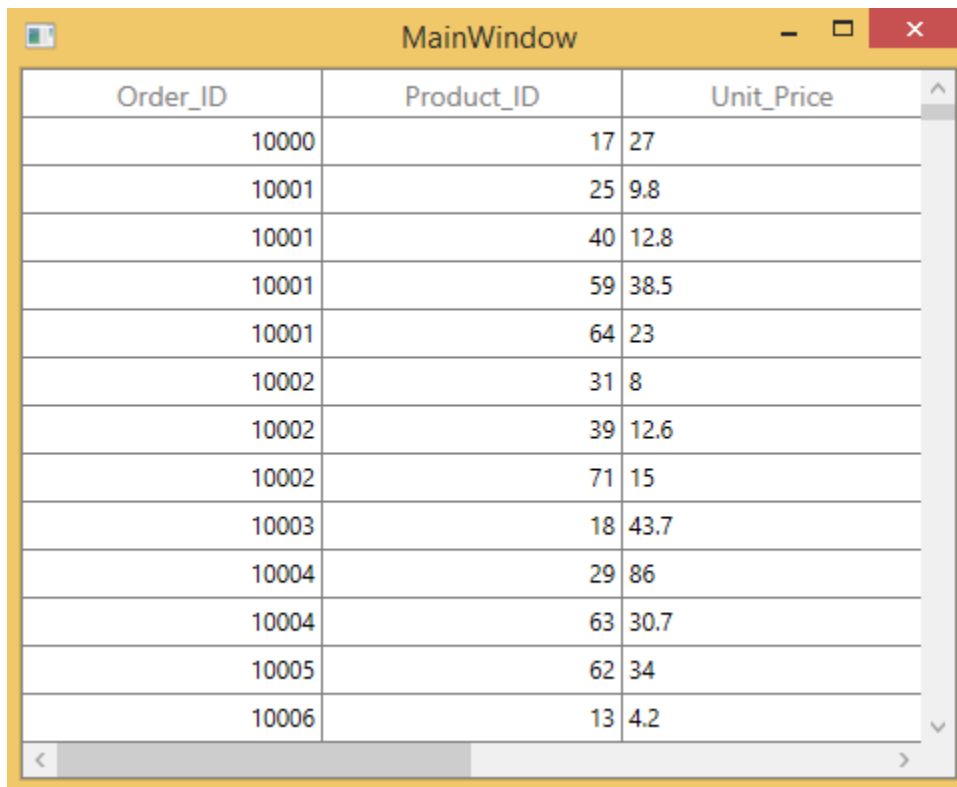
To populate the SfDataGrid using Entity Framework, bind the collection created in previous step to [SfDataGrid.ItemsSource](#) property and set the `DataContext` as ViewModel.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowEditing="True"
    AutoGenerateColumns="True"
    ItemsSource="{Binding OrderDetails}"/>
```

Now, run the application and you can see the following screenshot shows the SfDataGrid control populated with data from Entity Framework data service.

<br/>



Order_ID	Product_ID	Unit_Price
10000	17	27
10001	25	9.8
10001	40	12.8
10001	59	38.5
10001	64	23
10002	31	8
10002	39	12.6
10002	71	15
10003	18	43.7
10004	29	86
10004	63	30.7
10005	62	34
10006	13	4.2

### Save back to Database

You can save the row or cell level changes back to the data source by handling [SfDataGrid.RowValidated](#), [SfDataGrid.CurrentCellValidated](#) or [SfDataGrid.CurrentCellValueChanged](#) events.

The below code example will save back the changed value of row to data base.

### C#

```
NORTHWNDEntities northWind;
public MainWindow()
{
    InitializeComponent();
    northWind = new NORTHWNDEntities();
    this.dataGrid.RowValidated += dataGrid_RowValidated;
}
```

```
void dataGrid_RowValidated(object sender,
    Syncfusion.UI.Xaml.Grid.RowValidatedEventArgs args)
{
    Order_Detail newRecord = args.RowData as Order_Detail;
    Order_Detail order = northWind.Order_Details.First(i => i.OrderID ==
        newRecord.OrderID);
    order.OrderID = newRecord.OrderID;
    order.ProductID = newRecord.ProductID;
    order.Quantity = newRecord.Quantity;
    order.UnitPrice = newRecord.UnitPrice;
    order.Discount = newRecord.Discount;
    northWind.Entry(order).State = EntityState.Modified;
    northWind.SaveChanges();
}
```

### Binding data from LINQ to SQL

SfDataGrid control supports to bind data from LINQ to SQL. In this walkthrough, you will learn about binding data from LINQ to SQL and save back the changes to the Database.

1. Creating WPF Application
2. Adding data model using LINQ to SQL classes
3. Loading data from LINQ to SQL classes
4. Binding data to SfDataGrid

### References :

<https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/data-binding>

<https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/insert-update-and-delete-operations>

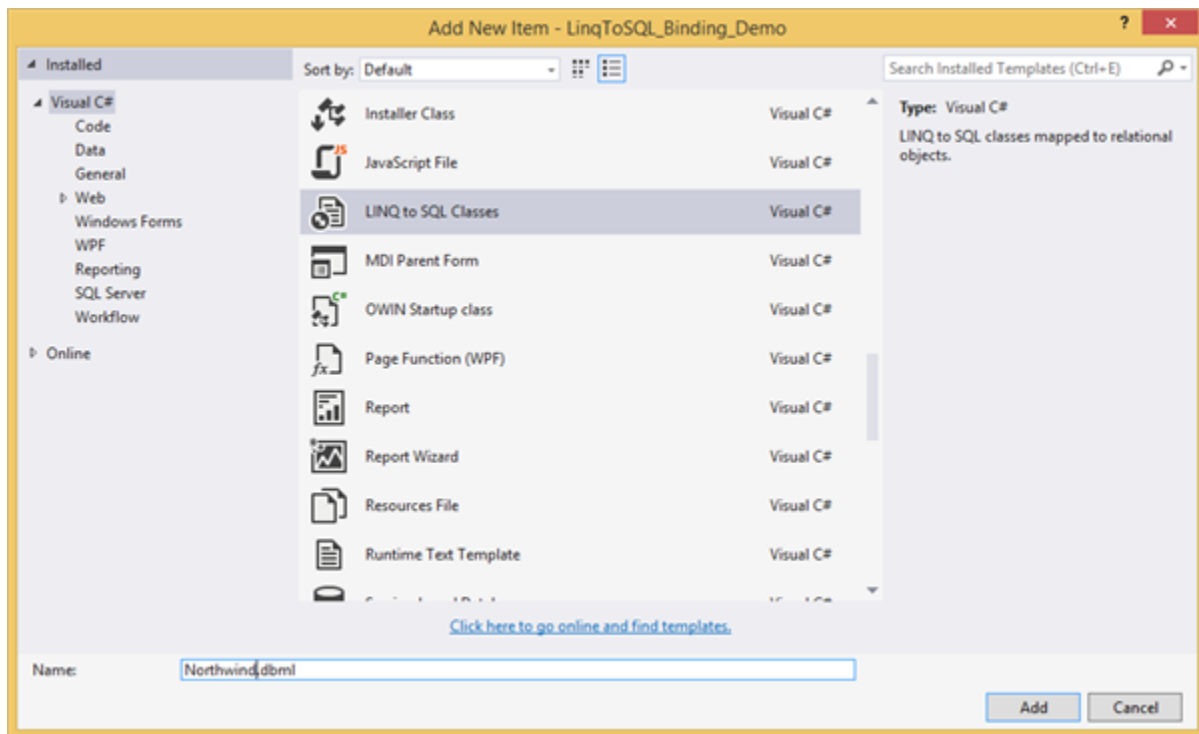
#### *Creating WPF Client Application*

To add LINQ to SQL, create a new WPF application and add the SfDataGrid control in your application.

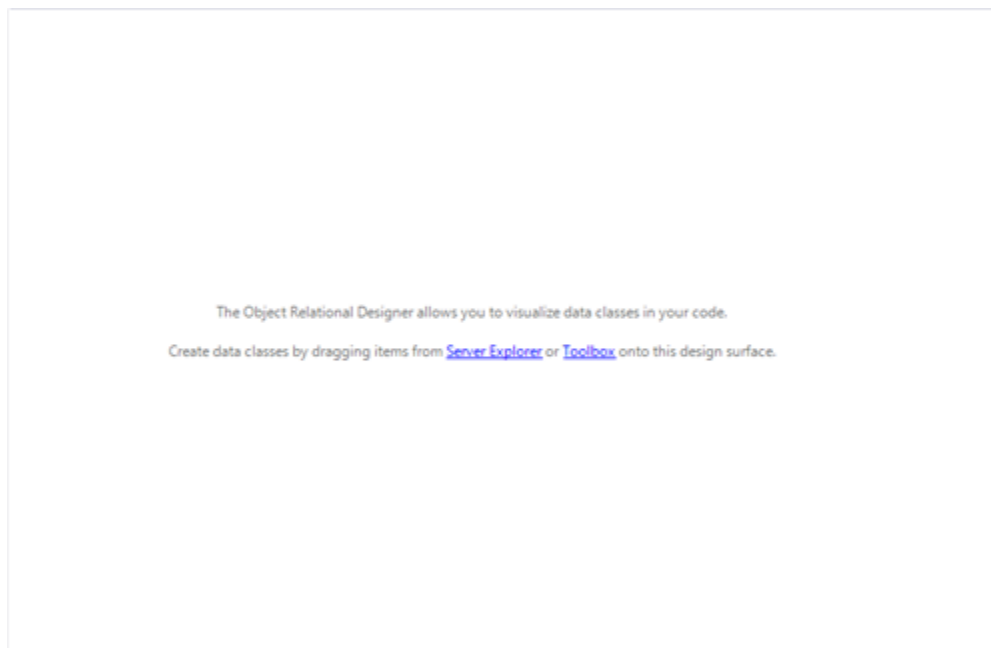
#### *Adding data model using LINQ to SQL*

To create data model using LINQ to SQL in WPF project follow the below steps.

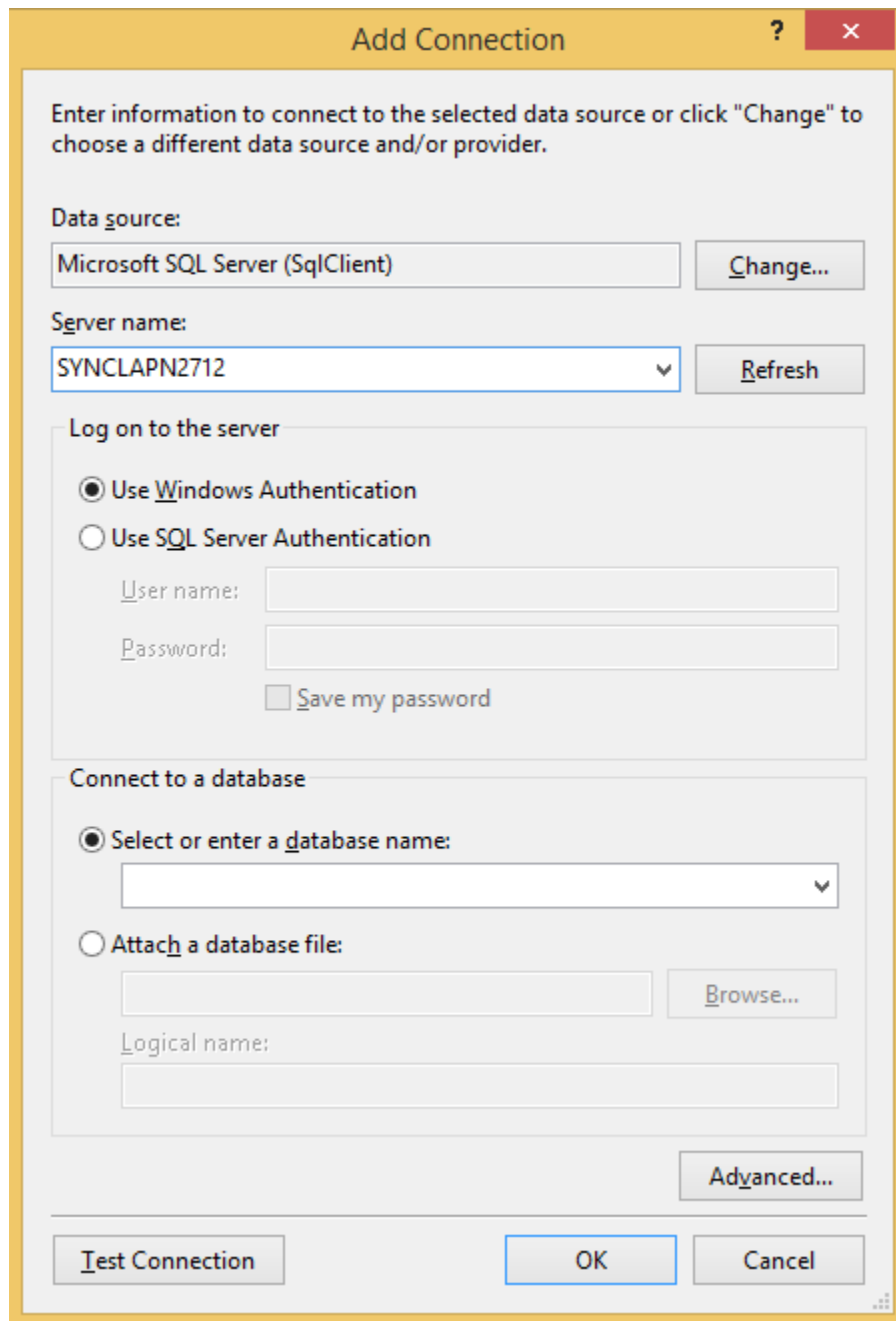
1. Right-click on your WPF project, select **Add** option and then click **New Item**.
2. The **Add New Item** wizard appears, select **LINQ to SQL Classes** from the **Visual C#**.
3. Name the file as **Northwind** and then select **Add** option to add the **Northwind.dbml** in your project.



4. Once the Northwind.dbml is added in your project, then the design view is opened.



5. You can add new Database connection by clicking add icon button in Server Explorer.
6. The **Add Connection** wizard appeared with the default data source as **Microsoft SQL Server Database File (SqlClient)**.



The image shows a 'Add Connection' dialog box with a yellow title bar. It contains several sections for configuring a data source connection. The 'Data source' section has a text box with 'Microsoft SQL Server (SqlClient)' and a 'Change...' button. The 'Server name' section has a dropdown menu showing 'SYNCLAPN2712' and a 'Refresh' button. The 'Log on to the server' section has two radio buttons: 'Use Windows Authentication' (selected) and 'Use SQL Server Authentication'. Below these are text boxes for 'User name' and 'Password', and a checkbox for 'Save my password'. The 'Connect to a database' section has two radio buttons: 'Select or enter a database name:' (selected) and 'Attach a database file:'. The first has a dropdown menu, and the second has a text box and a 'Browse...' button. Below these is a 'Logical name' text box. At the bottom right is an 'Advanced...' button. At the bottom are three buttons: 'Test Connection', 'OK', and 'Cancel'.

**Add Connection** ? X

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:  
Microsoft SQL Server (SqlClient) Change...

Server name:  
SYNCLAPN2712 Refresh

Log on to the server

☒ Use Windows Authentication  
☐ Use SQL Server Authentication

User name:   
Password:   
☐ Save my password

Connect to a database

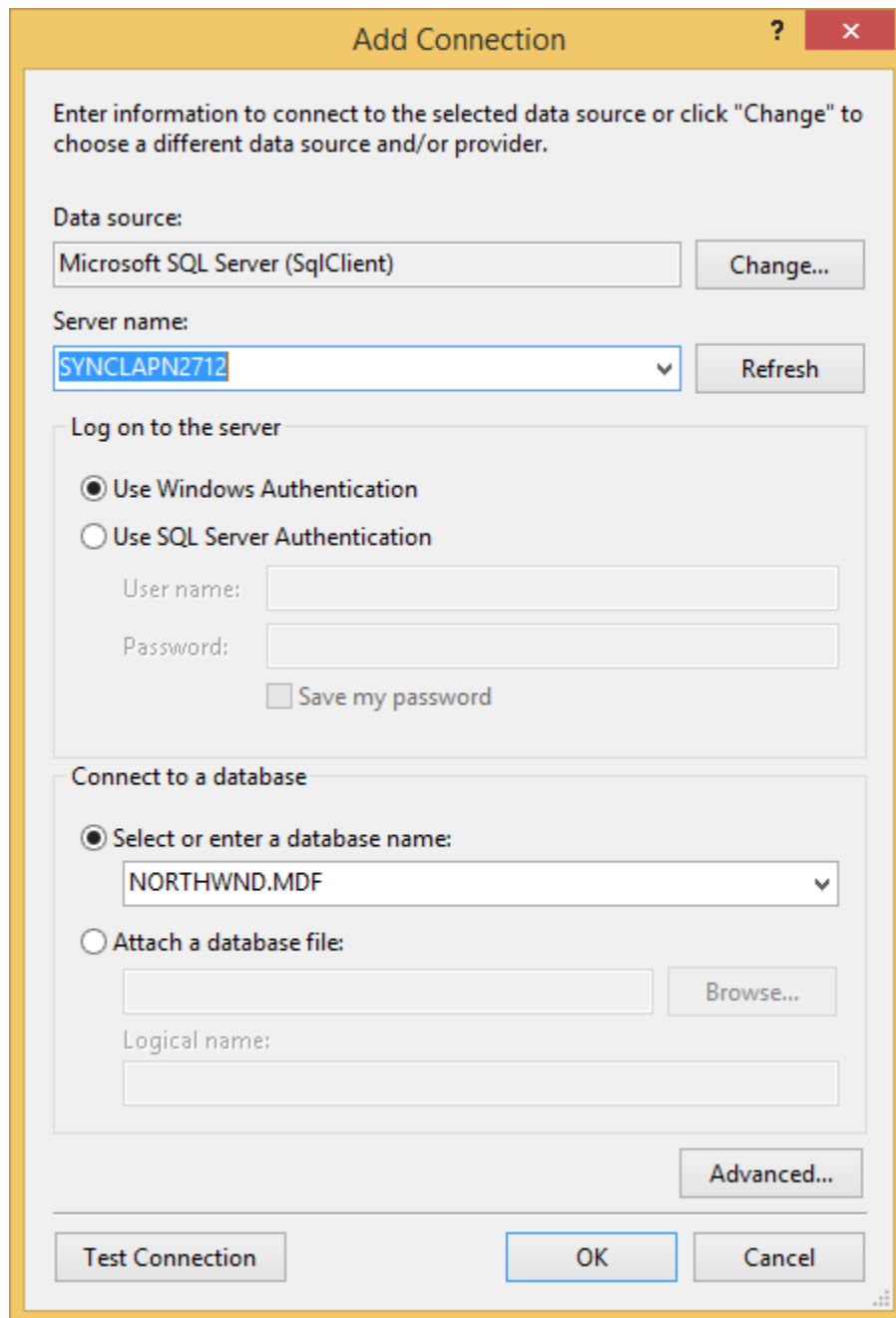
☒ Select or enter a database name:

☐ Attach a database file:  
 Browse...  
Logical name:

Advanced...

Test Connection OK Cancel

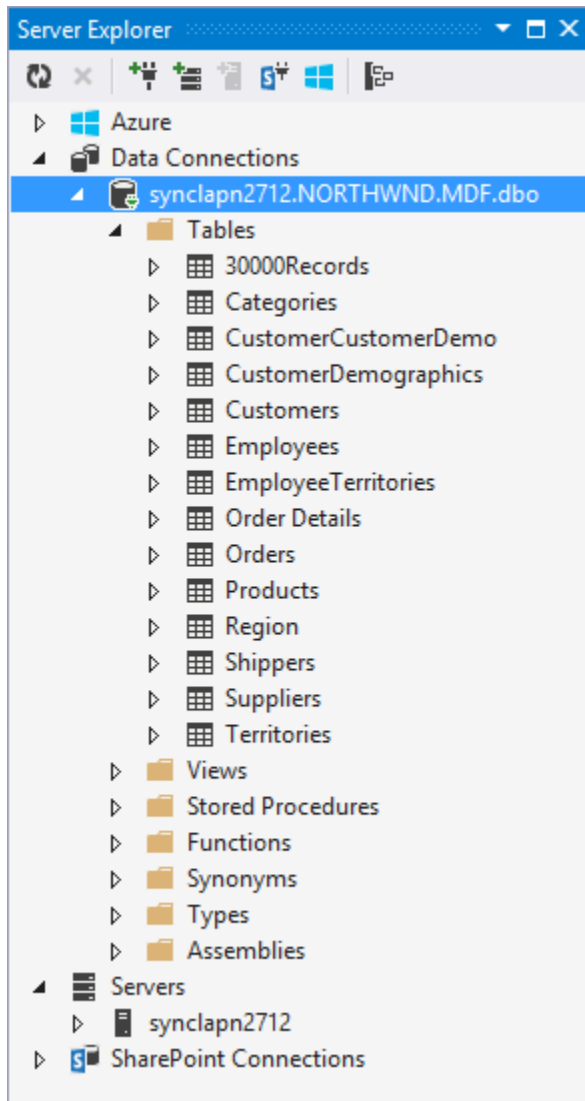
7. Click **Change** to open the **Change Data Source** dialog box and select the type of data source that you have owned.
8. In the **Server name** option, click the **Refresh** button to select the server from the drop down list.
9. Enable **select or enter a database name** radio button to select database from the drop down list under the **Connect to a database** option.



The image shows a 'Add Connection' dialog box with a yellow title bar. It contains the following sections:

- Header:** 'Add Connection' title, a help icon (?), and a close button (X).
- Instructions:** 'Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.'
- Data source:** A text box containing 'Microsoft SQL Server (SqlClient)' and a 'Change...' button.
- Server name:** A dropdown menu showing 'SYNCLAPN2712' and a 'Refresh' button.
- Log on to the server:** A group box containing:
  - Two radio buttons: 'Use Windows Authentication' (selected) and 'Use SQL Server Authentication'.
  - Text boxes for 'User name:' and 'Password:'.
  - A checkbox labeled 'Save my password'.
- Connect to a database:** A group box containing:
  - Two radio buttons: 'Select or enter a database name:' (selected) and 'Attach a database file:'.
  - A dropdown menu showing 'NORTHWND.MDF' under the first option.
  - Text boxes for 'Attach a database file:' and a 'Browse...' button.
  - A text box for 'Logical name:'.
- Buttons:** 'Test Connection', 'OK', 'Cancel', and 'Advanced...'.

10. Click **Test Connection** to check whether the connection with your database is succeeded or not.
11. Once the connection is succeeded, click **OK** button to add database in your server explorer.



12. Drag **Shippers** table in to design view of **Northwind.dbml**. The **Entity model diagram** for Shippers table is generated once it is dropped in to design view.

#### *Loading data from LINQ to SQL classes*

To load the database created in previous step in to SfDataGrid, create a ViewModel class with Shippers property and it is initialized with Shippers table in Northwind database.

#### **C#**

```
public class ViewModel
{
    private List<Shipper> shippers;
    public List<Shipper> Shippers
    {
        get { return shippers; }
        set { shippers = value; }
    }
    public ViewModel()
    {
    }
}
```



```
{
NorthwindDataContext northWind = new NorthwindDataContext();
shippers = (from data in northWind.Shippers
select data).ToList();
}
```

**Note:** NorthwindDataContext is from Northwind.Designer.cs file (it is from the file that is added with LINQ to SQL). Shippers are selected table from Database.

#### *Binding data to SfDataGrid*


To bind data from LINQ to SQL classes, assign the Shippers collection created in the previous step, to [SfDataGrid.ItemsSource](#) property and set the [DataContext](#) as ViewModel.

#### **C#**

```
this.dataGrid.ItemsSource = (this.dataGrid.DataContext as
ViewModel).Shippers;
```

Now, run the application and you can see the following screenshot shows the SfDataGrid control loaded with Shippers data.

<br/>



ShipperID	CompanyName	Phone
1	Speedy Express	(503) 512
2	United Package	(503)
3	Federal Shipping	(503) 555-9931

#### *Save back to Database*

You can save the row or cell level changes back to the data source by handling [SfDataGrid.RowValidated](#), [SfDataGrid.CurrentCellValidated](#) or [SfDataGrid.CurrentCellValueChanged](#) events.

The below code example will save back the changed value of row to database.

#### **C#**

```
this.dataGrid.RowValidated += dataGrid_RowValidated;
void dataGrid_RowValidated(object sender,
Syncfusion.UI.Xaml.Grid.RowValidatedEventArgs args)
{
    NorthwindDataContext northWind = new NorthwindDataContext();
    Shipper newRecord = args.RowData as Shipper;
    Shipper shipper = (from data in northWind.Shippers
where data.ShipperID == newRecord.ShipperID
select data).First() as Shipper;
    shipper.CompanyName = newRecord.CompanyName;
    shipper.Phone = newRecord.Phone;
    northWind.SubmitChanges();
}
```

### Binding data from ADO.NET

SfDataGrid control supports to load the data using [ADO.NET](#). In this walk-through, you will learn about binding data from ADO.NET service and save back the changes to the database.

1. Creating WPF application
2. Connecting database to WPF application
3. Loading data using ADO.NET

#### *Creating WPF Application*

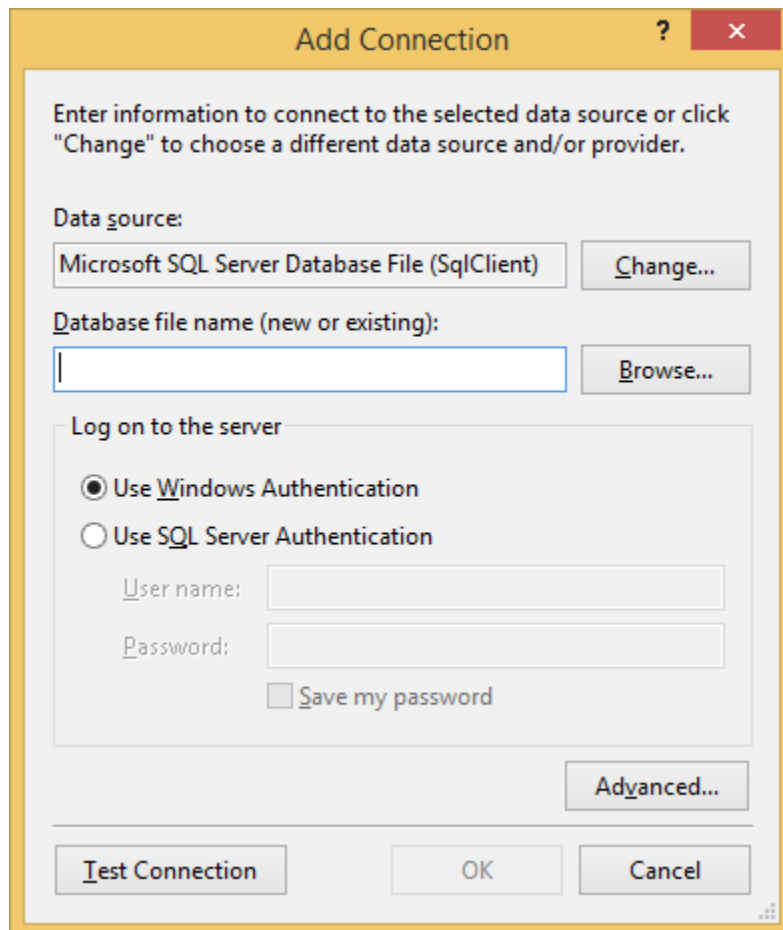
To display the data from the ADO.NET data service, create a new WPF Application.

#### *Connecting WPF application to Databases*

To connect SQL database to your WPF application, refer the below MSDN link or follow the below steps,

[https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/s4yys16a\(v=vs.90\)](https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/s4yys16a(v=vs.90))

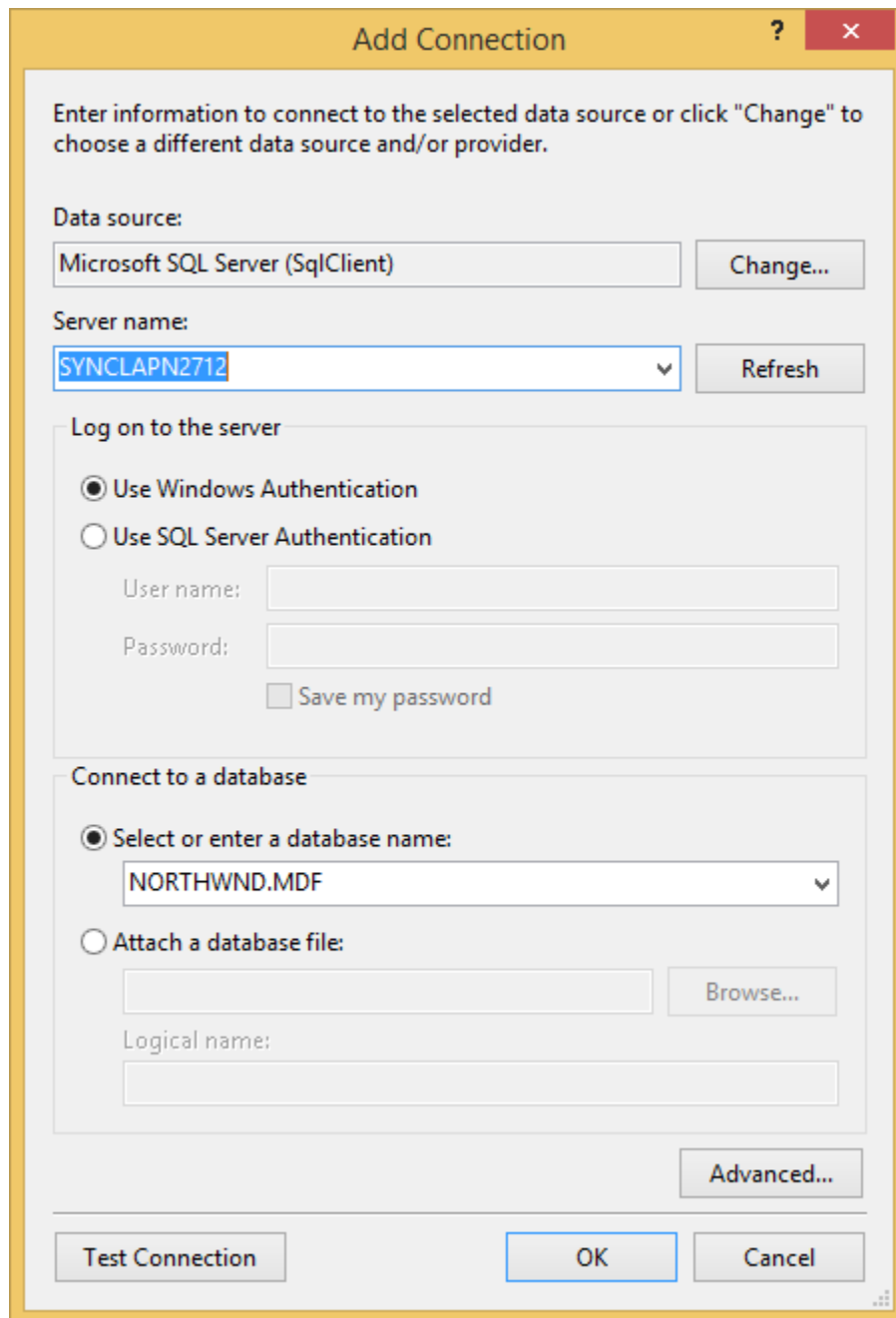
1. In the Tools menu, select the **Connect to Database**.
2. The **Add Connection** wizard appeared with the default data source as **Microsoft SQL Server Database File (SqlClient)**.



The image shows a Windows-style dialog box titled "Add Connection". It has a yellow header bar with a question mark icon and a red close button. The main content area is white and contains the following elements:

- Instructional text: "Enter information to connect to the selected data source or click 'Change' to choose a different data source and/or provider."
- Data source section: A text box containing "Microsoft SQL Server Database File (SqlClient)" and a "Change..." button.
- Database file name section: A label "Database file name (new or existing):", an empty text box, and a "Browse..." button.
- Log on to the server section: A group box containing two radio buttons: "Use Windows Authentication" (selected) and "Use SQL Server Authentication". Below these are text boxes for "User name:" and "Password:", and a checkbox labeled "Save my password".
- Advanced button: An "Advanced..." button.
- Footer: Three buttons: "Test Connection", "OK", and "Cancel".

3. Click **Change** to open the **Change Data Source** dialog box and select the type of data source that you have owned.



The image shows a 'Add Connection' dialog box with a yellow title bar. It contains several sections for configuring a database connection. The 'Data source' section has a text box with 'Microsoft SQL Server (SqlClient)' and a 'Change...' button. The 'Server name' section has a dropdown menu showing 'SYNCLAPN2712' and a 'Refresh' button. The 'Log on to the server' section has two radio buttons: 'Use Windows Authentication' (selected) and 'Use SQL Server Authentication'. Below these are text boxes for 'User name:', 'Password:', and a 'Save my password' checkbox. The 'Connect to a database' section has two radio buttons: 'Select or enter a database name:' (selected) and 'Attach a database file:'. The first has a dropdown menu showing 'NORTHWND.MDF'. The second has a text box and a 'Browse...' button. Below these is a 'Logical name:' text box. At the bottom right is an 'Advanced...' button. At the bottom are three buttons: 'Test Connection', 'OK', and 'Cancel'.

**Add Connection**

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

**Data source:**  
Microsoft SQL Server (SqlClient) Change...

**Server name:**  
SYNCLAPN2712 Refresh

**Log on to the server**

☒ Use Windows Authentication  
☐ Use SQL Server Authentication

User name:   
Password:   
☐ Save my password

**Connect to a database**

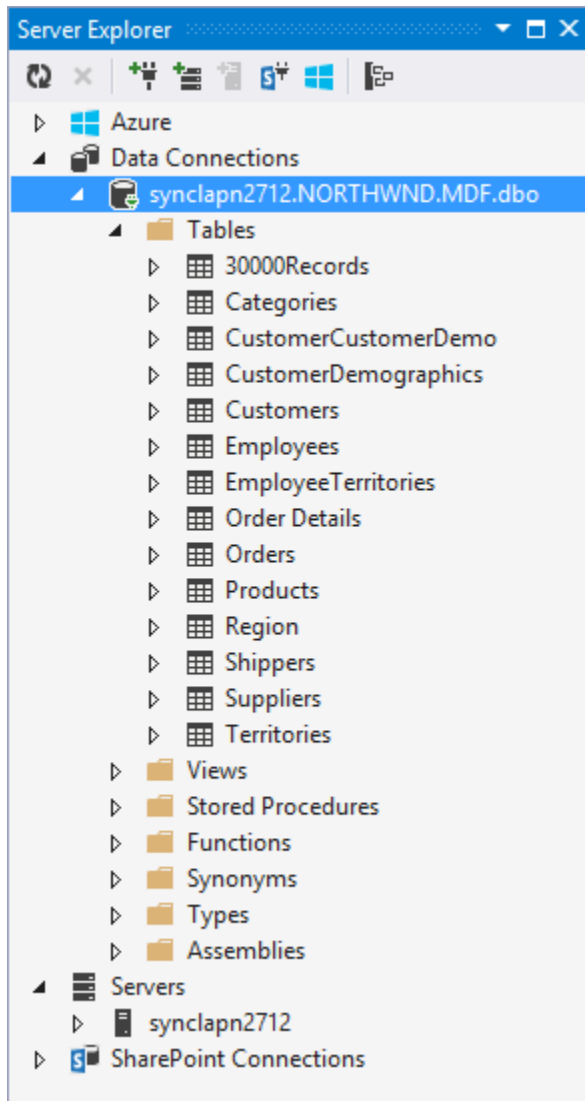
☒ Select or enter a database name:  
NORTHWND.MDF

☐ Attach a database file:  
 Browse...  
Logical name:

Advanced...

Test Connection OK Cancel

4. In the **Server name** option, click the **Refresh** button to select the server from the drop down list.
5. Enable **select or enter a database name** radio button to select your database from the drop down list under the **Connect to a database** option.
6. Click **Test Connection** to check whether the database connection is succeeded or not.
7. Once the connection is succeeded, click **OK** button to add database in your server explorer.



8. To get connection string for the database, right-click on your database and then click **properties** option.

#### *Loading data from ADO.NET*

To access the data from data source using ADO.NET, follow the below steps.

1. Create a user interface with SfDataGrid control and add the required assemblies to your WPF application.
2. Create a connection through any of the [.NET Framework data provider](#) based on the type of data source that you have owned.
3. Set the [ItemsSource](#) as Shippers table from data set. For more information refer [here](#).

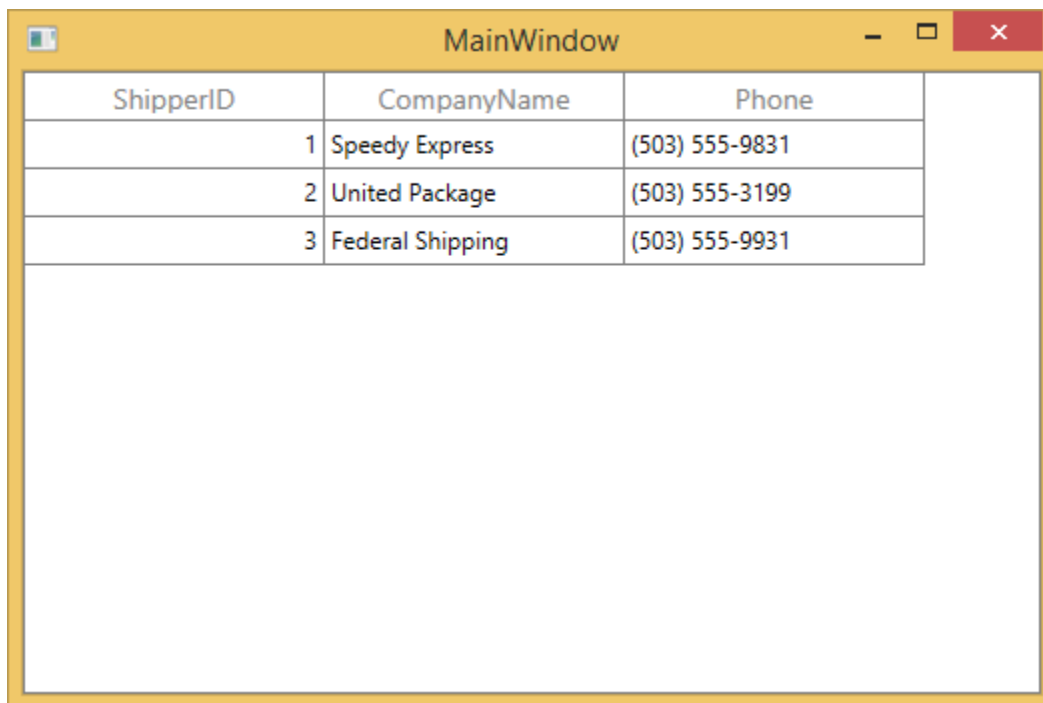
#### **C#**

```
public partial class MainWindow : Window
{
    DataSet ds = new DataSet();
```

```
SqlConnection connectionString = new SqlConnection("Data
Source=SYNCLAPN2712;Initial Catalog=NORTHWND.MDF;Integrated Security=True");
SqlDataAdapter adapter = null;
public MainWindow()
{
    InitializeComponent();
    BindData();
}
private void BindData()
{
    connectionString.Open();
    adapter = new SqlDataAdapter("Select * from Shippers ", connectionString);
    adapter.Fill(ds, "Shippers");
    dataGrid.ItemsSource = ds.Tables["Shippers"];
}
}
```

Now, run the application and you can see the following screenshot displays the SfDataGrid loaded the data using ADO.NET.

<br/>



#### *Save back to Database*

You can save the row or cell level changes back to the data source by handling [SfDataGrid.RowValidated](#), [SfDataGrid.CurrentCellValidated](#) or [SfDataGrid.CurrentCellValueChanged](#) events.

The below code example will save back the changed value of row to database.

You can refer the below MSDN link for more information.

<https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/updating-data-sources-with-dataadapters>

**C#**

```
this.dataGrid.RowValidated += dataGrid_RowValidated;
void dataGrid_RowValidated(object sender,
Syncfusion.UI.Xaml.Grid.RowValidatedEventArgs args)
{
    adapter.UpdateCommand = new SqlCommand("UPDATE Shippers SET CompanyName =
@CompanyName " + ", " + "Phone = @Phone " +
"WHERE ShipperID = @ShipperID", connectionString);
    adapter.UpdateCommand.Parameters.Add("@CompanyName", SqlDbType.NVarChar, 40,
"CompanyName");
    adapter.UpdateCommand.Parameters.Add("@Phone", SqlDbType.NVarChar, 24,
"Phone");
    SqlParameter parameter = adapter.UpdateCommand.Parameters.Add("@ShipperID",
SqlDbType.Int, 4);
    parameter.SourceColumn = "ShipperID";
    parameter.SourceVersion = DataRowVersion.Original;
    DataTable shippersTable = new DataTable();
    shippersTable.Clear();
    adapter.Fill(shippersTable);
    DataRow shippersRow = shippersTable.Rows[args.RowIndex - 1];
    shippersRow["CompanyName"] = (args.RowData as DataRowView).Row.ItemArray[1];
    shippersRow["Phone"] = (args.RowData as DataRowView).Row.ItemArray[2];
    adapter.Update(shippersTable);
}
```

## Binding data from Microsoft Access

SfDataGrid control supports to bind data from Microsoft Access database .In this section, you will learn about how to bind the data from Microsoft Access database to SfDataGrid.

To load the data from Microsoft Access database,

1. Create a new WPF project.
2. Import the Microsoft Access database file to WPF project.
3. You can connect the Microsoft Access Database through the **OleDbConnection**.
4. Create and open the connection.
5. Use **OleDbDataAdapter** to load the data in to **DataSet**.
6. Access the Employee table from the DataSet and set the Employee collection as **ItemsSource** of SfDataGrid.

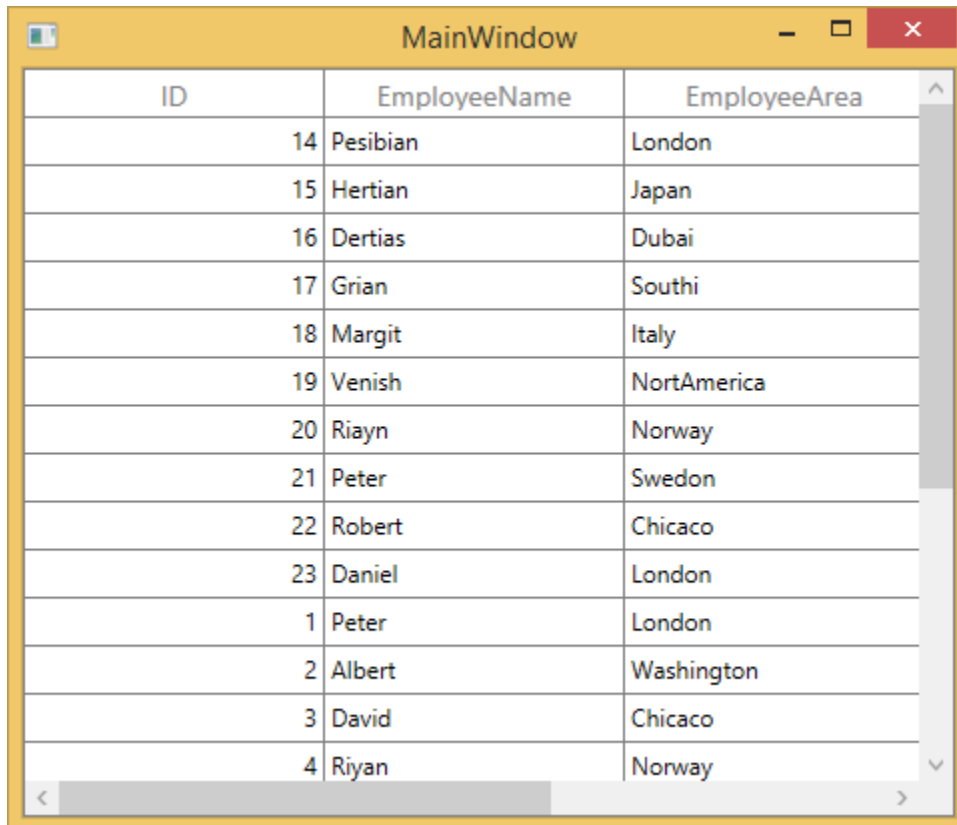
**C#**

```
this.dataGrid.Loaded+=dataGrid_Loaded;
void dataGrid_Loaded(object sender, RoutedEventArgs e)
{
    System.Data.OleDb.OleDbConnection con = new
    System.Data.OleDb.OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=EmployeeDatabase.accdb;Persist Security Info=True");
    System.Data.OleDb.OleDbDataAdapter myDataAdapter = new
    System.Data.OleDb.OleDbDataAdapter("SELECT * FROM Employee", con);
    DataSet ds = new DataSet();
    con.Open();
    myDataAdapter.Fill(ds, "Employee");
    con.Close();
}
```

```
dataGrid.ItemsSource = ds.Tables["Employee"];  
}
```

7. Now, run the application and you can see following screenshot shows the SfDataGrid control populated data from Microsoft Access database.

<br/>



ID	EmployeeName	EmployeeArea
14	Pesibian	London
15	Hertian	Japan
16	Dertias	Dubai
17	Grian	Southi
18	Margit	Italy
19	Venish	NortAmerica
20	Riayn	Norway
21	Peter	Swedon
22	Robert	Chicaco
23	Daniel	London
1	Peter	London
2	Albert	Washington
3	David	Chicaco
4	Riyan	Norway

How To

*Maintain scroll position when changing the ItemsSource for SfDataGrid*

By default, the scrollbar position is not maintained and gets reset when changing the ItemsSource of the SfDataGrid. But, you can maintain the scrollbar position of the SfDataGrid by setting the [SfDataGrid.CanMaintainScrollPosition](#) value to `true`.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"  
CanMaintainScrollPosition="True"  
ItemsSource="{Binding Orders}"/>
```

#### C#

```
dataGrid.CanMaintainScrollPosition = true;
```



### Columns in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) (SfDataGrid) allows you to add or remove columns using [SfDataGrid.Columns](#) property. You can choose the columns to be added from built-in column types or you can create your own column and add to the [SfDataGrid.Columns](#).

Below are the built-in column types supported in SfDataGrid. Each column has its own properties to handle different types of data.

Column Type	Description
<a href="#">GridTextColumn</a>	Use to display the string data.
<a href="#">GridNumericColumn</a>	Use to display the numeric data.
<a href="#">GridCurrencyColumn</a>	Use to display the currency value.
<a href="#">GridPercentColumn</a>	Use to display the percent value
<a href="#">GridMaskColumn</a>	Use to display the data to be masked.
<a href="#">GridTimeSpanColumn</a>	Use to display the time span value
<a href="#">GridDateTimeColumn</a>	Use to display the date time value
<a href="#">GridComboBoxColumn</a>	Use to display the IEnumerable data using ComboBox.
<a href="#">GridCheckBoxColumn</a>	Use to display the boolean type data
<a href="#">GridImageColumn</a>	Use to display the image in each row.
<a href="#">GridHyperlinkColumn</a>	Use to display the Uri data
<a href="#">GridTemplateColumn</a>	Use to display the custom template-specified content.
<a href="#">GridUnboundColumn</a>	Use to display custom information of each record.
<a href="#">GridMultiColumnDropDownList</a>	Use to display the IEnumerable data using SfMultiColumnDropDownControl.

### Defining columns

You can let the SfDataGrid to create columns or you can manually define columns to be displayed. Below sections explains both ways,

1. Automatically generating columns
2. Manually define columns

#### *Automatically generating columns*

The automatic column generation based on properties of data object can be enabled or disabled by setting [SfDataGrid.AutoGenerateColumns](#). Default value is `true`.

Columns are generated based on type of property. For example, [GridNumericColumn](#) is added for `int` type property. Below are table shows data type and its column type. For remaining types, [GridTextColumn](#) will be added.

Data Type	Column
string, object, dynamic	GridTextColumn
int, float, double, decimal and also it's nullable	GridNumericColumn
DateTime, DateTimeOffset and also it's nullable	GridDateTimeColumn
Uri, Uri?	GridHyperLinkColumn
bool, bool?	GridCheckBoxColumn
TimeSpan, TimeSpan?	GridTimeSpanColumn

**Note:** The order of columns in the collection will determine the order of that they will appear in SfDataGrid.

#### [AutoGenerateColumns with different modes](#)

Column auto generation is controlled using [SfDataGrid.AutoGenerateColumnsMode](#) property.

The [SfDataGrid.AutoGenerateColumnsMode](#) includes the following modes.

Mode	Behavior	When ItemsSource changed
Reset	Generates the columns based on the properties defined in the underlying data object.	Keeps the columns added manually. Clears the columns which are auto generated before and creates new columns based on new ItemsSource.
RetainOld	Generates the columns based on the properties defined in the underlying data object if the columns are not defined explicitly.	The same columns will be maintained when changing ItemsSource also. So filtering, sorting and grouping settings will be maintained.
ResetAll	Generates the columns based on the properties defined in the underlying data object.	Clear all the columns including the columns defined manually and creates new columns based on new ItemsSource.
None	Columns will not be generated.	Keeps old columns in DataGrid.Columns collection.

#### [Auto generating columns for complex type](#)

Custom type (complex type) properties in data object can be auto-generated by setting [AutoGenerateColumnsForCustomType](#) property as true. Default value is false.

Custom type properties will be auto-generated through [AutoGenerateColumnsModeForCustomType](#) property.

### XML

```
<syncfusion:SfDataGrid ItemsSource="{Binding Source}" x:Name="dataGrid"
AutoGenerateColumnsForCustomType="True"
AutoGenerateColumnsModeForCustomType="Parent"
AllowEditing="True"
AllowFiltering="True"
AllowSorting="True"
ShowGroupDropArea="True"
ColumnSizer="Star" >
</syncfusion:SfDataGrid>
```

### C#

```
this.dataGrid.AutoGenerateColumnsForCustomType = true;
this.dataGrid.AutoGenerateColumnsModeForCustomType =
AutoGenerateColumnsModeForCustomType.Parent;
```

The `AutoGenerateColumnsModeForCustomType` includes the following modes.

Mode	Behavior
Both	Specifies that the columns for both the custom type and its inner properties will be auto generated.
Child	Specifies that the columns for all inner properties of custom type column will be auto generated.
Parent	Specifies that the column for only the custom type will be auto generated.

You can download the sample demo [here](#) .

#### Customize auto-generated columns in DataGrid

You can customize or cancel the generated column by handling `AutoGeneratingColumn` event.

`AutoGeneratingColumn` event occurs when the individual column is auto-generated for public and non-static property of underlying data object.

### C#

```
this.dataGrid.AutoGeneratingColumn += dataGrid_AutoGeneratingColumn;
void dataGrid_AutoGeneratingColumn(object sender, AutoGeneratingColumnArgs
e)
{
}
```

`AutoGeneratingColumnArgs` provides the information about the auto-generated column to the `AutoGeneratingColumn` event. `AutoGeneratingColumnArgs.Column` property returns the newly created column.

#### Cancel column generation for particular property

You can cancel the specific column adding to the DataGrid by handling `AutoGeneratingColumn` event.

In the below code, column generation for `OrderID` property is canceled by setting `Cancel` property to `true`.

#### C#

```
this.dataGrid.AutoGeneratingColumn += dataGrid_AutoGeneratingColumn;
void dataGrid_AutoGeneratingColumn(object sender, AutoGeneratingColumnArgs e)
{
    if (e.Column.MappingName == "OrderID")
        e.Cancel = true;
}
```

#### Changing column type

You can change the type of column adding to SfDataGrid by setting the instance of column you want to add in `AutoGeneratingColumn` event.

In the below code, column type for `UnitPrice` property is changed to `GridTextColumn` by setting instance of `GridTextColumn` to `Column` property.

#### C#

```
this.dataGrid.AutoGeneratingColumn += dataGrid_AutoGeneratingColumn;
void dataGrid_AutoGeneratingColumn(object sender, AutoGeneratingColumnArgs e)
{
    if (e.Column.MappingName == "UnitPrice")
    {
        if (e.Column is GridNumericColumn)
        {
            e.Column = new GridTextColumn() { MappingName = "UnitPrice", HeaderText = "Unit Price" };
        }
    }
}
```

#### Changing property settings

You can change the column properties in `AutoGeneratingColumn` event handler.

#### C#

```
this.dataGrid.AutoGeneratingColumn += dataGrid_AutoGeneratingColumn;
void dataGrid_AutoGeneratingColumn(object sender, AutoGeneratingColumnArgs e)
{
    if (e.Column.MappingName == "OrderID")
    {
        e.Column.AllowEditing = false;
        e.Column.AllowSorting = true;
        e.Column.AllowFiltering = true;
        e.Column.AllowGrouping = false;
        e.Column.AllowFocus = true;
        e.Column.AllowResizing = false;
        e.Column.ColumnSizer = GridLengthUnitType.Auto;
        e.Column.AllowDragging = true;
    }
}
```

```
}
}
```

Setting template to auto-generated column

You can set [GridColumn.HeaderTemplate](#) and [GridColumn.CellTemplate](#) properties for auto-generated column in [AutoGeneratingColumn](#) event handler.

### XML

```
<Window.Resources>
<DataTemplate x:Key="headerTemplate">
<TextBlock Text="The product has been purchased by the following OrderID"
TextWrapping="Wrap" />
</DataTemplate>
</Window.Resources>
```

### C#

```
this.dataGrid.AutoGeneratingColumn += dataGrid_AutoGeneratingColumn;
void dataGrid_AutoGeneratingColumn(object sender, AutoGeneratingColumnArgs
e)
{
if (e.Column.MappingName == "OrderID")
{
e.Column.HeaderTemplate = this.FindResource("headerTemplate") as
DataTemplate;
}
}
```

Below screenshot shows the customized header template loaded on the header of OrderID column.

The product has been purchased by the following OrderID	CustomerID	CustomerName
1000	ALFKI	Maria Anders
1001	ANATR	Ana Trujilo
1002	FKI	Anders
1003	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

### Data Annotations with AutoGenerateColumns

WPF DataGrid (SfDataGrid) support to generate the columns based on built-in [Data Annotation Attributes](#). Data Annotations ignored, when the `AutoGenerateColumns` is set to False.

#### Exclude column

You can skip the column generation using `AutoGenerateField` property or set the `Bindable` attribute to false.

#### C#

```
[Display(AutoGenerateField = false, Description = "OrderID field is not generated in UI")]
public int OrderID
{
    get { return orderID; }
    set { orderID = value; }
}
```

#### Filtering

You can enable filtering automatically for the field using `Display.AutoGenerateFilter` property

#### C#

```
[Display(AutoGenerateFilter = true, Description = "Filter enabled for CustomerID column")]
public string CustomerID
{
    get { return customerId; }
    set { customerId = value; }
}
```

#### Editing

You can change the value of the property using `Editable` attribute.

#### C#

```
[Editable(true)]
public string Country
{
    get { return country; }
    set { country = value; }
}
```

#### Change the HeaderText of column

You can customize header text of column using `Display.Name` property.

#### C#

```
[Display(Name="Name of the Customer",Description="CustomerName is necessary for identification ")]
public string CustomerName
{
    get { return customerName; }
    set { customerName = value; }
}
```

Change the order of the columns

You can change the columns order using `DisplayAttribute.Order` property.

**C#**

```
[Display(Order=0)]
public int OrderID
{
    get { return orderID; }
    set { orderID = value; }
}
[Display(Order=-1)]
public string CustomerID
{
    get { return customerId; }
    set { customerId = value; }
}
```

The OrderID and CustomerID column rearranged based on specified order.

OrderID	Unit Price	CustomerID	CustomerName	Country
1001		25 ALFKI	Maria Anders	Germany
1002		33 ANATR	Ana Trujilo	Mexico
1003		25 ANTON	Antonio Moreno	Mexico
1004		50 AROUT	Thomas Hardy	UK
1005		33 BERGS	Christina Berglund	Sweden
1006		50 BLAUS	Hanna Moos	Germany
1007		25 BLONP	Frédérique Citeaux	France
1008		33 BOLID	Martin Sommer	Spain
1009		25 BONAP	Laurence Lebihan	France
1010		50 BOTTM	Elizabeth Lincoln	Canada

DataGrid column formatting

You can customize the data format using `DataTypeAttribute.DataType` property.

**C#**

```
[DataType(DataType.Currency)]
public double UnitPrice
{
    get { return unitPrice; }
    set { unitPrice = value; }
}
```

DataGrid read-only column

You can disable the editing for a column using `ReadOnly` attribute.

**C#**

```
[ReadOnly(true)]
public string Country
{
    get { return country; }
    set { country = value; }
}
```

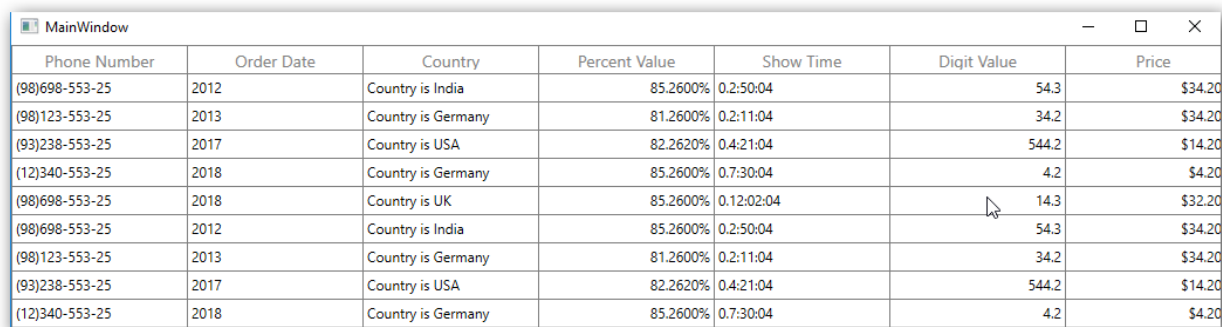
Format datagrid columns using DisplayFormat attribute

The auto-generated columns will be formatted using the [DataFormatString](#) property in the [DisplayFormat](#) attribute when the [DisplayFormat](#) attribute is defined for the properties defined in the view model. If the [DisplayFormat](#) attribute is defined with the [DataFormatString](#) property, the DataGrid formats the column only based on [DataFormatString](#), without considering other formatting property settings of columns.

### C#

```
[DisplayFormat(DataFormatString = "Country is {0}")]
public string Country
{
    get { return country; }
    set { country = value; }
}
[DisplayFormat(DataFormatString = "yyyy")]
public DateTime OrderDate
{
    get { return _orderDate; }
    set { orderDate = value; }
}
```

**Note:** The [DataFormatString](#) attribute will be considered only when the column is auto-generated.



Phone Number	Order Date	Country	Percent Value	Show Time	Digt Value	Price
(98)698-553-25	2012	Country is India	85.2600%	0.2:50:04	54.3	\$34.20
(98)123-553-25	2013	Country is Germany	81.2600%	0.2:11:04	34.2	\$34.20
(93)238-553-25	2017	Country is USA	82.2620%	0.4:21:04	544.2	\$14.20
(12)340-553-25	2018	Country is Germany	85.2600%	0.7:30:04	4.2	\$4.20
(98)698-553-25	2018	Country is UK	85.2600%	0.12:02:04	14.3	\$32.20
(98)698-553-25	2012	Country is India	85.2600%	0.2:50:04	54.3	\$34.20
(98)123-553-25	2013	Country is Germany	81.2600%	0.2:11:04	34.2	\$34.20
(93)238-553-25	2017	Country is USA	82.2620%	0.4:21:04	544.2	\$14.20
(12)340-553-25	2018	Country is Germany	85.2600%	0.7:30:04	4.2	\$4.20

### Manually defining columns

WPF DataGrid (SfDataGrid) control allows you to define the columns manually by adding desired column to the [SfDataGrid.Columns](#) collection.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="False"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.Columns>
    <syncfusion:GridTextColumn HeaderText="Order ID" MappingName="OrderID" />
    </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```



```
<syncfusion:GridTextColumn HeaderText="Customer ID" MappingName="CustomerID" />
<syncfusion:GridTextColumn HeaderText="Customer Name"
MappingName="CustomerName" />
<syncfusion:GridTextColumn HeaderText="Country" MappingName="Country" />
<syncfusion:GridNumericColumn HeaderText="Unit Price"
MappingName="UnitPrice" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#

```
this.dataGrid.Columns.Add(new GridTextColumn() { HeaderText = "Order ID",
MappingName = "OrderID" });
this.dataGrid.Columns.Add(new GridTextColumn() { HeaderText = "Customer ID",
MappingName = "CustomerID" });
this.dataGrid.Columns.Add(new GridTextColumn() { HeaderText = "Customer
Name", MappingName = "CustomerName" });
this.dataGrid.Columns.Add(new GridTextColumn() { HeaderText = "Country",
MappingName = "Country" });
this.dataGrid.Columns.Add(new GridNumericColumn() { HeaderText = "Unit
Price", MappingName = "UnitPrice" });
```

You can refer more information about handling the column level operations for manually defined columns in [Column types](#) section.

### Column manipulation

You can get the columns (added or auto-generated) from [SfDataGrid.Columns](#) property.

#### Adding column to DataGrid

You can add column at runtime by adding instance of column to [SfDataGrid.Columns](#) property.

### C#

```
this.dataGrid.Columns.Add(new GridTextColumn() { HeaderText = "Order ID",
MappingName = "OrderID" });
```

#### Accessing column

You can access the column through its column index or [GridColumn.MappingName](#) from the [SfDataGrid.Columns](#) collection.

### C#

```
GridColumn column = this.dataGrid.Columns[1];
//OR
GridColumn column = this.dataGrid.Columns["OrderID"];
```

#### Clearing or removing column in DataGrid

You can remove all the columns by clearing the [SfDataGrid.Columns](#) property.

### C#

```
this.dataGrid.Columns.Clear();
```

You can remove a column using Remove and RemoveAt methods.

### C#

```
dataGrid.Columns.Remove(column);
//OR
dataGrid.Columns.RemoveAt(1);
```

You can also remove the column under one stacked column from StackedHeaderRow.

### C#

```
var childColumns =
this.dataGrid.StackedHeaderRows[1].StackedColumns[0].ChildColumns.Split(',')
;
foreach (var name in childColumns)
{
var column = dataGrid.Columns[name];
if (column == null)
continue;
dataGrid.Columns.Remove(column);
}
```

### DataGrid column resizing

SfDataGrid allows to resize the columns like in excel by resizing column header. This can be enabled or disabled by setting [SfDataGrid.AllowResizingColumns](#) or [GridColumn.AllowResizing](#) property.

**Note:** Resizing considers MinWidth and MaxWidth of column.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AllowResizingColumns="True"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}" />
```

You can change the column width by clicking and dragging the resizing cursor at the edge of column header. The resizing cursor appears when you hover the grid line exists between two columns.

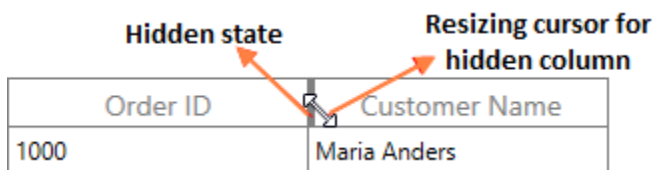
CustomerID	ProductName
ALFKI	Alice Mutton



### Hidden column resizing

SfDataGrid shows indication for hidden columns in column header and also allows end-users to resize the hidden columns when setting [SfDataGrid.AllowResizingHiddenColumns](#) property to `true`.

Order ID	Customer Name
1000	Maria Anders



### Disable resizing

You can cancel resizing of particular column by setting [GridColumn.AllowResizing](#) property to `false`. In another way, you can cancel the resizing by handling [SfDataGrid.ResizingColumns](#) event. The [ResizingColumns](#) event occurs when you start dragging by resizing cursor on headers.

[ResizingColumnsEventArgs](#) of [ResizingColumns](#) provides information about the columns's index and width.

### C#

```
this.dataGrid.ResizingColumns += dataGrid_ResizingColumns;
void dataGrid_ResizingColumns(object sender, ResizingColumnsEventArgs e)
{
    if (e.ColumnIndex == 1)
        e.Cancel = true;
}
```

### Identify resizing of the column gets completed

SfDataGrid allows you to identify the progress of the resizing of columns through [ResizingColumnsEventArgs.Reason](#) property. You can get the width of the column after resizing completed by getting [ResizingColumnsEventArgs.Width](#) when [ResizingColumnsEventArgs.Reason](#) is [ColumnResizingReason.Resized](#) in [ResizingColumns](#) event.

### C#

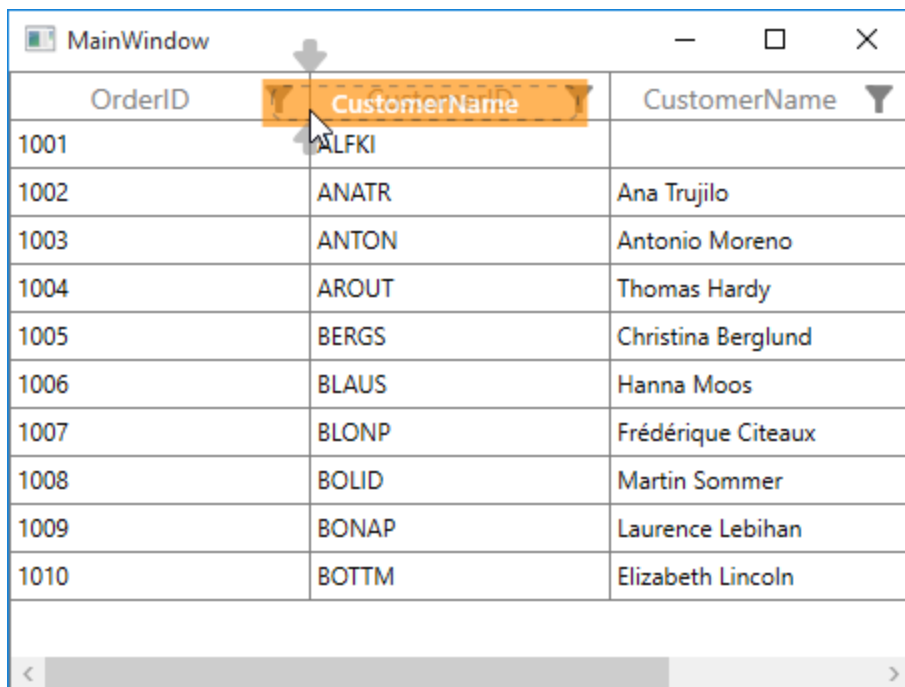
```
this.dataGrid.ResizingColumns += OnResizingColumns;
void OnResizingColumns(object sender, ResizingColumnsEventArgs e)
{
    if (e.Reason == Syncfusion.UI.Xaml.Grid.ColumnResizingReason.Resized)
    {
        var resizedWidth = e.Width;
    }
}
```

### DataGrid column drag and drop

You can allow end-users to rearrange the columns by drag and drop the column headers by setting [SfDataGrid.AllowDraggingColumns](#) to `true`.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowDraggingColumns="True"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}" />
```



OrderID	CustomerName	CustomerName
1001	ALFKI	
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

You can enable or disable dragging on particular column using [GridColumn.AllowDragging](#) property.

### XML

```
<syncfusion:GridTextColumn AllowDragging="False"
  HeaderText="Order ID"
  MappingName="OrderID" />
```

### Disable column reordering

You can cancel the particular column dragging by handling [SfDataGrid.QueryColumnDragging](#).

[QueryColumnDragging](#) event occurs when you start dragging the column header.

[QueryColumnDraggingEventArgs](#) of [QueryColumnDragging](#) event provides information about the column triggered this event.

[QueryColumnDraggingEventArgs.From](#) property returns the index of column triggered this event.

[QueryColumnDraggingEventArgs.To](#) property returns the index where you try to drop the column.

[QueryColumnDraggingEventArgs.Reason](#) returns column dragging details by

[QueryColumnDraggingReason](#).

### C#

```
this.dataGrid.QueryColumnDragging += dataGrid_QueryColumnDragging;
void dataGrid_QueryColumnDragging(object sender,
QueryColumnDraggingEventArgs e)
{
    var column = dataGrid.Columns[e.From];
    if (column.MappingName == "CustomerName" && e.Reason ==
QueryColumnDraggingReason.Dropping)
    {
        e.Cancel = true;
    }
}
```

### Column drag and drop customization

The drag-and-drop operations can be changed by overriding the virtual methods of [GridColumnDragDropController](#) class and assigning it to `SfDataGrid.GridColumnDragDropController`.

### C#

```
this.dataGrid.GridColumnDragDropController = new
CustomDragDropController(dataGrid);
public class CustomDragDropController:GridColumnDragDropController
{
    public CustomDragDropController(SfDataGrid dataGrid): base(dataGrid)
    {
    }
    //Returns whether the popup showed its header or not.
    public override bool CanShowPopup(GridColumn column)
    {
        return base.CanShowPopup(column);
    }
    //Get the corresponding GridRegion at a given point.
    public override GridRegion PointToGridRegion(System.Windows.Point point)
    {
        return base.PointToGridRegion(point);
    }
    //Occurs when the GridColumn.Hidden property value changed.
    protected override void OnColumnHiddenChanged(GridColumn column)
    {
        base.OnColumnHiddenChanged(column);
    }
    //Occurs when the popup content is created.
    protected override System.Windows.UIElement CreatePopupContent(GridColumn
column)
    {
        return base.CreatePopupContent(column);
    }
    //Occurs when the popup content is dropped on DataGrid.
    protected override void PopupContentDroppedOnGrid(System.Windows.Point
point)
    {
        base.PopupContentDroppedOnGrid(point);
    }
    //Occurs when the popup content is dropped on header row.
    protected override void PopupContentDroppedOnHeaderRow(int oldIndex, int
newColumnIndex)
    {
        base.PopupContentDroppedOnHeaderRow(oldIndex, newIndex);
    }
    //Occurs when the popup content is dropped.
    protected override void OnPopupContentDropped(System.Windows.Point point,
System.Windows.Point pointOverGrid)
    {
        base.OnPopupContentDropped(point, pointOverGrid);
    }
    //Occurs when the popup content is dropped on GroupDropArea
    protected override void PopupContentDroppedOnGroupDropArea(GridColumn
column)
```

```

{
    base.PopupContentDroppedOnGroupDropArea(column);
}
//Occurs when the popup content position changed.
protected override void OnPopupContentPositionChanged(double
HorizontalDelta, double VerticalDelta, System.Windows.Point mousePoint,
System.Windows.Point mousePointOverGrid)
{
    base.OnPopupContentPositionChanged(HorizontalDelta, VerticalDelta,
mousePoint, mousePointOverGrid);
}
}

```

#### Disabling drag & drop between frozen and non-frozen columns

By default, the columns re-ordering performed between any column regions of columns. You can cancel the dropping action between the frozen and non-frozen columns by handling [SfDataGrid.QueryColumnDragging](#) event.

#### C#

```

this.dataGrid.QueryColumnDragging += dataGrid_QueryColumnDragging;
void dataGrid_QueryColumnDragging(object sender,
QueryColumnDraggingEventArgs e)
{
    if (e.Reason == QueryColumnDraggingReason.Dropping)
    {
        var frozenColIndex = this.dataGrid.FrozenColumnCount +
this.dataGrid.ResolveToStartColumnIndex();
        if (e.From < frozenColIndex && e.To > frozenColIndex - 1)
            e.Cancel = true;
        if (e.From > frozenColIndex && e.To < frozenColIndex || (e.From ==
frozenColIndex && e.To < frozenColIndex))
            e.Cancel = true;
    }
}

```

**Note:** `FrozenColumnCount` and `FooterColumnCount` should be lesser than the number of Columns that can be displayed in View.

#### DataGrid freeze columns

You can freeze the columns in view at the left and right side like in excel by setting [SfDataGrid.FrozenColumnCount](#) and [SfDataGrid.FooterColumnCount](#) properties.

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
FrozenColumnCount="2"
ItemsSource="{Binding Orders}"/>

```



```
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

### Column Types in WPF DataGrid (SfDataGrid)

SfDataGrid provides support for various built-in column types. Each column has its own properties and renderer to handle different types of data.

You can also add or override existing columns and renderers as you need.

Column Type	Renderer	Description
<a href="#">GridTextColumn</a>	<a href="#">GridCellTextBoxRenderer</a>	Use to display the string data.
<a href="#">GridNumericColumn</a>	<a href="#">GridCellNumericRenderer</a>	Use to display the numeric data.
<a href="#">GridCurrencyColumn</a>	<a href="#">GridCellCurrencyRenderer</a>	Use to display the currency value.
<a href="#">GridPercentColumn</a>	<a href="#">GridCellPercentageRenderer</a>	Use to display the percent value.
<a href="#">GridMaskColumn</a>	<a href="#">GridCellMaskRenderer</a>	Use to display the data to be masked.
<a href="#">GridTimeSpanColumn</a>	<a href="#">GridCellTimeSpanRenderer</a>	Use to display the time span value.
<a href="#">GridDateTimeColumn</a>	<a href="#">GridCellDateTimeRenderer</a>	Use to display the date time value.



<a href="#">GridComboBoxColumn</a>	<a href="#">GridCellComboBoxRenderer</a>	Use to display the IEnumerable data using ComboBox.
<a href="#">GridCheckBoxColumn</a>	<a href="#">GridCellCheckBoxRenderer</a>	Use to display the boolean type data.
<a href="#">GridImageColumn</a>	<a href="#">GridCellImageRenderer</a>	Use to display the image in each row.
<a href="#">GridHyperlinkColumn</a>	<a href="#">GridCellHyperLinkRenderer</a>	Use to display the URI data.
<a href="#">GridTemplateColumn</a>	<a href="#">GridCellTemplateRenderer</a>	Use to display the custom template-specified content.
<a href="#">GridUnboundColumn</a>	<a href="#">GridUnBoundCellTextBoxRendererGridUnBoundCellTemplateRenderer</a>	Use to display custom information of each record.
<a href="#">GridMultiColumnDropDownList</a>	<a href="#">GridCellMultiColumnDropDownRenderer</a>	Use to display the IEnumerable data using <a href="#">SfMultiColumnDropDownControl</a> .
<a href="#">GridCheckBoxSelectorColumn</a>	<a href="#">GridCellCheckBoxSelectorRenderer</a>	Selects or deselects rows based on the check box value, which is not bound with data object.

## GridColumn

[GridColumn](#) is an abstract class provides base functionalities for all the column types in SfDataGrid.

### Column mapping

Column can be bound to a property in data object using [GridColumn.MappingName](#) property. In addition, it supports to format or bind different property for display and edit mode separately via [GridColumn.DisplayBinding](#) and [GridColumn.ValueBinding](#).

When you set MappingName, DisplayBinding and ValueBinding are created based on MappingName, if these properties are not defined explicitly.

You can use DisplayBinding property to format the column in display, by setting StringFormat or Converter properties of Binding.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    ColumnSizer="Star"
    AutoGenerateColumns="False"
    ItemsSource="{Binding Orders}">
  <syncfusion:SfDataGrid.Columns>
    <syncfusion:GridTextColumn DisplayBinding="{Binding Path=UnitPrice,
```

```
StringFormat='{0:C}'"
HeaderText="Unit Price"
MappingName="UnitPrice"
ValueBinding="{Binding Path=Quantity}" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

In the below screenshot, **Unit Price** column display value is formatted to currency by setting **DisplayBinding** property.

OrderID	Unit Price	CustomerID	CustomerName	Country
1001	\$25.00	ALFKI	Maria Anders	Germany
1002	3	ANATR	Ana Trujillo	Mexico
1003	\$25.00	ANTON	Antonio Moreno	Mexico
1004	\$50.00	AROUT	Thomas Hardy	UK
1005	\$33.00	BERGS	Christina Berglund	Sweden
1006	\$50.00	BLAUS	Hanna Moos	Germany
1007	\$25.00	BLONP	Frédérique Citeaux	France
1008	\$33.00	BOLID	Martin Sommer	Spain
1009	\$25.00	BONAP	Laurence Lebihan	France
1010	\$50.00	BOTTM	Elizabeth Lincoln	Canada

By default, Columns handling the data operations (sorting and grouping) based on **MappingName** property. You can perform data operations based on **ValueBinding** by setting [GridColumn.UseBindingValue](#) to **true**, when the standard reflection not works or binding column with complex or indexer properties.

### Column CellTemplate

You can load any WPF control in the display mode for all columns by setting [GridColumn.CellTemplate](#) property. In edit mode, corresponding editor will be loaded based on column type.

Using **CellTemplate**, you can format data or conditionally change the properties using [DataTrigger](#).

In the below code snippet, **GridNumericColumn** is loaded with **ProgressBar** and **TextBlock**. When you start editing **DoubleTextBox** will be loaded as Editor.

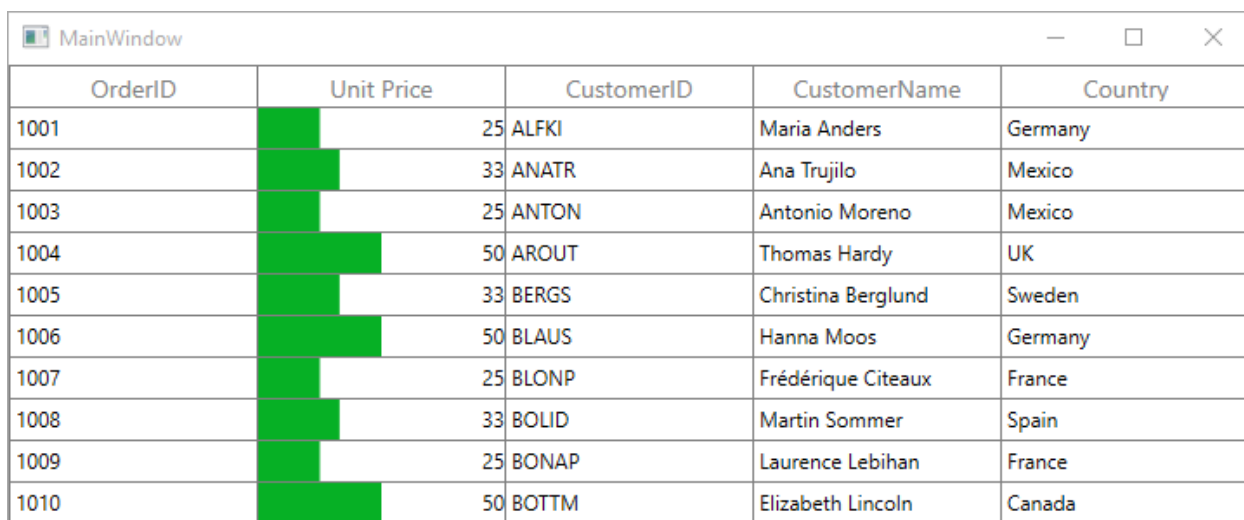
### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ColumnSizer="Star"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridNumericColumn HeaderText="Unit Price"
MappingName="UnitPrice">
<syncfusion:GridNumericColumn.CellTemplate>
<DataTemplate>
<Grid>
<ProgressBar x:Name="progressBar"
Background="Transparent"
Visibility="Visible"
```

```

Minimum="0"
Maximum="100"
BorderThickness="0"
Value="{Binding Path=UnitPrice}" />
<TextBlock Text="{Binding Path=UnitPrice}"
HorizontalAlignment="Right"
VerticalAlignment="Center"
TextAlignment="Center" />
</Grid>
</DataTemplate>
</syncfusion:GridNumericColumn.CellTemplate>
</syncfusion:GridNumericColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```



OrderID	Unit Price	CustomerID	CustomerName	Country
1001	25	ALFKI	Maria Anders	Germany
1002	33	ANATR	Ana Trujilo	Mexico
1003	25	ANTON	Antonio Moreno	Mexico
1004	50	AROUT	Thomas Hardy	UK
1005	33	BERGS	Christina Berglund	Sweden
1006	50	BLAUS	Hanna Moos	Germany
1007	25	BLONP	Frédérique Citeaux	France
1008	33	BOLID	Martin Sommer	Spain
1009	25	BONAP	Laurence Lebihan	France
1010	50	BOTTM	Elizabeth Lincoln	Canada

CellTemplate is not support by GridHyperlinkColumn, GridCheckboxColumn and GridImageColumn columns.

#### Reusing same DataTemplate for multiple columns

By default, underlying record is DataContext for CellTemplate. So you have to define, template for each column to display values based on MappingName.

You can use the same [DataTemplate](#) for all columns to display value based on MappingName by setting [GridColumn.SetCellBoundValue](#) property to true. Setting SetCellBoundValue to true, changes the DataContext for CellTemplate to [DataContextHelper](#) which has the following members,

- Value - Return the value base on MappingName.
- Record - Returns the underlying data object.

#### XML

```

<Window.Resources>
<DataTemplate x:Key="cellTemplate">
<TextBlock Foreground="Red"
Margin="3,0,0,0"
Text="{Binding Path=Value}" />

```

```

</DataTemplate>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ColumnSizer="Star"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn SetCellBoundValue="True"
MappingName="OrderID"
CellTemplate="{StaticResource cellTemplate}" />
<syncfusion:GridTextColumn MappingName="CustomerName"
SetCellBoundValue="True"
CellTemplate="{StaticResource cellTemplate}" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```



OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

Setting CellTemplate based on custom logic using template selector

**GridColumn** provides support to choose different [DataTemplate](#) based on underlying data object using [GridColumn.CellTemplateSelector](#) property.

For example, two different templates loaded alternatively in **OrderID** column.

#### XML

```

<Window.Resources>
<local:CustomCellTemplateSelector x:Key="cellTemplateSelector"/>
<DataTemplate x:Key="DefaultTemplate">
<TextBlock Background="Wheat"
Foreground="Red"
Text="{Binding Path=OrderID}"
TextAlignment="Center" />
</DataTemplate>
<DataTemplate x:Key="AlternateTemplate">
<TextBlock Background="Beige"
Foreground="Green"
Text="{Binding Path=OrderID}"
TextAlignment="Center" />
</DataTemplate>

```

```
</Window.Resources>
```

Below code returns the `DefaultTemplate` and `AlternateTemplate` based on `OrderID`'s value

.

#### C#

```
public class CustomCellTemplateSelector : DataTemplateSelector
{
    public override System.Windows.DataTemplate SelectTemplate(object item,
        System.Windows.DependencyObject container)
    {
        if (item == null)
            return null ;
        var data = item as OrderInfo;
        if (data.OrderID % 2 == 0)
            return Application.Current.MainWindow.FindResource("AlternateTemplate") as
                DataTemplate;
        else
            return Application.Current.MainWindow.FindResource("DefaultTemplate") as
                DataTemplate;
    }
}
```

In the below code, the custom template selector set to `GridColumn.CellTemplateSelector` and set `SetCellBoundValue` to `true`.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    ColumnSizer="Star"
    AutoGenerateColumns="False"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.Columns>
        <syncfusion:GridTextColumn MappingName="OrderID"
            CellTemplateSelector="{StaticResource cellTemplateSelector}"/>
    </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

**Note:** Non-Editable columns does not support `CellTemplate`.

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

#### Binding ViewModel properties with CellTemplate

You can bind properties in ViewModel with the controls in CellTemplate.

Below command defined in ViewModel is bound to **Button** inside CellTemplate. Below code, denote the base command.

#### C#

```
public class BaseCommand: ICommand
{
    #region Fields
    readonly Action<object> _execute;
    readonly Predicate<object> _canExecute;
    #endregion
    #region Constructors
    /// <summary>
    /// Creates a new command that always execute.
    /// </summary>
    /// <param name="execute">The execution logic.</param>
    public BaseCommand(Action<object> execute)
    : this(execute, null)
    {
    }
    /// <summary>
    /// Creates a new command.
    /// </summary>
    /// <param name="execute">The execution logic.</param>
    /// <param name="canExecute">The execution status logic.</param>
    public BaseCommand(Action<object> execute, Predicate<object> canExecute)
    {
        if (execute == null)
            throw new ArgumentNullException("execute");
        _execute = execute;
        _canExecute = canExecute;
    }
    #endregion
    #region ICommand Members
    public bool CanExecute(object parameter)
    {
    }
```

```

return _canExecute == null ? true : _canExecute(parameter);
}
public event EventHandler CanExecuteChanged
{
    add { CommandManager.RequerySuggested += value; }
    remove { CommandManager.RequerySuggested -= value; }
}
public void Execute(object parameter)
{
    _execute(parameter);
}
#endregion
}

```

Below code, defines the command for **Button** in **ViewModel**.

### C#

```

public class ViewModel
{
    private BaseCommand deleteRecord;
    public BaseCommand DeleteRecord
    {
        get
        {
            if (deleteRecord == null)
                deleteRecord = new BaseCommand(OnDeleteRecordClicked, OnCanDelete);
            return deleteRecord;
        }
    }
    private static bool OnCanDelete(object obj)
    {
        return true;
    }
    private void OnDeleteRecordClicked(object obj)
    {
        //TODO ACTION.
    }
}

```

In the below code, Button inside CellTemplate bound to the command in ViewModel.

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="False"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.Columns>
        <syncfusion:GridTextColumn MappingName="Delete">
            <syncfusion:GridTextColumn.CellTemplate>
                <DataTemplate>
                    <Button Command="{Binding DataContext.DeleteRecord, ElementName=dataGrid}"
                        CommandParameter="{Binding}"
                        Content="Delete" />
                <!--or-->
                    <Button Command="{Binding DeleteRecord,

```

```

Source={StaticResource viewModel}}}"
CommandParameter="{Binding}"
Content="Delete" />
</DataTemplate>
</syncfusion:GridTextColumn.CellTemplate>
</syncfusion:GridTextColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

### Column Formatting

GridColumn supports to format the data using [StringFormat](#) and [Converter](#) properties, by defining GridColumn.DisplayBinding and GridColumn.ValueBinding. GridColumn.DisplayBinding formats the data in display mode. GridColumn.ValueBinding formats the data in edit mode.

#### Column formatting using Binding

You can apply format for the column using [StringFormat](#) property by defining DisplayBinding. StringFormat applies to GridTextColumn alone. Refer the [Converter](#) section to format the other column types.

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
ColumnSizer="Star"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn DisplayBinding="{Binding Path=UnitPrice,
StringFormat='{0:C}'}"
HeaderText="Unit Price"
MappingName="UnitPrice"
ValueBinding="{Binding Path=UnitPrice}" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

When column is auto-generated, you can set the [StringFormat](#) by handling [AutoGeneratingColumn](#) event.

### C#

```

void dataGrid_AutoGeneratingColumn(object sender, AutoGeneratingColumnArgs e)
{
    if (e.Column.MappingName == "UnitPrice")
    {
        if (e.Column is GridNumericColumn)
        {
            e.Column = new GridTextColumn() { MappingName = "UnitPrice", HeaderText = "Unit Price" };
        }
        e.Column.DisplayBinding = new Binding() { Path = new PropertyPath(e.Column.MappingName), StringFormat = "{0:C}" };
        e.Column.ValueBinding = new Binding() { Path = new PropertyPath(e.Column.MappingName), StringFormat="{0:C}" };
    }
}

```



```
}

```

### Column formatting using converter

You can format the column using **Converter** property by defining **DisplayBinding**.

#### XML

```
<Window.Resources>
<local:CurrencyFormatConverter x:Key="currencyFormatConverter" />
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridNumericColumn DisplayBinding="{Binding Path=UnitPrice,
Converter={StaticResource currencyFormatConverter}}"
HeaderText="Unit Price"
MappingName="UnitPrice" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### C#

```
public class CurrencyFormatConverter:IValueConverter
{
    object IValueConverter.Convert(object value, Type targetType, object
parameter, System.Globalization.CultureInfo culture)
    {
        return string.Format("{0:C}", value);
    }
    object IValueConverter.ConvertBack(object value, Type targetType, object
parameter, System.Globalization.CultureInfo culture)
    {
        return value;
    }
}
```

When column is auto-generated, you can set the **Converter** by handling **AutoGeneratingColumn** event

#### C#

```
void dataGrid_AutoGeneratingColumn(object sender, AutoGeneratingColumnArgs
e)
{
    if (e.Column.MappingName == "UnitPrice")
    {
        if (e.Column is GridNumericColumn)
        {
            e.Column = new GridTextColumn() { MappingName = "UnitPrice", HeaderText =
"Unit Price" };
        }
        e.Column.DisplayBinding = new Binding() { Path = new
PropertyPath(e.Column.MappingName), Converter = new
CurrencyFormatConverter() };
    }
}
```

```
}
}
```

OrderID	Unit Price	CustomerID	CustomerName	Country
1001	\$25.00	ALFKI	Maria Anders	Germany
1002	\$33.00	ANATR	Ana Trujilo	Mexico
1003	\$25.00	ANTON	Antonio Moreno	Mexico
1004	\$50.00	AROUT	Thomas Hardy	UK
1005	\$33.00	BERGS	Christina Berglund	Sweden
1006	\$50.00	BLAUS	Hanna Moos	Germany
1007	\$25.00	BLONP	Frédérique Citeaux	France
1008	\$33.00	BOLID	Martin Sommer	Spain
1009	\$25.00	BONAP	Laurence Lebihan	France
1010	\$50.00	BOTTM	Elizabeth Lincoln	Canada

### Column styling

GridColumn support to customize the style of particular column using [GridColumn.CellStyle](#) property. For more information, refer [Styling and Template](#) section.

#### Change the column font setting

You can change the font settings such as FontSize, FontFamily, FontWeight etc. by writing style of TargetType [GridCell](#) or [GridColumn.CellStyle](#) property.

### XML

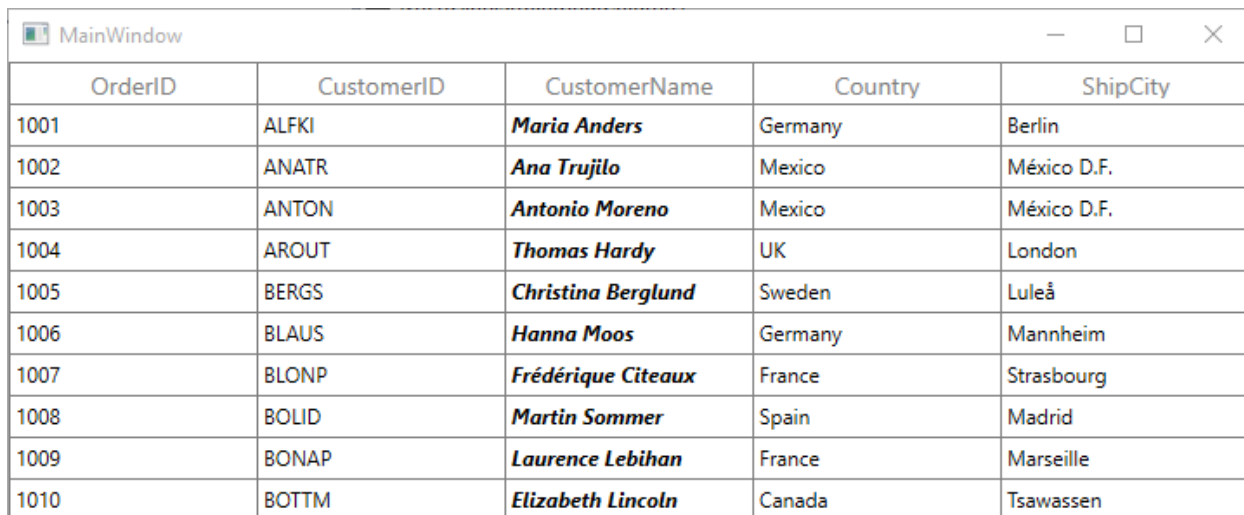
```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="CustomerName">
<syncfusion:GridTextColumn.CellStyle>
<Style TargetType="syncfusion:GridCell">
<Setter Property="FontSize" Value="12" />
<Setter Property="FontFamily" Value="Segoe UI" />
<Setter Property="FontWeight" Value="Bold" />
<Setter Property="FontStyle" Value="Italic" />
<Setter Property="FontStretch" Value="Condensed" />
</Style>
</syncfusion:GridTextColumn.CellStyle>
</syncfusion:GridTextColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

When column is auto-generated, you can style the column by handling [AutoGeneratingColumn](#) event

### C#

```
this.dataGrid.AutoGeneratingColumn += dataGrid_AutoGeneratingColumn;
void dataGrid_AutoGeneratingColumn(object sender, AutoGeneratingColumnArgs e)
```

```
{
    if (e.Column.MappingName == "CustomerName")
    e.Column.CellStyle = this.FindResource("cellStyle") as Style;
}
```



OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

#### Styles based on custom logic

You can apply the styles to columns based on certain condition using [GridColumn.CellStyleSelector](#) property.

Below code creates two different styles by TargetType `GridCell`.

#### XML

```
<Application.Resources>
<Style x:Key="cellStyle1" TargetType="syncfusion:GridCell">
<Setter Property="Background" Value="Bisque" />
</Style>
<Style x:Key="cellStyle2" TargetType="syncfusion:GridCell">
<Setter Property="Background" Value="Aqua" />
</Style>
</Application.Resources>
```

In the below code, returns the style based on `OrderID` value. Using `Container` you can format the columns data based on `GridCell`.

#### C#

```
public class CustomCellStyleSelector : StyleSelector
{
    public override Style SelectStyle(object item,
    System.Windows.DependencyObject container)
    {
        var gridCell = container as GridCell;
        var mappingName = gridCell.ColumnBase.GridColumn.MappingName;
        var record = gridCell.DataContext;
        var cellValue = record.GetType().GetProperty(mappingName).GetValue(record);
        if (mappingName.Equals("OrderID"))
        {

```

```

if (Convert.ToInt16(cellValue) <= 1005)
return App.Current.Resources["cellStyle1"] as Style;
else
return App.Current.Resources["cellStyle2"] as Style;
}
return base.SelectStyle(item, container);
}
}

```

Below code, sets the customized style selector to `GridColumn.CellStyleSelector` property.

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
AutoGenerateColumns="True" >
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="OrderID"
CellStyleSelector="{StaticResource cellStyleSelector}"/>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

When column is auto-generated, you can style the column by handling `AutoGeneratingColumn` event

#### C#

```

this.dataGrid.AutoGeneratingColumn += dataGrid_AutoGeneratingColumn;
void dataGrid_AutoGeneratingColumn(object sender, AutoGeneratingColumnArgs
e)
{
if (e.Column.MappingName == "OrderID")
{
e.Column.CellStyleSelector = new CustomCellStyleSelector();
}
}
}

```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

## End-user interaction

### Hide Column

You can hide or unhide the particular column programmatically by setting [GridColumn.IsHidden](#) property. For allowing end-user to hide or unhide column in UI refer [Resizing Columns](#) section.

### Disable column

You can disable column by setting [GridColumn.AllowFocus](#) property. Therefore, that column can't be selected or edited.

## Column width, alignment and padding

### Column width

The width of `GridColumn` can be changed by setting [Width](#) property. Column width set based on [GridColumn.MinimumWidth](#) and [GridColumn.MaximumWidth](#) properties.

---

**Note:** If the `GridColumn.Width` is defined explicitly takes priority than `GridColumn.ColumnSizer`.

---

### Column padding

`GridColumn` allows you to change the padding of cell content by setting [Padding](#) property.

### Column alignment

`GridColumn` allows you to change the alignment of `GridCell` and `GridHeaderCell` content using [TextAlignment](#), [VerticalAlignment](#) and [HorizontalHeaderContentAlignment](#) properties.

## GridTextColumnBase

[GridTextColumnBase](#) is the abstract class derived from `GridColumn`. The following columns are derived from the `GridTextColumnBase`.

1. `GridTextColumn`
2. `GridDateTimeColumn`
3. `GridTimeSpanColumn`
4. `GridMaskColumn`
5. `GridTemplateColumn`
6. `GridMultiColumnDropDownList`

### GridTextColumnBase properties

- **Text decorations** - You can [decorate](#) column's data using [TextDecorations](#) property.
- **Text trimming** - You can [trim](#) the column's data using [TextTrimming](#) property.
- **Text wrapping** - You can [wrap](#) the column's data using [TextWrapping](#) property.

## XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn HeaderText="Customer Name"
MappingName="CustomerName"
TextDecorations="StrikeThrough"
TextTrimming="WordEllipsis"
TextWrapping="Wrap" />
</syncfusion:SfDataGrid.Columns>
```

```
</syncfusion:SfDataGrid>
```

OrderID	CustomerID	Customer Name	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

### GridEditorColumn

[GridEditorColumn](#) is the abstract class derived from [GridTextColumnBase](#). The following columns derived from [GridEditorColumn](#) class.

1. [GridNumericColumn](#)
2. [GridCurrencyColumn](#)
3. [GridPercentColumn](#)

### Auto increment

You can allow end-user to increment or decrement the column value's when [MouseWheel](#) or pressing up and down arrow keys by setting [AllowScrollingOnCircle](#) property to [true](#).

### Null value support

[GridEditorColumn](#) provides support to restrict or allow null value in columns based on [AllowNullValue](#) property. Instead of displaying null values, you can display hint text using [NullText](#) property or you can set the default value when null value encountered using [NullValue](#) property.

The [NullText](#) and [NullValue](#) properties won't work, when the [AllowNullValue](#) is [false](#).

### Setting input range

You can restrict and display the input value with in the range using [MinValue](#) and [MaxValue](#) properties. You can set the constraint to validate the maximum value using [MaxValidation](#) and minimum value using [MinValidation](#) properties. Below are the two constraints that specify when to validate.

- [LostFocus](#) - Validates input when the focus is lost.
- [OnKeyPress](#) - Validates input, when user type the value.

### GridTextColumn

[GridTextColumn](#) derived from [GridTextColumnBase](#) which hosts [TextBox](#) in edit mode.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid">
```

```

AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn HeaderText="Customer Name"
MappingName="CustomerName" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

**C#**

```

this.dataGrid.Columns.Add(new GridTextColumn()
{
    HeaderText = "Customer Name",
    MappingName = "CustomerName",
});

```

**GridNumericColumn**

**GridNumericColumn** derived from **GridEditorColumn** which displays columns data as numeric. It hosts **DoubleTextBox** in editing mode.

**XML**

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridNumericColumn HeaderText="Quantity" MappingName="Quantity"
/>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

**C#**

```

this.dataGrid.Columns.Add(new GridNumericColumn() { MappingName =
"Quantity", HeaderText = "Quantity" });

```

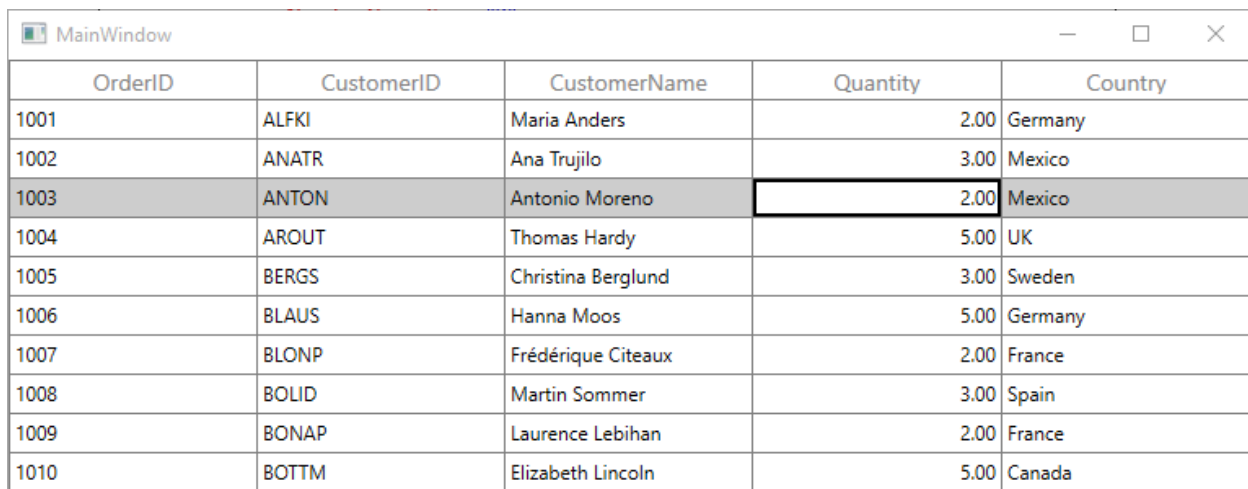
*Data formatting*

**GridNumericColumn** allows you to format the numeric data with culture-specific information.

- [NumberDecimalDigits](#) - You can change the [Number of decimal digits](#) to be displayed after the decimal point using **NumberDecimalDigits** property.
- [NumberDecimalSeparator](#) - By default, the dot (.) operator [separates the decimal part](#) of numeric value. You can use any operator as decimal separator using **NumberDecimalSeparator** property.
- [NumberGroupSeparator](#) - By default, the comma (,) [separates group of digits](#) before the decimal point. You can use any operator as group separator using **NumberGroupSeparator** property.
- [NumberGroupSizes](#) - You can change the [number of digits in each group](#) before the decimal point on numeric values using **NumberGroupSizes** property.
- **Formatting negative pattern** - You can format the [pattern of negative](#) numeric values using **NumberNegativePattern**.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridNumericColumn HeaderText="Quantity"
MappingName="Quantity"
NumberDecimalDigits="2"
NumberDecimalSeparator="."
NumberGroupSeparator=","
NumberGroupSizes="3"/>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```



OrderID	CustomerID	CustomerName	Quantity	Country
1001	ALFKI	Maria Anders	2.00	Germany
1002	ANATR	Ana Trujilo	3.00	Mexico
1003	ANTON	Antonio Moreno	2.00	Mexico
1004	AROUT	Thomas Hardy	5.00	UK
1005	BERGS	Christina Berglund	3.00	Sweden
1006	BLAUS	Hanna Moos	5.00	Germany
1007	BLONP	Frédérique Citeaux	2.00	France
1008	BOLID	Martin Sommer	3.00	Spain
1009	BONAP	Laurence Lebihan	2.00	France
1010	BOTTM	Elizabeth Lincoln	5.00	Canada

*ParsingMode support*

GridNumericColumn process int, double and decimal data types. By default, it treats all values as double internally and return the same. When you are binding dynamic property, we need to maintain the type which can be achieved by setting [GridNumericColumn.ParsingMode](#) property.

For example, you have dynamic object with property OrderID type int. When you edit the OrderID using GridNumericColumn, its type will be changed as double. To avoid this, you must set ParsingMode property based on the type of property you are binding to GridNumericColumn or you need to use ValueBinding converter change type according to your underlying property type.

**XML**

```
<Syncfusion:GridNumericColumn ParsingMode="Int" MappingName="OrderID"
HeaderText="OrderID"/>
```

**C#**

```
datagrid.Columns.Add(new GridNumericColumn() { MappingName = "OrderID",
ParsingMode = ParseMode.Int });
```



### GridCurrencyColumn

GridCurrencyColumn derived from GridEditorColumn and it displays columns data as currency. It hosts CurrencyTextBox element in editing mode.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridCurrencyColumn HeaderText="Unit Price"
MappingName="UnitPrice" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.Columns.Add(new GridCurrencyColumn() { MappingName =
"UnitPrice", HeaderText = "Unit Price" });
```

### Data formatting

GridCurrencyColumn allows you to format the parsing currency data with culture-specific information.

- [CurrencySymbol](#) - By default, the currency symbol will be displayed based on culture. You can change the symbol using CurrencySymbol property.
- [CurrencyDecimalDigits](#) - You can change the [number of decimal digits](#) to be displayed after the decimal point on currency values using CurrencyDecimalDigits property.
- [CurrencyDecimalSeparator](#) - By default, the dot (.) operator [separates the decimal part](#) of currency value .You can use any operator as decimal separator through CurrencyDecimalSeparator property.
- [CurrencyGroupSeparator](#) - By default, the comma (,) [separates the group](#) of digits before the decimal point on currency value .You can use any operator as group separator through CurrencyGroupSeparator property.
- [CurrencyGroupSizes](#) - You can specify [the number of digits in each group](#) before the decimal point on currency value using CurrencyGroupSizes property.
- [Pattern](#) - You can format the pattern for both [positive](#) and [negative](#) currency values through [CurrencyPositivePattern](#) and [CurrencyNegativePattern](#).

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridCurrencyColumn CurrencyDecimalDigits="2"
CurrencyDecimalSeparator="."
CurrencyGroupSeparator=","
CurrencyGroupSizes="3"
CurrencyPositivePattern="1"
CurrencySymbol="$"
HeaderText="Unit Price"
```

```
MappingName="UnitPrice" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

OrderID	CustomerID	CustomerName	Unit Price	Country
1001	ALFKI	Maria Anders	25.00\$	Germany
1002	ANATR	Ana Trujillo	33.00\$	Mexico
1003	ANTON	Antonio Moreno	25.00\$	Mexico
1004	AROUT	Thomas Hardy	50.00\$	UK
1005	BERGS	Christina Berglund	33.00\$	Sweden
1006	BLAUS	Hanna Moos	50.00\$	Germany
1007	BLONP	Frédérique Citeaux	25.00\$	France
1008	BOLID	Martin Sommer	33.00\$	Spain
1009	BONAP	Laurence Lebihan	25.00\$	France
1010	BOTTM	Elizabeth Lincoln	50.00\$	Canada

### GridPercentColumn

**GridPercentColumn** derived from **GridEditorColumn** and it displays columns data as percent. It hosts **PercentTextBox** element in editing mode.

You can display data as percent value or double value using **PercentEditMode** property.

**PercentEditMode.PercentMode** returns the value as percentage. **PercentEditMode.DoubleMode** returns the value as numeric.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridPercentColumn HeaderText="Discount" MappingName="Discount"
/>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#

```
this.dataGrid.Columns.Add(new GridPercentColumn() { HeaderText = "Discount",
MappingName = "Discount" });
```

### Data formatting

**GridPercentColumn** allows you to format the parsing percent data with culture-specific information.

- **PercentSymbol** - By default, the percent operator (%) will be loaded with the value. You can change the symbol using **PercentSymbol** property.
- **PercentDecimalDigits** - You can change the **number of decimal digits** to be displayed after the decimal point on percent value can be specified using **PercentDecimalDigits** property.

- [PercentDecimalSeparator](#) - By default, the dot (.) operator [separates the decimal part](#) of percent value .You can use any operator as decimal separator using `PercentDecimalSeparator` property.
- [PercentGroupSeparator](#) - By default, the comma (,) operator [separates the group](#) of digits left to the decimal point on currency value .You can use any operator as group separator using `PercentGroupSeparator` property.
- [PercentGroupSizes](#) - You can specify [the number of digits in each group](#) before the decimal point through `PercentGroupSizes` property.
- `Pattern` - You can specify the pattern for both [positive](#) and [negative](#) percent values through [PercentPositivePattern](#) and [PercentNegativePattern](#).

## XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridPercentColumn HeaderText="Discount"
MappingName="Discount"
PercentDecimalDigits="2"
PercentDecimalSeparator="."
PercentEditMode="PercentMode"
PercentGroupSeparator=","
PercentGroupSizes="2"
PercentPositivePattern="0" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

OrderID	CustomerID	CustomerName	Discount	Country
1001	ALFKI	Maria Anders	1,00.00 %	Germany
1002	ANATR	Ana Trujilo	0.00 %	Mexico
1003	ANTON	Antonio Moreno	3,00.00 %	Mexico
1004	AROUT	Thomas Hardy	0.00 %	UK
1005	BERGS	Christina Berglund	1,00.00 %	Sweden
1006	BLAUS	Hanna Moos	2,00.00 %	Germany
1007	BLONP	Frédérique Citeaux	3,00.00 %	France
1008	BOLID	Martin Sommer	0.00 %	Spain
1009	BONAP	Laurence Lebihan	1,00.00 %	France
1010	BOTTM	Elizabeth Lincoln	2,00.00 %	Canada

## GridDateTimeColumn

`GridDateTimeColumn` derived from `GridTextColumnBase` and it displays columns data as date time. It hosts `DateTimeEdit` element in editing mode.

## XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
```

```
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridDateTimeColumn HeaderText="Order Date"
MappingName="OrderDate" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

**C#**

```
this.dataGrid.Columns.Add(new GridDateTimeColumn() { HeaderText = "Order
Date", MappingName = "OrderDate" });
```

*Change the pattern of date time value*

You can format the date time value using [GridDateTimeColumn.Pattern](#) property, which contains the set of predefined date time patterns,

Pattern	Expected
LongDate	Monday, June 15, 2016
LongTime	1:45:30 PM
ShortDate	6/15/2016
ShortTime	1:45 PM
FullDateTime	Monday, June 15, 2016 1:45 PM
MonthDay	June 15 <sup>th</sup>
RFC1123	Mon, 15 Jun 2016 20:45:30 GMT
SortableDateTime	2016-06-15T13:45:30
UniversalSortableDateTime	2016-06-15 13:45:30Z
YearMonth	June, 2016

When the predefined **Pattern** does not meet your requirement, you can set the custom pattern using [CustomPattern](#) property.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridDateTimeColumn CustomPattern="dd-m-yyyy hh:mm:ss"
HeaderText="Order Date"
MappingName="OrderDate"
Pattern="CustomPattern" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

You can also change the format of standard date time pattern such as short pattern, long pattern, etc. by using [GridDateTimeColumn.DateTimeFormat](#) property.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridDateTimeColumn HeaderText="Order Date"
MappingName="OrderDate"
Pattern="LongDate">
<syncfusion:GridDateTimeColumn.DateTimeFormat>
<global:DateTimeFormatInfo LongDatePattern="dd-MM-yyyy hh:mm:ss" />
</syncfusion:GridDateTimeColumn.DateTimeFormat>
</syncfusion:GridDateTimeColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### Auto increment

You can allow end-user to increment or decrement the value when **MouseWheel** or pressing up and down arrow keys by setting [AllowScrollingOnCircle](#) property to **true**.

#### Null value support

**GridDateTimeColumn** provides support to restrict or allow null value in columns based on [AllowNullValue](#) property. Instead of displaying null values, you can display hint text using [NullText](#) property or you can set the default value using [NullValue](#) property.

The **NullText** and **NullValue** properties won't work, when the **AllowNullValue** is **false**.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridDateTimeColumn AllowNullValue="True"
HeaderText="Order Date"
MappingName="OrderDate"
NullValue="07/05/2010" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### Setting date time value range

You can restrict and display the input value within the range using [MinDateTime](#) and [MaxDateTime](#) properties.

#### Editing support

By default, the user can input the date time value by selecting through Calendar in dropdown. You allow users to input or delete the date time value from the key board by setting [GridDateTimeColumn.CanEdit](#) to **true**.

*Disable date selection*

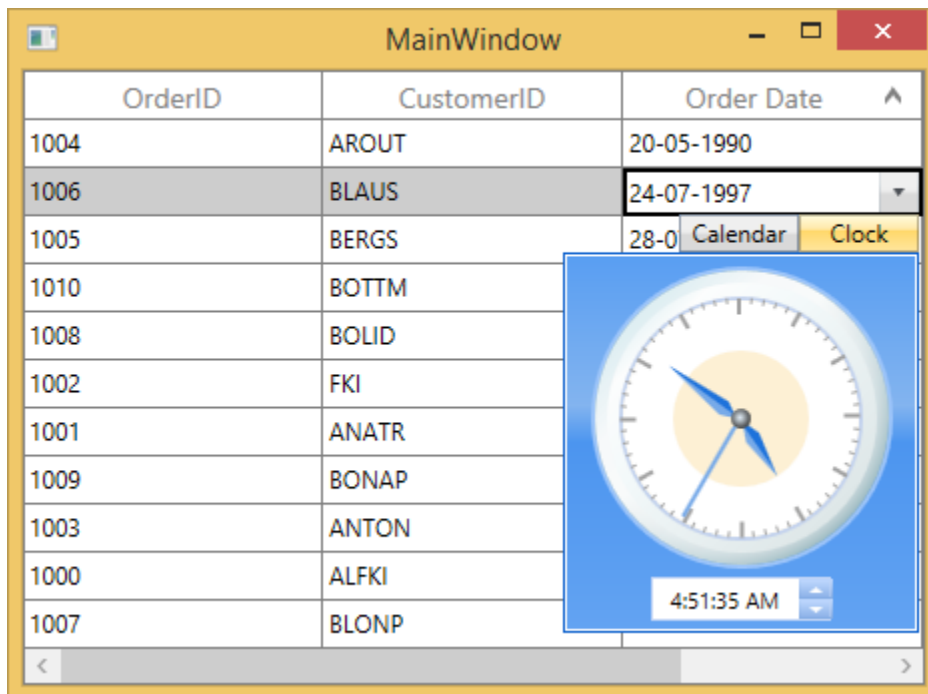
You can disable the date selection on dropdown popup by setting the [DisableDateSelection](#) property to true, so dropdown calendar displays months to select.

*EnableBackSpaceKey and EnableDeleteKey*

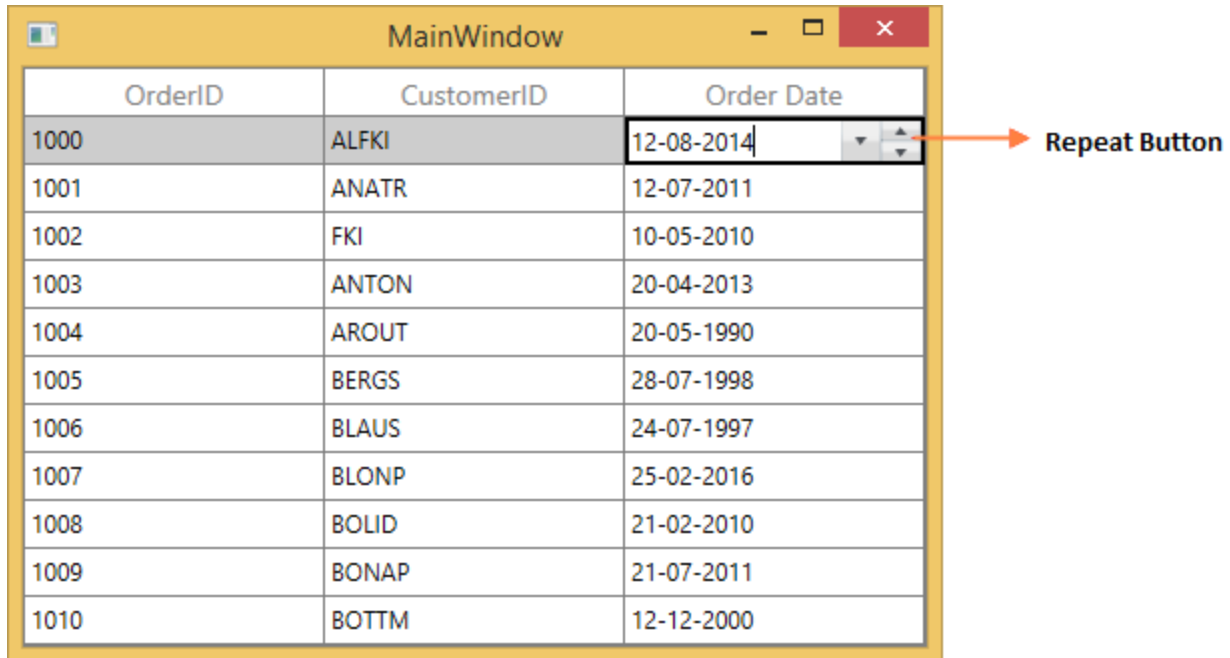
You can delete using backspace and delete keys by setting [EnableBackSpaceKey](#) and [EnableDeleteKey](#) properties to true.

*Enable clock and calendar*

By default, the Calendar displayed in dropdown popup. You can enable both Calendar and Clock control in dropdown popup by setting the [EnableClassicStyle](#) to true.

*Show repeat button*

You can increment or decrement the selected part of date time value by enabling the repeat button through [GridDateTimeColumn.ShowRepeatButton](#) property.



OrderID	CustomerID	Order Date
1000	ALFKI	12-08-2014
1001	ANATR	12-07-2011
1002	FKI	10-05-2010
1003	ANTON	20-04-2013
1004	AROUT	20-05-1990
1005	BERGS	28-07-1998
1006	BLAUS	24-07-1997
1007	BLONP	25-02-2016
1008	BOLID	21-02-2010
1009	BONAP	21-07-2011
1010	BOTTM	12-12-2000

#### Format using Converter

`GridDateTimeColumn` allows you to set different cultures by setting [ConverterCulture](#) property in [DisplayBinding](#) and [ValueBinding](#).

#### XML

```
<Window.Resources>
<local:CultureFormatConverter x:Key="converter"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridDateTimeColumn DisplayBinding="{Binding Path=OrderDate,
ConverterCulture=fr-FR,
Converter={StaticResource converter}}"
HeaderText="Order Date"
ValueBinding="{Binding Path=OrderDate,
ConverterCulture=fr-FR}" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

**Note:** By default, short date pattern is applied in `GridDateTimeColumn`.

Below code, returns the culture-specified date time value instead of default pattern.

#### C#

```
public class CultureFormatConverter: IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
return value;
}
```

```

}
public object ConvertBack(object value, Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
{
    throw new NotImplementedException();
}
}

```

### GridCheckBoxColumn

**GridCheckBoxColumn** derived from **GridColumn** and it used display and edit **Boolean** type data. It hosts **CheckBox** element as **GridCell** content.

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="False"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.Columns>
    <syncfusion:GridCheckBoxColumn HeaderText="Is Delivered"
        MappingName="IsDelivered" />
    </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

#### C#

```

this.dataGrid.Columns.Add(new GridCheckBoxColumn() { HeaderText = "Is
Delivered", MappingName = "IsDelivered" });

```

**GridCheckBoxColumn** allows you to customize check box state and its alignment.

- [IsThreeState](#) - By default, the **GridCheckBoxColumn** has **Checked** and **Unchecked** state. You can enable another **Intermediate** state setting **IsThreeState** property to **true**.
- [HorizontalAlignment](#) - You can change the horizontal alignment of **CheckBox** using **HorizontalAlignment** property.

### GridTemplateColumn

**GridTemplateColumn** derived from **GridTextColumnBase** and it displays the template-specified cell content. You can load any WPF control in the display mode for all columns by setting [CellTemplate](#) and [EditTemplate](#) properties.

Using **CellTemplate**, you can format data or conditionally change the properties using [DataTrigger](#).

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="False"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.Columns>
    <syncfusion:GridTemplateColumn MappingName="CustomerID">
    <syncfusion:GridTemplateColumn.CellTemplate>
    <DataTemplate>
    <TextBlock Text="{Binding CustomerID}" />
    </DataTemplate>
    </syncfusion:GridTemplateColumn>
    </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```



```

</syncfusion:GridTemplateColumn.CellTemplate>
<syncfusion:GridTemplateColumn.EditTemplate>
<DataTemplate>
<TextBox Text="{Binding CustomerID, Mode=TwoWay}" />
</DataTemplate>
</syncfusion:GridTemplateColumn.EditTemplate>
</syncfusion:GridTemplateColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

## C#

```

//CellTemplate creation.
DataTemplate cellTemplate = new DataTemplate();
FrameworkElementFactory frameworkElement1 = new
FrameworkElementFactory(typeof(TextBlock));
Binding displayBinding = new Binding() { Path = new
PropertyPath("CustomerID") };
frameworkElement1.SetValue(TextBlock.TextProperty, displayBinding);
cellTemplate.VisualTree = frameworkElement1;
//EditTemplate creation.
DataTemplate editTemplate = new DataTemplate();
FrameworkElementFactory frameworkElement2 = new
FrameworkElementFactory(typeof(TextBox));
Binding editBinding = new Binding() { Path = new PropertyPath("CustomerID"),
Mode = BindingMode.TwoWay };
frameworkElement2.SetValue(TextBox.TextProperty, editBinding);
editTemplate.VisualTree = frameworkElement2;
this.dataGrid.Columns.Add(new GridTemplateColumn() { HeaderText = "Customer
ID", MappingName = "CustomerID", CellTemplate = cellTemplate, EditTemplate =
editTemplate, SetCellBoundValue = false });

```

### Mouse interaction for UIElement loaded inside template

You can allow **UIElement** loaded inside **CellTemplate** or **EditTemplate** to handle mouse interaction in required cases by setting [VisualContainer.WantsMouseInput](#) attached property to the particular **UIElement** inside template.

## XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTemplateColumn MappingName="ProductName">
<syncfusion:GridTemplateColumn.CellTemplate>
<DataTemplate>
<ComboBox ItemsSource="{Binding ComboItems, Source={StaticResource
viewModel}}" syncfusion:VisualContainer.WantsMouseInput="True" />
</DataTemplate>
</syncfusion:GridTemplateColumn.CellTemplate>
</syncfusion:GridTemplateColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

*Keyboard interaction for UIElement loaded inside CellTemplate*

You can allow **UIElement** loaded inside **CellTemplate** to handle keyboard interaction by setting [FocusManagerHelper.WantsKeyInput](#) attached property to **GridColumn**. You can use this when loading edit element in CellTemplate.

In this case SfDataGrid handles the below key operations and other keys are handled by UIElement loaded inside **CellTemplate**.

- Tab
- Enter
- PageUp
- PageDown

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTemplateColumn MappingName="CustomerName"
syncfusion:FocusManagerHelper.WantsKeyInput="True">
<syncfusion:GridTemplateColumn.CellTemplate>
<TextBox Text="{Binding Country}" />
</syncfusion:GridTemplateColumn.CellTemplate>
</syncfusion:GridTemplateColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

*Setting focus to particular element inside Template when cell gets activated or edited*

You can allow logical focus to specific UIElement loaded inside **EditTemplate** or **CellTemplate** by setting [FocusManagerHelper.FocusedElement](#) attached property. You can use this property to start editing the template column value as like normal column when the user gets into edit mode.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTemplateColumn MappingName="CustomerName">
<syncfusion:GridTemplateColumn.CellTemplate>
<DataTemplate>
<Grid>
<TextBlock Text="{Binding CustomerName}" />
</Grid>
</DataTemplate>
</syncfusion:GridTemplateColumn.CellTemplate>
<syncfusion:GridTemplateColumn.EditTemplate>
<DataTemplate>
<Grid>
<TextBox Text="{Binding CustomerName}"
syncfusion:FocusManagerHelper.FocusedElement="True" />
</Grid>
</DataTemplate>
</syncfusion:GridTemplateColumn.EditTemplate>
</syncfusion:GridTemplateColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

```

</syncfusion:GridTemplateColumn.EditTemplate>
</syncfusion:GridTemplateColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

#### *Binding CellTemplate and EditTemplate based on MappingName*

By default, underlying record is `DataContext` for `CellTemplate`. So you have to define, template for each column to display values based on `MappingName`.

You can use the same [DataTemplate](#) for all columns to display value based on `MappingName` by setting [SetCellBoundValue](#) property to `true`. Setting `SetCellBoundValue` to `true`, changes the `DataContext` for `CellTemplate` to [DataContextHelper](#) which has the following members,

- `Value` - Return the value base on `MappingName`.
- `Record` - Returns the underlying data object.

---

**Note:** `EditTemplate` support available only for `GridTemplateColumn`.

---

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTemplateColumn MappingName="OrderID"
SetCellBoundValue="True">
<syncfusion:GridTemplateColumn.CellTemplate>
<DataTemplate>
<Grid>
<TextBlock Text="{Binding Value}" />
</Grid>
</DataTemplate>
</syncfusion:GridTemplateColumn.CellTemplate>
<syncfusion:GridTemplateColumn.EditTemplate>
<DataTemplate>
<Grid>
<TextBox Text="{Binding Value}" />
</Grid>
</DataTemplate>
</syncfusion:GridTemplateColumn.EditTemplate>
</syncfusion:GridTemplateColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

#### *Sets EditTemplate based on custom logic*

`GridTemplateColumn` provides support to load different edit elements based on underlying data object using [GridTemplateColumn.EditTemplateSelector](#) property.

Below code returns the `DefaultTemplate` and `AlternateTemplate` based on `OrderID`'s value.

#### C#

```

public class CustomEditTemplateSelector:DataTemplateSelector
{

```

```

public override System.Windows.DataTemplate SelectTemplate(object item,
System.Windows.DependencyObject container)
{
    if (item == null)
        return null ;
    var data = item as OrderInfo;
    if (data.OrderID % 2 == 0)
        return App.Current.Resources["AlternateTemplate"] as DataTemplate;
    else
        return App.Current.Resources["DefaultTemplate"] as DataTemplate;
}
}

```

In the below code, custom template selector set to `GridTemplateColumn.EditTemplateSelector`.

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.Columns>
        <syncfusion:GridTemplateColumn MappingName="OrderID"
CellTemplateSelector="{StaticResource cellTemplateSelector}"
EditTemplateSelector="{StaticResource editTemplateSelector}"/>
    </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

OrderID	CustomerID	CustomerName	Country
1001	ALFKI	Maria Anders	Germany
1002	ANATR	Ana Trujilo	Mexico
1003	ANTON	Antonio Moreno	Mexico
1004	AROUT	Thomas Hardy	UK
1005	BERGS	Christina Berglund	Sweden
1006	BLAUS	Hanna Moos	Germany
1007	BLONP	Frédérique Citeaux	France
1008	BOLID	Martin Sommer	Spain
1009	BONAP	Laurence Lebihan	France
1010	BOTTM	Elizabeth Lincoln	Canada

### GridComboBoxColumn

`GridComboBoxColumn` derived from `GridColumn` which hosts `ComboBox` as edit element. You can enable editing in `ComboBox` by setting `IsEditable` property to `true`. The data source to `ComboBox` can be set by using `GridComboBoxColumn.ItemsSource` property.

By default, `GridComboBoxColumn` displays the value using `MappingName` property. You can set `DisplayMemberPath` which denotes the path to a value on the source object(`GridComboBoxColumn.ItemsSource`) to serve as visual representation of object. You can set `SelectedValuePath` which denotes the path to get the `SelectedValue` from the `SelectedItem`.

### XML

```

<Window.Resources>
<local:ViewModel x:key="viewModel"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridComboBoxColumn HeaderText="Product Name"
ItemsSource="{Binding ComboItems,
Source={StaticResource viewModel}}"
MappingName="ProductName" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

### C#

```

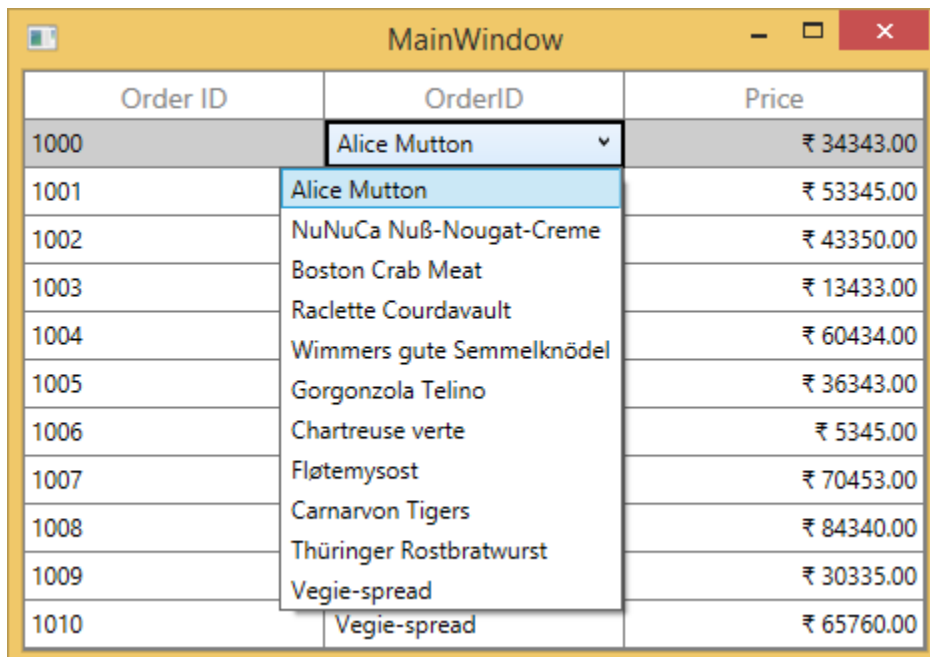
this.dataGrid.Columns.Add(new GridComboBoxColumn() { HeaderText = "Product
Name", MappingName = "ProductName", ItemsSource = viewModel.ComboItems });

```

SfDataGrid triggers, [CurrentCellDropDownSelectionChanged](#) event, when the SelectedValue is changed. [CurrentCellDropDownSelectionChangedEventArgs](#) of [CurrentCellDropDownSelectionChanged](#) event provides the information about the changed cell value.

[SelectedIndex](#) property returns the index of selected item.

[SelectedItem](#) property returns the selected item from drop down list.



*Keep the dropdown to be opened*

You can keep the drop-down control open when start editing on the text box of [ComboBox](#) also by setting [StaysOpenOnEdit](#) property to [true](#).

*Improving dropdown opening time*

You can improve the drop-down opening time on loading by setting [VirtualizingStackPanel][https://docs.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/ms617901\(v=vs.95\)\)](https://docs.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/ms617901(v=vs.95))) as [ItemsPanelTemplate](#) of [ComboBox](#), when the large number of items loaded in it.

**XML**

```
<Window.Resources>
<Style TargetType="ComboBox">
<Setter Property="ItemsPanel">
<Setter.Value>
<ItemsPanelTemplate>
<VirtualizingStackPanel />
</ItemsPanelTemplate>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>
```

*Opening dropdown popup in single-click*

You can open the drop down within single click by setting [ComboBox.IsDropDownOpen](#) property to `true` in [OnEditElementLoaded](#) method by overriding existing renderer.

Below code, creates [GridCellComboBoxRendererExt](#) to set [IsDropDownOpen](#) property. Replace the default renderer with created renderer in [SfDataGrid.CellRenderers](#) collection.

**C#**

```
this.dataGrid.CellRenderers.Remove("ComboBox");
this.dataGrid.CellRenderers.Add("ComboBox", new
GridCellComboBoxRendererExt());
public class GridCellComboBoxRendererExt:GridCellComboBoxRenderer
{
protected override void OnEditElementLoaded(object sender,
System.Windows.RoutedEventArgs e)
{
base.OnEditElementLoaded(sender, e);
var combobox = sender as ComboBox;
combobox.IsDropDownOpen = true;
}
}
```

**Note:** This is applicable when the [SfDataGrid.EditTrigger](#) is set as `OnTap`.

*Customizing GroupCaptionText based on DisplayMemberPath*

By default, the [GroupCaptionText](#) will be displayed based on [MappingName](#). You can display the [GroupCaptionText](#) based on [DisplayMemberPath](#) using [GroupColumnDescription.Converter](#) property.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
```

```
<syncfusion:GridComboBoxColumn DisplayMemberPath="OrderID"
HeaderText="Order ID"
IsEditable="True"
ItemsSource="{Binding ComboItems}"
MappingName="ProductName"
SelectedValuePath="ProductName" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

Below code returns the group caption based on `GridComboBoxColumn.ItemsSource`.

#### C#

```
public class GroupCaptionConverter: IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        if (!(parameter is GridComboBoxColumn))
            return value;
        var column = parameter as GridComboBoxColumn;
        var record = value as OrderInfo;
        foreach (var item in column.ItemsSource)
        {
            if (record.ProductName == (item as OrderDetails).ProductName)
                return (item as OrderDetails).OrderID;
        }
        return null;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

In the below code snippet, `GroupCaptionConverter` set to `GroupColumnDescription.Converter` while grouping.

#### C#

```
this.dataGrid.GroupColumnDescriptions.CollectionChanged +=
    GroupColumnDescriptions_CollectionChanged;
void GroupColumnDescriptions_CollectionChanged(object sender,
    System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
{
    if (e.Action == NotifyCollectionChangedAction.Add)
    {
        var groupDescription = e.NewItems[0] as GroupColumnDescription;
        if (groupDescription.ColumnName == "ProductName")
            groupDescription.Converter = new GroupCaptionConverter();
    }
}
```

*Loading Different ItemSource for each row of GridComboBoxColumn*

You can load the different ItemsSource to each row of GridComboBoxColumn by setting [SfDataGrid.ItemsSourceSelector](#) property.

*Implementing IItemsSourceSelector*

ItemsSourceSelector needs to implement [IItemsSourceSelector](#) interface which requires you to implement [GetItemsSource](#) method which receives the below parameters,

```
<ul>
<li> <b>Record</b> – data object associated with row.</li>
<li> <b>Data Context</b> – Data context of data grid.</li>
</ul>
```

In the below code, ItemsSource for ShipCity column returned based on ShipCountry column value using the record and data context of data grid passed to [GetItemsSource](#) method.

**XML**

```
<Window.Resources>
<local:ItemsSourceSelector x:Key="itemSourceSelector" />
</Window.Resources>
<syncfusion:SfDataGrid x:Name="sfdatagrid"
AllowEditing="True"
AutoGenerateColumns="False"
ItemsSource="{Binding OrderDetails}"
ColumnSizer="Star">
<interactivity:Interaction.Behaviors>
<local:ItemsSourceSelectorBehavior />
</interactivity:Interaction.Behaviors>
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridComboBoxColumn MappingName="ShipCountry"
ItemsSource="{Binding Path=DataContext.CountryList,
ElementName=sfdatagrid}" />
<syncfusion:GridComboBoxColumn HeaderText="ShipCity"
DisplayMemberPath="ShipCityName"
ItemsSourceSelector="{StaticResource itemSourceSelector}"
MappingName="ShipCityID" SelectedValuePath="ShipCityID" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

**C#**

```
/// <summary>
/// Implementation class for ItemsSourceSelector interface
/// </summary>
public class ItemsSourceSelector : IItemsSourceSelector
{
    public IEnumerable GetItemsSource(object record, object dataContext)
    {
        if (record == null)
            return null;
        var orderinfo = record as OrderDetails;
        var countryName = orderinfo.ShipCountry;
        var viewModel = dataContext as ViewModel;
```



```
//Returns ShipCity collection based on ShipCountry.
if (viewModel.ShipCities.ContainsKey(countryName))
{
    ObservableCollection<ShipCityDetails> shipCities = null;
    viewModel.ShipCities.TryGetValue(countryName, out shipCities);
    return shipCities.ToList();
}
return null;
}
```

The following screenshot illustrates the different ShipCity ItemsSource bound to each row of the ComboBox based on the Country Name.

OrderID	CustomerID	ProductName	NoOfOrders	ShipCountry	ShipCity
1000	FOLIG	NuNuCa Nub-Nouga	10	Austria	Cork
1001	FRANS	Alice Mutton	15	Austria	Århus
1002	RISCU	Konbu	20	Belgium	austriaAachen
1003	WARTH	NuNuCa Nub-Nouga	25	Brazil	Cork
1004	FURIB	Konbu	20	Canada	Århus
1005	ALFKI	Konbu	17	Denmark	Montréal
1006	LINOD	Raclette Courdavault	14	Finland	Graz
1007	LINOD	Konbu	11	Italy	Bruxelles
1008	RISCU	Wimmers gute Semm	7	US	Campinas
1009	FOLKO	Konbu	5	Belgium	Campinas
1010	WARTH	Alice Mutton	3	Brazil	Aachen
1011	FOLIG	Raclette Courdavault	7	Denmark	Bruxelles
1012	ALFKI	Konbu	13	Argentina	Graz

OrderID	CustomerID	ProductName	NoOfOrders	ShipCountry	ShipCity
1000	WELLI	Boston Crab Meat	10	Argentina	Bergamo
1001	FURIB	Konbu	15	Austria	austriaAachen
1002	ALFKI	Raclette Courdavault	20	Belgium	Campinas
1003	VAFFE	Raclette Courdavault	25	Brazil	Bruxelles
1004	VAFFE	Alice Mutton	20	Canada	Campinas
1005	ALFKI	Alice Mutton	17	Denmark	Lille
1006	PICCO	Boston Crab Meat	14	Finland	Bergamo
1007	FOLKO	Raclette Courdavault	11	Italy	Bruxelles
1008	SEVES	Konbu	7	US	Campinas
1009	FURIB	Boston Crab Meat	5	Belgium	Campinas
1010	BLONP	Raclette Courdavault	3	Brazil	Aachen
1011	RISCU	Alice Mutton	7	Denmark	Bruxelles
1012	SEVES	Alice Mutton	13	Argentina	Graz

You can download the sample from [here](#).

### GridMultiColumnDropDownList

GridMultiColumnDropDownList derived from GridTextColumnBase and it displays enumeration as cell contents. It hosts [SfMultiDropDownControl](#) in editing mode. GridMultiColumnDropDownList allows you to define the predefined columns in its drop-down like SfDataGrid.

You can change the value by selecting the item from drop down or you can edit the SfMultiColumnDropDownControl.Editor. You can disable the editing by setting [IsTextReadOnly](#).

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridMultiColumnDropDownList AutoGenerateColumns="False"
DisplayMember="CustomerID"
HeaderText="Customer ID"
ItemsSource="{Binding Orders}"
MappingName="OrderID"
ValueMember="OrderID">
<syncfusion:GridMultiColumnDropDownList.Columns>
<syncfusion:Columns>
<syncfusion:GridTextColumn HeaderText="Customer ID" MappingName="CustomerID"
/>
<syncfusion:GridTextColumn HeaderText="Order ID" MappingName="OrderID" />
</syncfusion:Columns>
</syncfusion:GridMultiColumnDropDownList.Columns>
</syncfusion:GridMultiColumnDropDownList>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#

```
Columns columns = new Columns();
columns.Add(new GridTextColumn() { HeaderText = "Order ID", MappingName =
"OrderID" });
columns.Add(new GridTextColumn() { HeaderText = "Customer ID", MappingName =
"CustomerID" });
this.dataGrid.Columns.Add(new GridMultiColumnDropDownList() { ItemsSource =
viewModel.Orders, MappingName = "OrderID", HeaderText = "Customer ID",
DisplayMember = "CustomerID", ValueMember = "OrderID", AutoGenerateColumns =
false, Columns = columns });
```

SfDataGrid triggers, [CurrentCellDropDownSelectionChanged](#) event, when the SelectedValue is changed. [CurrentCellDropDownSelectionChangedEventArgs](#) of [CurrentCellDropDownSelectionChanged](#) event provides the information about the changed cell value.

[SelectedIndex](#) property returns the index of selected item.

[SelectedItem](#) property returns the selected item from drop down list.

Product Name	Customer ID	Price
Alice Mutton	ALFKI	₹ 34343.00
NuNuCa Nuß-Not		₹ 53345.00
Boston Crab Meat	ALFKI	₹ 43350.00
Raclette Courdav	ANATR	₹ 13433.00
Wimmers gute Se	FKI	₹ 60434.00
Gorgonzola Telino	ANTON	₹ 36343.00
Chartreuse verte	AROUT	₹ 5345.00
Fløtemysost	BERGS	₹ 70453.00
Carnarvon Tigers	BLAUS	₹ 84340.00
Thüringer Rostbr	BLONP	₹ 30335.00
Vegie-spread	BOLID	₹ 65760.00
	BONAP	

#### Auto-complete support

You can allow SfMultiDropDownControl to complete the entered input value automatically by setting the [AllowAutoComplete](#) property to true. AllowAutoComplete property will autocomplete the input value by comparing it to each item of the underlying data source of GridMultiColumnDropDownList and returns the matched value by DisplayMember.

#### Filtering

You can allow SfMultiDropDownControl to filter the drop-down list items dynamically based on the text typed on editor by setting [AllowIncrementalFiltering](#) property to true.

GridMultiColumnDropDownList allows you to filter the items based on case by setting [AllowCasingforFilter](#) to true. This will help users to select from large number of items.

Product Name	Customer ID	Price
Alice Mutton	a	₹ 34343.00
NuNuCa Nuß-Not		₹ 53345.00
Boston Crab Meat	ALFKI	₹ 43350.00
Raclette Courdav	ANATR	₹ 13433.00
Wimmers gute Se	ANTON	₹ 60434.00
Gorgonzola Telino	AROUT	₹ 36343.00
Chartreuse verte		₹ 5345.00
Fløtemysost		₹ 70453.00
Carnarvon Tigers		₹ 84340.00
Thüringer Rostbr		₹ 30335.00
Vegie-spread		₹ 65760.00

#### Auto increment

You can increment or decrement the value via the Mouse-Wheel or up and down key by setting [AllowSpinOnMouseWheel](#) to true.

#### Null value support

You can allow the null values by setting the [AllowNullInput](#) property to true.

---

**Note:** The AllowNullInput will work only when the underlying property type is Nullable.

---

#### Popup customization

##### Size

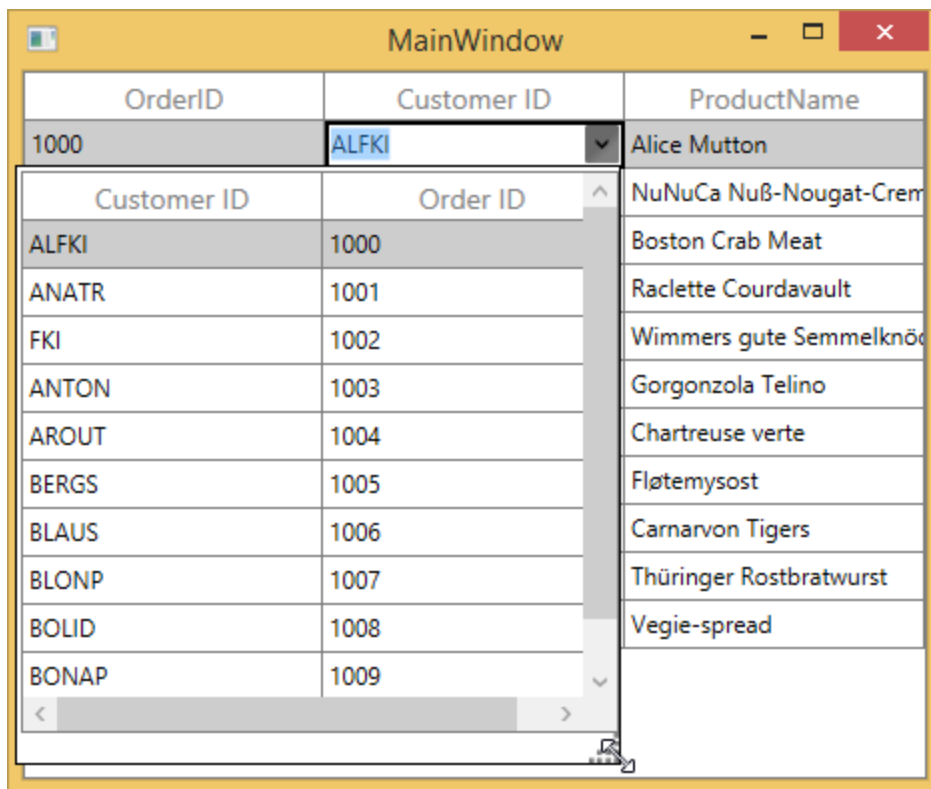
You can allow change the size of drop-down popup by setting [PopUpWidth](#) and [PopUpHeight](#) property. The [PopUpWidth](#) will be set based on [PopUpMinWidth](#) and [PopUpMaxWidth](#) properties when the value is not between them. By default, the [PopUpMinWidth](#) value is 200.0.

Similarly, the [PopUpHeight](#) based on [PopUpMinHeight](#) and [PopUpMaxHeight](#) when the value is not between them. The default value of [PopUpMinHeight](#) is 300.0.

SfMultiDropDownControl can automatically adjust the popup width based on the actual size of SfDataGrid when setting [IsAutoPopupSize](#) to true.

##### Resizing drop-down popup

You can allow end-user to resize the drop-down popup by showing resizing thumb by setting [ShowResizeThumb](#) property to Visible.



#### Keep popup open on editing

You can open the drop-down popup, when the user start editing on text editor using [AllowImmediatePopup](#) property to `true` in [OnEditElementLoaded](#) method by overriding existing renderer.

Below code, creates `GridCellMultiColumnDropDownRendererExt` to set `AllowImmediatePopup` property. Replace the default renderer with created renderer in [SfDataGrid.CellRenderers](#) collection.

#### C#

```
this.dataGrid.CellRenderers.Remove("MultiColumnDropDown");
this.dataGrid.CellRenderers.Add("MultiColumnDropDown", new
GridCellMultiColumnDropDownRendererExt());
public class GridCellMultiColumnDropDownRendererExt :
GridCellMultiColumnDropDownRenderer
{
    protected override void OnEditElementLoaded(object sender,
    System.Windows.RoutedEventArgs e)
    {
        (sender as SfMultiColumnDropDownControl).AllowImmediatePopup = true;
        (sender as SfMultiColumnDropDownControl).Text = PreviewInputText;
        base.OnEditElementLoaded(sender, e);
    }
}
```

#### Loading different ItemsSource for each row

You can load different `ItemsSource` to each row of `GridMultiColumnDropDownList` by setting the `SfDataGrid.ItemsSourceSelector` property.

*Implementing ItemsSourceSelector*

ItemsSourceSelector needs to implement the IItemsSourceSelector interface, which is required to implement the GetItemsSource method. The GetItemsSource method receives the following parameters:

- <ul>
- <li> <b>Record</b> – Data object associated with row.</li>
- <li> <b>Data Context</b> – Data context of data grid.</li>
- </ul>

In the following code, ItemsSource for ShipCity column is returned based on ShipCountry column value using the record and data context of data grid passed to the GetItemsSource method.

**XML**

```
<Window.Resources>
<local:ItemsSourceSelector x:Key="itemSourceSelector" />
</Window.Resources>
<syncfusion:SfDataGrid x:Name="sfdatagrid"
AllowEditing="True"
AutoGenerateColumns="False"
AllowFiltering="False"
ItemsSource="{Binding OrderDetails}"
ColumnSizer="Star">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="OrderID" />
<syncfusion:GridTextColumn MappingName="CustomerID" />
<syncfusion:GridTextColumn MappingName="NoOfOrders" />
<syncfusion:GridComboBoxColumn MappingName="ShipCountry"
ItemsSource="{Binding Path=DataContext.CountryList,
ElementName=sfdatagrid}"/>
<syncfusion:GridMultiColumnDropDownList AllowEditing="True"
HeaderText="ShipCity" DisplayMember="ShipCityName"
ItemsSourceSelector="{StaticResource itemSourceSelector}"
MappingName="ShipCityID" ValueMember="ShipCityID"/>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

**C#**

```
/// <summary>
/// Implementation class for ItemsSourceSelector interface
/// </summary>
public class ItemsSourceSelector : IItemsSourceSelector
{
public IEnumerable GetItemsSource(object record, object dataContext)
{
if (record == null)
return null;
var orderinfo = record as OrderDetails;
var countryName = orderinfo.ShipCountry;
var viewModel = dataContext as ViewModel;
//Returns ShipCity collection based on ShipCountry.
if (viewModel.ShipCities.ContainsKey(countryName))
```

```

{
    ObservableCollection<ShipCityDetails> shipCities = null;
    viewModel.ShipCities.TryGetValue(countryName, out shipCities);
    return shipCities.ToList();
}
return null;
}
}

```

The following screenshot illustrates different **ShipCity** ItemsSource bound to each row of the **MultiColumnDropDownList** based on country name.

OrderID	CustomerID	NoOfOrders	ShipCountry	ShipCity
1000	MEREP	10	Argentina	Bergamo
1001	LINOD	15	Austria	Århus
1002	FRANS	20	Belgium	ShipCityID ShipCityName
1003	PICCO	25	Brazil	106 austriaAachen
1004	SEVES	20	Canada	107 Cork
1005	ALFKI	17	Denmark	108 Århus
1006	SIMOB	14	Finland	109 Montréal
1007	MEREP	11	Italy	110 Graz
1008	PICCO	7	US	
1009	FRANS	5	Belgium	
1010	FOLKO	3	Brazil	
1011	SIMOB	7	Denmark	
1012	FOLIG	13	Argentina	
1013	PICCO	12	Canada	
1014	PICCO	5	Finland	Montréal
1015	VAFFE	1	US	Bruxelles
1016	WARTH	6	Austria	Cork
1017	SEVES	9	Italy	Montréal
1018	FOLIG	4	Denmark	Bergamo

OrderID	CustomerID	NoOfOrders	ShipCountry	ShipCity
1000	MEREP	10	Argentina	Bergamo
1001	LINOD	15	Austria	Århus
1002	FRANS	20	Belgium	Bruxelles
1003	PICCO	25	Brazil	ShipCityID ShipCityName
1004	SEVES	20	Canada	111 Bruxelles
1005	ALFKI	17	Denmark	112 Campinas
1006	SIMOB	14	Finland	113 Lille
1007	MEREP	11	Italy	114 Bergamo
1008	PICCO	7	US	
1009	FRANS	5	Belgium	
1010	FOLKO	3	Brazil	
1011	SIMOB	7	Denmark	
1012	FOLIG	13	Argentina	
1013	PICCO	12	Canada	
1014	PICCO	5	Finland	
1015	VAFFE	1	US	bruxelles
1016	WARTH	6	Austria	Cork
1017	SEVES	9	Italy	Montréal
1018	FOLIG	4	Denmark	Bergamo

You can download the sample from the following link: [Sample](#).

### GridHyperlinkColumn

GridHyperlinkColumn derived from GridTextColumn and it displays columns data as HyperLinkControl. It hosts TextBlock element as GridCell content.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridHyperlinkColumn HeaderText="Country Link"
MappingName="Country" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.Columns.Add(new GridHyperlinkColumn() { HeaderText = "Country
Link", MappingName = "Country" });
```

You can allow end-user to navigate the Uri when the cell value contains valid Uri address or using [CurrentCellRequestNavigate](#) event. The CurrentCellRequestNavigate occurs when the current cell in GridHyperlinkColumn is clicked for navigation.

[CurrentCellRequestNavigateEventArgs](#) of CurrentCellRequestNavigate event provide information about the hyperlink triggered this event. CurrentCellRequestNavigateEventArgs.NavigateText returns the value using ValueBinding or MappingName to navigate.

#### C#

```
this.dataGrid.CurrentCellRequestNavigate +=
dataGrid_CurrentCellRequestNavigate;
void dataGrid_CurrentCellRequestNavigate(object sender,
CurrentCellRequestNavigateEventArgs args)
{
string address = "https://en.wikipedia.org/wiki/" + args.NavigateText;
Process.Start(new ProcessStartInfo(address));
}
```

#### Cancel the navigation

You can cancel the navigation by setting [CurrentCellRequestNavigateEventArgs.Handled](#) to true.

#### C#

```
this.dataGrid.CurrentCellRequestNavigate +=
dataGrid_CurrentCellRequestNavigate;
void dataGrid_CurrentCellRequestNavigate(object sender,
CurrentCellRequestNavigateEventArgs args)
{
args.Handled = true;
}
```



*Customize HyperLink**Change the alignment*

You can change the horizontal alignment of `GridHyperlinkColumn` using [HorizontalAlignment](#) property.

*Change the foreground color*

You can change the foreground color of `GridHyperlinkColumn` by writing the style with target type `Hyperlink`.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowEditing="True"
    AutoGenerateColumns="False"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.Columns>
    <syncfusion:GridHyperlinkColumn MappingName="HyperLink">
    <syncfusion:GridHyperlinkColumn.CellStyle>
    <Style>
    <Style.Resources>
    <Style TargetType="Hyperlink">
    <Setter Property="Foreground" Value="Purple" />
    </Style>
    </Style.Resources>
    </Style>
    </syncfusion:GridHyperlinkColumn.CellStyle>
    </syncfusion:GridHyperlinkColumn>
    </syncfusion:SfDataGrid.Columns>
    </syncfusion:SfDataGrid>
```

*GridImageColumn*

`GridImageColumn` derived from `GridColumn` and it displays columns data as `Image`. It hosts `Image` element as `GridCell` content.

**C#**

```
public class OrderInfo
{
    private BitmapImage _imageLink;
    public BitmapImage ImageLink
    {
        get { return _imageLink; }
        set { _imageLink = value; }
    }
    public OrderInfo(int orderId, string customerName, string country, string
        customerId, string shipCity)
    {
        ImageLink = new System.Windows.Media.Imaging.BitmapImage(new
            Uri(string.Format(@"..\..\Images\{0}", this.Country + ".jpg"),
            UriKind.Relative));
    }
}
```

In the below code `GridImageColumn` defined with `ImageLink` property using `MappingName`

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridImageColumn MappingName="ImageLink"
HeaderText="Image"
Stretch="Uniform"
TextAlignment="Center" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

Read image from path using ValueBinding

You can use converter to read image from resource by setting Converter in ValueBinding definition.

Below code, returns the Image URI using ValueBinding property.

### C#

```
public class StringToImageConverter:IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        string imageName = value as string+".jpg";
        return new BitmapImage(new Uri(string.Format(@"..\..\Images\{0}",
            imageName), UriKind.Relative));
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        return null;
    }
}
```

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridImageColumn MappingName="Country"
HeaderText="Image"
Stretch="Uniform"
TextAlignment="Center"
ValueBinding="{Binding Path=Country,
Converter={StaticResource ImageConverter}}" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#

```
this.dataGrid.Columns.Add(new GridImageColumn() { HeaderText = "Image",
MappingName = "Country", ValueBinding = new Binding() { Path = new
PropertyPath("Country"), Converter = new StringToImageConverter() },
TextAlignment = TextAlignment.Center, Stretch = Stretch.Uniform });
```

### Customize Image

**GridImageColumn** allows you to customize the image with below properties.

- **Width and Height**- You can change the height and width of the image using [GridImageColumn.ImageHeight](#) and [GridImageColumn.ImageWidth](#) properties.
- **Stretch** - The image can be stretch by setting **Stretch** property.
- **Scale** - You can scale the image using [StretchDirection](#) property.

### GridMaskColumn

**GridMaskColumn** derived from **GridTextColumnBase** and it displays columns data with specified mask pattern. It hosts **MaskedTextBox** element in editing mode.

You can set the input mask at runtime by setting [GridMaskColumn.Mask](#) property.

For example,

In the below code snippet, **Mask** applied to format and validate the user input to get five digit numeric value for phone number.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridMaskColumn HeaderText="Contact Number"
MappingName="ContactNumber"
Mask="(99)999" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.Columns.Add(new GridMaskColumn() { HeaderText = "Contact
Number", MappingName = "ContactNumber", Mask = "(99)999" });
```

OrderID	CustomerID	CustomerName	Contact Number	Country
1001	ALFKI	Maria Anders	(97)865	Germany
1002	ANATR	Ana Trujillo	(93)865	Mexico
1003	ANTON	Antonio Moreno	(92)347	Mexico
1004	AROUT	Thomas Hardy	(96)875	UK
1005	BERGS	Christina Berglund	(97)505	Sweden
1006	BLAUS	Hanna Moos	(97)234	Germany
1007	BLONP	Frédérique Citeaux	(90)067	France
1008	BOLID	Martin Sommer	(97)509	Spain
1009	BONAP	Laurence Lebihan	(96)645	France
1010	BOTTM	Elizabeth Lincoln	(95)777	Canada

Mask for numeric value not exceeds two digits to the left of the decimal point.

In the below code snippet, `Mask` applied to format and validate the user input to enter one digit double value.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridMaskColumn MappingName="UnitPrice" Mask="#.0" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

OrderID	CustomerID	CustomerName	UnitPrice	Country
1001	ALFKI	Maria Anders	2.5	Germany
1002	ANATR	Ana Trujillo	3.3	Mexico
1003	ANTON	Antonio Moreno	2.5	Mexico
1004	AROUT	Thomas Hardy	5.0	UK
1005	BERGS	Christina Berglund	3.3	Sweden
1006	BLAUS	Hanna Moos	5.0	Germany
1007	BLONP	Frédérique Citeaux	2.5	France
1008	BOLID	Martin Sommer	3.3	Spain
1009	BONAP	Laurence Lebihan	2.5	France
1010	BOTTM	Elizabeth Lincoln	5.0	Canada

#### Specifying prompt character

By default, an underscore (`_`) is displayed when the user input is absent. This can be changed by setting [GridMaskColumn.PromptChar](#) property.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridMaskColumn MappingName="OrderID"
Mask="#####"
PromptChar="^" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### Setting mask format

You can decide whether the value contains the literals and prompt characters in it through [GridMaskColumn.MaskFormat](#) property.

The `GridMaskColumn.MaskFormat` property has the following format.

- `ExcludePromptAndLiterals`

- IncludePrompt
- IncludeLiterals
- IncludePromptAndLiterals

### GridTimeSpanColumn

**GridTimeSpanColumn** derived from **GridTextColumnBase** and it displays columns data as time span. It hosts **TimeSpanEdit** element in editing mode.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTimeSpanColumn HeaderText="Delivery Time"
MappingName="DeliveryTime" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.Columns.Add(new GridTimeSpanColumn() { HeaderText =
"DeliveryTime", MappingName = "DeliveryTime" });
```

#### Auto increment

You can allow end-user to increment or decrement the column value's when **MouseWheel** or pressing up arrow keys by setting **AllowScrollingOnCircle** property to true. You can also use **ArrowButtons** to change the value by setting **ShowArrowButtons** to true.

#### Null value support

**GridTimeSpanColumn** provides support to restrict or allow null value in columns based on **AllowNull** property. Instead of displaying null values, you can display hint text using **NullText** property.

The **NullText** properties won't work, when the **AllowNull** is false.

#### Setting input value range

You can restrict and display the input value with in the range using **MinValue** and **MaxValue** properties.

#### Data formatting

You can format the time span values by setting **Format** property.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTimeSpanColumn Format="d 'days' h'hours'"
HeaderText="Delivery Time"
MappingName="DeliveryTime" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

OrderID	CustomerID	CustomerName	Delivery Time	Country
1001	ALFKI	Maria Anders	3 days 6hours	Germany
1002	ANATR	Ana Trujilo	8 days 6hours	Mexico
1003	ANTON	Antonio Moreno	3 days 6hours	Mexico
1004	AROUT	Thomas Hardy	16 days 6hours	UK
1005	BERGS	Christina Berglund	8 days 6hours	Sweden
1006	BLAUS	Hanna Moos	16 days 6hours	Germany
1007	BLONP	Frédérique Citeaux	3 days 6hours	France
1008	BOLID	Martin Sommer	8 days 6hours	Spain
1009	BONAP	Laurence Lebihan	3 days 6hours	France
1010	BOTTM	Elizabeth Lincoln	16 days 6hours	Canada

### GridCheckBoxSelectorColumn

SfDataGrid allows you to select or deselect individual rows through **CheckBox** using [GridCheckBoxSelectorColumn](#), which is not bound with data object from underlying data source, and it can be added like normal columns. The selector column supports row selection alone, and selection in selector column works based on the [SelectionMode](#).

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridCheckBoxSelectorColumn MappingName="SelectorColumn"
Width="30"/>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#

```
this.dataGrid.Columns.Add(new GridCheckBoxSelectorColumn()
{
MappingName = "SelectorColumn",
Width = 30
});
```

By default, check box is displayed in the header of selector column, which is used to select or deselect all the rows in the datagrid.

<input checked="" type="checkbox"/>	ID	Model	Brand	Type
<input type="checkbox"/>	10000	Xperia_Tipo	SonyMobile	Mobile
<input type="checkbox"/>	10001	XPS15	Dell	Laptop
<input type="checkbox"/>	10002	Men_Black	FastTrack	Watch
<input type="checkbox"/>	10003	Macbook_Pro2	Apple	Laptop
<input checked="" type="checkbox"/>	10004	Submariner	ROLEX	Watch
<input type="checkbox"/>	10005	Xperia_Z	SonyMobile	Mobile
<input checked="" type="checkbox"/>	10006	Monaco	Geneva	Watch
<input type="checkbox"/>	10007	Xperia_Z	SonyMobile	Mobile
<input type="checkbox"/>	10008	XPS12	Dell	Laptop
<input type="checkbox"/>	10009	Vaio	Sony	Laptop
<input checked="" type="checkbox"/>	10010	XPS12	Dell	Laptop
<input type="checkbox"/>	10011	Vaio	Sony	Laptop
<input type="checkbox"/>	10012	XPS12	Dell	Laptop
<input type="checkbox"/>	10013	Lumia_800	Nokia	Mobile
<input type="checkbox"/>	10014	S3	Samsung	Mobile
<input checked="" type="checkbox"/>	10015	One_X	HTC	Mobile
<input checked="" type="checkbox"/>	10016	XPS15	Dell	Laptop
<input type="checkbox"/>	10017	Xperia_Tipo	SonyMobile	Mobile

*Text on column header*

You can display text instead of check box in header of selector column by setting the [AllowCheckBoxOnHeader](#) property to `False`.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="False"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.Columns>
    <syncfusion:GridCheckBoxSelectorColumn MappingName="SelectorColumn"
        AllowCheckBoxOnHeader="False"
        HeaderText="Selector"
        Width="70"/>
    </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.Columns.Add(new GridCheckBoxSelectorColumn()
{
    HeaderText = "Selector",
    MappingName = "SelectorColumn",
    AllowCheckBoxOnHeader = false,
    Width = 70
});
```

```
});
```

Selector	ID	Model	Brand	Type
<input type="checkbox"/>	10000	Lumia_800	Nokia	Mobile
<input type="checkbox"/>	10001	Fastrack	FastTrack	Watch
<input type="checkbox"/>	10002	G-Shock	Casio	Watch
<input type="checkbox"/>	10003	Men_Black	FastTrack	Watch
<input checked="" type="checkbox"/>	10004	G-Shock	Casio	Watch
<input type="checkbox"/>	10005	Lumia_800	Nokia	Mobile
<input checked="" type="checkbox"/>	10006	Xperia_Z	SonyMobile	Mobile
<input type="checkbox"/>	10007	Fastrack	FastTrack	Watch
<input type="checkbox"/>	10008	Submariner	ROLEX	Watch
<input type="checkbox"/>	10009	Vaio	Sony	Laptop
<input checked="" type="checkbox"/>	10010	XPS15	Dell	Laptop
<input type="checkbox"/>	10011	Lumia_800	Nokia	Mobile
<input type="checkbox"/>	10012	Lumia_920	Nokia	Mobile
<input type="checkbox"/>	10013	S3	Samsung	Mobile
<input type="checkbox"/>	10014	XPS15	Dell	Laptop
<input checked="" type="checkbox"/>	10015	Vaio	Sony	Laptop
<input checked="" type="checkbox"/>	10016	Lumia_800	Nokia	Mobile
<input type="checkbox"/>	10017	Envy_X2	HP	Laptop

### Styling selector column

The style of checkbox in record cells can be customized using the `CellStyle` property.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridCheckBoxSelectorColumn MappingName="SelectorColumn"
Width="30">
<syncfusion:GridCheckBoxSelectorColumn.CellStyle>
<Style TargetType="syncfusion:GridCell">
<Style.Resources>
<Style TargetType="CheckBox">
<Setter Property="BorderBrush" Value="Red"/>
</Style>
</Style.Resources>
</Style>
</syncfusion:GridCheckBoxSelectorColumn.CellStyle>
</syncfusion:GridCheckBoxSelectorColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```



<input type="checkbox"/>	ID	Model	Brand	Type
<input type="checkbox"/>	10000	Pavilion_G6	HP	Laptop
<input type="checkbox"/>	10001	Fastrack	FastTrack	Watch
<input type="checkbox"/>	10002	XPS12	Dell	Laptop
<input type="checkbox"/>	10003	Macbook_Pro2	Apple	Laptop
<input checked="" type="checkbox"/>	10004	G-Shock	Casio	Watch
<input type="checkbox"/>	10005	Pavilion_G6	HP	Laptop
<input checked="" type="checkbox"/>	10006	Carrera	Geneva	Watch
<input type="checkbox"/>	10007	Submariner	ROLEX	Watch
<input type="checkbox"/>	10008	Transformer	Asus	Laptop
<input type="checkbox"/>	10009	G-Shock	Casio	Watch
<input checked="" type="checkbox"/>	10010	Transformer	Asus	Laptop
<input type="checkbox"/>	10011	Vaio	Sony	Laptop
<input type="checkbox"/>	10012	Macbook_Pro2	Apple	Laptop
<input type="checkbox"/>	10013	Pavilion_G6	HP	Laptop
<input type="checkbox"/>	10014	8x	HTC	Mobile
<input checked="" type="checkbox"/>	10015	XPS15	Dell	Laptop
<input checked="" type="checkbox"/>	10016	XPS15	Dell	Laptop
<input type="checkbox"/>	10017	Xperia_Tipo	SonyMobile	Mobile

As above, style of the header check box can be customized using the `HeaderCellStyle` property.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridCheckBoxSelectorColumn MappingName="SelectorColumn"
Width="30">
<syncfusion:GridCheckBoxSelectorColumn.HeaderStyle>
<Style TargetType="syncfusion:GridHeaderCellControl">
<Style.Resources>
<Style TargetType="CheckBox">
<Setter Property="BorderBrush" Value="Blue"/>
<Setter Property="BorderThickness" Value="2"/>
</Style>
</Style.Resources>
</Style>
</syncfusion:GridCheckBoxSelectorColumn.HeaderStyle>
</syncfusion:GridCheckBoxSelectorColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

<input checked="" type="checkbox"/>	ID	Model	Brand	Type
<input type="checkbox"/>	10000	Monaco	Geneva	Watch
<input type="checkbox"/>	10001	S3	Samsung	Mobile
<input type="checkbox"/>	10002	Pavilion_G6	HP	Laptop
<input type="checkbox"/>	10003	Pavilion_G6	HP	Laptop
<input checked="" type="checkbox"/>	10004	Fastrack	FastTrack	Watch
<input type="checkbox"/>	10005	One_X	HTC	Mobile
<input checked="" type="checkbox"/>	10006	Pavilion_G6	HP	Laptop
<input type="checkbox"/>	10007	Carrera	Geneva	Watch
<input type="checkbox"/>	10008	S3	Samsung	Mobile
<input type="checkbox"/>	10009	Carrera	Geneva	Watch
<input checked="" type="checkbox"/>	10010	Envy_X2	HP	Laptop
<input type="checkbox"/>	10011	Lumia_800	Nokia	Mobile
<input type="checkbox"/>	10012	Xperia_Tipo	SonyMobile	Mobile
<input type="checkbox"/>	10013	XPS12	Dell	Laptop
<input type="checkbox"/>	10014	Lumia_920	Nokia	Mobile
<input checked="" type="checkbox"/>	10015	Transformer	Asus	Laptop
<input checked="" type="checkbox"/>	10016	Xperia_Z	SonyMobile	Mobile
<input type="checkbox"/>	10017	Macbook_Pro2	Apple	Laptop

### Limitations

The following are the limitations of `GridCheckBoxSelectorColumn`:

- Selector column does not support cell selection.
- Selector column does not support data operations such as sorting, filtering, and grouping.
- Selector column will be excluded in operations such as printing and exporting.
- Selector column does not have filter row support.

### Custom column support

SfDataGrid allows you to create your own column by overriding predefined column type or creating a new custom column.

#### Creating column from existing column

You can create your own column by overriding the [predefined](#) column types in SfDataGrid.

For example, the `GridDateTimeColumn` loads the `DateTime` value by default. If you want to display [DateTimeOffset](#) value, you can create a new column by overriding the `GridDateTimeColumn` class.

In the below code snippet, converter created to format the `DateTimeOffset` value to `DateTime` by defining `ValueBinding` (edit) and `DisplayBinding` (non-edit).

### C#

```
public class DateTimeOffsetFormatConverter : IValueConverter
{
    private GridDateTimeOffsetColumn cachedColumn;
    public DateTimeOffsetFormatConverter(GridDateTimeOffsetColumn column)
    {
        cachedColumn = column;
    }
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        value = ((DateTimeOffset)value).DateTime;
        var column = cachedColumn as GridDateTimeColumn;
        if (value == null || DBNull.Value == value)
        {
            if (column.AllowNullValue && column.MaxDateTime != System.DateTime.MaxValue
                && column.NullText == string.Empty)
                return column.MaxDateTime;
            if (column.AllowNullValue && column.NullValue != null)
                return column.NullValue;
            else if (column.AllowNullValue && column.NullText != string.Empty)
                return column.NullText;
            if (column.MaxDateTime != System.DateTime.MaxValue)
                return column.MaxDateTime;
        }
        DateTime _columnValue;
        _columnValue = (DateTime)value;
        if (_columnValue < column.MinDateTime)
            _columnValue = column.MinDateTime;
        if (_columnValue > column.MaxDateTime)
            _columnValue = column.MaxDateTime;
        return DateTimeFormatString(_columnValue, column);
    }
    private string DateTimeFormatString(DateTime columnValue, GridDateTimeColumn
        column)
    {
        switch (column.Pattern)
        {
            case DateTimePattern.ShortDate:
                return columnValue.ToString("d", column.DateTimeFormat);
            case DateTimePattern.LongDate:
                return columnValue.ToString("D", column.DateTimeFormat);
            case DateTimePattern.LongTime:
                return columnValue.ToString("T", column.DateTimeFormat);
            case DateTimePattern.ShortTime:
                return columnValue.ToString("t", column.DateTimeFormat);
            case DateTimePattern.FullDateTime:
                return columnValue.ToString("F", column.DateTimeFormat);
            case DateTimePattern.RFC1123:
                return columnValue.ToString("R", column.DateTimeFormat);
            case DateTimePattern.SortableDateTime:
                return columnValue.ToString("s", column.DateTimeFormat);
            case DateTimePattern.UniversalSortableDateTime:
                return columnValue.ToString("u", column.DateTimeFormat);
            case DateTimePattern.YearMonth:
                return columnValue.ToString("Y", column.DateTimeFormat);
            case DateTimePattern.MonthDay:
                return columnValue.ToString("M", column.DateTimeFormat);
        }
    }
}
```

```

case DateTimePattern.CustomPattern:
return columnValue.ToString(column.CustomPattern, column.DateTimeFormat);
default:
return columnValue.ToString("MMMM", column.DateTimeFormat);
}
}
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
throw new NotImplementedException();
}
}
public class DateTimeOffsetToDateTimeConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
if (value == null)
return null;
return ((DateTimeOffset)value).DateTime;
}
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
if (value == null)
return null;
return value is DateTimeOffset ? value : new
DateTimeOffset((DateTime)value);
}
}
}

```

In the below code snippet, `GridDateTimeOffsetColumn` column created from `GridDateTimeColumn`.

### C#

```

public class GridDateTimeOffsetColumn : GridDateTimeColumn
{
protected override void SetDisplayBindingConverter()
{
if ((DisplayBinding as Binding).Converter == null)
(DisplayBinding as Binding).Converter = new
DateTimeOffsetFormatConverter(this);
if ((ValueBinding as Binding).Converter == null)
(ValueBinding as Binding).Converter = new
DateTimeOffsetToDateTimeConverter();
}
}
}

```

In the below code snippet, created `GridDateTimeOffsetColumn` added to [SfDataGrid.Columns](#) collection and specify the Pattern as `FullDateTime`. Since the `ShortDate` is the default pattern of `GridDateTimeColumn`.

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"

```

```

AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<local:GridDateTimeOffsetColumn MappingName="OrderDate"
Pattern="FullDateTime" UseBindingValue="True"/>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

**C#**

```

this.datagrid1.Columns.Add(new GridDateTimeOffsetColumn()
{
    MappingName = "OrderDate",
    Pattern = Syncfusion.Windows.Shared.DateTimePattern.FullDateTime,
    UseBindingValue = true
});

```

You can get the sample from [here](#).

*Customize column renderer*

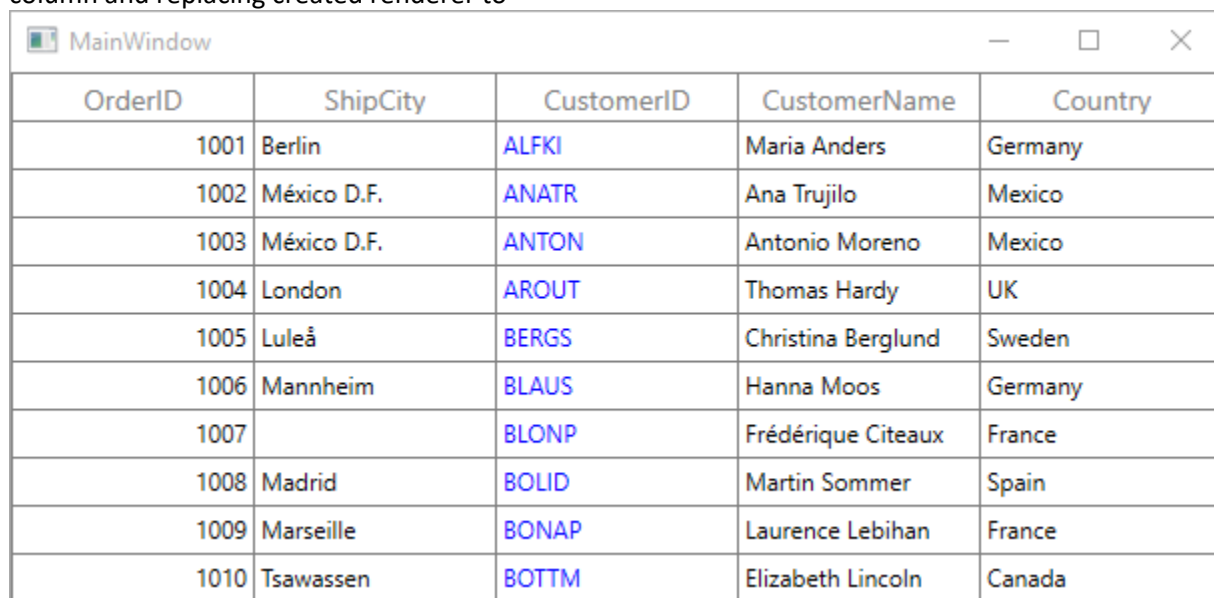
SfDataGrid allows you to customize the column related operations like key navigation and UI related interactions by overriding the corresponding renderer associated with the column. Each column has its own renderer with set of virtual methods for handling the column level operations.

Below table lists the available cell types for columns and its renderers.

Column Name	Renderer	Cell Type
GridTextColumn	<a href="#">GridCellTextBoxRenderer</a>	TextBox
GridNumericColumn	<a href="#">GridCellNumericRenderer</a>	Numeric
GridCheckBoxColumn	<a href="#">GridCellCheckBoxRenderer</a>	CheckBox
GridTemplateColumn	<a href="#">GridCellTemplateRenderer</a>	Template
GridImageColumn	<a href="#">GridCellImageRenderer</a>	Image
GridUnBoundColumn	<a href="#">GridUnBoundCellTextBoxRendererGridUnBoundCellTemplateRenderer</a>	UnBoundTextColumnUnBoundTemplateColumn
GridComboBoxColumn	<a href="#">GridCellComboBoxRenderer</a>	ComboBox
GridDateTimeColumn	<a href="#">GridCellDateTimeRenderer</a>	DateTime
GridHyperlinkColumn	<a href="#">GridCellHyperLinkRenderer</a>	HyperLink
GridMaskColumn	<a href="#">GridCellMaskRenderer</a>	Mask
GridPercentColumn	<a href="#">GridCellPercentageRenderer</a>	Percent
GridCurrencyColumn	<a href="#">GridCellCurrencyRenderer</a>	Currency

GridMultiColumnDropDownList	<a href="#">GridCellMultiColumnDropDownRenderer</a>	MultiColumnDropDown
GridTimeSpanColumn	<a href="#">GridCellTimeSpanRenderer</a>	TimeSpan
GridCheckBoxSelectorColumn	<a href="#">GridCellCheckBoxSelectorRenderer</a>	Selector

Below code, creates the `GridCellTextBoxRendererExt` to change the fore ground of CustomerID column and replacing created renderer to



OrderID	ShipCity	CustomerID	CustomerName	Country
1001	Berlin	ALFKI	Maria Anders	Germany
1002	México D.F.	ANATR	Ana Trujilo	Mexico
1003	México D.F.	ANTON	Antonio Moreno	Mexico
1004	London	AROUT	Thomas Hardy	UK
1005	Luleå	BERGS	Christina Berglund	Sweden
1006	Mannheim	BLAUS	Hanna Moos	Germany
1007		BLONP	Frédérique Citeaux	France
1008	Madrid	BOLID	Martin Sommer	Spain
1009	Marseille	BONAP	Laurence Lebihan	France
1010	Tsawassen	BOTTM	Elizabeth Lincoln	Canada

CellRenderers.

**C#**

```

this.dataGrid.CellRenderers.Remove("TextBox");
this.dataGrid.CellRenderers.Add("TextBox", new
GridCellTextBoxRendererExt());
public class GridCellTextBoxRendererExt:GridCellTextBoxRenderer
{
    public override void OnInitializeDisplayElement(DataColumnBase dataColumn,
    TextBlock uiElement, object dataContext)
    {
        base.OnInitializeDisplayElement(dataColumn, uiElement, dataContext);
        if(dataColumn.GridColumn.MappingName.Equals("CustomerID"))
        uiElement.Foreground = new SolidColorBrush(Colors.Blue);
    }
    public override void OnUpdateDisplayBinding(DataColumnBase dataColumn,
    TextBlock uiElement, object dataContext)
    {
        base.OnUpdateDisplayBinding(dataColumn, uiElement, dataContext);
        if (dataColumn.GridColumn.MappingName.Equals("CustomerID"))
        uiElement.Foreground = new SolidColorBrush(Colors.Blue);
    }
}

```

![WPF DataGrid column with Custom Renderer](Column-Types\_images/wpf-datagrid-custom-renderer.png)

#### Create the renderer of existing column

You can change the renderer of existing column by removing the predefined cell type value from [CellRenderers](#) collection and add the newly derived renderer from [GridVirtualizingCellRenderer](#).

For Non-Editable columns, [SupportsRenderOptimization](#) property should be set as `false` to set focus for the control loaded as `DisplayElement` in the column when interact through mouse.

Below code creates the new `GridComboBoxRenderer` with `ComboBoxAdv` as edit element for `GridComboBoxColumn` and replacing created renderer to `CellRenderers`.

#### C#

```
dataGrid.CellRenderers.Remove("ComboBox");
dataGrid.CellRenderers.Add("ComboBox", new GridComboBoxRenderer());
public class GridComboBoxRenderer : GridVirtualizingCellRenderer<TextBlock,
ComboBoxAdv>
{
    public GridComboBoxRenderer()
    {
    }
    /// <summary>
    /// Create new display element.
    /// </summary>
    /// <returns></returns>
    protected override TextBlock OnCreateDisplayUIElement()
    {
        return new TextBlock();
    }
    /// <summary>
    /// Create new edit element.
    /// </summary>
    /// <returns></returns>
    protected override ComboBoxAdv OnCreateEditUIElement()
    {
        return new ComboBoxAdv();
    }
    /// <summary>
    /// Initialize binding for display element.
    /// </summary>
    /// <param name="dataColumn"></param>
    /// <param name="uiElement"></param>
    /// <param name="dataContext"></param>
    public override void
    OnInitializeDisplayElement(Syncfusion.UI.Xaml.Grid.DataColumnBase
dataColumn, TextBlock uiElement, object dataContext)
    {
        base.OnInitializeDisplayElement(dataColumn, uiElement, dataContext);
        SetDisplayBinding(uiElement, dataColumn.GridColumn, dataContext);
    }
    /// <summary>
    /// custom binding for display element.
    /// </summary>
```

```

/// <param name="element"></param>
/// <param name="column"></param>
/// <param name="dataContext"></param>
private static void SetDisplayBinding(TextBlock element, GridColumn column,
object dataContext)
{
var comboBoxColumn = (GridColumnComboBox)column;
var binding = new Binding
{
Path = new PropertyPath(comboBoxColumn.MappingName),
Mode = BindingMode.TwoWay,
UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
Converter = new DisplayConverter(comboBoxColumn),
};
element.SetBinding(TextBlock.TextProperty, binding);
}
/// <summary>
/// Update binding for display element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnUpdateDisplayBinding(DataColumnBase dataColumn,
TextBlock uiElement, object dataContext)
{
base.OnUpdateDisplayBinding(dataColumn, uiElement, dataContext);
SetDisplayBinding(uiElement, dataColumn.GridColumn, dataContext);
}
/// <summary>
/// Initialize binding for edit element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnInitializeEditElement(DataColumnBase dataColumn,
ComboBoxAdv uiElement, object dataContext)
{
base.OnInitializeEditElement(dataColumn, uiElement, dataContext);
SetEditBinding(uiElement, dataColumn.GridColumn, dataContext);
}
/// <summary>
/// Update binding for edit element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnUpdateEditBinding(DataColumnBase dataColumn,
ComboBoxAdv element, object dataContext)
{
base.OnUpdateEditBinding(dataColumn, element, dataContext);
SetEditBinding(element, dataColumn.GridColumn, dataContext);
}
/// <summary>
/// custom binding for display element.
/// </summary>
/// <param name="element"></param>
/// <param name="column"></param>

```



```

/// <param name="dataContext"></param>
private static void SetEditBinding(ComboBoxAdv element, GridColumn column,
object dataContext)
{
var comboboxColumn = (GridColumn)column;
var binding = new Binding
{
Source = dataContext,
Path = new PropertyPath(comboboxColumn.MappingName),
Mode = BindingMode.TwoWay,
UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
};
element.SetBinding(ComboBoxAdv.SelectedItemsProperty, binding);
var itemsSourceBinding = new Binding { Path = new
PropertyPath("ItemsSource"), Mode = BindingMode.TwoWay, Source =
comboboxColumn };
element.SetBinding(ComboBoxAdv.ItemsSourceProperty, itemsSourceBinding);
var displayMemberBinding = new Binding { Path = new
PropertyPath("DisplayMemberPath"), Mode = BindingMode.TwoWay, Source =
comboboxColumn };
element.SetBinding(ComboBoxAdv.DisplayMemberPathProperty,
displayMemberBinding);
var selectedValuePathBinding = new Binding { Path = new
PropertyPath("SelectedValuePath"), Mode = BindingMode.TwoWay, Source =
comboboxColumn };
element.SetBinding(ComboBoxAdv.SelectedValuePathProperty,
selectedValuePathBinding);
element.AllowMultiSelect = true;
}
/// <summary>
/// Let Renderer decide whether the parent grid should be allowed to handle
keys and prevent
/// the key event from being handled by the visual UIElement for this
renderer.
/// </summary>
/// <param name="e">A <see cref="KeyEventArgs" /> object.</param>
/// <returns>
/// True if the parent grid should be allowed to handle keys; false
otherwise.
/// </returns>
protected override bool
ShouldGridTryToHandleKeyDown(System.Windows.Input.KeyEventArgs e)
{
if (!HasCurrentCellState || !IsInEditing)
return true;
switch (e.Key)
{
case Key.End:
case Key.Home:
case Key.Enter:
case Key.Escape:
return !((ComboBoxAdv)CurrentCellRendererElement).IsDropDownOpen;
case Key.Down:
case Key.Up:
case Key.Left:
case Key.Right:
return !((ComboBoxAdv)CurrentCellRendererElement).IsDropDownOpen;
}
}

```

```

}
return base.ShouldGridTryToHandleKeyDown(e);
}
/// <summary>
/// Gets the control value.
/// </summary>
public override object GetControlValue()
{
    if (!HasCurrentCellState)
        return base.GetControlValue();
    return CurrentCellRendererElement.GetValue(IsInEditing ?
        ComboBoxAdv.SelectedValueProperty : TextBlock.TextProperty);
}
/// <summary>
/// Sets the control value.
/// </summary>
/// <param name="value">The value.</param>
public override void SetControlValue(object value)
{
    if (!HasCurrentCellState)
        return;
    if (IsInEditing)
        ((ComboBoxAdv)CurrentCellRendererElement).SelectedValue = value;
    else
        throw new Exception("Value cannot be Set for Unloaded Editor");
}
}

```

Below code, returns the display value from multiple selected items.

### C#

```

public class DisplayConverter : IValueConverter
{
    GridColumn cachedColumn;
    public DisplayConverter()
    {
    }
    public DisplayConverter(GridColumn column)
    {
        cachedColumn = column;
    }
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        var selectedItems = value as IEnumerable;
        var displayMemberPath = string.Empty;
        var column = cachedColumn as GridComboBoxColumn;
        displayMemberPath = column.DisplayMemberPath;
        if (selectedItems == null)
            return null;
        string selectedItem = string.Empty;
        PropertyDescriptorCollection pdc = null;
        var enumerator = selectedItems.GetEnumerator();
        while (enumerator.MoveNext())
        {

```

```

var type = enumerator.Current.GetType();
pdc = pdc ?? TypeDescriptor.GetProperties(type);
if (!string.IsNullOrEmpty(displayMemberPath))
selectedItem += pdc.GetValue(enumerator.Current, displayMemberPath) + " - ";
}
return selectedItem.Substring(0, selectedItem.Length - 2);
}
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
return value;
}
}

```

OrderID	CustomerID	CustomerName	SelectedItems
1001	ALFKI	Maria Anders	Germany - Mexico - Mexico - UK
1002	ANATR	Ana Trujillo	Mexico - UK
1003	ANTON	Antonio Moreno	<input type="checkbox"/> Germany
1004	AROUT	Thomas Hardy	<input checked="" type="checkbox"/> Mexico
1005	BERGS	Christina Berglund	<input type="checkbox"/> Mexico
1006	BLAUS	Hanna Moos	<input checked="" type="checkbox"/> UK
1007	BLONP	Frédérique Citeaux	<input type="checkbox"/> Sweden
1008	BOLID	Martin Sommer	<input type="checkbox"/> Germany
1009	BONAP	Laurence Lebihan	<input type="checkbox"/> France
1010	BOTTM	Elizabeth Lincoln	<input type="checkbox"/> Spain
			<input type="checkbox"/> France
			<input type="checkbox"/> Canada

#### Creating new column and renderer

You can create a new column by deriving [GridColumn](#), rendered in UI using customized [CellType](#) using [GridVirtualizingCellRenderer](#).

Below steps to create custom column in SfDataGrid.

- [Creating custom column.](#)
- [Creating renderer.](#)
- [Adding the custom renderer to SfDataGrid.CellRenderers collection.](#)
- [Defining custom column.](#)

#### Creating custom column

You can create custom column by overriding a new class from `GridColumn` class.

Below code creates the converter to format the date time value.

#### C#

```

public class CustomConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
if (string.IsNullOrEmpty(value.ToString()))
return null;
}
}

```

```

return new ConvertToDateTimeClass().ConvertToDateTime(value);
}
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
return new ConvertToDateTimeClass().ConvertToDateTime(value);
}
}
public class ConvertToDateTimeClass
{
public string ConvertToDateTime(object value)
{
DateTime date = Convert.ToDateTime(value);
return date.Year + "/" + date.Month + "/" + date.Day;
}
}

```

In the below code, new column created with converter using `SetDisplayBindingConverter` method.

### C#

```

public class DatePickerColumn : GridColumn
{
public DatePickerColumn()
{
SetCellType("DatePickerRenderer");
}
protected override void SetDisplayBindingConverter()
{
(this.DisplayBinding as Binding).Converter = new CustomConverter();
}
protected override Freezable CreateInstanceCore()
{
return new DatePickerColumn();
}
}

```

### Creating renderer

After creating custom column, you need to create renderer for the custom column. Below are the steps to create custom renderer. You can create custom renderer either by deriving

`GridVirtualizingCellRenderer` class or overriding [existing renderers](#).

In the below code snippet, display and edit `UIElement` defined via `GridVirtualizingCellRenderer` parameter.

### C#

```

/// <summary>
/// CustomRenderer Creation
/// </summary>
/// <param name="TextBlock">Display Control</param>
/// <param name="DatePicker">Edit Control</param>
public class DatePickerRenderer : GridVirtualizingCellRenderer<TextBlock,
DatePicker>
{

```

```
public DatePickerRenderer()
{
}
}
```

With the below code snippet, you can create the display and edit element for renderer by overriding [OnCreateDisplayUIElement](#) and [OnCreateEditUIElement](#) methods.

### C#

```
/// <summary>
/// Create new display element.
/// </summary>
/// <returns></returns>
protected override TextBlock OnCreateDisplayUIElement()
{
    return new TextBlock();
}
/// <summary>
/// Create new edit element.
/// </summary>
/// <returns></returns>
protected override DatePicker OnCreateEditUIElement()
{
    return new DatePicker();
}
```

With the below code snippet, you can initialize the binding for display element by overriding the [OnInitializeDisplayElement](#) method.

### C#

```
/// <summary>
/// Initialize binding for display element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void
OnInitializeDisplayElement(Syncfusion.UI.Xaml.Grid.DataColumnBase
dataColumn, TextBlock uiElement, object dataContext)
{
    base.OnInitializeDisplayElement(dataColumn, uiElement, dataContext);
    SetDisplayBinding(uiElement, dataColumn.GridColumn, dataContext);
}
/// <summary>
/// custom binding for display element.
/// </summary>
/// <param name="element"></param>
/// <param name="column"></param>
/// <param name="dataContext"></param>
private static void SetDisplayBinding(TextBlock element, GridColumn column,
object dataContext)
{
    var customColumn = (DatePickerColumn)column;
    var binding = new Binding
```

```
{
    Path = new PropertyPath(customColumn.MappingName),
    Mode = BindingMode.TwoWay,
    UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
    Converter = (column.DisplayBinding as Binding).Converter,
};
element.SetBinding(TextBlock.TextProperty, binding);
}
```

With the below code snippet, updates the binding while UI interaction by overriding [OnUpdateDisplayBinding](#) method.

### C#

```
/// <summary>
/// Update binding for display element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnUpdateDisplayBinding(DataColumnBase dataColumn,
    TextBlock uiElement, object dataContext)
{
    base.OnUpdateDisplayBinding(dataColumn, uiElement, dataContext);
    SetDisplayBinding(uiElement, dataColumn.GridColumn, dataContext);
}
```

Similarly, you can initialize and update the binding for edit element by overriding [OnInitializeEditElement](#) and [OnUpdateEditBinding](#) methods.

### C#

```
/// <summary>
/// Initialize binding for edit element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnInitializeEditElement(DataColumnBase dataColumn,
    DatePicker uiElement, object dataContext)
{
    base.OnInitializeEditElement(dataColumn, uiElement, dataContext);
    SetEditBinding(uiElement, dataColumn.GridColumn, dataContext);
}

/// <summary>
/// Update binding for edit element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnUpdateEditBinding(DataColumnBase dataColumn,
    DatePicker element, object dataContext)
{
    base.OnUpdateEditBinding(dataColumn, element, dataContext);
    SetEditBinding(element, dataColumn.GridColumn, dataContext);
}
```

```

/// <summary>
/// custom binding for display element.
/// </summary>
/// <param name="element"></param>
/// <param name="column"></param>
/// <param name="dataContext"></param>
private static void SetEditBinding(DatePicker element, GridColumn column,
object dataContext)
{
var customColumn = (DatePickerColumn)column;
var binding = new Binding
{
Source = dataContext,
Path = new PropertyPath(customColumn.MappingName),
Mode = BindingMode.TwoWay,
UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
};
element.SetBinding(DatePicker.TextProperty, binding);
}

```

You can customize the editor control while loading by overriding [OnEditElementLoaded](#) method.

#### C#

```

/// <summary>
/// Handling operations on edit mode UIElement.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected override void OnEditElementLoaded(object sender, RoutedEventArgs
e)
{
var datePicker = (sender as DatePicker);
datePicker.Focus();
DatePickerTextBox datePickerTextBox =
(DatePickerTextBox)GridUtil.FindDescendantChildByType(datePicker,
typeof(DatePickerTextBox));
if ((this.DataGrid.EditorSelectionBehavior ==
EditorSelectionBehavior.SelectAll ||
this.DataGrid.IsAddNewIndex(this.CurrentCellIndex.RowIndex)) &&
PreviewInputText == null)
{
datePickerTextBox.SelectAll();
}
else
{
datePickerTextBox.Select(datePickerTextBox.SelectedText.Length, 0);
}
PreviewInputText = null;
}

```

With the below code snippet, you can customize the keyboard interactions for the custom column by overriding [ShouldGridTryToHandleKeyDown](#) method.

#### C#

```

/// <summary>
/// Let Renderer decide whether the parent grid should be allowed to handle
keys and prevent
/// the key event from being handled by the visual UIElement for this
renderer.
/// </summary>
/// <param name="e">A <see cref="KeyEventArgs" /> object.</param>
/// <returns>
/// True if the parent grid should be allowed to handle keys; false
otherwise.
/// </returns>
protected override bool
ShouldGridTryToHandleKeyDown(System.Windows.Input.KeyEventArgs e)
{
    if (!HasCurrentCellState || !IsInEditing)
        return true;
    DatePickerTextBox datePickerTextBox =
        (DatePickerTextBox)
        GridUtil.FindDescendantChildByType(CurrentCellRendererElement as DatePicker,
        typeof(DatePickerTextBox));
    switch (e.Key)
    {
        case Key.End:
        case Key.Home:
        case Key.Enter:
        case Key.Escape:
            return !((DatePicker)CurrentCellRendererElement).IsDropDownOpen;
        case Key.Down:
        case Key.Up:
        case Key.Left:
        case Key.Right:
            return !((DatePicker)CurrentCellRendererElement).IsDropDownOpen;
    }
    return base.ShouldGridTryToHandleKeyDown(e);
}

```

You can handle the cell value for the custom renderer by overriding [GetControlValue](#) and [SetControlValue](#) methods.

### C#

```

/// <summary>
/// Gets the control value.
/// </summary>
public override object GetControlValue()
{
    if (!HasCurrentCellState)
        return base.GetControlValue();
    return CurrentCellRendererElement.GetValue(IsInEditing ?
    DatePicker.TextProperty : TextBlock.TextProperty);
}
/// <summary>
/// Sets the control value.
/// </summary>
/// <param name="value">The value.</param>
public override void SetControlValue(object value)

```



```

{
    if (!HasCurrentCellState)
    {
        return;
    }
    if (IsInEditing)
    {
        ((TextBox)CurrentCellRenderElement).Text = value.ToString();
    }
    else
    {
        throw new Exception("Value cannot be Set for Unloaded Editor");
    }
}

```

Adding the custom renderer to SfDataGrid.CellRenderers collection

By below code, you can add the previous created custom renderer to [SfDataGrid.CellRenderers](#) collection.

### C#

```
dataGrid.CellRenderers.Add("DatePickerRenderer", new DatePickerRenderer());
```

Loading custom column

By below code, you can define the custom column in SfDataGrid.

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="False"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.Columns>
    <local:DatePickerColumn MappingName="OrderDate"/>
    </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

### C#

```
this.dataGrid.Columns.Add(new DatePickerColumn() {AllowEditing=true,
MappingName = "DateOfMonth"} );
```

OrderID	OrderDate	CustomerID	CustomerName	Country
1001	2/10/2016	ALFKI	Maria Anders	Germany
1002	(calendar popup)	ANATR	Ana Trujillo	Mexico
1003	(calendar popup)	ANTON	Antonio Moreno	Mexico
1004	(calendar popup)	AROUT	Thomas Hardy	UK
1005	(calendar popup)	BERGS	Christina Berglund	Sweden
1006	(calendar popup)	BLAUS	Hanna Moos	Germany
1007	(calendar popup)	BLONP	Frédérique Citeaux	France
1008	(calendar popup)	BOLID	Martin Sommer	Spain
1009	2016/2/20	BONAP	Laurence Lebihan	France
1010	2016/2/12	BOTTM	Elizabeth Lincoln	Canada

## How To

### *Restrict the input content length*

You can restrict the range of input using **MaxLength** property on **GridColumn** in below ways.

- Using Converter property in DisplayBinding and ValueBinding
- Using control style
- Overriding existing cell types

### Using Converter

You can restrict the length of user input in both display and edit element using **Converter** using **DisplayBinding** and **ValueBinding**.

### XML

```
<Window.Resources>
<local:MaxLengthConverter x:Key="converter" />
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn DisplayBinding="{Binding CustomerName,
Converter={StaticResource converter}}"
HeaderText="Customer Name"
MappingName="CustomerName"
ValueBinding="{Binding CustomerName,
Converter={StaticResource converter}}" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#

```
public class MaxLengthConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        int maxLength = 5;
        var columnValue = System.Convert.ToString(value);
        if (columnValue.Length < maxLength)
            return columnValue;
        else
            return columnValue.Substring(0, maxLength);
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        return value;
    }
}
```

### Using control style

You can set the `MaxLength` property in edit mode by writing style of `TargetType` edit element of the corresponding column.

**Note:** `TextBlock` does not have the `MaxLength` property. Therefore, you can use the converter to format in display.

### XML

```
<Window.Resources>
<Style TargetType="TextBox">
<Setter Property="MaxLength" Value="7" />
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="CustomerName"/>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### Overriding existing cell types

You can set the `MaxLength` property to the edit element of the particular column by overriding existing cell types.

Below code, overrides the [OnInitializeEditElement](#) method of the corresponding renderer and set the `MaxLength` to the `UIElement` and add the renderer to [SfDataGrid.CellRenderers](#) collection.

### C#

```
this.dataGrid.CellRenderers.Remove("TextBox");
this.dataGrid.CellRenderers.Add("TextBox", new
GridCellTextBoxRendererExt());
public class GridCellTextBoxRendererExt:GridCellTextBoxRenderer
{
public override void
OnInitializeEditElement(Syncfusion.UI.Xaml.Grid.DataColumnBase dataColumn,
System.Windows.Controls.TextBox uiElement, object dataContext)
{
if (dataColumn.GridColumn != null && dataColumn.GridColumn.MappingName ==
"ProductName")
uiElement.MaxLength = 7;
else
uiElement.MaxLength = 0;
base.OnInitializeEditElement(dataColumn, uiElement, dataContext);
}
}
```

### AutoSize Columns in WPF DataGrid (SfDataGrid)

`DataGrid` allows you to set the column widths based on certain logic using [SfDataGrid.ColumnSizer](#) or [GridColumn.ColumnSizer](#) property. Below is the list of predefined column sizing options available.

Type	Column width
------	--------------

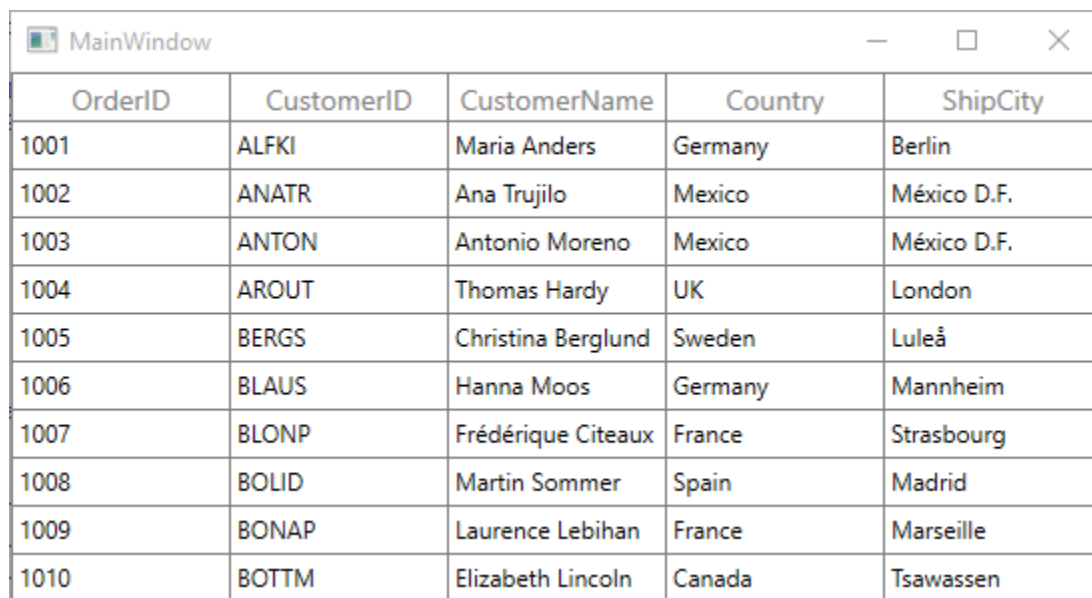
Star	Divides the total width equally for columns.
Auto	Calculates the width of column based on header and cell contents. So that header and cell contentâ€™s are not truncated.
AutoWithLastColumnFill	Applies <code>GridLengthUnitType.Auto</code> width to all the columns except last column which is visible and the remaining width from total width of SfDataGrid is set to last column.
AutoLastColumnFill	Applies <code>GridLengthUnitType.Auto</code> width to all the columns except last column which is visible and sets the maximum between last column auto spacing width and remaining width to last column.
SizeToCells	Calculates the width of column based on cell contents. So that cell contentâ€™s are not truncated.
SizeToHeader	Calculates the width of column based on header content. So that header content is not truncated.
None	Default column width or defined width set to column.

**Note:** ColumnSizer will not work when the column width defined explicitly. ColumnSizer calculates column width based on `MinWidth` and `MaxWidth` properties.

Below code, applies `GridLengthUnitType.Star` to equally set width for `SfDataGrid.Columns`.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ColumnSizer="Star"
    ItemsSource="{Binding Orders}" />
```



OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

**Note:** The `GridColumn.ColumnSizer` takes higher priority than the `SfDataGrid.ColumnSizer`.

Fill remaining width for any column instead of last column when `ColumnSizer` is `AutoLastColumnFill` or `AutoWithLastColumnFill`

In `SfDataGrid` while setting `SfDataGrid.ColumnSizer` as `AutoLastColumnFill` or `AutoWithLastColumnFill` remaining width is applied to last column. You can apply the remaining width to specific column by setting [GridColumn.ColumnSizer](#) property as like below.

#### XML

```
<Syncfusion:SfDataGrid x:Name="datagrid"
    ColumnSizer="AutoWithLastColumnFill"
    ItemsSource="{Binding OrderInfoCollection }">
    <Syncfusion:SfDataGrid.Columns>
    <Syncfusion:GridTextColumn MappingName="OrderID" HeaderText="OrderID"
        ColumnSizer="AutoLastColumnFill"/>
    <Syncfusion:GridTextColumn MappingName="CustomerID" HeaderText="CustomerID"
        />
    <Syncfusion:GridTextColumn MappingName="CustomerName"
        HeaderText="CustomerName"/>
    <Syncfusion:GridTextColumn MappingName="Country" HeaderText="Country"/>
    <Syncfusion:GridTextColumn MappingName="ShipCity" HeaderText="ShipCity"/>
    </Syncfusion:SfDataGrid.Columns>
</Syncfusion:SfDataGrid>
```

#### C#

```
this.datagrid.ColumnSizer = GridLengthUnitType.AutoLastColumnFill;
this.datagrid.Columns["OrderID"].ColumnSizer =
    GridLengthUnitType.AutoWithLastColumnFill;
```

SfDataGrid Demo

OrderID	CustomerID	CustomerName	ShipCity	Country	IsDelivered
1001	ALFKI	Alfreds	Beverton	US	<input checked="" type="checkbox"/>
1002	ANATR	Oliver	Consolidated	US	<input type="checkbox"/>
1003	ANTON	Brendon	Johanesberg	China	<input checked="" type="checkbox"/>
1004	YHGT	Around Horn	Chicago	UK	<input checked="" type="checkbox"/>
1005	BERGS	France	Spain	China	<input type="checkbox"/>
1006	TGVFD	Dintin	Britain	US	<input type="checkbox"/>
1007	YTREW	Friedo	Campinas	Switzerland	<input type="checkbox"/>
1008	HANAR	John	Oregon	British	<input checked="" type="checkbox"/>
1009	BGFDE	Sireert	Bulk	US	<input checked="" type="checkbox"/>
1010	BOTTM	Rakesh	hertion	Britain	<input type="checkbox"/>
1011	TGVFD	George	Holdings	US	<input checked="" type="checkbox"/>
1012	Nantes	Charles	Spain	Australia	<input checked="" type="checkbox"/>
1013	Johanesberg	Lanchonetes	Campinas	UK	<input type="checkbox"/>
1014	London	Charles	Spain	China	<input checked="" type="checkbox"/>
1015	TGVFD	Dintin	Britain	US	<input type="checkbox"/>
1016	Mannheim	Carnes	James	US	<input type="checkbox"/>
1017	Berlin	Gourmet	Frankenversand	Englan	<input checked="" type="checkbox"/>
1018	KOENE	Trading	Versailles	Barcelona	<input checked="" type="checkbox"/>
1019	MAISD	Island	Típica	Elgin	<input checked="" type="checkbox"/>
1020	MAGAA	Dewey	Vancouver	Australia	<input checked="" type="checkbox"/>
1021	LEHMS	Maison	Bergamo	Switzerland	<input checked="" type="checkbox"/>
1022	TGVFD	Dintin	Britain	US	<input type="checkbox"/>
1023	TGVFD	George	Cocina	US	<input checked="" type="checkbox"/>

Refreshing autosize calculation at runtime

You can refresh the autosize calculation at runtime by calling [SfDataGrid.GridColumnSizer.Refresh](#) method.

DataGrid support to recalculates the column auto width by calling reset methods of `GridColumnSizer`. [GridColumnSizer.ResetAutoCalculationforAllColumns](#) method reset widths to all columns. [GridColumnSizer.ResetAutoCalculation](#) method reset the width to particular column.

**Note:** The `GridColumnSizer.ResetAutoCalculationforAllColumns` or `GridColumnSizer.ResetAutoCalculation` methods applicable for Auto, AutoWithLastColumnFill, AutoLastColumnFill, SizeToCells types.

For example, you can refresh all the column's width based on the cell contents of newly added records at runtime.

### C#

```
var viewModel = this.dataGrid.DataContext as ViewModel;
viewModel.Orders.Add(new OrderInfo(11, "BLFKI", "Maria Joseph Anders"));
this.dataGrid.GridColumnSizer.ResetAutoCalculationforAllColumns();
this.dataGrid.GridColumnSizer.Refresh();
```

### Resetting column width to apply autosize calculation

When the width of the column is explicitly defined or column is resized, then column width is not changed based on `GridColumnSizer`. You can reset [GridColumn.Width](#) by setting `double.NaN` to apply column width based on column sizer.

### C#

```
foreach (var column in dataGrid.Columns)
{
    if (!double.IsNaN(column.Width))
        column.Width = double.NaN;
}
this.dataGrid.GridColumnSizer.Refresh();
```

### Customizing built-in column auto-sizing logic

SfDataGrid process column sizing operations in [GridColumnSizer](#) class. You can customize the column sizing operations by overriding `GridColumnSizer` and set it to `SfDataGrid.GridColumnSizer`.

### C#

```
this.dataGrid.GridColumnSizer = new GridColumnSizerExt(dataGrid);
public class GridColumnSizerExt:GridColumnSizer
{
    public GridColumnSizerExt(SfDataGrid dataGrid)
        :base(dataGrid)
    {
    }
    // Calculate Width for column when ColumnSizer is SizeToCells.
    protected override double CalculateCellWidth(GridColumn column, bool
setWidth = true)
    {
        return base.CalculateCellWidth(column, setWidth);
    }
    //Calculate Width for the column when ColumnSizer is SizeToHeader
    protected override double CalculateHeaderWidth(GridColumn column, bool
setWidth = true)
    {
    }
```

```
return base.CalculateHeaderWidth(column, setWidth);
}
}
```

Auto width calculation based on font settings

By default, the ColumnSizer calculates column's width based on fixed **FontSize**, **FontFamily**, **Margin**, **SortIconWidth**, **FilterIconWidth**. You can change the calculation by customized settings.

#### *Changing Sort and Filter Icon width*

You can change the filter icon and sort icon widths for column width calculation by setting [GridColumnSizer.SortIconWidth](#) and [GridColumnSizer.FilterIconWidth](#) properties.

#### **C#**

```
dataGrid.GridColumnSizer.SortIconWidth = 20;
dataGrid.GridColumnSizer.FilterIconWidth = 20;
```

#### *Changing Font settings for DataGrid*

You can change the font settings for column width calculation by setting [GridColumnSizer.FontSize](#), [GridColumnSizer.FontFamily](#) and [GridColumnSizer.Margin](#) properties. This settings will be considered for all columns.

#### **C#**

```
this.dataGrid.GridColumnSizer.FontSize = 10.0;
this.dataGrid.GridColumnSizer.FontFamily = new FontFamily("TimesNewRoman");
this.dataGrid.GridColumnSizer.Margin = new Thickness(9, 3, 1, 3);
```

#### *Changing Font settings for one Column*

You can change the font setting for one column width calculation using [GridColumnSizer.SetFontFamily](#), [GridColumnSizer.SetFontSize](#) and [GridColumnSizer.SetMargin](#) static methods of **GridColumnSizer** to **GridColumn**.

#### **C#**

```
var gridColumn = this.dataGrid.Columns[0];
GridColumnSizer.SetFontFamily(gridColumn, new FontFamily("TimesNewRoman"));
GridColumnSizer.SetFontSize(gridColumn, 10.0);
GridColumnSizer.SetMargin(gridColumn, new Thickness(9, 3, 1, 3));
```

You can also change the font setting for one column width calculation using [GetFormattedText](#) method.

#### **C#**

```
this.dataGrid.GridColumnSizer = new ColumnSizerExt(this.dataGrid);
public class ColumnSizerExt : GridColumnSizer
{
    public ColumnSizerExt(SfDataGrid grid) : base(grid)
    {
    }
    protected override FormattedText GetFormattedText(GridColumn column, object
record, string displayText)
```

```
{
    var formattedText = base.GetFormattedText(column, record, displayText);
    if (column.MappingName.Equals("OrderID"))
    {
        formattedText.SetFontFamily(new FontFamily("TimesNewRoman"));
        formattedText.SetFontSize(10);
    }
    return formattedText;
}
```

### Star column sizer ratio support

You can customize the `ColumnSizer.Star` width calculation logic by overriding [SetStarWidth](#) method of [GridColumnSizer](#).

For example, you can calculate the column width, with specified ratios instead of dividing equal width for all columns in Star calculation using `ColumnRatio` attached property.

### C#

```
public static class StarRatio
{
    public static int GetColumnRatio(DependencyObject obj)
    {
        return (int)obj.GetValue(ColumnRatioProperty);
    }
    public static void SetColumnRatio(DependencyObject obj, int value)
    {
        obj.SetValue(ColumnRatioProperty, value);
    }
    public static readonly DependencyProperty ColumnRatioProperty =
        DependencyProperty.RegisterAttached("ColumnRatio", typeof(int),
        typeof(StarRatio), new PropertyMetadata(1, null));
}
```

Below code to define the star width calculation based on the `ColumnRatio`.

### C#

```
//Assign the customized GridColumnSizerExt to SfDataGrid.GridColumnSizer
this.dataGrid.GridColumnSizer = new GridColumnSizerExt(dataGrid);
public class GridColumnSizerExt : GridColumnSizer
{
    public GridColumnSizerExt(SfDataGrid grid)
    : base(grid)
    {
    }
    protected override void SetStarWidth(double remainingColumnWidth,
        IEnumerable<GridColumn> remainingColumns)
    {
        var removedColumn = new List<GridColumn>();
        var columns = remainingColumns.ToList();
        var totalRemainingStarValue = remainingColumnWidth;
        double removedWidth = 0;
        bool isRemoved;
```



```
while (columns.Count > 0)
{
    isRemoved = false;
    removedWidth = 0;
    var columnsCount = 0;
    columns.ForEach((col) =>
    {
        columnsCount += StarRatio.GetColumnRatio(col);
    });
    double starWidth = Math.Floor((totalRemainingStarValue / columnsCount));
    var column = columns.First();
    starWidth *= StarRatio.GetColumnRatio(column);
    double computedWidth = SetColumnWidth(column, starWidth);
    if (starWidth != computedWidth && starWidth > 0)
    {
        isRemoved = true;
        columns.Remove(column);
        foreach (var remColumn in removedColumn)
        {
            if (!columns.Contains(remColumn))
            {
                removedWidth += remColumn.ActualWidth;
                columns.Add(remColumn);
            }
        }
        removedColumn.Clear();
        totalRemainingStarValue += removedWidth;
    }
    totalRemainingStarValue = totalRemainingStarValue - computedWidth;
    if (!isRemoved)
    {
        columns.Remove(column);
        if (!removedColumn.Contains(column))
            removedColumn.Add(column);
    }
}
}}
```

Below code uses the **ColumnRatio** to apply the defined star width for each column.

#### C#

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
ColumnSizer="Star"
AutoGenerateColumns="False">
<syncfusion:SfDataGrid.Columns>;
<syncfusion:GridTextColumn HeaderText="Order ID"
MappingName="OrderID"
local:StarRatio.ColumnRatio="1" />
<syncfusion:GridTextColumn HeaderText="Customer ID"
MappingName="CustomerID"
local:StarRatio.ColumnRatio="2" />
<syncfusion:GridTextColumn HeaderText="Customer Name"
MappingName="CustomerName"
```

```
local:StarRatio.ColumnRatio="3" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

Change the width of DataGrid ComboBoxColumn based on it's ItemsSource

By default, the `ColumnSizer` calculates auto width based on the column content. You can change the auto width calculation for `GridComboBoxColumn` based on its items source by overriding the `CalculateCellWidth` virtual method.

Below code creates `CustomColumnSizer` to change the width of `GridComboboxColumn` and set to `SfDataGrid.GridColumnSizer`.

#### C#

```
this.dataGrid.GridColumnSizer = new CustomColumnSizer(this.dataGrid);
public class CustomColumnSizer:GridColumnSizer
{
    public CustomColumnSizer(SfDataGrid grid)
    : base(grid)
    {
    }
    protected override double CalculateCellWidth(GridColumn column, bool
setWidth = true)
    {
        if (column is GridComboBoxColumn)
        {
            double colWidth = double.MaxValue;
            var source = (column as GridComboBoxColumn).ItemsSource;
            string maximumComboItemsText = string.Empty;
            var clientSize = new Size(colWidth, DataGrid.RowHeight);
            foreach (var comboItems in source)
            {
                string comboItemText = (string)comboItems;
                if (maximumComboItemsText.Length < comboItemText.Length)
                    maximumComboItemsText = comboItemText;
            }
        }
    }
}
```

```

var measureSize=
MeasureText(clientSize,maximumComboItemsText,column,null,GridQueryBounds.Width);
return measureSize.Width + SystemParameters.ScrollWidth;
}
else
return base.CalculateCellWidth(column, setWidth);
}
}

```

## Stacked Headers in WPF DataGrid (SfDataGrid)

DataGrid supports additional unbound header rows known as **stacked header rows** or that span across the DataGrid columns using [StackedHeaderRows](#). You can group one or more columns under each stacked header.

Each [StackedHeaderRow](#) contains [StackedColumns](#) where each [StackedColumn](#) contains a number of child columns. The [StackedColumn.ChildColumns](#) property returns the columns grouped under the stacked header row. The [StackedColumn.MappingName](#) is a unique name used for mapping a specific child columns grouped under the same stacked header row whereas, the [StackedColumn.HeaderText](#) returns the text displayed in the stacked header row.

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.StackedHeaderRows>
<syncfusion:StackedHeaderRow>
<syncfusion:StackedHeaderRow.StackedColumns>
<syncfusion:StackedColumn
ChildColumns="OrderID, CustomerID, CustomerName, ShipCity, Country"
HeaderText="Sales Details" MappingName="SalesDetails"/>
</syncfusion:StackedHeaderRow.StackedColumns>
</syncfusion:StackedHeaderRow>
<syncfusion:StackedHeaderRow>
<syncfusion:StackedHeaderRow.StackedColumns>
<syncfusion:StackedColumn ChildColumns="OrderID" HeaderText="Order Details"
MappingName="OrderDetails"/>
<syncfusion:StackedColumn ChildColumns="CustomerID, CustomerName"
HeaderText="Customer Details" MappingName="CustomerDetails"/>
<syncfusion:StackedColumn ChildColumns="ShipCity, Country"
HeaderText="Shipping Details" MappingName="ShippingDetails"/>
</syncfusion:StackedHeaderRow.StackedColumns>
</syncfusion:StackedHeaderRow>
</syncfusion:SfDataGrid.StackedHeaderRows>
</syncfusion:SfDataGrid>

```

### C#

```

var stackedHeaderRow = new StackedHeaderRow();
stackedHeaderRow.StackedColumns.Add(new StackedColumn() { ChildColumns =
"OrderID" + "," + "CustomerID" + "," + "CustomerName" + "," + "ShipCity" +
"," + "Country", HeaderText = "Sales Details" , MappingName="SalesDetails"
});
dataGrid.StackedHeaderRows.Add(stackedHeaderRow);

```

```

var stackedHeaderRow1 = new StackedHeaderRow();
stackedHeaderRow1.StackedColumns.Add(new StackedColumn() { ChildColumns =
"OrderID", HeaderText = "Order Details" , MappingName="OrderDetails" });
stackedHeaderRow1.StackedColumns.Add(new StackedColumn() { ChildColumns =
"CustomerID" + "," + "CustomerName", HeaderText = "Customer Details" ,
MappingName="CustomerDetails" });
stackedHeaderRow1.StackedColumns.Add(new StackedColumn() { ChildColumns =
"ShipCity" + "," + "Country", HeaderText = "Shipping Details" ,
MappingName="ShippingDetails" });
dataGrid.StackedHeaderRows.Add(stackedHeaderRow1);

```

Sales Details				
Order Details	Customer Details		Shipping Details	
OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

### Stacked Headers using Data Annotation

You can also add the stacked headers using `GroupName` property of [Data Annotations Display attributes](#).

### C#

```

public class OrderInfo
{
    private int orderID;
    private string customerId;
    private string country;
    private string customerName;
    private string shippingCity;
    public int OrderID
    {
        get { return orderID; }
        set { orderID = value; }
    }
    [Display(GroupName = "Customer Details")]
    public string CustomerID
    {
        get { return customerId; }
        set { customerId = value; }
    }
    [Display(GroupName = "Customer Details")]
    public string CustomerName
    {

```

```

get { return customerName; }
set { customerName = value; }
}
[Display(GroupName = "Shipping Details")]
public string Country
{
    get { return country; }
    set { country = value; }
}
[Display(GroupName = "Shipping Details")]
public string ShipCity
{
    get { return shippingCity; }
    set { shippingCity = value; }
}
}

```

OrderID	Customer Details		Shipping Details	
	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Leblan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

### Adding ChildColumns

You can add the child columns in particular stacked header directly.

#### C#

```

var childColumn =
this.dataGrid.StackedHeaderRows[1].StackedColumns[0].ChildColumns;
this.dataGrid.StackedHeaderRows[1].StackedColumns[0].ChildColumns =
childColumn + ", " + "OrderDate" + ", " + "Discount";

```

### Removing ChildColumns

Similarly, you can remove the child columns from particular stacked header directly.

#### C#

```

var removingColumns =
this.dataGrid.StackedHeaderRows[0].StackedColumns[0].ChildColumns.Split(',')
.ToList<string>();
string childColumns = string.Empty;
foreach(var stackedColumnName in removingColumns.ToList())
{
    if (stackedColumnName.Equals("OrderID"))

```

```
{
    removingColumns.Remove(stackedColumnName);
}
else
    childColumns = childColumns + stackedColumnName + ",";
}
this.dataGrid.StackedHeaderRows[0].StackedColumns[0].ChildColumns =
    childColumns;
```

### Changing stacked header row height

You can change the height of stacked header rows by using [VisualContainer.RowHeights](#) property.

#### C#

```
using Syncfusion.UI.Xaml.Grid.Helpers;
var visualContainer = dataGrid.GetVisualContainer();
int count = dataGrid.StackedHeaderRows.Count;
for (int i = 0; i < count; i++)
{
    visualContainer.RowHeights[i] = 50;
}
visualContainer.InvalidateMeasure();
```

You can also change the height of stacked header rows using [SfDataGrid.QueryRowHeight](#) event.

#### C#

```
using Syncfusion.UI.Xaml.Grid;
this.dataGrid.QueryRowHeight += dataGrid_QueryRowHeight;
void dataGrid_QueryRowHeight(object sender,
    Syncfusion.UI.Xaml.Grid.QueryRowHeightEventArgs e)
{
    if(e.RowIndex < this.dataGrid.GetHeaderIndex())
    {
        e.Height = 50;
        e.Handled = true;
    }
}
```

### Sorting in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) (SfDataGrid) allows you to sort the data against one or more columns either in ascending or descending order. When sorting is applied, the rows are rearranged based on sort criteria. You can allow users to sort the data by touching or clicking the column header using [SfDataGrid.AllowSorting](#) property to **true**.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowSorting="True"
    ItemsSource="{Binding Orders}">
```

#### C#

```
dataGrid.AllowSorting = true;
```

In another way, you can enable or disable the sorting for particular column by setting the [GridColumn.AllowSorting](#) property.

#### XML

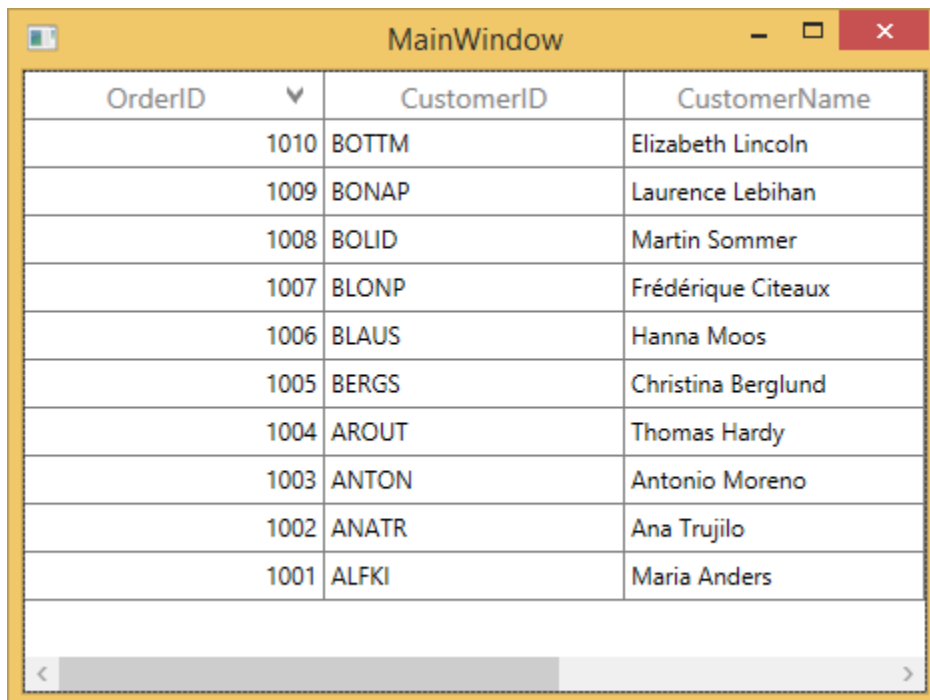
```
<syncfusion:SfDataGrid x:Name="dataGrid"
AllowSorting="False"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn AllowSorting="True" MappingName="OrderID" />
<syncfusion:GridTextColumn AllowSorting="False" MappingName="CustomerID" />
<syncfusion:GridTextColumn AllowSorting="False" MappingName="CustomerName"
/>
<syncfusion:GridTextColumn AllowSorting="True" MappingName="Country" />
<syncfusion:GridTextColumn AllowSorting="True" MappingName="ShipCity" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.Columns["OrderID"].AllowSorting = true;
this.dataGrid.Columns["CustomerID"].AllowSorting = false;
```

**Note:** The [GridColumn.AllowSorting](#) takes higher priority than [SfDataGrid.AllowSorting](#) property.

End users can sort the column by clicking column header cell. Once the columns get sorted, the sort indicator will be displayed on the right side of the column header.



OrderID ▼	CustomerID	CustomerName
1010	BOTTM	Elizabeth Lincoln
1009	BONAP	Laurence Lebihan
1008	BOLID	Martin Sommer
1007	BLONP	Frédérique Citeaux
1006	BLAUS	Hanna Moos
1005	BERGS	Christina Berglund
1004	AROUT	Thomas Hardy
1003	ANTON	Antonio Moreno
1002	ANATR	Ana Trujilo
1001	ALFKI	Maria Anders

### Sort column in double click

By default, column gets sorted when column header clicked. You can change this behavior to sort the column in double click action by setting [SfDataGrid.SortClickAction](#) property to `DoubleClick`.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowSorting="True"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"
    SortClickAction="DoubleClick" />
```

#### C#

```
this.dataGrid.SortClickAction = SortClickAction.DoubleClick;
```

### Sorting order

By default, the data is sorted in ascending or descending order when clicking column header. You can rearrange the data to its initial order from descending, when clicking column header by setting [SfDataGrid.AllowTriStateSorting](#) property.

Following are the sequence of sorting orders when clicking column header,

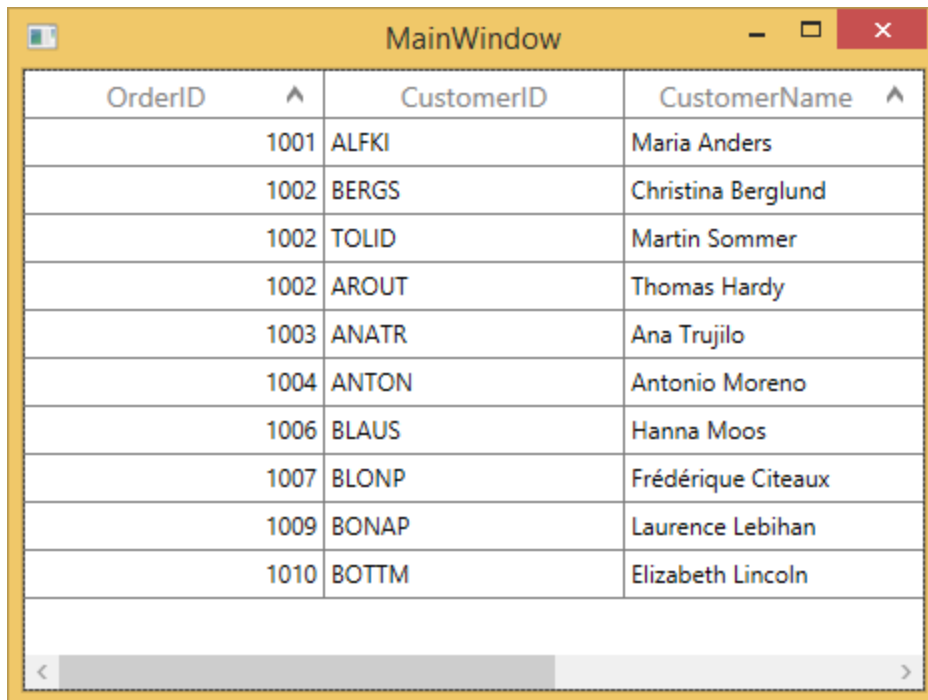
- Sorts the data in ascending order
- Sorts the data in descending order
- Clears the sorting and records displayed in its initial order

### Multi column sorting

WPF DataGrid (SfDataGrid) control allows you sort more than one column, where sorting is applied one column against other columns. To apply sorting on multiple columns, user have to click the column header by pressing the Ctrl key.

In the below screen shot, the OrderID column sorted. Then the CustomerName column is sorted against the OrderID data by clicking column header by pressing Ctrl key. The sorting state of OrderID column is preserved and CustomerName column sorted against OrderID column.





OrderID ^	CustomerID	CustomerName ^
1001	ALFKI	Maria Anders
1002	BERGS	Christina Berglund
1002	TOLID	Martin Sommer
1002	AROUT	Thomas Hardy
1003	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

#### Display sort order

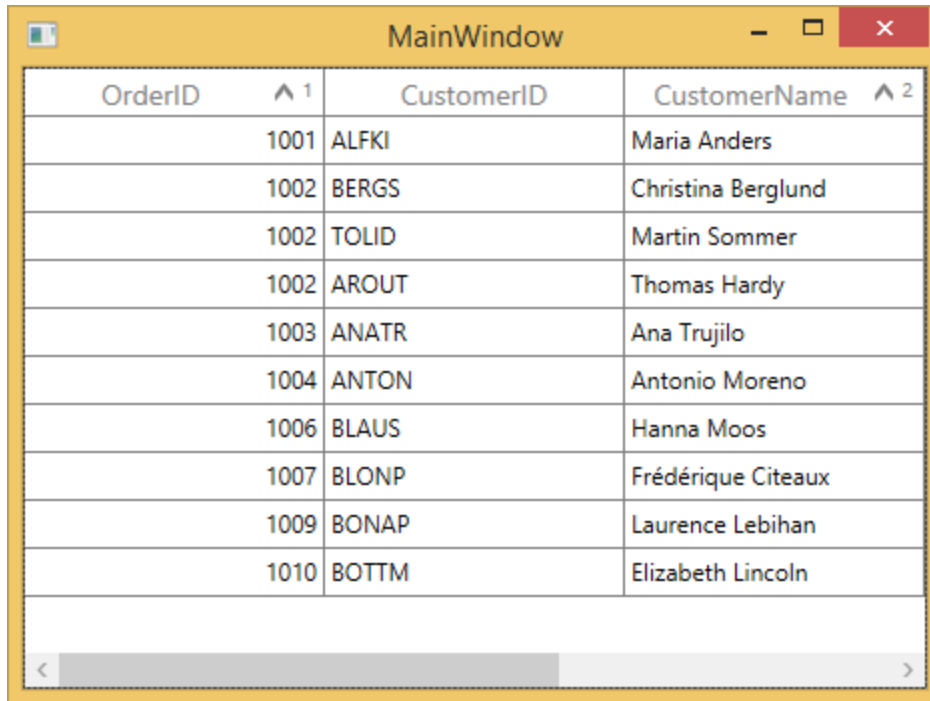
It is also possible to display sorted order of columns in header by setting [SfDataGrid.ShowSortNumbers](#) property to `true`.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowSorting="True"
    ShowSortNumbers="True"
    ItemsSource="{Binding Orders}" />
```

#### C#

```
this.dataGrid.ShowSortNumbers = true;
```



OrderID ^ 1	CustomerID	CustomerName ^ 2
1001	ALFKI	Maria Anders
1002	BERGS	Christina Berglund
1002	TOLID	Martin Sommer
1002	AROUT	Thomas Hardy
1003	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

### Programmatic sorting

You can sort the data programmatically by adding or removing the [SortColumnDescription](#) in [SfDataGrid.SortColumnDescriptions](#) property.

**Note:** [SfDataGrid.SortColumnsChanging](#) and [SfDataGrid.SortColumnsChanged](#) events are not raised when the data sorted programmatically through [SfDataGrid.SortColumnDescriptions](#).

### Adding sort columns

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoExpandGroups="True"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"
    ShowGroupDropArea="True">
    <syncfusion:SfDataGrid.SortColumnDescriptions>
    <syncfusion:SortColumnDescription ColumnName="OrderID"
        SortDirection="Ascending" />
    <syncfusion:SortColumnDescription ColumnName="CustomerName"
        SortDirection="Descending" />
    </syncfusion:SfDataGrid.SortColumnDescriptions>
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.SortColumnDescriptions.Add(new SortColumnDescription() {
    ColumnName = "OrderID", SortDirection = ListSortDirection.Ascending });
this.dataGrid.SortColumnDescriptions.Add(new SortColumnDescription() {
    ColumnName = "CustomerName", SortDirection = ListSortDirection.Descending
});
```

### Removing sort columns

You can unsort the data by removing the corresponding [SortColumnDescription](#) from the [SfDataGrid.SortColumnDescriptions](#) property.

#### C#

```
var sortColumnDescription =  
this.dataGrid.SortColumnDescriptions.FirstOrDefault(col => col.ColumnName ==  
"OrderID");  
if (sortColumnDescription!=null)  
{  
    this.dataGrid.SortColumnDescriptions.Remove(sortColumnDescription);  
}
```

### Clear sorting

You can clear sorting, by clearing the [SfDataGrid.SortColumnDescriptions](#).

#### C#

```
this.dataGrid.SortColumnDescriptions.Clear();
```

### Custom sorting

SfDataGrid allows you to sort the columns based on the custom logic. The custom sorting can be applied by adding the [SortComparer](#) instance to [SfDataGrid.SortComparers](#).

The [SortComparer](#) have the following properties,

[PropertyName](#) - Gets or sets the name of the column to apply custom sorting.

[Comparer](#) - Gets or sets the custom comparer in which you can code to compare the data using custom logic.

You can implement [ISortDirection](#) interface in comparer to get the sort direction. So you can apply different custom logics for ascending and descending.

Follow the below steps to add custom comparer to sort using custom logic,

### Define custom comparer with custom sort logic

In the below code snippet, **CustomerName** property is compared based on its string length, instead of default string comparison.

#### C#

```
Public class CustomComparer: IComparer<object>, ISortDirection  
{  
    public int Compare(object x, object y)  
    {  
        int nameX;  
        int nameY;  
        //While data object passed to comparer  
        if (x.GetType() == typeof(OrderInfo))  
        {  
            nameX = ((OrderInfo)x).CustomerName.Length;  
            nameY = ((OrderInfo)y).CustomerName.Length;  
        }  
        //While sorting groups  
        else if (x.GetType() == typeof(Group))
```

```

{
    //Calculating the group key length
    nameX = ((Group)x).Key.ToString().Length;
    nameY = ((Group)y).Key.ToString().Length;
}
else
{
    nameX = x.ToString().Length;
    nameY = y.ToString().Length;
}
//returns the comparison result based in SortDirection.
if (nameX.CompareTo(nameY) > 0)
return SortDirection == ListSortDirection.Ascending ? 1 : -1;
else if (nameX.CompareTo(nameY) == -1)
return SortDirection == ListSortDirection.Ascending ? -1 : 1;
else
return 0;
}
private ListSortDirection _SortDirection;
/// <summary>
/// Gets or sets the property that denotes the sort direction.
/// </summary>
/// <remarks>
/// SortDirection gets updated only when sorting the groups. For other
cases, SortDirection is always ascending.
/// </remarks>
public ListSortDirection SortDirection
{
    get { return _SortDirection; }
    set { _SortDirection = value; }
}
}

```

#### Adding custom comparer to SfDataGrid

Custom comparer can be added to [SfDataGrid.SortComparers](#) property. **SortComparers** maintains custom comparers and the custom comparer gets called when corresponding column gets sorted by clicking column header or programmatically.

#### XML

```

<xmlns:linq="clr-namespace:Syncfusion.Data;assembly=Syncfusion.Data.WPF">
<syncfusion:SfDataGrid x:Name="dataGrid"
AllowSorting="True"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.SortComparers>
<linq:SortComparer Comparer="{StaticResource Comparer}"
PropertyName="CustomerName" />
</syncfusion:SfDataGrid.SortComparers>
</syncfusion:SfDataGrid>

```

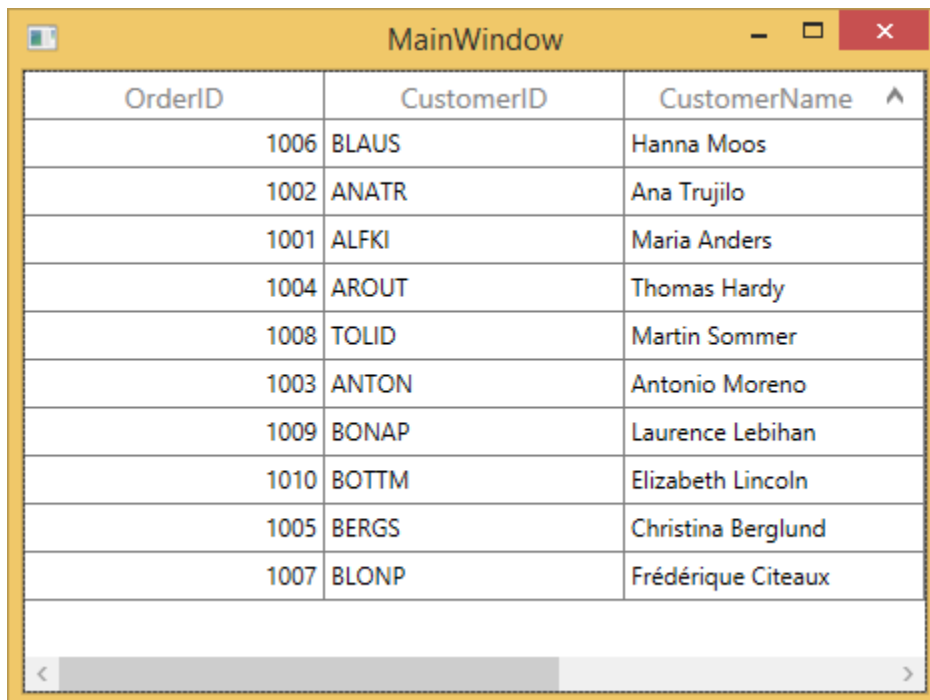
#### C#

```

this.dataGrid.SortComparers.Add(new SortComparer() { Comparer = new
CustomComparer(), PropertyName = "CustomerName" });

```

Sorting CustomerName column sorts the data using custom comparer available in `SfDataGrid.SortComparers`.



OrderID	CustomerID	CustomerName ▲
1006	BLAUS	Hanna Moos
1002	ANATR	Ana Trujilo
1001	ALFKI	Maria Anders
1004	AROUT	Thomas Hardy
1008	TOLID	Martin Sommer
1003	ANTON	Antonio Moreno
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln
1005	BERGS	Christina Berglund
1007	BLONP	Frédérique Citeaux

#### Sorting the underlying collection

SfDataGrid sorts the records in UI and maintains in its internal `CollectionView` and it will not change the order of data in underlying collection.

You can get sorted data from `SfDataGrid.View.Records` when groups is not in place and `SfDataGrid.View.TopLevelGroup.DisplayElements` when grouping in place.

If you want to sort the underlying collection when sorting takes place, then this can be achieved by handling [SfDataGrid.SortColumnsChanged](#) event.

#### C#

```
this.dataGrid.SortColumnsChanged += dataGrid_SortColumnsChanged;
void dataGrid_SortColumnsChanged(object sender,
GridSortColumnsChangedEventArgs e)
{
    var viewModel = this.DataContext as ViewModel;
    IEnumerable<OrderInfo> OrderedSource = viewModel.Orders;
    foreach (var sortColumn in this.dataGrid.View.SortDescriptions)
    {
        var columnName = sortColumn.PropertyName;
        if (sortColumn.Direction == ListSortDirection.Ascending)
            OrderedSource = OrderedSource.OrderBy(source => GetOrderSource(source,
            columnName));
        else
            OrderedSource = OrderedSource.OrderByDescending(source =>
            GetOrderSource(source, columnName));
    }
}
```

```
}  
private object GetOrderSource(OrderInfo source, string name)  
{  
    var propInfo = source.GetType().GetRuntimeProperty(name);  
    if (propInfo != null)  
        // get the current sort column value  
        return propInfo.GetValue(source);  
    return null;  
}
```

Handling events

*SortColumnsChanging event*

[SfDataGrid.SortColumnsChanging](#) event occurs while sorting the columns by clicking column header.

[GridSortColumnsChangingEventArgs](#) has following members which provides information for [SortColumnsChanging](#) event.

[Action](#) – Gets the action triggered this event.

[Cancel](#) – Setting value to `true`, cancels the triggered action.

[AddedItems](#) - Gets the list of new [SortColumnDescription](#)'s that are added.

[RemovedItems](#) - Gets the list of [SortColumnDescription](#)'s that are removed.

[CancelScroll](#) - Gets or sets a value that indicates, whether scroll and bring [SelectedItem](#) in view after sorting takes place.

You can prevent sorting for the particular column through [GridSortColumnsChangingEventArgs.Cancel](#) property of [SortColumnsChanging](#) event.

## C#

```
this.dataGrid.SortColumnsChanging += dataGrid_SortColumnsChanging;  
void dataGrid_SortColumnsChanging(object sender,  
Syncfusion.UI.Xaml.Grid.GridSortColumnsChangingEventArgs e)  
{  
    if (e.AddedItems[0].ColumnName == "OrderID")  
    {  
        e.Cancel = true;  
    }  
}
```

*SortColumnsChanged event*

[SfDataGrid.SortColumnsChanged](#) event occurs when the sorting is applied to the column.

[GridSortColumnsChangedEventArgs](#) provides information for [SortColumnsChanged](#) event.

See Also

[How to sort a column in WPF DataGrid?](#)

[How to customize the Filtering and Sorting icons in the SfDataGrid ?](#)

[How to sort multiple column without pressing Ctrl key in SfDataGrid?](#)

[How to change the position of FilterToggleButton and SortIcon in header cell of SfDataGrid?](#)

[How to sort your binded collection of ViewModel?](#)

## Grouping in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) allows you to group the data against one or more columns. When grouping is applied, the data is organized into a hierarchical structure based on matching column values and it is sorted by ascending order.

SfDataGrid allows you to group the data in below ways,

- UI Grouping
- Programmatic Grouping

## UI grouping

You can allow end-user to group the data by setting [SfDataGrid.AllowGrouping](#) property to `true`, where user can drag and drop the column into `GroupDropArea` to group based on that column.

When the column is grouped, records that have an identical value in the column are combined to form a group. The `GroupDropArea` can be enabled by setting the [SfDataGrid.ShowGroupDropArea](#) property to `true`.

XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowGrouping="False"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"
    ShowGroupDropArea="True" />
```

C#

```
this.dataGrid.AllowGrouping = true;
```

You can enable or disable grouping on particular column by setting the [GridColumn.AllowGrouping](#) property.

XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="False"
    ItemsSource="{Binding Orders}"
    ShowGroupDropArea="True">
    <syncfusion:SfDataGrid.Columns>
    <syncfusion:GridTextColumn AllowGrouping="True" MappingName="OrderID" />
    <syncfusion:GridTextColumn AllowGrouping="False" MappingName="CustomerID" />
    </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

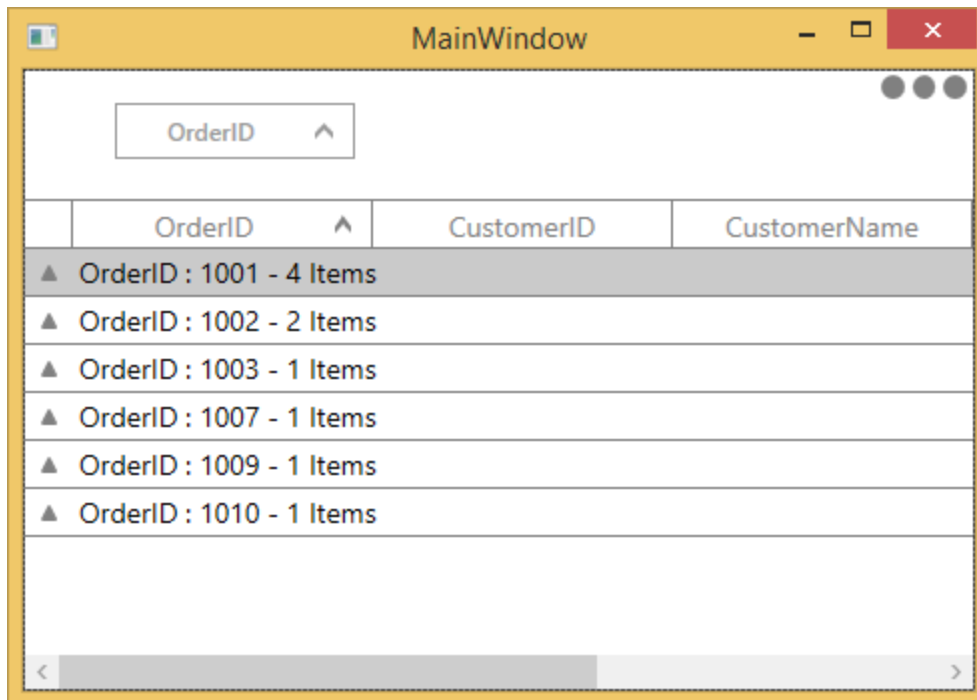
C#

```
this.dataGrid.Columns["OrderID"].AllowGrouping = true;
this.dataGrid.Columns["CustomerID"].AllowGrouping = true;
```

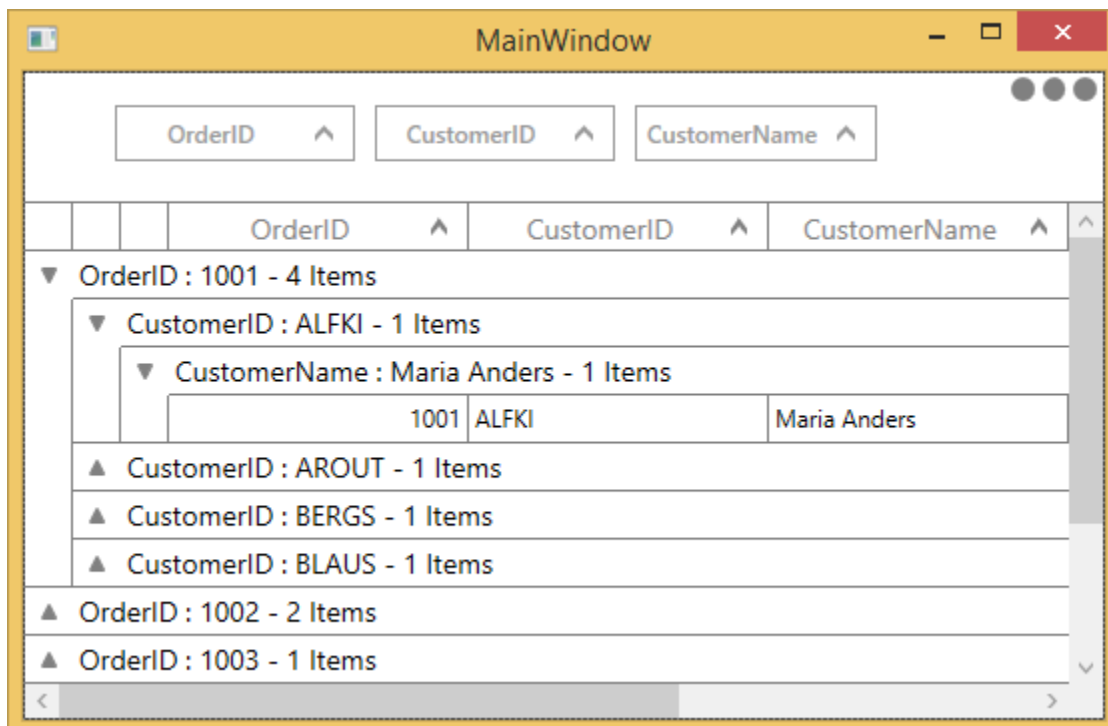
---

**Note:** [GridColumn.AllowGrouping](#) takes higher priority than [SfDataGrid.AllowGrouping](#).

---



The data can be grouped by an unlimited number of columns. To group more than one columns, drag-and-drop the desired columns in to **GroupDropArea**.



Each group is identified by its **CaptionSummaryRows** and it is used to organize the data into a hierarchical tree structure based on identical values of that column. The underlying records in each caption summary row can be expanded or collapsed by clicking its group caption.



Each `CaptionSummaryRow` carries information about a particular group like group name, number of items (records) in the group, etc. You can refer [Caption Summaries](#) section, for more information about `CaptionSummaryRow`.

#### Programmatic grouping

The WPF DataGrid (SfDataGrid) allows you to group the data programmatically by adding or removing [GroupColumnDescription](#) to [SfDataGrid.GroupColumnDescriptions](#) collection.

For example, if you want to group the `OrderID` column programmatically, define its [MappingName](#) to [ColumnName](#) property of `GroupColumnDescription`. Then add the `GroupColumnDescription` to the `SfDataGrid.GroupColumnDescriptions` collection.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True">
<syncfusion:SfDataGrid.GroupColumnDescriptions>
<syncfusion:GroupColumnDescription ColumnName="OrderID" />
</syncfusion:SfDataGrid.GroupColumnDescriptions>
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.GroupColumnDescriptions.Add(new GroupColumnDescription() {
ColumnName = "OrderID" });
```

You can group more than one column programmatically.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True">
<syncfusion:SfDataGrid.GroupColumnDescriptions>
<syncfusion:GroupColumnDescription ColumnName="OrderID" />
<syncfusion:GroupColumnDescription ColumnName="CustomerID" />
</syncfusion:SfDataGrid.GroupColumnDescriptions>
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.View.BeginInit();
this.dataGrid.GroupColumnDescriptions.Add(new GroupColumnDescription() {
ColumnName = "OrderID" });
this.dataGrid.GroupColumnDescriptions.Add(new GroupColumnDescription() {
ColumnName = "CustomerID" });
this.dataGrid.View.EndInit();
```

#### Group based on display text

You can group the column in DataGrid based on the value being displayed in cell by setting [GridColumn.GroupMode](#) as `Display`.

In the below example, OrderID column displays value with one decimal digit in cell. But when you group, groups will be created based on actual value considering all decimal digits of value (Refer right side screen shot). You can group based value displayed in the cell by setting [GridColumn.GroupMode](#) as `Display` (Refer left side screen shot for the same data).

### XML

```
<Syncfusion:GridNumericColumn NumberDecimalDigits="1" MappingName="OrderID"
HeaderText="OrderID" GroupMode="Display"/>
```

### C#

```
this.datagrid.Columns["OrderID"].GroupMode = DataReflectionMode.Display;
```

OrderID	ShipId
<b>Display based Grouping</b>	
▼ OrderID : 1001.0 - 2 Items	
1001.0	UK
1001.0	China
▼ OrderID : 1001.1 - 7 Items	
1001.1	Australia
1001.1	Switzerland
1001.1	British
1001.1	Britain
1001.1	US
1001.1	Australia
1001.1	Britain
▼ OrderID : 1001.2 - 1 Items	
1001.2	Switzerland
▼ OrderID : 1001.3 - 2 Items	
1001.3	British
<b>Value based Grouping</b>	
▼ OrderID : 1001.01 - 2 Items	
1001.0	UK
1001.0	China
▼ OrderID : 1001.05 - 3 Items	
1001.1	Switzerland
1001.1	British
1001.1	Australia
▼ OrderID : 1001.07 - 1 Items	
1001.1	Britain
▼ OrderID : 1001.12 - 2 Items	
1001.1	Australia
1001.1	Britain
▼ OrderID : 1001.13 - 1 Items	
1001.1	US
▲ OrderID : 1001.17 - 1 Items	

*Group caption based on DisplayMember when grouping GridComboBoxColumn and GridMultiColumnDropDownList*

In WPF DataGrid (SfDataGrid), you can group the column based on display value and also the same can be displayed in caption summary by setting [GridColumn.GroupMode](#) as `Display`.

### XML

```
<Syncfusion:GridComboBoxColumn ItemsSource="{Binding ComboItemsSource}"
MappingName="ShipId" DisplayMemberPath="ShipCity" SelectedValuePath="ShipId"
GroupMode="Display"/>
```

**C#**

```
this.datagrid.Columns.Add(new GridComboBoxColumn()
{
    ItemsSource = viewModel.ComboItemsSource,
    DisplayMemberPath = "ShipCity",
    MappingName = "ShipId",
    SelectedValuePath = "ShipId",
    GroupMode = DataReflectionMode.Display
});
```

ShipId ^

Display based Grouping

ShipId ^	OrderID ^
▼ ShipId : Australia - 2 Items	
Australia	1001
Australia	1001
▼ ShipId : Britain - 3 Items	
Britain	1001
Britain	1001
Britain	1001
▼ ShipId : British - 2 Items	
British	1001
British	1001
▼ ShipId : China - 1 Items	
China	1001
▼ ShipId : Switzerland - 1 Items	
Switzerland	1001
▼ ShipId : UK - 1 Items	

ShipId ^

Value based Grouping

ShipId ^	OrderID ^
▼ ShipId : 101 - 1 Items	
UK	1001
▼ ShipId : 102 - 1 Items	
China	1001
▼ ShipId : 103 - 2 Items	
Australia	1001
Australia	1001
▼ ShipId : 104 - 1 Items	
Switzerland	1001
▼ ShipId : 105 - 2 Items	
British	1001
British	1001
▼ ShipId : 106 - 3 Items	
Britain	1001
Britain	1001
Britain	1001

## Clearing or removing group

You can remove all the groups by clearing [SfDataGrid.GroupColumnDescriptions](#) collection.

**C#**

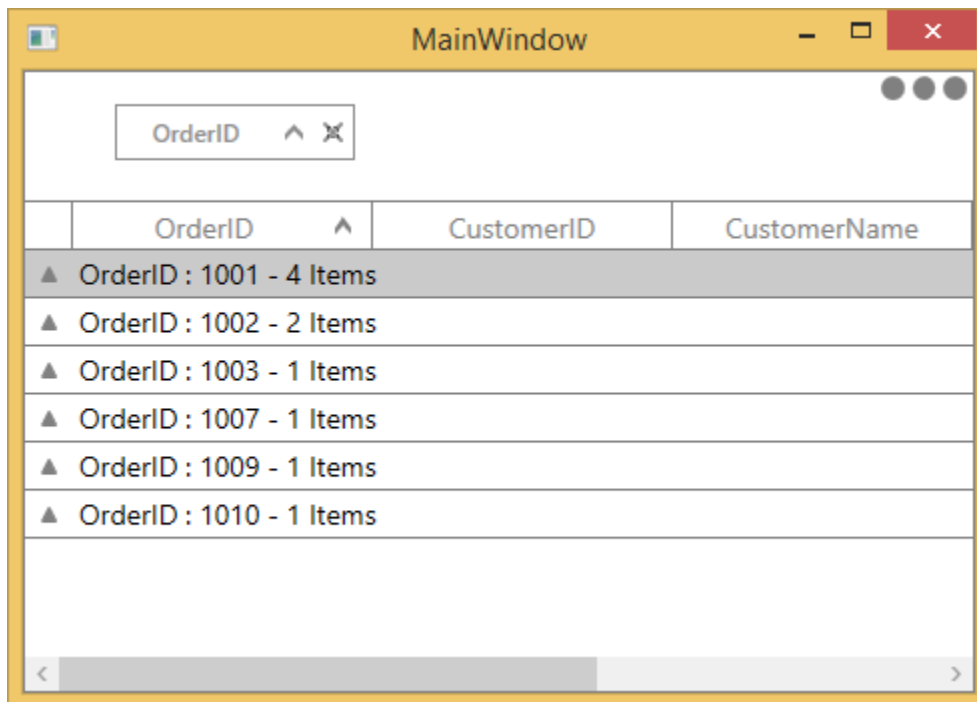
```
this.datagrid.GroupColumnDescriptions.Clear();
```

You can ungroup the column programmatically at runtime by removing [GroupColumnDescription](#) from [SfDataGrid.GroupColumnDescriptions](#) collection.

**C#**

```
this.datagrid.View.BeginInit();
this.datagrid.GroupColumnDescriptions.Remove(this.datagrid.GroupColumnDescriptions[0]);
this.datagrid.View.EndInit();
```

To ungroup the column in UI, click the close button on column header or drag the column header from the GroupDropArea and drop it on the header row.



Hiding the column when grouped

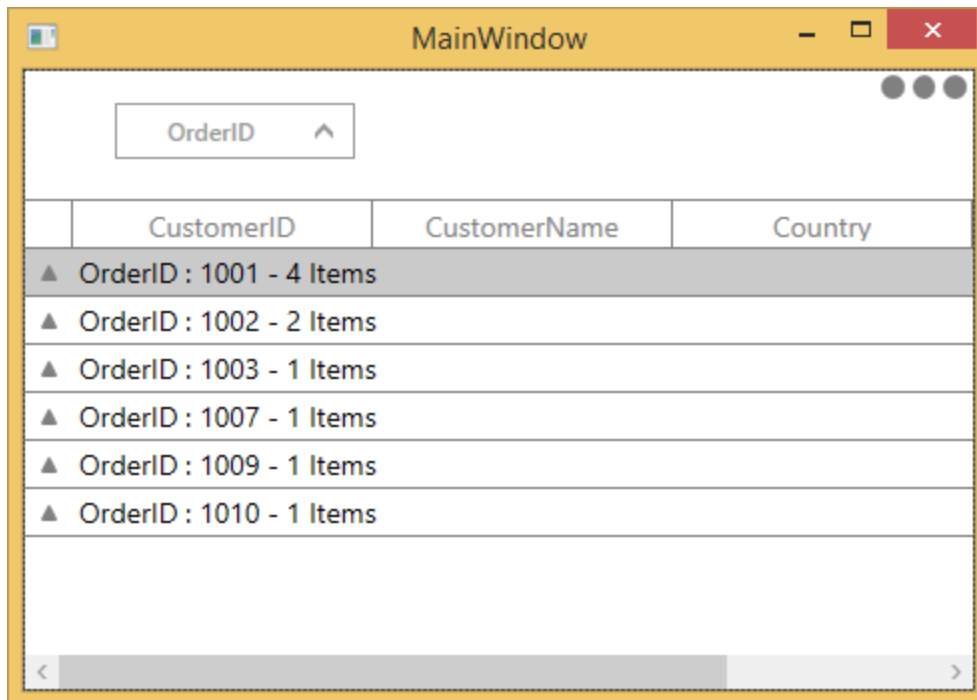
You can hide the column header when the particular column gets grouped by setting [SfDataGrid.ShowColumnWhenGrouped](#) property to `false`.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"
    ShowColumnWhenGrouped="False"
    ShowGroupDropArea="True" />
```

#### C#

```
this.dataGrid.ShowColumnWhenGrouped = false;
```



### Freezing caption rows when scrolling

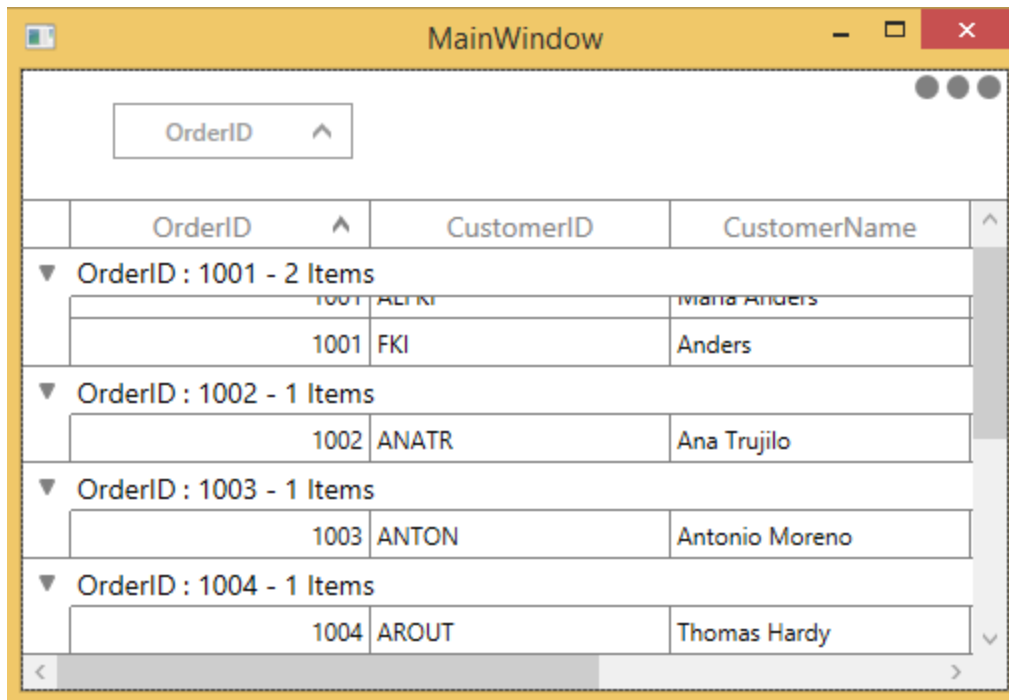
You can freeze the group caption of the group in view until its records scrolled out of the view by setting the [SfDataGrid.AllowFrozenGroupHeaders](#) property to `true`.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    AllowFrozenGroupHeaders="True"
    ItemsSource="{Binding Orders}"
    ShowGroupDropArea="True" />
```

### C#

```
this.dataGrid.AllowFrozenGroupHeaders = true;
```



### Expanding or collapsing the groups

By default, you can view the records in each group by expanding or collapsing its group caption.

You can allow end-user to expand or collapse the groups programmatically at runtime.

#### *Expand groups while grouping*

You can expand all the groups while grouping by setting [SfDataGrid.AutoExpandGroups](#) to `true`. So, when user group any column, then all groups will be in expanded state.

### **XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoExpandGroups="True"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"
    ShowGroupDropArea="True" />
```

### **C#**

```
this.dataGrid.AutoExpandGroups = true;
```

#### *Programmatically expand or collapse the groups*

##### *Expand or collapse all the Groups*

You can expand or collapse all the groups at programmatically at runtime by using [SfDataGrid.ExpandAllGroup](#) and [SfDataGrid.CollapseAllGroup](#) methods.

### **C#**

```
this.dataGrid.ExpandAllGroup();
this.dataGrid.CollapseAllGroup();
```

Expand or Collapse the Group based on its level

You can expand or collapse the group based on its level by using [SfDataGrid.ExpandGroupsAtLevel](#) and [SfDataGrid.CollapseGroupsAtLevel](#) methods.

#### C#

```
this.dataGrid.ExpandGroupsAtLevel(2);  
this.dataGrid.CollapseGroupsAtLevel(2);
```

Expand or Collapse the specific Group

You can expand or collapse specific group by using [SfDataGrid.ExpandGroup](#) and [SfDataGrid.CollapseGroup](#) methods.

#### C#

```
var group = (dataGrid.View.Groups[0] as Group);  
this.dataGrid.ExpandGroup(group);  
this.dataGrid.CollapseGroup(group);
```

Customize indent column width

You can customize the width of IndentColumn in SfDataGrid by using [IndentColumnWidth](#) property as like below.

#### XML

```
<Syncfusion:SfDataGrid x:Name="dataGrid"  
AllowGrouping="True"  
IndentColumnWidth="50"  
ShowGroupDropArea="True"  
ItemsSource="{Binding OrderInfoCollection }">
```

#### C#

```
this.dataGrid.IndentColumnWidth = 50;
```

GroupDropArea customization

*GroupDropArea text*

You can change the **GroupDropArea**'s text can by setting [SfDataGrid.GroupDropAreaText](#) property.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"  
GroupDropAreaText="Drag and drop the columns here"  
ItemsSource="{Binding Orders}"  
ShowGroupDropArea="True" />
```



OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1001	ANATR	Ana Trujilo	Mexico	México D.F.
1001	ANTON	Antonio Moreno	Mexico	México D.F.
1002	AROUT	Thomas Hardy	UK	London
1002	BERGS	Christina Berglund	Sweden	Luleå
1003	BLAUS	Hanna Moos	Germany	Mannheim
1003	BLONP	Frédérique Citeaux	France	Strasbourg
1003	BOLID	Martin Sommer	Spain	Madrid
1004	BONAP	Laurence Lebihan	France	Marseille
1005	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

#### GroupDropArea height and appearance

SfDataGrid allows you to customize the appearance and height of **GroupDropArea** by writing the style of TargetType **GroupDropArea**.

#### XML

```
<Window.Resources>
<Style TargetType="syncfusion:GroupDropArea">
<Setter Property="Height" Value="80" />
<Setter Property="Foreground" Value="Red" />
<Setter Property="Background" Value="Pink" />
</Style>
</Window.Resources>
```



OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen



### Expanding GroupDropArea while loading

By default, the `GroupDropArea` will be expanded while dragging the column towards the `GroupDropArea`. You can set the `GroupDropArea` to be always expanded by setting the [SfDataGrid.IsGroupDropAreaExpanded](#) property to `true`.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
IsGroupDropAreaExpanded="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True" />
```

### C#

```
this.dataGrid.IsGroupDropAreaExpanded = true;
```

### Custom grouping

DataGrid allows you to group the data based on custom logic when the built-in grouping functionality doesn't meet your requirement.

To perform custom grouping on a particular column, specify the custom logic through [GroupColumnDescription.Converter](#) property and the column name to [GroupColumnDescription.ColumnName](#) property.

For an example, the Date column is grouped based on the week basis in the following example.

### C#

```
public class GroupDateTimeConverter : IValueConverter
{
    public object Convert(object value, System.Type targetType, object
parameter, CultureInfo culture)
    {
        var saleinfo = value as SalesByDate;
        var dt = DateTime.Now;
        var days = (int)Math.Floor((dt - saleinfo.Date).TotalDays);
        var dayofweek = (int)dt.DayOfWeek;
        var difference = days - dayofweek;
        if (days <= dayofweek)
        {
            if (days == 0)
                return "TODAY";
            if (days == 1)
                return "YESTERDAY";
            return saleinfo.Date.DayOfWeek.ToString().ToUpper();
        }
        if (difference > 0 && difference <= 7)
            return "LAST WEEK";
        if (difference > 7 && difference <= 14)
            return "TWO WEEKS AGO";
        if (difference > 14 && difference <= 21)
            return "THREE WEEKS AGO";
        if (dt.Year == saleinfo.Date.Year && dt.Month == saleinfo.Date.Month)
            return "EARLIER THIS MONTH";
    }
}
```

```

if (DateTime.Now.AddMonths(-1).Month == saleinfo.Date.Month)
return "LAST MONTH";
return "OLDER";
}
public object ConvertBack(object value, System.Type targetType, object
parameter, CultureInfo culture)
{
throw new NotImplementedException();
}
}

```

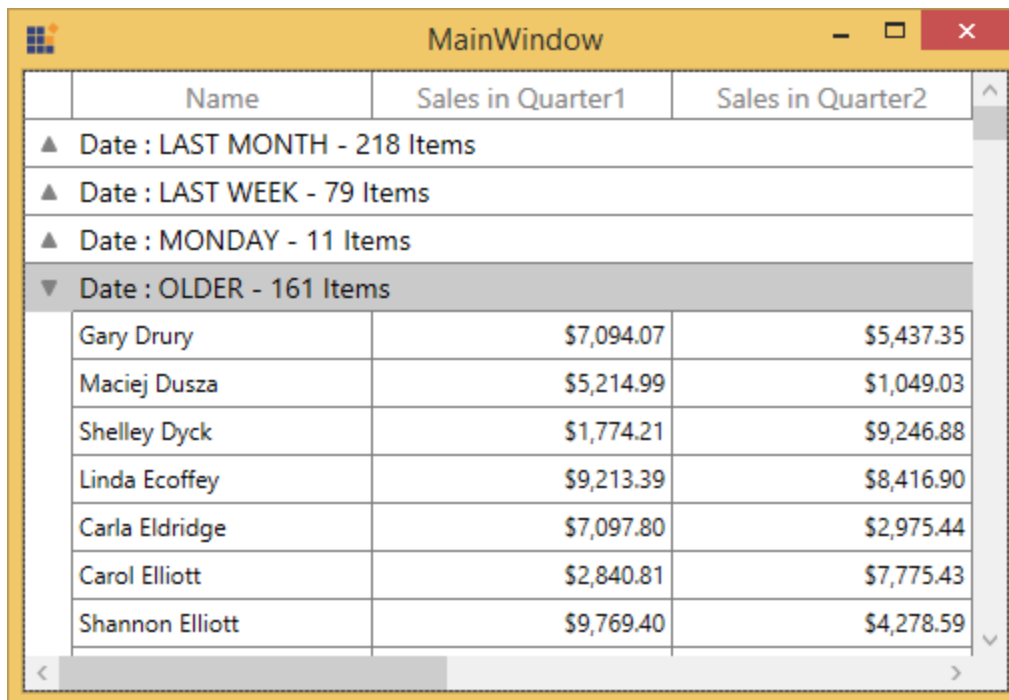
Now, assign the `GroupDateTimeConverter` into `GroupColumnDescription.Converter` and set `Date` property to `GroupColumnDescription.ColumnName` property.

#### XML

```

<Window.Resources>
<local:GroupDateTimeConverter x:Key="customGroupDateTimeConverter" />
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding DailySalesDetails}">
<syncfusion:SfDataGrid.GroupColumnDescriptions>
<syncfusion:GroupColumnDescription ColumnName="Date"
Converter="{StaticResource customGroupDateTimeConverter}" />
</syncfusion:SfDataGrid.GroupColumnDescriptions>
</syncfusion:SfDataGrid>

```



	Name	Sales in Quarter1	Sales in Quarter2
▲	Date : LAST MONTH - 218 Items		
▲	Date : LAST WEEK - 79 Items		
▲	Date : MONDAY - 11 Items		
▼	Date : OLDER - 161 Items		
	Gary Drury	\$7,094.07	\$5,437.35
	Maciej Dusza	\$5,214.99	\$1,049.03
	Shelley Dyck	\$1,774.21	\$9,246.88
	Linda Ecoffey	\$9,213.39	\$8,416.90
	Carla Eldridge	\$7,097.80	\$2,975.44
	Carol Elliott	\$2,840.81	\$7,775.43
	Shannon Elliott	\$9,769.40	\$4,278.59

You can download samples from below location,

Refer [sample](#) for Custom grouping when ItemsSource is ObservableCollection.

Refer [sample](#) for Custom grouping when ItemsSource is DataTable.

You can refer [here](#) to apply custom sorting when grouping is applied.

#### Sorting the grouped column records

In custom grouping, you can sort all the inner records of each group by setting

#### [GroupColumnDescription.SortGroupRecords](#)

sorted based on the column name described in [GroupColumnDescription](#).

#### XML

```
<syncfusion:SfDataGrid.GroupColumnDescriptions>
<syncfusion:GroupColumnDescription ColumnName="SickLeaveHours"
Converter="{StaticResource customGrouping}"
SortGroupRecords="True" />
</syncfusion:SfDataGrid.GroupColumnDescriptions>
```

#### C#

```
GroupColumnDescription groupColumnDesc = new GroupColumnDescription()
{
    ColumnName = "SickLeaveHours",
    Converter = new CustomGroupingConverter(),
    SortGroupRecords = true
};
dataGrid.GroupColumnDescriptions.Add(groupColumnDesc);
```

In the below screenshot custom grouping is applied based on **SickLeaveHours** column and the inner records in each group are sorted based on **SickLeaveHours** value.

SfDataGrid Demo

Name	EmployeeID	SickLeaveHours ^	Gender	Salary
▼ SickLeaveHours : SickLeaveHours above 15 - 2 Items				
James	151	19.00	FeMale	\$25000.00
Christina	106	21.00	FeMale	\$15000.00
▼ SickLeaveHours : SickLeaveHours between 1 and 5 - 5 Items				
Hannah	101	2.00	Male	\$25000.00
Maxwell	104	2.00	Male	\$20000.00
Stephen	102	4.00	FeMale	\$25000.00
Humberto	103	5.00	FeMale	\$20000.00
Maxwell	121	5.00	Male	\$25000.00
▼ SickLeaveHours : SickLeaveHours between 10 and 15 - 3 Items				
Thomas	108	12.00	Male	\$55000.00
Robert	107	13.00	Male	\$75000.00
Christina	151	13.00	Male	\$26000.00
▼ SickLeaveHours : SickLeaveHours between 5 and 10 - 5 Items				
Michael	111	8.00	FeMale	\$45000.00
Humberto	131	8.00	Male	\$35000.00
Milton	101	9.00	FeMale	\$25000.00
Ronald	105	10.00	Male	\$25000.00
Cory	201	10.00	FeMale	\$10000.00

### Sorting groups based on summary values

DataGrid allows you to sort the groups based its summary values. You can sort the groups based on summary aggregate value by using [SfDataGrid.SummaryGroupComparer](#) property.

Follow the below steps to sort the groups based on caption aggregate value.

#### Define custom group comparer with custom sort logic

In the below code snippet, the **OrderID** column compared and sorted based on the number of order count in each **OrderID**.

#### C#

```
public class SortGroupComparers : IComparer<Group>, ISortDirection
{
    public ListSortDirection SortDirection { get; set; }
    public int Compare(Group x, Group y)
    {
        int cmp = 0;
        var xgroupSummary = Convert.ToInt32((x as
        Group).GetSummaryValue(x.SummaryDetails.SummaryRow.SummaryColumns[0].Mapping
        Name, "Count"));
        var ygroupSummary = Convert.ToInt32((y as
        Group).GetSummaryValue(x.SummaryDetails.SummaryRow.SummaryColumns[0].Mapping
        Name, "Count"));
        cmp = ((IComparable)xgroupSummary).CompareTo(ygroupSummary);
        if (this.SortDirection == ListSortDirection.Descending)
            cmp = -cmp;
        return cmp;
    }
}
```

#### Defining custom group comparer to DataGrid

Custom group comparer can be defined in SfDataGrid using [SfDataGrid.SummaryGroupComparer](#) property. **SummaryGroupComparer** maintains the custom comparers and the custom comparer gets called when corresponding column is grouped.

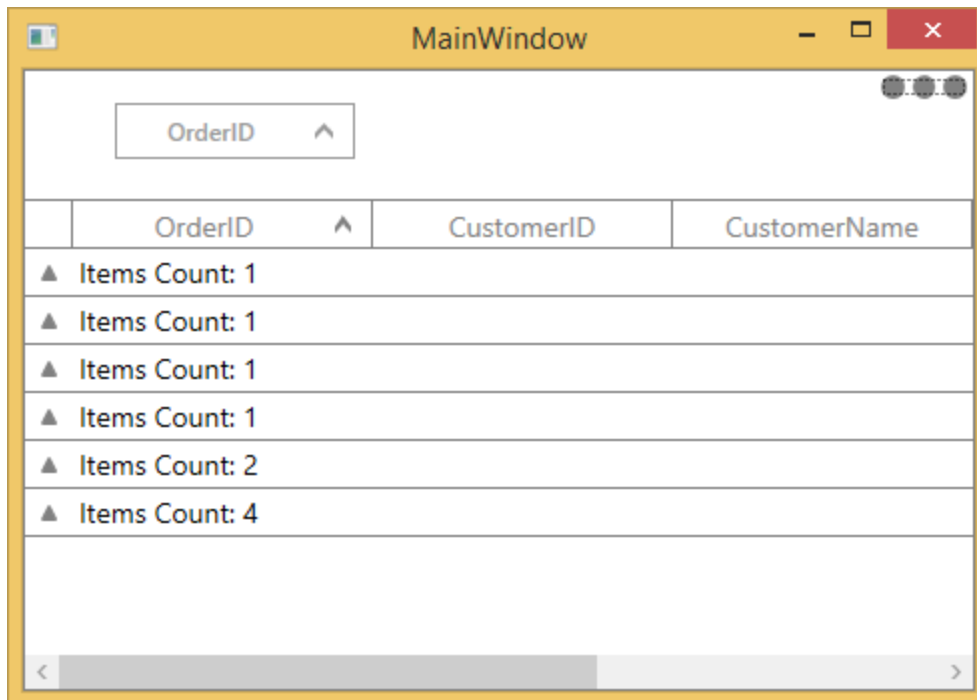
#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowSorting="True"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"
    ShowGroupDropArea="True"
    SummaryGroupComparer="{StaticResource groupComparer}">
    <syncfusion:SfDataGrid.GroupColumnDescriptions>
    <syncfusion:GroupColumnDescription ColumnName="OrderID" />
    </syncfusion:SfDataGrid.GroupColumnDescriptions>
    <syncfusion:SfDataGrid.CaptionSummaryRow>
    <syncfusion:GridSummaryRow Title="Items Count: {IDCount}"
        ShowSummaryInRow="True">
    <syncfusion:GridSummaryRow.SummaryColumns>
    <syncfusion:GridSummaryColumn Name="IDCount"
        Format="' {Count} '"
        MappingName="OrderID"
        SummaryType="CountAggregate" />
    </syncfusion:GridSummaryRow.SummaryColumns>
```

```

</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>

```



You can download the sample demo [here](#).

### Grouping events

#### *GroupExpanding event*

The [SfDataGrid.GroupExpanding](#) event occurs when the group is being expanded.

The [GroupChangingEventArgs](#) of the [GroupExpanding](#) event provides the information about the expanding group and it has the following members.

[Group](#) - Gets the group that's being expanded.

[Cancel](#) – Decides whether to cancel the group expansion.

You can cancel the group expansion by setting [GroupChangingEventArgs.Cancel](#) to `true`.

### C#

```

this.dataGrid.GroupExpanding += dataGrid_GroupExpanding;
void dataGrid_GroupExpanding(object sender,
Syncfusion.UI.Xaml.Grid.GroupChangingEventArgs e)
{
    if (e.Group.Key.Equals(1001))
    {
        e.Cancel = true;
    }
}

```

#### *GroupExpanded event*

The [SfDataGrid.GroupExpanded](#) event occurs after the group is expanded.

The [GroupChangedEventArgs](#) of the **GroupExpanded** event provides the information about the expanded group and it has the following member.

[Group](#) - Gets the expanded group.

#### *GroupCollapsing event*

The [SfDataGrid.GroupCollapsing](#) event occurs when the group is being collapsed.

The [GroupChangingEventArgs](#) of the **GroupCollapsing** event provides the information about the collapsing group and it contains the following member.

[Group](#) - Gets the group that's being collapsed.

[Cancel](#) – Decides whether to cancel the group collapsing.

You can cancel the group is being collapsed by using [GroupChangingEventArgs.Cancel](#) of **GroupCollapsing** event.

#### **C#**

```
this.dataGrid.GroupCollapsing += dataGrid_GroupCollapsing;
void dataGrid_GroupCollapsing(object sender, GroupChangingEventArgs e)
{
    if (e.Group.Key.Equals(1001))
        e.Cancel = true;
}
```

#### *GroupCollapsed event*

The [SfDataGrid.GroupCollapsed](#) event occurs after the group is collapsed.

[GroupChangedEventArgs](#) of the **GroupCollapsed** event provides the information about collapsed group and it contains the following member.

[Group](#) - Gets the collapsed group.

See Also

[How to remove the gridline of WPF DataGrid \(SfDataGrid\) with grouping?](#)

[How to maintain expanded state of groups in printing?](#)

[How to show vertical border to the column wise summary rows?](#)

[How to define summary rows using AttachedProperty in datagrid](#)

[How to displaying Group header name based on other column](#)

[How to clear selection while grouping/ungrouping?](#)

[How to add selection to data rows of each group on expanding its CaptionSummaryRow?](#)

[How to change the CaptionSummaryRow Style based on the grouping level?](#)

[How to Change the GroupCaptionText based on Display member of the GridComboBoxColumn?](#)

[How to add controls like TextBox in GroupDropArea?](#)

[How to customize the CaptionSummaryCell text in the SfDataGrid?](#)

[How to apply the Custom Grouping while grouping the column using GroupDropArea?](#)

[How to avoid selection while grouping and ungrouping in SfDataGrid?](#)

[How to customize the GroupDropArea?](#)

## Summaries in WPF DataGrid (SfDataGrid)

SfDataGrid provides support to display the concise information about the data objects using summaries. SfDataGrid provides below three different types of summary rows.

- **Table Summary** – Used to display summary information of table either at top or bottom of SfDataGrid.
- **Group Summary** – used to display summary information of data objects in each group.
- **Caption Summary** – used to display summary information in the caption of the group.

OrderID ^					
OrderID ^	Order ID	Customer ID	Customer Name	Product Name	Price
Total Income : \$463,138.00					
▼ Total Discount : (\$10.00) for 1 Products					
1000	1000	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53345.00
Total Discount : (\$10.00)					
▼ Total Discount : \$0.00 for 4 Products					
1001	1001	FKI	Anders	Boston Crab Meat	\$43350.00
1001	1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13433.00
1001	1001	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	\$60434.00
1001	1001	BERGS	Christina Berglund	Gorgonzola Telino	\$36343.00
Total Discount : \$0.00					
▼ Total Discount : (\$53.00) for 3 Products					
1002	1002	BLAUS	Hanna Moos	Chartreuse verte	\$5345.00
1002	1002	BLONP	Frédérique Citeaux	Fløtemysost	\$70453.00
1002	1002	BOLID	Martin Sommer	Carnarvon Tigers	\$84340.00
Total Discount : (\$53.00)					
▼ Total Discount : (\$32.00) for 2 Products					
1005	1005	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	\$30335.00
1005	1005	BOTTM	Elizabeth Lincoln	Vegie-spread	\$65760.00
Total Discount : (\$32.00)					
Total Products Count: 10					

Summary rows are represented by using [GridSummaryRow](#) and each [GridSummaryRow](#) hold summary information of columns in [SummaryColumns](#) property . The [SummaryColumns](#) contains the collection of [GridSummaryColumn](#) which carries the name of column ,format and its summary aggregate type.

### Table Summary

The table summary calculates the summary value over all the records. SfDataGrid allows you to add any number of table summary rows in top and bottom of SfDataGrid.

You can add table summary row in SfDataGrid by adding [GridTableSummaryRow](#) to [SfDataGrid.TableSummaryRows](#) collection.

*Defining summary for column*

You can display summary information in the column by setting [GridSummaryRow.ShowSummaryInRow](#) to `false` and defining summary columns. To calculate summary based on column you have to specify the below properties,

1. [GridSummaryColumn.MappingName](#) – MappingName of the column (Property name of data object) that you want calculate summary.
2. [GridSummaryColumn.SummaryType](#) – SfDataGrid provides different built-in summary calculation functions for various types.
3. [GridSummaryColumn.Format](#) – Used to define format string for summary based on support function name's in specified SummaryType.

Refer [Formatting Summary](#) section to know more about how to format summary and [Aggregate Types](#) section to know about different SummaryTypes.

In the below code snippet, summary is defined for `UnitPrice` and `ProductName` columns.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="'Total UnitPrice : {Sum:c} '"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="'Total Product Count : {Count:d} '"
MappingName="ProductName"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>
```

**C#**

```
this.dataGrid.TableSummaryRows.Add(new GridTableSummaryRow()
{
    ShowSummaryInRow = false,
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name = "PriceAmount",
            MappingName="UnitPrice",
            SummaryType= SummaryType.Int32Aggregate,
            Format="Total UnitPrice : {Sum:c}"
        },
        new GridSummaryColumn()
        {

```



```

Name="ProductCount",
MappingName="ProductName",
SummaryType=SummaryType.CountAggregate,
Format="Total Product Count : {Count:d}"
},
}
}
});

```

Order ID	Customer ID	Customer Name	Product Name	Price
1000	ALFKI	Maria Anders	Alice Mutton	\$34343.00
1001	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53345.00
1002	FKI	Anders	Boston Crab Meat	\$43350.00
1003	ANTON	Antonio Moreno	Raclette Courdavault	\$13433.00
1004	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	\$60434.00
1005	BERGS	Christina Berglund	Gorgonzola Telino	\$36343.00
1006	BLAUS	Hanna Moos	Chartreuse verte	\$5345.00
1007	BLONP	Frédérique Citeaux	Fløtemysost	\$70453.00
1008	BOLID	Martin Sommer	Carnarvon Tigers	\$84340.00
1009	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	\$30335.00
1010	BOTTM	Elizabeth Lincoln	Vegie-spread	\$65760.00
Total Product Count : 11				Total UnitPrice : \$497,481.00

#### Displaying summary for Row

You can display summary information in row by setting [GridSummaryRow.ShowSummaryInRow](#) to `true` and defining summary columns. You have to set [GridSummaryRow.Title](#) based on [GridSummaryColumn.Name](#) property to format summary columns values in row.

Refer [Formatting Summary](#) section to know more about how to format summary.

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow Title=" Total Price : {PriceAmount} for
{ProductCount} products " ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>

```

**C#**

```

this.dataGrid.TableSummaryRows.Add(new GridTableSummaryRow()
{
    ShowSummaryInRow = true,
    Title = "Total Price: {PriceAmount} for {ProductCount} Products",
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name = "PriceAmount",
            MappingName="UnitPrice",
            SummaryType= SummaryType.Int32Aggregate,
            Format="{Sum:c}"
        },
        new GridSummaryColumn()
        {
            Name="ProductCount",
            MappingName="ProductName",
            SummaryType=SummaryType.CountAggregate,
            Format="{Count:d}"
        },
    }
});

```

Order ID	Customer ID	Customer Name	Product Name	Price
1000	ALFKI	Maria Anders	Alice Mutton	\$34343.00
1001	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53345.00
1002	FKI	Anders	Boston Crab Meat	\$43350.00
1003	ANTON	Antonio Moreno	Raclette Courdavault	\$13433.00
1004	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	\$60434.00
1005	BERGS	Christina Berglund	Gorgonzola Telino	\$36343.00
1006	BLAUS	Hanna Moos	Chartreuse verte	\$5345.00
1007	BLONP	Frédérique Citeaux	Fløtemysost	\$70453.00
1008	BOLID	Martin Sommer	Carnarvon Tigers	\$84340.00
1009	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	\$30335.00
1010	BOTTM	Elizabeth Lincoln	Vegie-spread	\$65760.00
Total Price : \$497,481.00 for 11 products				

*Table summary template*

The data grid hosts any view(s) inside a table summary for the entire row or for individual columns by loading a template.

*Displaying template for a row*

The template for the row can be loaded by using [DataTemplate](#) or [DataTemplateSelector](#) as follows,

*Using template*

The template for the table summary row can be loaded using the [GridSummaryRow.TitleTemplate](#) property with [GridSummaryRow.ShowSummaryInRow](#) enabled.

**XML**

```

<syncfusion:ChromelessWindow.Resources>
<local:TableSummaryRowConverter x:Key="summaryConverter" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow Title=" Total Price : {PriceAmount} for
{ProductCount} products " ShowSummaryInRow="True" Position="Bottom">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryRow.TitleTemplate>
<DataTemplate>
<TextBlock Text="{Binding Converter={StaticResource summaryConverter},
ConverterParameter= {x:Reference Name= dataGrid}}" Foreground="Blue"
Background="Yellow" FontSize="15"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryRow.TitleTemplate>
</syncfusion:GridTableSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>

```

**C#**

```

class TableSummaryRowConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
{
var data = value != null ? value as SummaryRecordEntry : null;
if (data != null)
{
SfDataGrid dataGrid = (SfDataGrid)parameter;
var unitPrice = SummaryCreator.GetSummaryDisplayText(data, "UnitPrice",
dataGrid.View);
var count = SummaryCreator.GetSummaryDisplayText(data, "ProductName",
dataGrid.View);
return "Total Price : " + unitPrice.ToString() + " for " + count.ToString()
+ " Products ";
}
return null;
}
public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
{
return null;
}
}

```

```
}
}
```

Order ID	Customer ID	Customer Name	Product Name	Price
1000.00	ALFKI	Maria Anders	Alice Mutton	5000.00
1001.00	ALFKI	Anders	Boston Crab Meat	30000.00
1001.00	ANTON	Antonio Moreno	Raclette Courdavault	300000.00
1001.00	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	20000.00
1001.00	BERGS	Christina Berglund	Gorgonzola Telino	40000.00
1002.00	BLAUS	Hanna Moos	Chartreuse verte	250000.00
1003.00	BLAUS	Frédérique Citeaux	Carnarvon Tigers	35900.00
1004.00	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	55099.00
1005.00	BOTTM	Elizabeth Lincoln	Vegie-spread	25000.00
Total Price : \$760,999.00 for 9 Products				

Using template selector

The template for the table summary row can also be loaded based on data object and the data-bound element using the [GridSummaryRow.TitleTemplateSelector](#) property with [GridSummaryRow.ShowSummaryInRow](#) enabled.

### XML

```
<Window.Resources>
<local:TemplateSelector x:Key="templateSelector" />
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow Title="Total Price : {PriceAmount} for
{ProductCount} products" ShowSummaryInRow="True" Position="Bottom"
TitleTemplateSelector="{StaticResource templateSelector}">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>
```

### C#

```
public class TemplateSelector : DataTemplateSelector
{
```

```

public override DataTemplate SelectTemplate(object item, DependencyObject
container)
{
    var summaryRecordEntry = item as SummaryRecordEntry;
    if (summaryRecordEntry.SummaryRow.ShowSummaryInRow)
    {
        return Application.Current.MainWindow.Resources["GridSummaryRowTemplate"] as
DataTemplate;
    }
    else
    {
        return Application.Current.MainWindow.Resources["GridSummaryColumnTemplate"]
as DataTemplate;
    }
    return null;
}
}

```

#### Displaying template for a column

The template for the summary column can be loaded by using `DataTemplate` or `DataTemplateSelector` as follows,

#### Using template

The template for the defined summary column can be loaded using the [GridSummaryColumn.Template](#) property.

#### XML

```

<syncfusion:ChromelessWindow.Resources>
<local:SummaryColumnConverter x:Key="summaryColumnConverter" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow Position="Bottom" ShowSummaryInRow="False">
<syncfusion:GridTableSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="'Total Price : {Sum:c} '"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" >
<syncfusion:GridSummaryColumn.Template>
<DataTemplate>
<TextBlock Text="{Binding Converter={ StaticResource
summaryColumnConverter}, ConverterParameter=UnitPrice }" Foreground="Red"
Background="LightBlue"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryColumn.Template>
</syncfusion:GridSummaryColumn>
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="'Total Product Count : {Count:d} '"
MappingName="ProductName"
SummaryType="CountAggregate">
<syncfusion:GridSummaryColumn.Template>
<DataTemplate>

```

```
<TextBlock Text="{Binding Converter={ StaticResource
summaryColumnConverter}, ConverterParameter=ProductName }" Foreground="Red"
Background="LightBlue"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryColumn.Template>
</syncfusion:GridSummaryColumn>
</syncfusion:GridTableSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>
```

## C#

```
class SummaryColumnConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var summaryRecordEntry = value as SummaryRecordEntry;
        if (summaryRecordEntry != null)
        {
            var columnName = parameter.ToString();
            var summaryRow = summaryRecordEntry.SummaryRow;
            var summaryCol = summaryRow.SummaryColumns.FirstOrDefault(s => s.MappingName
                == columnName);
            var summaryItems = summaryRecordEntry.SummaryValues;
            if (summaryItems != null && summaryCol != null)
            {
                var item = summaryItems.FirstOrDefault(s => s.Name == summaryCol.Name);
                if (item != null)
                {
                    if (columnName == "ProductName")
                        return string.Format("Total Product Count : {0:d}",
                            item.AggregateValues.Values.ToArray());
                    if (columnName == "UnitPrice")
                        return string.Format("Total Price : {0:c}",
                            item.AggregateValues.Values.ToArray());
                }
            }
            return "Value is wrong";
        }
        public object ConvertBack(object value, Type targetType, object parameter,
            CultureInfo culture)
        {
            return null;
        }
    }
}
```

Order ID	Customer ID	Customer Name	Product Name	Price
1000.00	ALFKI	Maria Anders	Alice Mutton	5000.00
1001.00	ALFKI	Anders	Boston Crab Meat	30000.00
1001.00	ANTON	Antonio Moreno	Raclette Courdavault	300000.00
1001.00	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	20000.00
1001.00	BERGS	Christina Berglund	Gorgonzola Telino	40000.00
1002.00	BLAUS	Hanna Moos	Chartreuse verte	250000.00
1003.00	BLAUS	Frédérique Citeaux	Carnarvon Tigers	35900.00
1004.00	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	55099.00
1005.00	BOTTM	Elizabeth Lincoln	Vegie-spread	25000.00
			Total Product Count : 9	Total Price : \$760,999.00

Using template selector

The template for the defined summary column can also be loaded based on data object and the data-bound element using the [GridSummaryColumn.TemplateSelector](#) property.

#### XML

```
<syncfusion:ChromelessWindow.Resources>
<local:TemplateSelector x:Key="templateSelector" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="'Total Price : {Sum:c}'"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" TemplateSelector="{StaticResource
templateSelector}">
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>
```

#### C#

```
public class TemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject
container)
    {
        if (item == null)
            return null;
        var summaryRecordEntry = item as SummaryRecordEntry;
        if (summaryRecordEntry.SummaryRow.ShowSummaryInRow)
        {
            return Application.Current.MainWindow.Resources["GridSummaryRowTemplate"] as
DataTemplate;
        }
    }
}
```

```

}
else
{
    return Application.Current.MainWindow.Resources["GridSummaryColumnTemplate"]
    as DataTemplate;
}
}
}

```

### Displaying column summary with title

SfDataGrid supports to show column summary and title summary at the same time. You can show column summary along with title by defining the [GridSummaryRow.Title](#) and [GridSummaryRow.TitleColumnCount](#) property along with defining summary columns. Showing column summary along with title can be only supported if [GridSummaryRow.ShowSummaryInRow](#) is disabled.

Refer [Defining summary for column](#) section to know more about how to define summary columns.

In the below code snippet, [GridSummaryRow.TitleColumnCount](#) is set as 2 and [GridSummaryRow.Title](#) is defined along with summary columns.

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"
    ShowGroupDropArea="True">
    <syncfusion:SfDataGrid.TableSummaryRows>
    <syncfusion:GridTableSummaryRow ShowSummaryInRow="False"
        TitleColumnCount="2" Position="Top"
        Title="Total Price : {PriceAmount} for {ProductCount} products">
    <syncfusion:GridSummaryRow.SummaryColumns>
    <syncfusion:GridSummaryColumn Name="CustomerName"
        Format=" {Count:d}"
        MappingName="CustomerName"
        SummaryType="CountAggregate" />
    <syncfusion:GridSummaryColumn Name="PriceAmount"
        Format=" {Sum:c}"
        MappingName="UnitPrice"
        SummaryType="DoubleAggregate" />
    <syncfusion:GridSummaryColumn Name="ProductCount"
        Format=" {Count:d}"
        MappingName="ProductName"
        SummaryType="CountAggregate" />
    </syncfusion:GridSummaryRow.SummaryColumns>
    </syncfusion:GridTableSummaryRow>
    <syncfusion:GridTableSummaryRow ShowSummaryInRow="False"
        TitleColumnCount="2" Position="Bottom"
        Title="Total Price : {PriceAmount} for {ProductCount} products">
    <syncfusion:GridSummaryRow.SummaryColumns>
    <syncfusion:GridSummaryColumn Name="CustomerName"
        Format=" {Count:d}"
        MappingName="CustomerName"
        SummaryType="CountAggregate" />
    <syncfusion:GridSummaryColumn Name="PriceAmount"
        Format=" {Sum:c}"
        MappingName="UnitPrice"

```



```

SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format=" {Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>

```

**C#**

```

this.dataGrid.TableSummaryRows.Add(new GridTableSummaryRow()
{
    ShowSummaryInRow = false,
    Position = TableSummaryRowPosition.Top,
    Title = "Total Price : {PriceAmount} for {ProductCount} products",
    TitleColumnCount = 2,
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name="CustomerName",
            Format="{Count:d}",
            MappingName="CustomerName",
            SummaryType=SummaryType.CountAggregate
        },
        new GridSummaryColumn()
        {
            Name = "PriceAmount",
            MappingName="UnitPrice",
            SummaryType= SummaryType.DoubleAggregate,
            Format="{Sum:c}"
        },
        new GridSummaryColumn()
        {
            Name="ProductCount",
            MappingName="ProductName",
            SummaryType=SummaryType.CountAggregate,
            Format="{Count:d}"
        }
    }
});
this.dataGrid.TableSummaryRows.Add(new GridTableSummaryRow()
{
    ShowSummaryInRow = false,
    Position = TableSummaryRowPosition.Bottom,
    Title = "Total Price : {PriceAmount} for {ProductCount} products",
    TitleColumnCount = 2,
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name="CustomerName",
            Format="{Count:d}",
            MappingName="CustomerName",

```

```

SummaryType=SummaryType.CountAggregate
},
new GridSummaryColumn()
{
    Name = "PriceAmount",
    MappingName="UnitPrice",
    SummaryType= SummaryType.DoubleAggregate,
    Format="{Sum:c}"
},
new GridSummaryColumn()
{
    Name="ProductCount",
    MappingName="ProductName",
    SummaryType=SummaryType.CountAggregate,
    Format="{Count:d}"
}
}
});

```

The following screenshot illustrates displaying summary columns with title at same time for `TableSummaryRow`.

Order ID	Customer ID	Customer Name	Product Name	Unit Price
Total Price : \$355.00 for 10 products		10	10	\$355.00
1000	ALFKI	Maria Anders	Alice Mutton	50.00
1001	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Crem	45.00
1002	ALFKI	Anders	Boston Crab Meat	30.00
1003	ANTON	Antonio Moreno	Raclette Courdavault	30.00
1004	AROUT	Thomas Hardy	Wimmers gute Semmelknö	20.00
1005	BERGS	Christina Berglund	Gorgonzola Telino	40.00
1006	BLAUS	Hanna Moos	Chartreuse verte	25.00
1007	BLAUS	Frédérique Citeaux	Carnarvon Tigers	35.00
1009	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	55.00
1010	BOTTM	Elizabeth Lincoln	Vegie-spread	25.00
Total Price : \$355.00 for 10 products		10	10	\$355.00



**SummaryColumns along with title in TableSummaryRow**

### Limitations

The following are the limitations of displaying column summary along with title at same time for `TableSummaryRow`:

- If `FrozenColumnCount` is defined lesser than `GridSummaryRow.TitleColumnCount`, the title summary will be spanned to `FrozenColumnCount` range, since spanned range and frozen range cannot be vary.
- Summary columns defined in the `GridSummaryRow.TitleColumnCount` range will not be shown.

Displaying template for column summary with title

The template for column summary can also be displayed with title template using `DataTemplate` or `DataTemplateSelector` as follows,

Using template

The template for column summary can also be displayed with title template using `GridSummaryRow.TitleColumnCount` property along with defining `GridSummaryRow.TitleTemplate` and `GridSummaryColumn.Template` properties.

### XML

```
<syncfusion:ChromelessWindow.Resources>
<local:TableSummaryRowConverter x:Key="summaryRowConverter"/>
<local:SummaryColumnConverter x:Key="summaryColumnConverter" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True"
AllowResizingColumns="True">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridSummaryRow Title="Total Price : {PriceAmount} for
{ProductCount} products" ShowSummaryInRow="False" TitleColumnCount="2">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" >
<syncfusion:GridSummaryColumn.Template>
<DataTemplate>
<TextBlock Text="{Binding Converter={ StaticResource
summaryColumnConverter}, ConverterParameter=UnitPrice }" Foreground="Red"
Background="LightBlue"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryColumn.Template>
</syncfusion:GridSummaryColumn>
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate">
<syncfusion:GridSummaryColumn.Template>
<DataTemplate>
<TextBlock Text="{Binding Converter={ StaticResource
summaryColumnConverter}, ConverterParameter=ProductName }" Foreground="Red"
Background="LightBlue"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryColumn.Template>
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryRow.TitleTemplate>
<DataTemplate>
<TextBlock Text="{Binding Converter={StaticResource summaryRowConverter},
ConverterParameter= {x:Reference Name= dataGrid}}" Foreground="Blue"
Background="Yellow" FontSize="15"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryRow.TitleTemplate>
```

```

</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>

```

Order ID	Customer ID	Customer Name	Product Name	Price
1000.00	ALFKI	Maria Anders	Alice Mutton	5000.00
1001.00	ALFKI	Anders	Boston Crab Meat	30000.00
1001.00	ANTON	Antonio Moreno	Raclette Courdavault	300000.00
1001.00	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	20000.00
1001.00	BERGS	Christina Berglund	Gorgonzola Telino	40000.00
1002.00	BLAUS	Hanna Moos	Chartreuse verte	250000.00
1003.00	BLAUS	Frédérique Citeaux	Carnarvon Tigers	35900.00
1004.00	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	55099.00
1005.00	BOTTM	Elizabeth Lincoln	Vegie-spread	25000.00
Total Price : \$760,999.00 for 9 Products			Total Product Count : 9	Total Price : \$760,999.00

Using template selector

The template for column summary can also be displayed with title template based on data object and the data-bound element using `GridSummaryRow.TitleColumnCount` property along with defining `GridSummaryRow.TitleTemplateSelector` and `GridSummaryColumn.TemplateSelector` properties.

### XML

```

<syncfusion:ChromelessWindow.Resources>
<local:TemplateSelector x:Key="templateSelector" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True"
AllowResizingColumns="True">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridSummaryRow Title="Total Price : {PriceAmount} for
{ProductCount} products" ShowSummaryInRow="False" TitleColumnCount="2"
TitleTemplateSelector = "{StaticResource templateSelector}" >
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" TemplateSelector = "{StaticResource
templateSelector}">
</syncfusion:GridSummaryColumn>
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate" TemplateSelector = "{StaticResource
templateSelector}">
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>

```

*Positioning TableSummaryRow*

You can position the table summary either at top or bottom of SfDataGrid by setting [GridTableSummaryRow.Position](#) property.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.TableSummaryRows>
    <syncfusion:GridTableSummaryRow Position="Top" ShowSummaryInRow="False">
    <syncfusion:GridSummaryRow.SummaryColumns>
    <syncfusion:GridSummaryColumn Name="PriceAmount"
    Format="{Sum:c}"
    MappingName="UnitPrice"
    SummaryType="DoubleAggregate" />
    </syncfusion:GridSummaryRow.SummaryColumns>
    </syncfusion:GridTableSummaryRow>
    <syncfusion:GridTableSummaryRow Title=" Total Product count: {ProductCount}"
    Position="Bottom"
    ShowSummaryInRow="True">
    <syncfusion:GridSummaryRow.SummaryColumns>
    <syncfusion:GridSummaryColumn Name="ProductCount"
    Format="{Count:d}"
    MappingName="ProductName"
    SummaryType="CountAggregate" />
    </syncfusion:GridSummaryRow.SummaryColumns>
    </syncfusion:GridTableSummaryRow>
    </syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>
```

**C#**

```
GridTableSummaryRow tablesummaryrow1 = new GridTableSummaryRow()
{
    ShowSummaryInRow = false,
    Position = TableSummaryRowPosition.Top,
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name="PriceAmount",
            Format="{Sum:c}",
            MappingName="UnitPrice",
            SummaryType=SummaryType.DoubleAggregate
        }
    }
};
GridTableSummaryRow tablesummaryrow2 = new GridTableSummaryRow()
{
    ShowSummaryInRow = true,
    Position = TableSummaryRowPosition.Bottom,
    Title="Total Product count: {ProductCount}",
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
```

```

{
    Name="ProductCount",
    Format="{Count:d}",
    MappingName="ProductName",
    SummaryType=SummaryType.CountAggregate
}
};
this.dataGrid.TableSummaryRows.Add(tablesummaryrow1);
this.dataGrid.TableSummaryRows.Add(tablesummaryrow2);

```

Table summary at top position

Order ID	Customer ID	Customer Name	Product Name	Unit Price
				\$463,138.00
1001	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53,345.00
1002	FKI	Anders	Boston Crab Meat	\$43,350.00
1003	ANTON	Antonio Moreno	Raclette Courdavault	\$13,433.00
1004	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	\$60,434.00
1005	BERGS	Christina Berglund	Gorgonzola Telino	\$36,343.00
1006	BLAUS	Hanna Moos	Chartreuse verte	\$5,345.00
1007	BLONP	Frédérique Citeaux	Fløtemysost	\$70,453.00
1008	BOLID	Martin Sommer	Carnarvon Tigers	\$84,340.00
1009	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	\$30,335.00
1010	BOTTM	Elizabeth Lincoln	Vegie-spread	\$65,760.00
Total Product count: 10				

Table summary at bottom position

### Group Summary

Group summary values calculated based on the records in the group and the summary information will be displayed at the bottom of each group. You can view the group summary row by expanding the corresponding group header. SfDataGrid allows you to add any number of group summary rows.

You can add the group summary rows in SfDataGrid by adding the [GridSummaryRow](#) to [SfDataGrid.GroupSummaryRows](#) collection.

### Defining summary for column

You can display summary information in the column by setting [GridSummaryRow.ShowSummaryInRow](#) to `false` and defining summary columns. To calculate summary based on column you have to specify the below properties,

1. [GridSummaryColumn.MappingName](#) – MappingName of the column (Property name of data object) that you want calculate summary.
2. [GridSummaryColumn.SummaryType](#) – SfDataGrid provides different built-in summary calculation functions for various types.
3. [GridSummaryColumn.Format](#) – Used to define format string for summary based on support function name's in specified SummaryType.

Refer [Formatting Summary](#) section to know more about how to format summary and [Aggregate Types](#) section to know about different Summary Type's.

In the below code snippet, summary is defined for `UnitPrice` and `ProductName` columns.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True">
<syncfusion:SfDataGrid.GroupSummaryRows>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="'Amount - {Sum:c} '"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="'Count - {Count:d} '"
MappingName="ProductName"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.GroupSummaryRows.Add(new GridSummaryRow()
{
    ShowSummaryInRow = false,
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name = "PriceAmount",
            MappingName="UnitPrice",
            SummaryType= SummaryType.Int32Aggregate,
            Format="Amount - {Sum:c}"
        },
        new GridSummaryColumn()
        {
            Name="ProductCount",
            MappingName="ProductName",
            SummaryType=SummaryType.CountAggregate,
            Format="Count - {Count:d}"
        },
    },
}
```

```
}
});
```

Order ID ^

Order ID ^	Customer ID	Customer Name	Product Name	Price
▼ Order ID : 1000 - 1 Items				
1000	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53345.00
			Count - 1	Amount - \$53,345.00
▼ Order ID : 1001 - 4 Items				
1001	ALFKI	Anders	Boston Crab Meat	\$43350.00
1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13433.00
1001	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	\$60434.00
1001	BERGS	Christina Berglund	Gorgonzola Telino	\$36343.00
			Count - 4	Amount - \$153,560.00
▲ Order ID : 1002 - 3 Items				
▲ Order ID : 1005 - 2 Items				

#### Displaying summary for Row

You can display summary information in row by setting [GridSummaryRow.ShowSummaryInRow](#) to `true` and defining summary columns. You have to define [GridSummaryRow.Title](#) based on [GridSummaryColumn.Name](#) property to format summary columns values in row.

Refer [Formatting Summary](#) section to know more about how to format summary.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"
    ShowGroupDropArea="True">
    <syncfusion:SfDataGrid.GroupSummaryRows>
    <syncfusion:GridSummaryRow Title="Total Price : {PriceAmount} for
    {ProductCount} products" ShowSummaryInRow="True">
    <syncfusion:GridSummaryRow.SummaryColumns>
    <syncfusion:GridSummaryColumn Name="PriceAmount"
    Format="{Sum:c}"
    MappingName="UnitPrice"
    SummaryType="DoubleAggregate" />
    <syncfusion:GridSummaryColumn Name="ProductCount"
    Format="{Count:d}"
    MappingName="ProductName"
    SummaryType="CountAggregate" />
    </syncfusion:GridSummaryRow.SummaryColumns>
    </syncfusion:GridSummaryRow>
    </syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>
```

#### C#



```

this.dataGrid.GroupSummaryRows.Add(new GridSummaryRow()
{
    Title = "Total Price : {PriceAmount} for {ProductCount} products",
    ShowSummaryInRow = true,
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name = "PriceAmount",
            MappingName="UnitPrice",
            SummaryType= SummaryType.Int32Aggregate,
            Format="{Sum:c}"
        },
        new GridSummaryColumn()
        {
            Name="ProductCount",
            MappingName="ProductName",
            SummaryType=SummaryType.CountAggregate,
            Format="{Count:d}"
        },
    }
});

```

Order ID

^

Order ID	Customer ID	Customer Name	Product Name	Price
▼ Order ID : 1000 - 1 Items				
1000	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Crem	\$53345.00
Total Price : \$53,345.00 for 1 products				
▼ Order ID : 1001 - 4 Items				
1001	ALFKI	Anders	Boston Crab Meat	\$43350.00
1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13433.00
1001	AROUT	Thomas Hardy	Wimmers gute Semmelknö	\$60434.00
1001	BERGS	Christina Berglund	Gorgonzola Telino	\$36343.00
Total Price : \$153,560.00 for 4 products				
▲ Order ID : 1002 - 3 Items				
▲ Order ID : 1005 - 2 Items				

#### Group summary template

The data grid hosts any view(s) inside a group summary for the entire row or for individual columns by loading a template.

#### Displaying template for a row

The template for the row can be loaded by using `DataTemplate` or `DataTemplateSelector` as follows,

#### Using template

The template for the group summary row can be loaded using the [GridSummaryRow.TitleTemplate](#) property with [GridSummaryRow.ShowSummaryInRow](#) enabled.

#### XML

```

<syncfusion:ChromelessWindow.Resources>
<local:GroupSummaryRowConverter x:Key="groupSummaryRowConverter"/>
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.GroupSummaryRows>
<syncfusion:GridSummaryRow Title="Total Price : {PriceAmount} for
{ProductCount} products" ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryRow.TitleTemplate>
<DataTemplate>
<TextBlock Text="{Binding Converter={StaticResource
groupSummaryRowConverter}, ConverterParameter= {x:Reference Name=
dataGrid}}" Foreground="Blue" Background="Yellow" FontSize="15"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryRow.TitleTemplate>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>

```

**C#**

```

class GroupSummaryRowConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var data = value != null ? value as SummaryRecordEntry : null;
        if (data != null)
        {
            SfDataGrid dataGrid = (SfDataGrid)parameter;
            var unitPrice = SummaryCreator.GetSummaryDisplayText(data, "UnitPrice",
                dataGrid.View);
            var count = SummaryCreator.GetSummaryDisplayText(data, "ProductName",
                dataGrid.View);
            return "Total Price : " + unitPrice.ToString() + " for " + count.ToString()
                + " Products ";
        }
        return null;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        return null;
    }
}

```

Order ID ^

Order ID ^	Customer ID	Customer Name	Product Name	Price
▼ Order ID : 1000 - 1 Items				
1000.00	ALFKI	Maria Anders	Alice Mutton	5000.00
Total Price : \$5,000.00 for 1 Products				
▼ Order ID : 1001 - 4 Items				
1001.00	ALFKI	Anders	Boston Crab Meat	30000.00
1001.00	ANTON	Antonio Moreno	Raclette Courdavault	300000.00
1001.00	AROUT	Thomas Hardy	Wimmers gute Semmelknöde	20000.00
1001.00	BERGS	Christina Berglund	Gorgonzola Telino	40000.00
Total Price : \$390,000.00 for 4 Products				
▲ Order ID : 1002 - 1 Items				
▲ Order ID : 1003 - 1 Items				
▲ Order ID : 1004 - 1 Items				
▲ Order ID : 1005 - 1 Items				

Using template selector

The template for the group summary row can also be loaded based on data object and the data-bound element using the [GridSummaryRow.TitleTemplateSelector](#) property with [GridSummaryRow.ShowSummaryInRow](#) enabled.

### XML

```
<syncfusion:ChromelessWindow.Resources>
<local:TemplateSelector x:Key="templateSelector" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.GroupSummaryRows>
<syncfusion:GridSummaryRow Title="Total Price : {PriceAmount} for
{ProductCount} products" ShowSummaryInRow="True"
TitleTemplateSelector="{StaticResource templateSelector}">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>
```

### C#

```

public class TemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
        var summaryRecordEntry = item as SummaryRecordEntry;
        if (summaryRecordEntry.SummaryRow.ShowSummaryInRow)
        {
            return Application.Current.MainWindow.Resources["GridSummaryRowTemplate"] as DataTemplate;
        }
        else
        {
            return Application.Current.MainWindow.Resources["GridSummaryColumnTemplate"] as DataTemplate;
        }
        return null;
    }
}

```

#### Displaying template for a column

The template for the summary column can be loaded by using `DataTemplate` or `DataTemplateSelector` as follows,

#### Using Template

The template for the defined summary column can be loaded using the [GridSummaryColumn.Template](#) property.

#### XML

```

<syncfusion:ChromelessWindow.Resources>
<local:SummaryColumnConverter x:Key="summaryColumnConverter" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.GroupSummaryRows>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="'Total Price : {Sum:c} '"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" >
<syncfusion:GridSummaryColumn.Template>
<DataTemplate>
<TextBlock Text="{Binding Converter={ StaticResource
summaryColumnConverter}, ConverterParameter=UnitPrice }" Foreground="Red"
Background="LightBlue"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryColumn.Template>
</syncfusion:GridSummaryColumn>
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="'Total Product Count : {Count:d} '"
MappingName="ProductName"
SummaryType="CountAggregate">
<syncfusion:GridSummaryColumn.Template>

```

```
<DataTemplate>
<TextBlock Text="{Binding Converter={ StaticResource
summaryColumnConverter}, ConverterParameter=ProductName }" Foreground="Red"
Background="LightBlue"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryColumn.Template>
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>
```

## C#

```
class SummaryColumnConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var summaryRecordEntry = value as SummaryRecordEntry;
        if (summaryRecordEntry != null)
        {
            var columnName = parameter.ToString();
            var summaryRow = summaryRecordEntry.SummaryRow;
            var summaryCol = summaryRow.SummaryColumns.FirstOrDefault(s => s.MappingName
                == columnName);
            var summaryItems = summaryRecordEntry.SummaryValues;
            if (summaryItems != null && summaryCol != null)
            {
                var item = summaryItems.FirstOrDefault(s => s.Name == summaryCol.Name);
                if (item != null)
                {
                    if (columnName == "ProductName")
                        return string.Format("Total Product Count : {0:d}",
                            item.AggregateValues.Values.ToArray());
                    if (columnName == "UnitPrice")
                        return string.Format("Total Price : {0:c}",
                            item.AggregateValues.Values.ToArray());
                }
            }
            return "Value is wrong";
        }
        public object ConvertBack(object value, Type targetType, object parameter,
            CultureInfo culture)
        {
            return null;
        }
    }
}
```

Order ID ^

Order ID ^	Customer ID	Customer Name	Product Name	Price
▼ Order ID : 1000 - 1 Items				
1000.00	ALFKI	Maria Anders	Alice Mutton	5000.00
			Total Product Count : 1	Total Price : \$5,000.00
▼ Order ID : 1001 - 4 Items				
1001.00	ALFKI	Anders	Boston Crab Meat	30000.00
1001.00	ANTON	Antonio Moreno	Raclette Courdavault	300000.00
1001.00	AROUT	Thomas Hardy	Wimmers gute Semmelknö	20000.00
1001.00	BERGS	Christina Berglund	Gorgonzola Telino	40000.00
			Total Product Count : 4	Total Price : \$390,000.00
▲ Order ID : 1002 - 1 Items				
▲ Order ID : 1003 - 1 Items				
▲ Order ID : 1004 - 1 Items				
▲ Order ID : 1005 - 1 Items				

Using template selector

The template for the defined summary column can also be loaded based on data object and the data-bound element using the [GridSummaryColumn.TemplateSelector](#) property.

#### XML

```
<syncfusion:ChromelessWindow.Resources>
<local:TemplateSelector x:Key="templateSelector" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.GroupSummaryRows>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="'Total Price : {Sum:c}'"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" TemplateSelector="{StaticResource
templateSelector}">
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>
```

#### C#

```
public class TemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject
    container)
    {
        if (item == null)
        return null;
    }
}
```

```

var summaryRecordEntry = item as SummaryRecordEntry;
if (summaryRecordEntry.SummaryRow.ShowSummaryInRow)
{
    return Application.Current.MainWindow.Resources["GridSummaryRowTemplate"] as
    DataTemplate;
}
else
{
    return Application.Current.MainWindow.Resources["GridSummaryColumnTemplate"]
    as DataTemplate;
}
}
}

```

#### Displaying column summary with title

SfDataGrid supports to show column summary and title summary at the same time. You can show column summary along with title by defining the [GridSummaryRow.Title](#) and [GridSummaryRow.TitleColumnCount](#) property along with defining summary columns. Showing column summary along with title can be only supported if [GridSummaryRow.ShowSummaryInRow](#) is disabled.

Refer [Defining summary for column](#) section to know more about how to define summary columns.

In the below code snippet, `GridSummaryRow.TitleColumnCount` is set as 2 and [GridSummaryRow.Title](#) is defined along with summary columns.

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"
    ShowGroupDropArea="True">
    <syncfusion:SfDataGrid.GroupSummaryRows>
    <syncfusion:GridSummaryRow ShowSummaryInRow="False" TitleColumnCount="2"
        Title="Total Price: {PriceAmount} for {ProductCount} Products" >
    <syncfusion:GridSummaryRow.SummaryColumns>
    <syncfusion:GridSummaryColumn Name="CustomerName"
        Format=" {Count:d}"
        MappingName="CustomerName"
        SummaryType="CountAggregate" />
    <syncfusion:GridSummaryColumn Name="PriceAmount"
        Format=" {Sum:c}"
        MappingName="UnitPrice"
        SummaryType="DoubleAggregate" />
    <syncfusion:GridSummaryColumn Name="ProductCount"
        Format=" {Count:d}"
        MappingName="ProductName"
        SummaryType="CountAggregate" />
    </syncfusion:GridSummaryRow.SummaryColumns>
    </syncfusion:GridSummaryRow>
    </syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>

```

#### C#

```

this.dataGrid.GroupSummaryRows.Add(new GridSummaryRow()

```

```

{
    ShowSummaryInRow = false,
    Title = "Total Price: {PriceAmount} for {ProductCount} Products",
    TitleColumnCount = 2,
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name="CustomerName",
            Format="{Count:d}",
            MappingName="CustomerName",
            SummaryType=SummaryType.CountAggregate
        },
        new GridSummaryColumn()
        {
            Name = "PriceAmount",
            MappingName="UnitPrice",
            SummaryType= SummaryType.DoubleAggregate,
            Format="{Sum:c}"
        },
        new GridSummaryColumn()
        {
            Name="ProductCount",
            MappingName="ProductName",
            SummaryType=SummaryType.CountAggregate,
            Format="{Count:d}"
        }
    }
});

```

The following screenshot illustrates displaying summary columns with title at same time for **GroupSummaryRow**.

Customer ID ^				
Order ID	Customer ID ^	Customer Name	Product Name	Unit Price
▼ Customer ID : ALFKI - 2 Items				
1000	ALFKI	Maria Anders	Alice Mutton	50.00
1002	ALFKI	Anders	Boston Crab Meat	30.00
Total Price: \$80.00 for 2 Products		2	2	\$80.00
▼ Customer ID : ANATR - 1 Items				
1001	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Crem	45.00
Total Price: \$45.00 for 1 Products		1	1	\$45.00
▼ Customer ID : ANTON - 1 Items				
1003	ANTON	Antonio Moreno	Raclette Courdavault	30.00
Total Price: \$30.00 for 1 Products		1	1	\$30.00

SummaryColumns along with title in GroupSummaryRow



### Limitations

The following are the limitations of displaying column summary along with title at same time for `GroupSummaryRow`:

- If `FrozenColumnCount` is defined lesser than `GridSummaryRow.TitleColumnCount`, the title summary will be spanned to `FrozenColumnCount` range, since spanned range and frozen range cannot be vary.
- Summary columns defined in the `GridSummaryRow.TitleColumnCount` range will not be shown.

### Displaying template for column summary with title

The template for column summary can also be displayed with title template using `DataTemplate` or `DataTemplateSelector` as follows,

#### Using template

The template for column summary can also be displayed with title template using `GridSummaryRow.TitleColumnCount` property along with defining `GridSummaryRow.TitleTemplate` and `GridSummaryColumn.Template` properties.

### XML

```
<syncfusion:ChromelessWindow.Resources>
<local:GroupSummaryRowConverter x:Key="groupSummaryRowConverter"/>
<local:SummaryColumnConverter x:Key="summaryColumnConverter" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True"
AllowResizingColumns="True">
<syncfusion:SfDataGrid.GroupSummaryRows>
<syncfusion:GridSummaryRow Title="Total Price : {PriceAmount} for
{ProductCount} products" ShowSummaryInRow="False" TitleColumnCount="2">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" >
<syncfusion:GridSummaryColumn.Template>
<DataTemplate>
<TextBlock Text="{Binding Converter={ StaticResource
summaryColumnConverter}, ConverterParameter=UnitPrice }" Foreground="Red"
Background="LightBlue"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryColumn.Template>
</syncfusion:GridSummaryColumn>
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate">
<syncfusion:GridSummaryColumn.Template>
<DataTemplate>
```

```

<TextBlock Text="{Binding Converter={ StaticResource
summaryColumnConverter}, ConverterParameter=ProductName }" Foreground="Red"
Background="LightBlue"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryColumn.Template>
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryRow.TitleTemplate>
<DataTemplate>
<TextBlock Text="{Binding Converter={StaticResource
groupSummaryRowConverter}, ConverterParameter= {x:Reference Name=
dataGrid}}" Foreground="Blue" Background="Yellow"
FontSize="15"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryRow.TitleTemplate>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>

```

Order ID	Customer ID	Customer Name	Product Name	Price
▼ Order ID : 1000 - 1 Items				
1000.00	ALFKI	Maria Anders	Alice Mutton	5000.00
Total Price : \$5,000.00 for 1 Products			Total Product Count : 1	Total Price : \$5,000.00
▼ Order ID : 1001 - 4 Items				
1001.00	ALFKI	Anders	Boston Crab Meat	30000.00
1001.00	ANTON	Antonio Moreno	Raclette Courdavault	300000.00
1001.00	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	20000.00
1001.00	BERGS	Christina Berglund	Gorgonzola Telino	40000.00
Total Price : \$390,000.00 for 4 Products			Total Product Count : 4	Total Price : \$390,000.00
▲ Order ID : 1002 - 1 Items				
▲ Order ID : 1003 - 1 Items				
▲ Order ID : 1004 - 1 Items				
▲ Order ID : 1005 - 1 Items				

Using template selector

The template for column summary can also be displayed with title template based on data object and the data-bound element using `GridSummaryRow.TitleColumnCount` property along with defining `GridSummaryRow.TitleTemplateSelector` and `GridSummaryColumn.TemplateSelector` properties.

### XML

```

<syncfusion:ChromelessWindow.Resources>
<local:TemplateSelector x:Key="templateSelector" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True"
AllowResizingColumns="True">
<syncfusion:SfDataGrid.GroupSummaryRows>

```

```

<syncfusion:GridSummaryRow Title="Total Price : {PriceAmount} for
{ProductCount} products" ShowSummaryInRow="False" TitleColumnCount="2"
TitleTemplateSelector = "{StaticResource templateSelector}" >
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="'Total Price : {Sum:c} '"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" TemplateSelector = "{StaticResource
templateSelector}">
</syncfusion:GridSummaryColumn>
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="'Total Product Count : {Count:d} '"
MappingName="ProductName"
SummaryType="CountAggregate" TemplateSelector = "{StaticResource
templateSelector}">
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>

```

### Caption Summaries

SfDataGrid provides built-in support for caption summaries. The caption summary value calculated based on the records in a group and the summary information will be displayed in the caption of group.

Below screen shot shows the built-in caption summary of Group.

ColumnName      Key      ItemsCount

Order ID ^	Customer ID	Customer Name	Product Name	Unit Price
▼ Order ID: 1001 - 3 Items				
1001	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53,345.00
1001	FKI	Anders	Boston Crab Meat	\$43,350.00
1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13,433.00
▲ Order ID : 1002 - 2 Items				
▲ Order ID : 1003 - 2 Items				
▲ Order ID : 1004 - 2 Items				
▲ Order ID : 1005 - 1 Items				

### Formatting built-in caption summary

By default, the summary value displayed in `CaptionSummaryRow` based on [SfDataGrid.GroupCaptionTextFormat](#) property.

The default group caption format is `{ColumnName}: {Key} - {ItemsCount} Items`.

- **ColumnName** - Displays the name of the column currently grouped.
- **Key** - Displays the key value of group.
- **ItemsCount** - Displays the number of items in group.

Diagram illustrating the SfDataGrid grouping interface. Arrows point from labels to the corresponding parts of the group caption:

- ColumnName** points to the "Order ID" column header.
- Key** points to the "1001" value in the group caption.
- ItemsCount** points to the "3 Items" text in the group caption.

	Order ID ^	Customer ID	Customer Name	Product Name	Unit Price
▼	Order ID: 1001	- 3 Items			
	1001	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53,345.00
	1001	FKI	Anders	Boston Crab Meat	\$43,350.00
	1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13,433.00
▲	Order ID : 1002 - 2 Items				
▲	Order ID : 1003 - 2 Items				
▲	Order ID : 1004 - 2 Items				
▲	Order ID : 1005 - 1 Items				

You can change group caption format to display column name and count alone by setting `GroupCaptionTextFormat` as below,

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowGrouping="True"
    AutoGenerateColumns="True"
    GroupCaptionTextFormat=" ColumnName: {Key} "
    ItemsSource="{Binding Orders}"
    ShowGroupDropArea="True"/>
```

Order ID ^					
	Order ID ^	Customer ID	Customer Name	Product Name	Unit Price
▼ ColumnName: 1001					
	1001	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53,345.00
	1001	FKI	Anders	Boston Crab Meat	\$43,350.00
	1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13,433.00
▼ ColumnName: 1002					
	1002	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	\$60,434.00
	1002	BERGS	Christina Berglund	Gorgonzola Telino	\$36,343.00
▲ ColumnName: 1003					
▲ ColumnName: 1004					
▲ ColumnName: 1005					

#### Defining summary for column

You can display summary information in the column by setting [GridSummaryRow.ShowSummaryInRow](#) to false and defining summary columns. To calculate summary based on column you have to specify the below properties,

1. [GridSummaryColumn.MappingName](#) – MappingName of the column (Property name of data object) that you want calculate summary.
2. [GridSummaryColumn.SummaryType](#) – SfDataGrid provides different built-in summary calculation functions for various types.
3. [GridSummaryColumn.Format](#) – Used to define format string for summary based on support function name's in specified SummaryType.

Refer [Formatting Summary](#) section to know more about how to format summary and [Aggregate Types](#) section to know about different Summary Type's.

In the below code snippet, summary is defined for `UnitPrice` and `ProductName` columns.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
```

```

SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>

```

**C#**

```

this.dataGrid.CaptionSummaryRow = new GridSummaryRow()
{
    ShowSummaryInRow = false,
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name = "PriceAmount",
            MappingName="UnitPrice",
            SummaryType= SummaryType.Int32Aggregate,
            Format="{Sum:c}"
        },
        new GridSummaryColumn()
        {
            Name="ProductCount",
            MappingName="ProductName",
            SummaryType=SummaryType.CountAggregate,
            Format="{Count:d}"
        },
    }
};

```

Order ID ^					
	Order ID ^	Customer ID	Customer Name	Product Name	Price
▼	1				\$53,345.00
	1000	ANATR	Ana Trujillo	NuNuCa Nuß-Nougat-Crem	\$53345.00
▼	4				\$153,560.00
	1001	ALFKI	Anders	Boston Crab Meat	\$43350.00
	1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13433.00
	1001	AROUT	Thomas Hardy	Wimmers gute Semmelknö	\$60434.00
	1001	BERGS	Christina Berglund	Gorgonzola Telino	\$36343.00
▲	3				\$160,138.00
▲	2				\$96,095.00

*Displaying summary for Row*

You can display summary information in row by setting [GridSummaryRow.ShowSummaryInRow](#) to **true** and defining summary columns. You have to define [GridSummaryRow.Title](#) based on [GridSummaryColumn.Name](#) property to format summary columns values in row.

Refer [Formatting Summary](#) section to know more about how to format summary.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow Title="Total Price : {PriceAmount} for
{ProductCount} Products" ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>
```

## C#

```
this.dataGrid.CaptionSummaryRow = new GridSummaryRow()
{
    ShowSummaryInRow = true,
    Title = "Total Price: {PriceAmount} for {ProductCount} Products",
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name = "PriceAmount",
            MappingName="UnitPrice",
            SummaryType= SummaryType.Int32Aggregate,
            Format="{Sum:c}"
        },
        new GridSummaryColumn()
        {
            Name="ProductCount",
            MappingName="ProductName",
            SummaryType=SummaryType.CountAggregate,
            Format="{Count:d}"
        },
    }
};
```

Order ID

^

Order ID	Customer ID	Customer Name	Product Name	Price
▼ Total Price : \$53,345.00 for 1 Products				
1000	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Crem	\$53345.00
▼ Total Price : \$153,560.00 for 4 Products				
1001	ALFKI	Anders	Boston Crab Meat	\$43350.00
1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13433.00
1001	AROUT	Thomas Hardy	Wimmers gute Semmelknöde	\$60434.00
1001	BERGS	Christina Berglund	Gorgonzola Telino	\$36343.00
▲ Total Price : \$160,138.00 for 3 Products				
▲ Total Price : \$96,095.00 for 2 Products				

### Caption summary template

The data grid hosts any view(s) inside a caption summary for the entire row or for individual columns by loading a template.

### Displaying template for a row

The template for the row can be loaded by using `DataTemplate` or `DataTemplateSelector` as follows,

### Using template

The template for the caption summary row can be loaded using the `GridSummaryRow.TitleTemplate` property with `GridSummaryRow.ShowSummaryInRow` enabled.

### XML

```
<syncfusion:ChromelessWindow.Resources>
<local:CaptionSummaryRowConverter x:Key="captionSummaryRowConverter"/>
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow Title="Total Price : {PriceAmount} for
{ProductCount} Products" ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryRow.TitleTemplate>
<DataTemplate>
<TextBlock Text="{Binding Converter={StaticResource
captionSummaryRowConverter}, ConverterParameter= {x:Reference Name=
dataGrid}}" Foreground="Blue" Background="Yellow" FontSize="15"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryRow.TitleTemplate>
</syncfusion:GridSummaryRow>
```



```
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>
```

**C#**

```
class CaptionSummaryRowConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var data = value != null ? value as Group : null;
        if (data != null)
        {
            SfDataGrid dataGrid = (SfDataGrid)parameter;
            var unitPrice = SummaryCreator.GetSummaryDisplayText(data.SummaryDetails,
                "UnitPrice", dataGrid.View);
            var count = SummaryCreator.GetSummaryDisplayText(data.SummaryDetails,
                "ProductName", dataGrid.View);
            return "Total Price : " + unitPrice.ToString() + " for " + count.ToString()
                + " Products ";
        }
        return null;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        return null;
    }
}
```

Order ID ^

Order ID ^	Customer ID	Customer Name	Product Name	Price
▼ Total Price : \$5,000.00 for 1 Products				
1000.00	ALFKI	Maria Anders	Alice Mutton	5000.00
▼ Total Price : \$390,000.00 for 4 Products				
1001.00	ALFKI	Anders	Boston Crab Meat	30000.00
1001.00	ANTON	Antonio Moreno	Raclette Courdavault	300000.00
1001.00	AROUT	Thomas Hardy	Wimmers gute Semmelknö	20000.00
1001.00	BERGS	Christina Berglund	Gorgonzola Telino	40000.00
▲ Total Price : \$250,000.00 for 1 Products				
▲ Total Price : \$35,900.00 for 1 Products				
▲ Total Price : \$55,099.00 for 1 Products				
▲ Total Price : \$25,000.00 for 1 Products				

Using template selector

The template for the caption summary row can also be loaded based on data object and the data-bound element using the [GridSummaryRow.TitleTemplateSelector](#) property with [GridSummaryRow.ShowSummaryInRow](#) enabled.

**XML**

```

<syncfusion:ChromelessWindow.Resources>
<local:TemplateSelector x:Key="templateSelector" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow Title="Total Price : {PriceAmount} for
{ProductCount} Products" ShowSummaryInRow="True"
TitleTemplateSelector="{StaticResource templateSelector}">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>

```

## C#

```

public class TemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject
container)
    {
        var summaryRecordEntry = item as SummaryRecordEntry;
        if (summaryRecordEntry.SummaryRow.ShowSummaryInRow)
        {
            return Application.Current.MainWindow.Resources["GridSummaryRowTemplate"] as
DataTemplate;
        }
        else
        {
            return Application.Current.MainWindow.Resources["GridSummaryColumnTemplate"]
as DataTemplate;
        }
        return null;
    }
}

```

### Displaying template for a column

The template for the summary column can be loaded by using `DataTemplate` or `DataTemplateSelector` as follows,

#### Using template

The template for the defined summary column can be loaded using the [GridSummaryColumn.Template](#) property.

## XML

```

<syncfusion:ChromelessWindow.Resources>
<local:CaptionSummaryColumnConverter x:Key="captionSummaryColumnConverter"
/>
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="'Total Price : {Sum:c} '"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" >
<syncfusion:GridSummaryColumn.Template>
<DataTemplate>
<TextBlock Text="{Binding Converter={ StaticResource
captionSummaryColumnConverter}, ConverterParameter=UnitPrice }"
Foreground="Red" Background="LightBlue"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryColumn.Template>
</syncfusion:GridSummaryColumn>
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="'Total Product Count : {Count:d} '"
MappingName="ProductName"
SummaryType="CountAggregate">
<syncfusion:GridSummaryColumn.Template>
<DataTemplate>
<TextBlock Text="{Binding Converter={ StaticResource
captionSummaryColumnConverter}, ConverterParameter=ProductName }"
Foreground="Red" Background="LightBlue"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryColumn.Template>
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>

```

## C#

```

class CaptionSummaryColumnConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
{
var summaryRecordEntry = value as Group;
if (summaryRecordEntry != null)
{
var columnName = parameter.ToString();
var summaryRow = summaryRecordEntry.SummaryDetails.SummaryRow;
var summaryCol = summaryRow.SummaryColumns.FirstOrDefault(s => s.MappingName
== columnName);
var summaryItems = summaryRecordEntry.SummaryDetails.SummaryValues;
if (summaryItems != null && summaryCol != null)
{

```

```

var item = summaryItems.FirstOrDefault(s => s.Name == summaryCol.Name);
if (item != null)
{
    if (columnName == "ProductName")
        return string.Format("Total Product Count : {0:d}",
            item.AggregateValues.Values.ToArray());
    if (columnName == "UnitPrice")
        return string.Format("Total Price : {0:c}",
            item.AggregateValues.Values.ToArray());
    }
}
return "Value is wrong";
}
public object ConvertBack(object value, Type targetType, object parameter,
    CultureInfo culture)
{
    return null;
}
}

```

Order ID ^					
Order ID ^	Customer ID	Customer Name	Product Name	Price ^	
Total Product Count : 1				Total Price : \$5,000.00	
1000.00	ALFKI	Maria Anders	Alice Mutton	5000.00	
Total Product Count : 4				Total Price : \$390,000.00	
1001.00	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	20000.00	
1001.00	ALFKI	Anders	Boston Crab Meat	30000.00	
1001.00	BERGS	Christina Berglund	Gorgonzola Telino	40000.00	
1001.00	ANTON	Antonio Moreno	Raclette Courdavault	300000.00	
Total Product Count : 1				Total Price : \$250,000.00	
Total Product Count : 1				Total Price : \$35,900.00	
Total Product Count : 1				Total Price : \$55,099.00	
Total Product Count : 1				Total Price : \$25,000.00	

Using template selector

The template for the defined summary column can also be loaded based on data object and the data-bound element using the [GridSummaryColumn.TemplateSelector](#) property.

### XML

```

<syncfusion:ChromelessWindow.Resources>
<local:TemplateSelector x:Key="templateSelector" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"

```

```

Format="'Total Price : {Sum:c}'"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" TemplateSelector="{StaticResource
templateSelector}">
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>

```

**C#**

```

public class TemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject
container)
    {
        if (item == null)
            return null;
        var summaryRecordEntry = item as SummaryRecordEntry;
        if (summaryRecordEntry.SummaryRow.ShowSummaryInRow)
        {
            return Application.Current.MainWindow.Resources["GridSummaryRowTemplate"] as
DataTemplate;
        }
        else
        {
            return Application.Current.MainWindow.Resources["GridSummaryColumnTemplate"]
as DataTemplate;
        }
    }
}

```

*Displaying column summary with title*

SfDataGrid supports to show column summary and title summary at the same time. You can show column summary along with title by defining the [GridSummaryRow.Title](#) and [GridSummaryRow.TitleColumnCount](#) property along with defining summary columns. Showing column summary along with title can be only supported if [GridSummaryRow.ShowSummaryInRow](#) is disabled.

Refer [Defining summary for column](#) section to know more about how to define summary columns.

In the below code snippet, `GridSummaryRow.TitleColumnCount` is set as 2 and [GridSummaryRow.Title](#) is defined along with summary columns.

**XML**

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow ShowSummaryInRow="False" TitleColumnCount="2"
Title=" {ColumnName} : {Key} - {ItemsCount} Items ">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="CustomerName"

```

```

Format="{Count:d}"
MappingName="CustomerName"
SummaryType="CountAggregate" />
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>

```

**C#**

```

this.dataGrid.CaptionSummaryRow = new GridSummaryRow()
{
    ShowSummaryInRow = false,
    Title = "{ColumnName} : {Key} - {ItemsCount} Items",
    TitleColumnCount = 2,
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name="CustomerName",
            Format="{Count:d}",
            MappingName="CustomerName",
            SummaryType=SummaryType.CountAggregate
        },
        new GridSummaryColumn()
        {
            Name = "PriceAmount",
            MappingName="UnitPrice",
            SummaryType= SummaryType.DoubleAggregate,
            Format="{Sum:c}"
        },
        new GridSummaryColumn()
        {
            Name="ProductCount",
            MappingName="ProductName",
            SummaryType=SummaryType.CountAggregate,
            Format="{Count:d}"
        }
    }
};

```

The following screenshot illustrates displaying summary columns with title at same time for `CaptionSummaryRow`.

Customer ID ^					
	Order ID ▼	Customer ID ^ ▼	Customer Name ▼	Product Name ▼	Unit Price ▼
▼	Customer ID : ALFKI - 2 Items		2	2	\$80.00
	1000	ALFKI	Maria Anders	Alice Mutton	50.00
	1002	ALFKI	Anders	Boston Crab Meat	30.00
▼	Customer ID : ANATR - 1 Items		1	1	\$45.00
	1001	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Crem	45.00
▼	Customer ID : ANTON - 1 Items		1	1	\$30.00
	1003	ANTON	Antonio Moreno	Raclette Courdavault	30.00
▲	Customer ID : AROUT - 1 Items		1	1	\$20.00
▲	Customer ID : BERGS - 1 Items		1	1	\$40.00
▲	Customer ID : BLAUS - 2 Items		2	2	\$60.00

SummaryColumns along with title in CaptionSummaryRow

#### Limitations

The following are the limitations of displaying column summary along with title at same time for CaptionSummaryRow:

- If [FrozenColumnCount](#) is defined lesser than `GridSummaryRow.TitleColumnCount`, the title summary will be spanned to [FrozenColumnCount](#) range, since spanned range and frozen range cannot be vary.
- Summary columns defined in the `GridSummaryRow.TitleColumnCount` range will not be shown.

#### Displaying template for column summary with title

The template for column summary can also be displayed with title template using `DataTemplate` or `DataTemplateSelector` as follows,

#### Using template

The template for column summary can also be displayed with title template using `GridSummaryRow.TitleColumnCount` property along with defining `GridSummaryRow.TitleTemplate` and `GridSummaryColumn.Template` properties.

#### XML

```
<syncfusion:ChromelessWindow.Resources>
  <local:CaptionSummaryRowConverter x:Key="captionSummaryRowConverter" />
  <local:CaptionSummaryColumnConverter x:Key="captionSummaryColumnConverter" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
  AutoGenerateColumns="True"
  ItemsSource="{Binding Orders}"
  ShowGroupDropArea="True"
  AllowResizingColumns="True">
  <syncfusion:SfDataGrid.CaptionSummaryRow>
```

```

<syncfusion:GridSummaryRow Title="Total Price : {PriceAmount} for
{ProductCount} products" ShowSummaryInRow="False" TitleColumnCount="2">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" >
<syncfusion:GridSummaryColumn.Template>
<DataTemplate>
<TextBlock Text="{Binding Converter={ StaticResource
captionsummaryColumnConverter}, ConverterParameter=UnitPrice }"
Foreground="Red" Background="LightBlue"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryColumn.Template>
</syncfusion:GridSummaryColumn>
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate">
<syncfusion:GridSummaryColumn.Template>
<DataTemplate>
<TextBlock Text="{Binding Converter={ StaticResource
captionsummaryColumnConverter}, ConverterParameter=UnitPrice }"
Foreground="Red" Background="LightBlue"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryColumn.Template>
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryRow.TitleTemplate>
<DataTemplate>
<TextBlock Text="{Binding Converter={StaticResource
captionSummaryRowConverter}, ConverterParameter= {x:Reference Name=
dataGrid}}" Foreground="Blue" Background="Yellow" FontSize="15"></TextBlock>
</DataTemplate>
</syncfusion:GridSummaryRow.TitleTemplate>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>

```



Order ID ^					
	Order ID ^	Customer ID	Customer Name	Product Name	Price
▼	Total Price : \$5,000.00 for 1 Products			Total Price : \$5,000.00	Total Price : \$5,000.00
	1000.00	ALFKI	Maria Anders	Alice Mutton	5000.00
▼	Total Price : \$390,000.00 for 4 Products			Total Price : \$390,000.00	Total Price : \$390,000.00
	1001.00	ALFKI	Anders	Boston Crab Meat	30000.00
	1001.00	ANTON	Antonio Moreno	Raclette Courdavault	300000.00
	1001.00	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	20000.00
	1001.00	BERGS	Christina Berglund	Gorgonzola Telino	40000.00
▲	Total Price : \$250,000.00 for 1 Products			Total Price : \$250,000.00	Total Price : \$250,000.00
▲	Total Price : \$35,900.00 for 1 Products			Total Price : \$35,900.00	Total Price : \$35,900.00
▲	Total Price : \$55,099.00 for 1 Products			Total Price : \$55,099.00	Total Price : \$55,099.00
▲	Total Price : \$25,000.00 for 1 Products			Total Price : \$25,000.00	Total Price : \$25,000.00

Using template selector

The template for column summary can also be displayed with title template based on data object and the data-bound element using `GridSummaryRow.TitleColumnCount` property along with defining `GridSummaryRow.TitleTemplateSelector` and `GridSummaryColumn.TemplateSelector` properties.

#### XML

```
<syncfusion:ChromelessWindow.Resources>
<local:TemplateSelector x:Key="templateSelector" />
</syncfusion:ChromelessWindow.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True"
AllowResizingColumns="True">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow Title="Total Price : {PriceAmount} for
{ProductCount} products" ShowSummaryInRow="False" TitleColumnCount="2"
TitleTemplateSelector = "{StaticResource templateSelector}" >
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" TemplateSelector = "{StaticResource
templateSelector}">
</syncfusion:GridSummaryColumn>
<syncfusion:GridSummaryColumn Name="ProductCount"
Format="{Count:d}"
MappingName="ProductName"
SummaryType="CountAggregate" TemplateSelector = "{StaticResource
templateSelector}">
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>
```

### Formatting summary

In the below sections formatting is explained using TableSummary. In the same way, you can format group and caption summaries also.

### Defining Summary Function

In the below code snippet `Format` property is defined to display sum of `UnitPrice` by specifying the function name inside curly braces.

**Note:** `DoubleAggregate` is used as `SummaryType` which has Count, Max, Min, Average and Sum functions.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>
```

Order ID	Customer ID	Customer Name	Product Name	Price
1000	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53345.00
1001	ALFKI	Anders	Boston Crab Meat	\$43350.00
1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13433.00
1001	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	\$60434.00
1001	BERGS	Christina Berglund	Gorgonzola Telino	\$36343.00
1002	BLAUS	Hanna Moos	Chartreuse verte	\$5345.00
1002	BLONP	Frédérique Citeaux	Fløtemysost	\$70453.00
1002	BOLID	Martin Sommer	Carnarvon Tigers	\$84340.00
1005	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	\$30335.00
1005	BOTTM	Elizabeth Lincoln	Vegie-spread	\$65760.00
				463138

### Formatting Summary Value

You can format the summary value by setting the appropriate format after the aggregate function followed by colon(:) in `GridSummaryColumn.Format` property.

In the below code snippet `UnitPrice` column summary is formatted using `c` format specifier. Refer [here](#) to know about how to set different format.

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>

```

Order ID	Customer ID	Customer Name	Product Name	Price
1000	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53345.00
1001	ALFKI	Anders	Boston Crab Meat	\$43350.00
1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13433.00
1001	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	\$60434.00
1001	BERGS	Christina Berglund	Gorgonzola Telino	\$36343.00
1002	BLAUS	Hanna Moos	Chartreuse verte	\$5345.00
1002	BLONP	Frédérique Citeaux	Fløtemysost	\$70453.00
1002	BOLID	Martin Sommer	Carnarvon Tigers	\$84340.00
1005	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	\$30335.00
1005	BOTTM	Elizabeth Lincoln	Vegie-spread	\$65760.00
				\$463,138.00

#### Displaying additional Content in Summary

You can append additional content with summary value using [GridSummaryColumn.Format](#) property.

In the below code snippet **Total UnitPrice:** text is appended before summary value.

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="PriceAmount"
Format="'Total UnitPrice : {Sum:c} '"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>

```

Order ID	Customer ID	Customer Name	Product Name	Price
1000	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53345.00
1001	ALFKI	Anders	Boston Crab Meat	\$43350.00
1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13433.00
1001	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	\$60434.00
1001	BERGS	Christina Berglund	Gorgonzola Telino	\$36343.00
1002	BLAUS	Hanna Moos	Chartreuse verte	\$5345.00
1002	BLONP	Frédérique Citeaux	Fløtemysost	\$70453.00
1002	BOLID	Martin Sommer	Carnarvon Tigers	\$84340.00
1005	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	\$30335.00
1005	BOTTM	Elizabeth Lincoln	Vegie-spread	\$65760.00
Total UnitPrice :				\$463,138.00

#### Formatting Summary for Row using Title Property

You can format the summary value for row using [GridSummaryRow.Title](#) when ShowSummaryInRow set to true.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.TableSummaryRows>
    <syncfusion:GridTableSummaryRow Title=" Total Price : {PriceAmount} "
        ShowSummaryInRow="True">
        <syncfusion:GridSummaryRow.SummaryColumns>
        <syncfusion:GridSummaryColumn Name="PriceAmount"
            Format="' {Sum:c} ' "
            MappingName="UnitPrice"
            SummaryType="DoubleAggregate" />
        </syncfusion:GridSummaryRow.SummaryColumns>
        </syncfusion:GridTableSummaryRow>
    </syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>
```

Order ID	Customer ID	Customer Name	Product Name	Price
1000	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53345.00
1001	ALFKI	Anders	Boston Crab Meat	\$43350.00
1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13433.00
1001	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	\$60434.00
1001	BERGS	Christina Berglund	Gorgonzola Telino	\$36343.00
1002	BLAUS	Hanna Moos	Chartreuse verte	\$5345.00
1002	BLONP	Frédérique Citeaux	Fløtemysost	\$70453.00
1002	BOLID	Martin Sommer	Carnarvon Tigers	\$84340.00
1005	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	\$30335.00
1005	BOTTM	Elizabeth Lincoln	Vegie-spread	\$65760.00
Total Price : \$463,138.00				

### Aggregate Types

You can specify the different summary aggregate types by using [GridSummaryColumn.SummaryType](#) property and use the built-in function in [GridSummaryColumn.Format](#).

The following are the list of predefined aggregate types and its built-in functions.

Aggregate Type	Built-in function
CountAggregate	Count
Int32Aggregate	Count, Max, Min, Average and Sum
DoubleAggregate	Count, Max, Min, Average and Sum
Custom	Used for custom summaries

### Calculate summary for selected rows

SfDataGrid calculates the summaries for all records by default. You can calculate the summaries for selected records by using the [SfDataGrid.SummaryCalculationUnit](#) or [GridSummaryRow.CalculationUnit](#) property.

This is applicable for all type of summary rows such as table, caption and group summary.

In the below code snippet, the summaries for selected records are calculated for the top positioned `TableSummaryRow` and the summaries for all records are calculated for the bottom positioned `TableSummaryRow`.

### XML

```
<syncfusion:SfDataGrid x:Name="sfDataGrid"
ItemsSource="{Binding Path=Orders}"
SelectionMode="Multiple"
AutoGenerateColumns="True">
  <syncfusion:SfDataGrid.TableSummaryRows>
    <syncfusion:GridTableSummaryRow ShowSummaryInRow="True" Title="Total Price
for all records: {UnitPrice}">
      <syncfusion:GridSummaryRow.SummaryColumns>
        <syncfusion:GridSummaryColumn Name="UnitPrice"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
      </syncfusion:GridSummaryRow.SummaryColumns>
    </syncfusion:GridTableSummaryRow>
    <syncfusion:GridTableSummaryRow ShowSummaryInRow="True" Title="Total price
for selected records: {UnitPrice}" CalculationUnit="SelectedRows"
Position="Top">
      <syncfusion:GridSummaryRow.SummaryColumns>
        <syncfusion:GridSummaryColumn Name="UnitPrice"
Format="{Sum:c}"
MappingName="UnitPrice"
SummaryType="DoubleAggregate" />
      </syncfusion:GridSummaryRow.SummaryColumns>
    </syncfusion:GridTableSummaryRow>
  </syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>
```

**C#**

```

GridTableSummaryRow tableSummaryRow1 = new GridTableSummaryRow()
{
    ShowSummaryInRow = true,
    Position = TableSummaryRowPosition.Top,
    CalculationUnit = SummaryCalculationUnit.SelectedRows,
    Title = "Total price for selected records: {PriceAmount}",
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name="PriceAmount",
            Format="{Sum:c}",
            MappingName="UnitPrice",
            SummaryType=SummaryType.DoubleAggregate
        }
    }
};

GridTableSummaryRow tableSummaryRow2 = new GridTableSummaryRow()
{
    ShowSummaryInRow = true,
    Position = TableSummaryRowPosition.Bottom,
    Title = "Total price for all records: {UnitPrice}",
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name="UnitPrice",
            Format="{Sum:c}",
            MappingName="UnitPrice",
            SummaryType=SummaryType.DoubleAggregate
        }
    }
};

this.sfDataGrid.TableSummaryRows.Add(tableSummaryRow1);
this.sfDataGrid.TableSummaryRows.Add(tableSummaryRow2);

```

Order ID	Customer ID	Customer Name	Country	Unit Price
Total price for selected records: ₹ 1,60,870.00				
1001	ALFKI	Maria Anders	Germany	25000
1002	ANATR	Ana Trujillo	Mexico	36000
1003	ANTON	Antonio Moreno	Mexico	40040
1004	AROUT	Thomas Hardy	UK	10700
1005	BERGS	Christina Berglund	Sweden	20300
1006	BLAUS	Hanna Moos	Germany	50700
1007	BLONP	Frederique Citeaux	France	80100
1008	BOLID	Martin Sommer	Spain	35000
1009	BONAP	Laurence Lebihan	France	20030
1010	BOTTM	Elizabeth Lincoln	Canada	54000
1001	ALFKI	Maria Anders	Germany	22000
1001	ANATR	Ana Trujillo	Mexico	30050
1001	ANTON	Antonio Moreno	Mexico	40800
1002	AROUT	Thomas Hardy	UK	10020
1003	BERGS	Christina Berglund	Sweden	20050
1003	BLAUS	Hanna Moos	Germany	50080
1004	BLONP	Frederique Citeaux	France	89000
Total Price for all records: ₹ 7,34,430.00				

---

**Note:** The `GridSummaryRow.CalculationUnit` takes higher priority than the `SfDataGrid.SummaryCalculationUnit`.

---

#### Limitation

`SummaryCalculationUnit.SelectedRows` or `SummaryCalculationUnit.Mixed` will not be considered for cell selection.

#### Custom summaries

SfDataGrid allows you to implement your own aggregate functions, when the built-in aggregate functions don't meet your requirement.

You can calculate the summary values based on custom logic using [GridSummaryColumn.CustomAggregate](#) property.

#### Implementing custom aggregate

1. Create custom aggregate class by deriving from [ISummaryAggregate](#) interface.
2. In the `CalculateAggregateFunc` method, you have to calculate the summary and assign it to the property.

In the below code snippet, the Standard Deviation is calculated for quantity of products.

#### C#

```
public class CustomAggregate:ISummaryAggregate
{
    public CustomAggregate()
    {
    }
    public double StdDev { get; set; }
    public Action<System.Collections.IEnumerable, string,
    System.ComponentModel.PropertyDescriptor> CalculateAggregateFunc()
    {
        return (items, property, pd) =>
        {
            var enumerableItems = items as IEnumerable<OrderInfo>;
            if (pd.Name == "StdDev")
            {
                this.StdDev = enumerableItems.StdDev<OrderInfo>(q => q.Quantity);
            }
        };
    }
}

public static class LinqExtensions
{
    public static double StdDev<T>(this IEnumerable<T> values, Func<T, double?>
    selector)
    {
        double ret = 0;
        var count = values.Count();
        if (count > 0)
        {
            double? avg = values.Average(selector);
            double sum = values.Select(selector).Sum(d =>
```

```

if (d.HasValue)
{
    return Math.Pow(d.Value - avg.Value, 2);
}
return 0.0;
});
ret = Math.Sqrt((sum) / (count - 1));
}
return ret;
}
}

```

3. Assign the custom aggregate to `GridSummaryColumn.CustomAggregate` property and set the `SummaryType` as `Custom`. `GridSummaryColumn.Format` property is defined based on property name in custom aggregate `StdDev`.

#### XML

```

<Window.Resources>
<local:CustomAggregate x:Key="customAggregate" />
</Window.Resources>
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow Name="Total Products"
Title="Standard Deviation: {Discount}"
Position="Top"
ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="Discount"
CustomAggregate="{StaticResource customAggregate}"
Format="{StdDev:##.##}"
MappingName="Discount"
SummaryType="Custom" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>

```

#### C#

```

this.dataGrid.TableSummaryRows.Add(new GridTableSummaryRow()
{
    ShowSummaryInRow = true,
    Position=TableSummaryRowPosition.Top,
    Title = "Standard Deviation: {Discount}",
    SummaryColumns = new ObservableCollection<ISummaryColumn>()
    {
        new GridSummaryColumn()
        {
            Name = "Discount",
            MappingName="Discount",
            CustomAggregate=new CustomAggregate(),
            SummaryType=SummaryType.Custom,
            Format="{StdDev}"
        },
    }
});

```



OrderID	CustomerID	CustomerName	ProductName	UnitPrice	Quantity	Discount
Standard Deviation: 50.02						
1000	ALFKI	Maria Anders	Alice Mutton	50.00	50.00	5.00
1001	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	45.00	100.00	10.00
1002	ALFKI	Anders	Boston Crab Meat	30.00	150.00	8.00
1003	ANTON	Antonio Moreno	Raclette Courdavault	30.00	120.00	20.00
1004	AROUT	Thomas Hardy	Wimmers gute Semmelknödel	20.00	130.00	15.00
1005	BERGS	Christina Berglund	Gorgonzola Telino	40.00	150.00	20.00
1006	BLAUS	Hanna Moos	Chartreuse verte	25.00	70.00	12.00
1007	BLAUS	Frédérique Citeaux	Carnarvon Tigers	35.00	8.00	25.00
1009	BONAP	Laurence Lebihan	Thüringer Rostbratwurst	55.00	50.00	30.00
1010	BOTTM	Elizabeth Lincoln	Vegie-spread	25.00	40.00	20.00

You can download the sample demo [here](#) .

### Overriding Summary Renderer

Each summary cell is associated with its own renderer and you can customize it by overriding the available virtual methods in the renderer.

There will be specific key for each summary row. You can remove and add the key from [SfDataGrid.CellRenderers](#) collection, when the renderer is need to be customized.

Types of summary	Renderer	Key
Table summary	<a href="#">GridTableSummaryCellRenderer</a>	TableSummary
Caption summary	<a href="#">GridCaptionSummaryCellRenderer</a>	CaptionSummary
Group summary	<a href="#">GridSummaryCellRenderer</a>	GroupSummary

### Formatting Table Summary

You can apply number format for numeric values displayed on `GridTableSummaryRow` by overriding the `OnUpdateEditBinding` method in [GridTableSummaryCellRenderer](#) class.

### C#

```

this.dataGrid.CellRenderers.Remove("TableSummary");
this.dataGrid.CellRenderers.Add("TableSummary", new
GridTableSummaryCellRendererExt());
public class GridTableSummaryCellRendererExt : GridTableSummaryCellRenderer
{
    public override void
    OnUpdateEditBinding(Syncfusion.UI.Xaml.Grid.DataColumnBase dataColumn,
    Syncfusion.UI.Xaml.Grid.GridTableSummaryCell element, object dataContext)
    {
        //Check whether the dataContext is SummaryRecordEntry
        var record = dataContext as SummaryRecordEntry;
        if (!(dataContext is SummaryRecordEntry))
            return;
        //Process each SummaryColumn and get the display text of corresponding
        summary
        foreach (ISummaryColumn summaryColumn in record.SummaryRow.SummaryColumns)

```

```

{
    if (!summaryColumn.MappingName.Contains(dataColumn.GridColumn.MappingName))
        continue;
    string summaryText = string.Empty;
    if (record.SummaryRow.ShowSummaryInRow)
        summaryText = SummaryCreator.GetSummaryDisplayTextForRow(record,
            this.DataGrid.View);
    else
        summaryText = SummaryCreator.GetSummaryDisplayText(record,
            dataColumn.GridColumn.MappingName, this.DataGrid.View);
    if (!string.IsNullOrEmpty(summaryText))
    {
        //Create new number format and apply it to summary columns
        NumberFormatInfo format = new NumberFormatInfo();
        format.NumberDecimalDigits = 3;
        format.NumberDecimalSeparator = ".";
        format.NumberGroupSeparator = ",";
        //Number format is applied to summary columns
        element.Content = Convert.ToDouble(double.Parse(summaryText,
            NumberStyles.Currency)).ToString("N", format);
    }
}
}
}
}

```

Order ID	Customer ID	Customer Name	Unit Price
34,345*000			
1000	ALFKI	Maria Anders	\$5000.00
1001	ANATR	Ana Trujilo	\$4500.00
1002	ALFKI	Anders	\$1233.00
1003	ANTON	Antonio Moreno	\$3456.00
1004	AROUT	Thomas Hardy	\$2066.00
1005	BERGS	Christina Berglund	\$4090.00
1006	BLAUS	Hanna Moos	\$2500.00
1007	BLAUS	Frédérique Citeaux	\$3500.00
1009	BONAP	Laurence Lebihan	\$5500.00
1010	BOTTM	Elizabeth Lincoln	\$2500.00

You can download the sample demo [here](#).

#### Customizing GroupCaptionText

You can apply different group caption text format for available groups based on certain conditions by deriving the [GridCaptionSummaryCellRenderer](#) class .

For example, the group caption text is customized based on the group name and items count of the column.

#### C#

```

this.dataGrid.CellRenderers.Remove("CaptionSummary");

```

```

this.dataGrid.CellRenderers.Add("CaptionSummary", new
GridCaptionSummaryCellRendererExt());
public class GridCaptionSummaryCellRendererExt :
GridCaptionSummaryCellRenderer
{
public override void
OnUpdateEditBinding(Syncfusion.UI.Xaml.Grid.DataColumnBase dataColumn,
Syncfusion.UI.Xaml.Grid.GridCaptionSummaryCell element, object dataContext)
{
if (element.DataContext is Group &&
this.DataGrid.View.GroupDescriptions.Count > 0)
{
var groupRecord = element.DataContext as Group;
var groupedColumn = this.GetGroupedColumn(groupRecord);
if (this.DataGrid.CaptionSummaryRow == null)
{
if (this.DataGrid.View.GroupDescriptions.Count < groupRecord.Level)
return;
//Set the CaptionSummaryCell text as customized.
element.Content = GetCustomizedCaptionText(groupedColumn.HeaderText,
groupRecord.Key, groupRecord.ItemsCount);
}
else if (this.DataGrid.CaptionSummaryRow.ShowSummaryInRow)
{
element.Content =
SummaryCreator.GetSummaryDisplayTextForRow(groupRecord.SummaryDetails,
this.DataGrid.View, groupedColumn.HeaderText);
}
else
element.Content =
SummaryCreator.GetSummaryDisplayText(groupRecord.SummaryDetails,
dataColumn.GridColumn.MappingName, this.DataGrid.View);
}
}
private GridColumn GetGroupedColumn(Group group)
{
var groupDesc = this.DataGrid.View.GroupDescriptions[group.Level - 1] as
PropertyGroupDescription;
foreach (var column in this.DataGrid.Columns)
{
if (column.MappingName == groupDesc.PropertyName)
{
return column;
}
}
return null;
}
private string GetCustomizedCaptionText(string columnName, object groupName,
int itemsCount)
{
//entryText - instead of "Items", the entryText is assigned to Customize the
CaptionSummaryCell Text.
string entryText = string.Empty;
if (itemsCount < 20)
entryText = "entries in the Group";
else if (itemsCount < 40)
entryText = "elements in the Group";

```

```

else if (itemsCount < 60)
entryText = "list in the Group";
else
entryText = "items in the Group";
if (groupName.ToString().Equals("1001"))
groupName = "Thousand and One";
else if (groupName.ToString().Equals("1002"))
groupName = "Thousand and Two";
else if (groupName.ToString().Equals("1004"))
groupName = "Thousand and Four";
return string.Format("{0}: {1} - {2} {3}", columnName, groupName,
itemsCount, entryText);
}
}

```

<div> <div>Order ID ^</div> <div>Customer ID ^</div> </div>					
	Order ID ^	Customer ID ^	Customer Name	Product Name	Unit Price
▼ Order ID: Thousand and One - 3 entries in the Group					
▼ Customer ID: ANATR - 1 entries in the Group					
	1001	ANATR	Ana Trujilo	NuNuCa Nuß-Nougat-Creme	\$53,345.00
▼ Customer ID: ANTON - 1 entries in the Group					
	1001	ANTON	Antonio Moreno	Raclette Courdavault	\$13,433.00
▼ Customer ID: FKI - 1 entries in the Group					
	1001	FKI	Anders	Boston Crab Meat	\$43,350.00
▲ Order ID: Thousand and Two - 2 entries in the Group					
▲ Order ID: 1003 - 2 entries in the Group					
▲ Order ID: Thousand and Four - 2 entries in the Group					
▲ Order ID: 1005 - 1 entries in the Group					

You can download the sample demo [here](#).

See Also

[How to show vertical border to the column wise summary rows?](#)

[How to define summary rows using AttachedProperty in datagrid](#)

[How to add a textbox under each group and binding a underlying property on that?](#)

[How to load a button in CaptionSummaryRow and prevent expanding and collapsing of groups upon clicking it?](#)

[How to display the NumberFormatInfo in GridSummaryColumn?](#)

[How to apply different styles to the GridTableSummaryRow?](#)

[How to use NumberFormatInfo in the SummaryColumn of the GridTableSummaryRow?](#)

[How to Change the GroupCaptionText based on Display member of the GridComboboxColumn?](#)

[How to customize the CaptionSummaryCell text in the SfDataGrid?](#)

[How to Set the horizontal alignment on summary columns?](#)

### Filtering in WPF DataGrid (SfDataGrid)

Filtering is the process of retrieving the values from the collection which satisfy the specified condition. In the SfDataGrid the filtering can be applied through the UI as well as the programmatic filters.

#### Programmatic filtering

The [WPF DataGrid](#) Filter allows you to filter the data programmatically in below ways,

- Through View Predicate
- Through Column Filter

#### View Filtering

View filtering can be achieved by setting [SfDataGrid.View.Filter](#) delegate. You can refresh the view by calling [SfDataGrid.View.RefreshFilter](#) method.

Here, FilterRecords delegate filters the data based on Country name. FilterRecords delegate is assigned to [SfDataGrid.View.Filter](#) predicate to filter Country column. After that, [SfDataGrid.View.RefreshFilter](#) method is called to refresh the records. If the record satisfies the filter conditions, true will be returned. Else false is returned.

#### C#

```
public bool FilterRecords(object o)
{
    string filterText = "Germany";
    var item = o as OrderInfo;
    if (item != null)
    {
        if (item.Country.Equals(filterText))
            return true;
    }
    return false;
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    dataGrid.View.Filter = FilterRecords;
    dataGrid.View.RefreshFilter();
}
```

---

**Note:** View filter is not supported when ItemsSource is [DataTable](#).

---

#### Column Filtering

Column filtering is achieved by using [GridColumn.FilterPredicates](#) property and adding [FilterPredicate](#) to it.

Here, OrderID column is filtered for the data which has OrderID as 1005.

#### C#

```
dataGrid.Columns["OrderID"].FilterPredicates.Add(new FilterPredicate() {
    FilterType = FilterType.Equals, FilterValue = "1005" });
```

### Filter Behavior

The [FilterBehavior](#) property is used to specify whether to consider the [FilterValue](#) as the string or specific data type.

- **StringTyped** - Records are filtered without considering the type and it takes [FilterValue](#) type as string.
- **StronglyTyped** - Records are filtered by considering the FilterValue underlying type.

---

**Note:** When you use [DataTable](#) as items Source, [IsCaseSensitive](#) property in [FilterPredicate](#) is not applicable, since DataTable does not support CaseSensitive filtering.

---

### Improving performance while adding multiple FilterPredicates to the column in loop

You can improve the performance of filtering by suspending the data operation while adding [FilterPredicates](#) to the column for bulk updates by calling [SfDataGrid.View.BeginInit](#) and [SfDataGrid.View.EndInit](#) method, before and after the data operation.

### C#

```
private void OnApplyFilterPredicate(object obj)
{
    var dataGrid = obj as SfDataGrid;
    var gridColumn = dataGrid.Columns["EmployeeId"];
    dataGrid.View.BeginInit();
    foreach (var filterValue in FilterValues)
    {
        gridColumn.FilterPredicates.Add(new FilterPredicate()
        {
            FilterType = FilterType.Equals,
            FilterValue = filterValue,
            FilterBehavior = FilterBehavior.StronglyTyped,
            FilterMode = ColumnFilter.Value,
            PredicateType = PredicateType.Or,
            IsCaseSensitive = true
        });
    }
    dataGrid.View.EndInit();
}
```

### Clear Filtering

The WPF DataGrid (SfDataGrid) allows you to clear the filters by clearing the filter predicates. This is achieved by invoking the following methods.

- [SfDataGrid.ClearFilters](#) - Clears filters for all the columns programmatically.
- [SfDataGrid.ClearFilter\(String columnName\)](#) - Clears the filter for particular column that has the columnName as MappingName.
- [SfDataGrid.ClearFilter\(GridColumn column\)](#) - Clears the filter for particular column alone.

### C#

```
this.dataGrid.ClearFilters();
this.dataGrid.ClearFilter("OrderID");
this.dataGrid.ClearFilter(this.dataGrid.Columns[0]);
```

### Excel like UI Filtering

The WPF DataGrid (SfDataGrid) provides excel like filtering UI and also advanced filter UI to filter the data easily. UI filtering can be enabled by setting [SfDataGrid.AllowFiltering](#) property to `true`, where you can open filter UI by clicking the Filter icon in column header and filter the records.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowFiltering="True"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}" />
```

#### C#

```
dataGrid.AllowFiltering = true;
```

You can enable/disable filtering for particular column by setting [GridColumn.AllowFiltering](#) property.

#### XML

```
<syncfusion:GridTextColumn AllowFiltering="True"
    MappingName="OrderID" />
```

#### C#

```
dataGrid.Columns["OrderID"].AllowFiltering = true;
```

#### Note:

1. [GridColumn.AllowFiltering](#) has higher priority than [SfDataGrid.AllowFiltering](#) property.
2. UI filtering is not supported when using on-demand paging by setting [UseOnDemandPaging](#) to `true`.

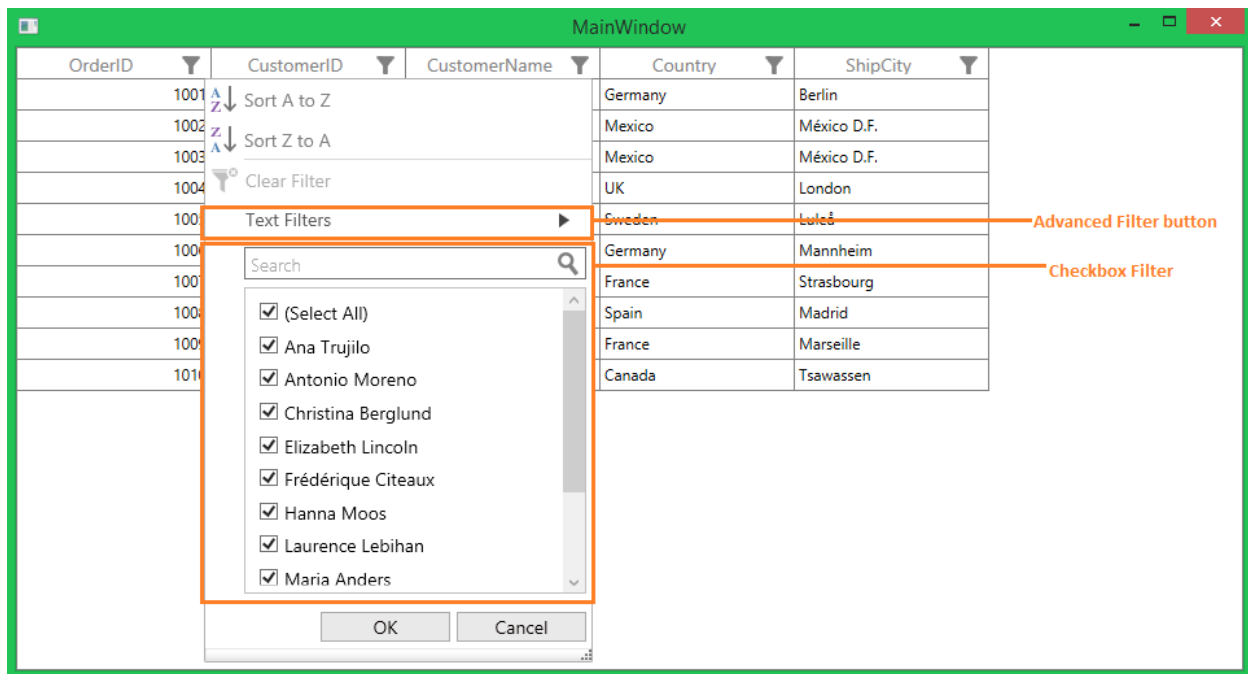
### Built-in UI Views

SfDataGrid filter UI comprises of two different UIs.

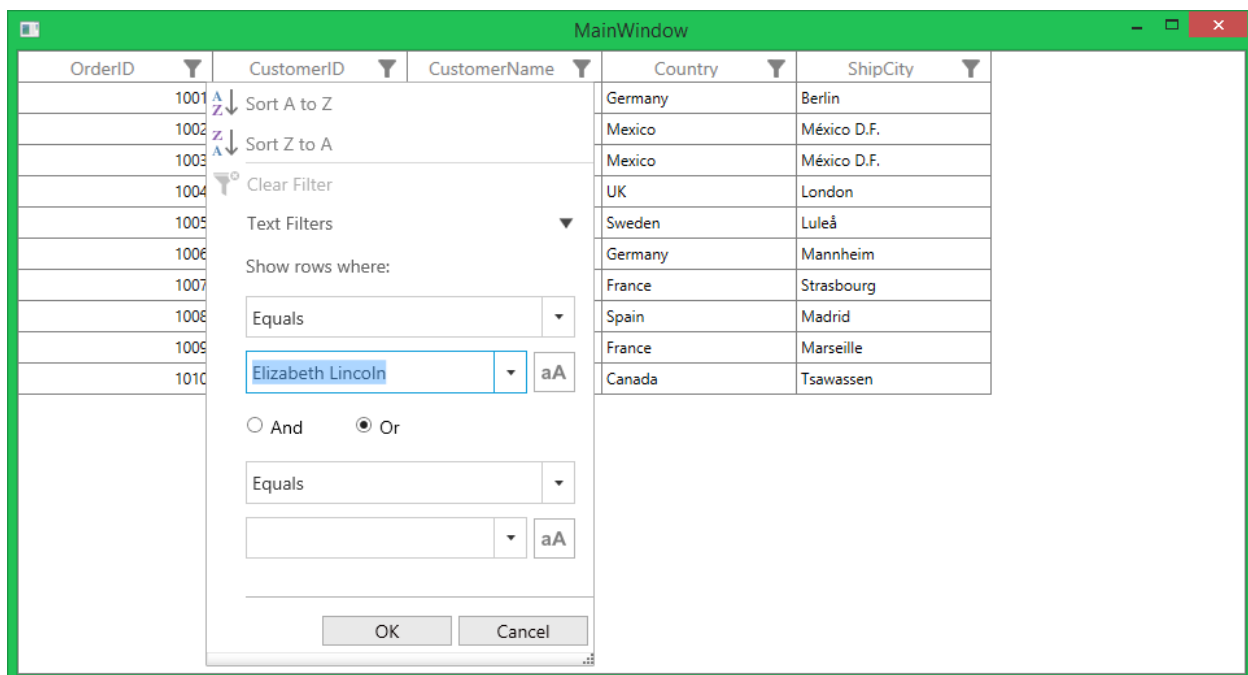
- **Checkbox Filter UI** - Provides excel like filter interface with list of check box's.
- **Advanced Filter UI** - Provides advanced filter options to filter the data.

By default, both Checkbox Filter and Advanced Filter are loaded while opening the filter pop-up. You can switch between AdvancedFilter and CheckboxFilter by using AdvancedFilter button in the UI View.

SfDataGrid with Checkbox Filter View:



SfDataGrid with Advanced Filter View:



### Choose between built-in UI Views

The WPF DataGrid (SfDataGrid) lets you to customize the UI Views displayed for particular column or grid using [FilterMode](#) property in [GridFilterControl](#).

Below are the options,

1. **CheckboxFilter** – Displays only Checkbox filter View.
2. **AdvancedFilter** – Displays only Advanced filter View.



### 3. Both – Displays both filters Views.

#### Changing filter UI View for Grid

Filter UI view can be changed for all the columns in grid by changing [FilterMode](#) in [GridFilterControl](#) by writing style and assign it to [SfDataGrid.FilterPopupStyle](#).

#### XML

```
<Style x:Key="filterControlStyle" TargetType="syncfusion:GridFilterControl">
<Setter Property="FilterMode" Value="AdvancedFilter" />
</Style>
<syncfusion:SfDataGrid x:Name="dataGrid"
AllowFiltering="True"
AutoGenerateColumns="True"
FilterPopupStyle="{StaticResource filterControlStyle}"
ItemsSource="{Binding Orders}"/>
```

#### Changing filter UI View for columns

Filter UI view can be changed for the particular column by changing [FilterMode](#) in [GridFilterControl](#) by writing style and assign it to [GridColumn.FilterPopupStyle](#).

#### XML

```
<Style x:Key="filterControlStyle" TargetType="syncfusion:GridFilterControl">
<Setter Property="FilterMode" Value="AdvancedFilter" />
</Style>
<syncfusion:GridTextColumn MappingName="OrderID"
FilterPopupStyle="{StaticResource filterControlStyle}" />
```

#### Changing filter UI View programmatically

You can change [FilterMode](#) programmatically by using [FilterItemsPopulating](#) event.

#### C#

```
this.dataGrid.FilterItemsPopulating += dataGrid_FilterItemsPopulating;
void dataGrid_FilterItemsPopulating(object sender,
Syncfusion.UI.Xaml.Grid.GridFilterItemsPopulatingEventArgs e)
{
if (e.Column.MappingName == "OrderID")
e.FilterControl.FilterMode = FilterMode.AdvancedFilter;
}
```

#### Setting Default Filter popup style for particular column

You can skip the [GridFilterControl](#) styling for particular column from [SfDataGrid.FilterPopupStyle](#) by setting [GridColumn.FilterPopupStyle](#) to `null`.

#### XML

```
<Window.Resources>
<Style x:Key="filterControlStyle" TargetType="syncfusion:GridFilterControl">
<Setter Property="FilterMode" Value="AdvancedFilter" />
</Style>
</Window.Resources>
<syncfusion:SfDataGrid Name="dataGrid"
AllowFiltering="True"
```

```

FilterPopupStyle="{StaticResource filterControlStyle}"
ItemsSource="{Binding OrderList}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn FilterPopupStyle="{x:Null}" MappingName="OrderID"
/>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

## C#

```
this.dataGrid.Columns["OrderID"].FilterPopupStyle = null;
```




Here, advanced filter will be loaded for all the columns in grid except OrderID column since [GridColumn.FilterPopupStyle](#) is set as `null` for OrderID column. So both checkbox filter and advanced filter (default style) will be loaded for OrderID column.

## Advanced Filter UI

Advanced filter UI provides multiple filter options to filter the data easily. Filter menu options are loaded based on Advanced filter type by automatically detecting the underlying data type.

Below are the built-in filter types supported.

- **Text Filters** – Loads various menu options to filter the display text effectively.
- **Number Filters** – Loads various menu options to filter the numeric data.
- **Date Filters** – Loads various menu options and [DatePicker](#) to filter DateTime type column.

Text Filters	Number Filters	Date Filters
When the string value is bounded to the <code>[GridColumn](http://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.GridColumn.html)</code> or the items source is <a href="#">dynamic</a> , then <code>TextFilters</code> are loaded in <a href="#">AdvancedFilterControl</a> .	When integer, double, short, decimal, byte or long are bound to the <code>{{ 'GridColumn'   markdownify }}</code> then <code>Number Filters</code> are loaded in <code>[AdvancedFilterControl](http://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.AdvancedFilterControl.html)</code> .	When the DateTime type value is bound to the <a href="#">GridColumn</a> , then <code>Date Filters</code> are loaded in <code>{{ 'AdvancedFilterControl'   markdownify }}</code> .
		

		Filter menu options
<p>Filter menu options</p> <ol style="list-style-type: none"> <li>1. Equals</li> <li>2. Does Not Equal</li> <li>3. Begins With</li> <li>4. Does Not Begin With</li> <li>5. Ends With</li> <li>6. Does Not End With</li> <li>7. Contains</li> <li>8. Does Not Contain</li> <li>9. Empty</li> <li>10. Not Empty</li> <li>11. Null</li> <li>12. Not Null</li> </ol>	<p>Filter menu options</p> <ol style="list-style-type: none"> <li>1. Equals</li> <li>2. Does Not Equal</li> <li>3. Null</li> <li>4. Not Null</li> <li>5. Less Than</li> <li>6. Less Than or Equal</li> <li>7. Greater Than</li> <li>8. Greater Than or Equal</li> </ol>	<ol style="list-style-type: none"> <li>1. Equals</li> <li>2. Does Not Equal</li> <li>3. Begins With</li> <li>4. Does Not Begin With</li> <li>5. Ends With</li> <li>6. Does Not End With</li> <li>7. Contains</li> <li>8. Does Not Contain</li> <li>9. Empty</li> <li>10. Not Empty</li> <li>11. Null</li> <li>12. Not Null</li> </ol>

**Note:**

1. **Null** and **Not Null** options are available only when [AllowBlankFilters](#) is set to **True**.
2. If the column is [GridUnboundColumn](#) or [GridMaskColumn](#), then **Text Filters** will be loaded.

#### Changing Advanced Filter type

[FilterBehavior](#) determines the Advanced filter type loaded in [GridFilterControl](#). By using [FilterBehavior](#), you can change Advanced filter type.

- **StringTyped** - **TextFilters** will be loaded in [AdvancedFilterControl](#).
- **Strongly Typed** – Advanced filter type is automatically detected based on underlying data type.

#### XML

```
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="OrderID"
FilterBehavior="StringTyped"/>
</syncfusion:SfDataGrid.Columns>
```

#### C#

```
dataGrid.Columns["OrderID"].FilterBehavior = FilterBehavior.StringTyped;
```

Advanced filter type can be changed programmatically by using [FilterItemsPopulating](#) event also.

#### C#

```
this.dataGrid.FilterItemsPopulating += dataGrid_FilterItemsPopulating;
void dataGrid_FilterItemsPopulating(object sender,
Syncfusion.UI.Xaml.Grid.GridFilterItemsPopulatingEventArgs e)
{
if (e.Column.MappingName != "OrderID")
return;
e.FilterControl.AdvancedFilterType = AdvancedFilterType.TextFilter;
e.FilterControl.SetColumnDataType(typeof(string));
e.FilterControl.AscendingSortString =
GridResourceWrapper.SortStringAscending;
e.FilterControl.DescendingSortString =
GridResourceWrapper.SortStringDescending;
}
```

#### Case Sensitive

By default, casing is not considered while filtering. Because, filter predicates will be created with [IsCaseSensitive](#) as **false**. If you want to filter the records with [IsCaseSensitive](#) as **true**, you need to click case sensitive button present in Advanced Filter.

---

**Note:** When you use [DataTable](#) as items Source, [CaseSensitive](#) button will not be available in Filter popup as [DataTable](#) does not support [CaseSensitive](#) filtering.

---

#### Performance tips

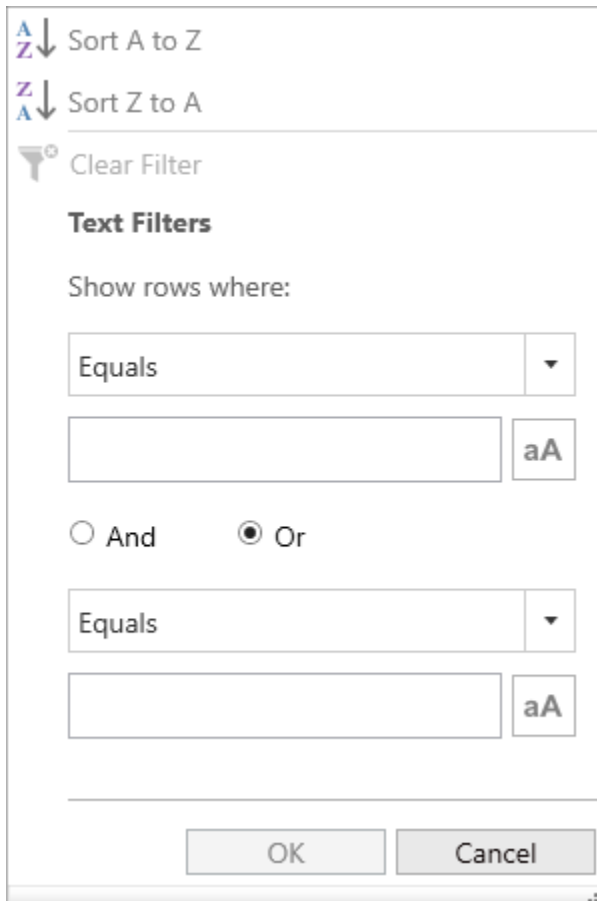
[GridFilterControl's](#) loading performance can be increased by setting [FilterMode](#) as **AdvancedFilter** and [CanGenerateUniqueItems](#) as **False**. Because a textbox is loaded instead of AdvancedFilter ComboBox that allows you to manually enter text for filtering.

**XML**

```

<Style TargetType="syncfusion:AdvancedFilterControl">
<Setter Property="CanGenerateUniqueItems" Value="False" />
</Style>
<Style x:Key="filterControlStyle" TargetType="syncfusion:GridFilterControl">
<Setter Property="FilterMode" Value="AdvancedFilter" />
</Style>
<syncfusion:SfDataGrid x:Name="dataGrid"
AllowFiltering="True"
AutoGenerateColumns="True"
FilterPopupStyle="{StaticResource filterControlStyle}"
ItemsSource="{Binding Orders}"/>

```



By default, [CanGenerateUniqueItems](#) is true. So all the unique items in the column are loaded in the AdvancedFilter ComboBox that allows you to select the value easily from the combo box and filter it.

**Filtering null values**

To filter the null values, you need to set [AllowBlankFilters](#) property as **True**. So null values will be included in filter items list. If you want to exclude the null values from filter items list, you need to set [AllowBlankFilters](#) as **False**.

**XML**

```

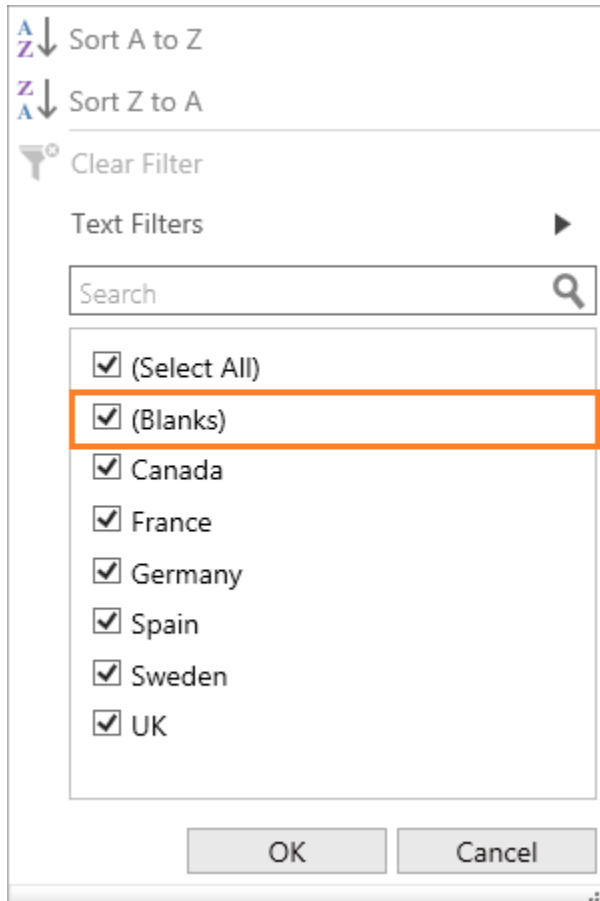
<syncfusion:GridTextColumn AllowBlankFilters="False" MappingName="Country"
/>

```

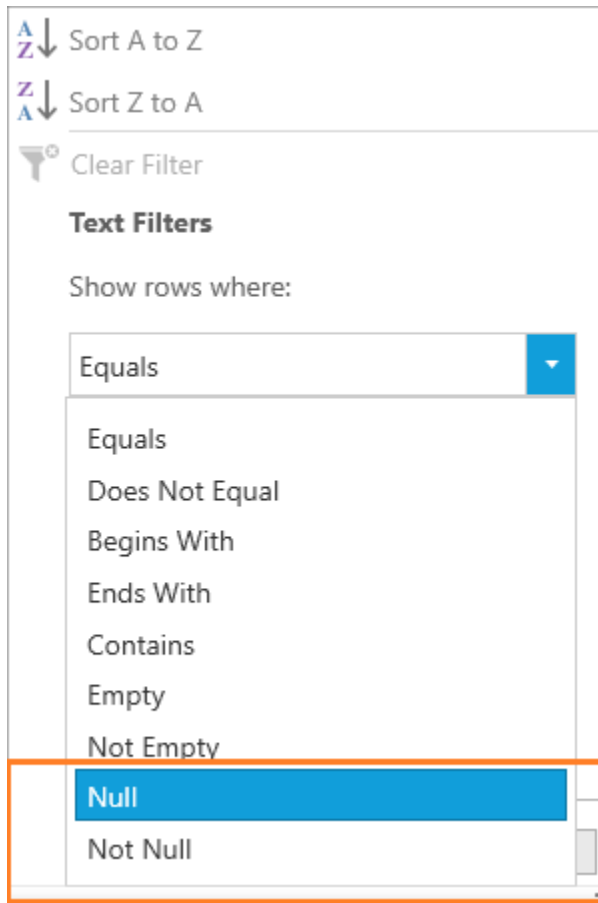
**C#**

```
dataGrid.Columns["Country"].AllowBlankFilters = false;
```

Checkbox Filter with `AllowBlankFilters` as `True`



Advanced Filter with `AllowBlankFilters` as `True`



#### Instant Filtering

By default, filters are applied to the columns when OK button is clicked in UI filtering. If you want to update the filters immediately whenever update in filter popup, you need to set [ImmediateUpdateColumnFilter](#) as True.

#### XML

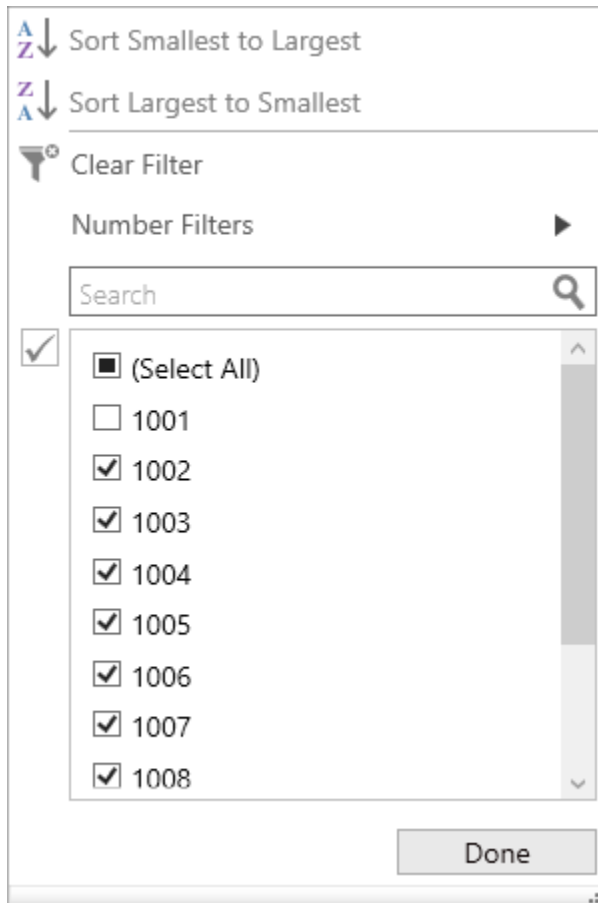
```
<syncfusion:GridTextColumn ImmediateUpdateColumnFilter="True"
    MappingName="OrderID" />
```

#### C#

```
dataGrid.Columns["OrderID"].ImmediateUpdateColumnFilter = true;
```

Here, the OK and Cancel buttons are unavailable and Done button is available to just close the popup.

Checkbox Filter with `ImmediateUpdateColumnFilter` is `True`



Advanced Filter with `ImmediateUpdateColumnFilter` is `True`



**Note:** In Checkbox Filter, the `SelectAll` option is not reflected in the filter updates if `ImmediateUpdateColumnFilter` is true.

#### Filtering based on DisplayText

In UI filtering, records are filtered based on actual value by default. If you want to filter the records based on `DisplayText`, you need to set `ColumnFilter` property as `DisplayText`.

#### XML

```
<syncfusion:GridDateTimeColumn MappingName="OrderDate"
    ColumnFilter="DisplayText"/>
```

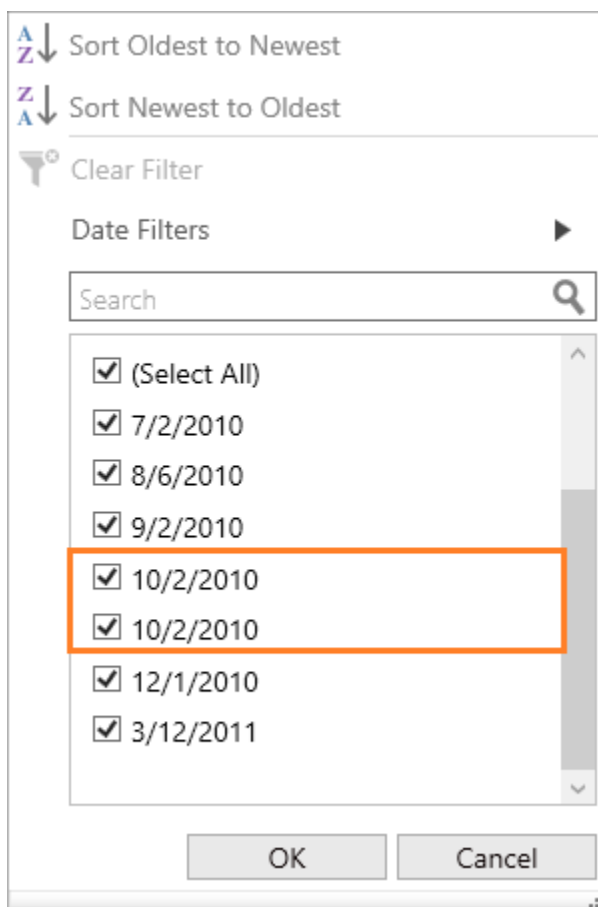
#### C#

```
dataGrid.Columns["OrderDate"].ColumnFilter = ColumnFilter.DisplayText;
```

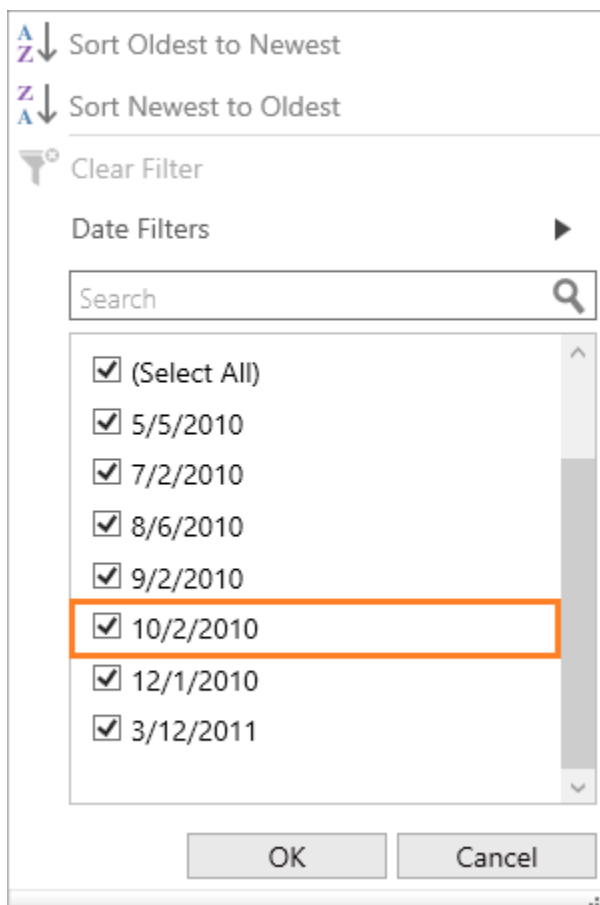
Consider in the following dataGrid, first and second records have same display value for OrderDate column but both have different actual value (E.g. 2/10/2010 12:00:00 AM and 2/10/2010 6:30:00 PM).

OrderID	CustomerID	CustomerName	ShipCity	Country	OrderDate
1001	ALFKI	Maria Anders	Berlin	Germany	10/2/2010
1002	ANATR	Ana Trujillo	México D.F.	Mexico	10/2/2010
1003	ANTON	Antonio Moreno	México D.F.	Mexico	8/6/2010
1004	AROUT	Thomas Hardy	London	UK	9/2/2010
1005	BERGS	Christina Berglund	Luleå	Sweden	4/4/2010
1006	BLAUS	Hanna Moos	Mannheim	Germany	12/1/2010
1007	BLONP	Frédérique Citeaux	Strasbourg	France	7/2/2010
1008	BOLID	Martin Sommer	Madrid	Spain	3/12/2011
1009	BONAP	Laurence Lebihan	Marseille	France	5/5/2010
1010	BOTTM	Elizabeth Lincoln	Tsawassen	Canada	4/20/2010

By default, based on the actual value only filter will be applied. So it will consider both values as different. And while opening filter popup, both values will be displayed like below.



If you set [ColumnFilter](#) as DisplayText, display value only will be considered for filtering. So filter popup will be shown like below.



After filtering, both records having the same OrderDate display value will be displayed in view.

MainWindow					
OrderID	CustomerID	CustomerName	ShipCity	Country	OrderDate
1001	ALFKI	Maria Anders	Berlin	Germany	10/2/2010
1002	ANATR	Ana Trujillo	México D.F.	Mexico	10/2/2010

## Events

SfDataGrid provides the following events for filtering.

### FilterChanging event

[FilterChanging](#) event is raised while applying filters to a particular column. You can use this event to change the [FilterPredicates](#), [FilterType](#) and [FilterBehavior](#).

### C#

```
this.dataGrid.FilterChanging += dataGrid_FilterChanging;
void dataGrid_FilterChanging(object sender, GridFilterEventArgs e)
{
}
```

### FilterChanged event

[FilterChanged](#) event is raised after filter is applied. You can use this event to get filtered records.

**C#**

```
this.dataGrid.FilterChanged += dataGrid_FilterChanged;
void dataGrid_FilterChanged(object sender, GridFilterEventArgs e)
{
}
```

*FilterItemsPopulating event*

[FilterItemsPopulating](#) event is raised while populating the filter list items in [GridFilterControl](#). You can change GridFilterControl properties by using this event.

**C#**

```
this.dataGrid.FilterItemsPopulating += dataGrid_FilterItemsPopulating;
void dataGrid_FilterItemsPopulating(object sender,
Syncfusion.UI.Xaml.Grid.GridFilterItemsPopulatingEventArgs e)
{
}
```

*FilterItemsPopulated event*

[FilterItemsPopulated](#) event is raised after filter list items are populated. You can change GridFilterControl [ItemSource](#) by using this event.

**C#**

```
this.dataGrid.FilterItemsPopulated += dataGrid_FilterItemsPopulated;
void dataGrid_FilterItemsPopulated(object sender,
GridFilterItemsPopulatedEventArgs e)
{
}
```

*Getting the filtered records*

You can get the filtered records from [View](#) in [FilterChanged](#) event. When filter is applied, the filtered records are available in [View.Records](#).

**C#**

```
this.dataGrid.FilterChanged += dataGrid_FilterChanged;
void dataGrid_FilterChanged(object sender, GridFilterEventArgs e)
{
    //OrderInfo is Model Class
    ObservableCollection<OrderInfo> order = new
    ObservableCollection<OrderInfo>();
    // Get filtered records
    var records = (sender as SfDataGrid).View.Records;
    foreach (RecordEntry record in records)
        order.Add(record.Data as OrderInfo);
}
```

*Show image in CheckBoxFilterControl instead of image path*

By default, in SfDataGrid image path is shown inside the CheckBoxFilterControl instead of image but you can show the image in CheckBoxFilterControl by setting [CheckBoxFilterControl.ItemTemplate](#) as like below.

**XML**

```

<syncfusion:GridTextColumn AllowEditing="False" HeaderText="Country"
MappingName="ImageLink">
<syncfusion:GridTextColumn.FilterPopupStyle>
<Style TargetType="syncfusion:GridFilterControl">
<Setter Property="CheckboxFilterStyle">
<Setter.Value>
<Style TargetType="syncfusion:CheckboxFilterControl">
<Setter Property="Background" Value="Red"/>
<Setter Property="ItemTemplate">
<Setter.Value>
<DataTemplate>
<CheckBox Margin="4"
HorizontalAlignment="Stretch"
HorizontalContentAlignment="Stretch"
Focusable="False"
Content="{Binding}"
FontWeight="{Binding FontWeight,RelativeSource={RelativeSource Self}}"
Foreground="{Binding Foreground,RelativeSource={RelativeSource Self}}"
IsChecked="{Binding IsSelected,
Mode=TwoWay}">
<CheckBox.ContentTemplate>
<DataTemplate>
<Image Source="{Binding Path=ActualValue, Converter={StaticResource
stringToImageConverter}}"
HorizontalAlignment="Left"
Height="25"/>
</DataTemplate>
</CheckBox.ContentTemplate>
</CheckBox>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
</Setter.Value>
</Setter>
</Style>
</syncfusion:GridTextColumn.FilterPopupStyle>
<syncfusion:GridTextColumn.CellTemplate>
<DataTemplate>
<Grid>
<Image Source="{Binding Path=ImageLink,Converter={StaticResource
stringToImageConverter}}"/>
</Grid>
</DataTemplate>
</syncfusion:GridTextColumn.CellTemplate>
</syncfusion:GridTextColumn>

```

**C#**

```

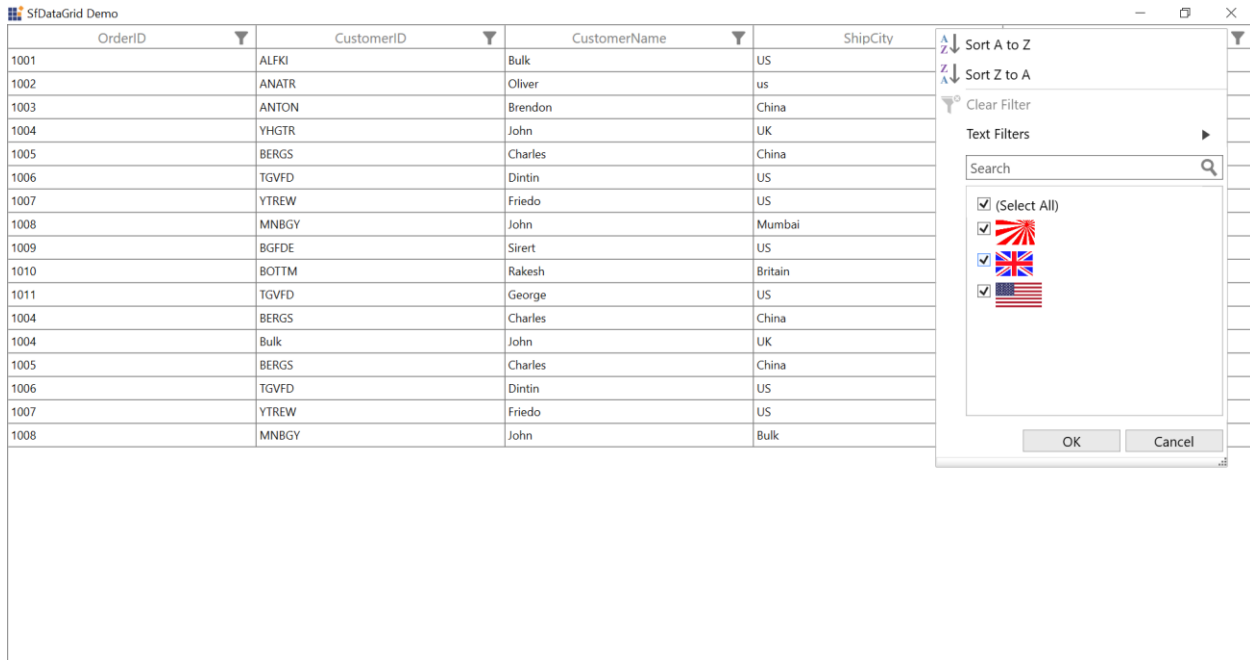
public class StringToImageConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {

```

```

string imagename = value as string;
return new BitmapImage(new Uri(string.Format(@"..\..\Images\{0}",
imagename), UriKind.Relative));
}
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
return null;
}
}

```



You can get the sample from [here](#).

Apply `ICollectionView.Filter` and `DataView.RowFilter` on initial loading

By default, the default filter created by `ICollectionView.Filter` and `DataView.RowFilter` will not be applied to the data on initial loading. These filters can be applied on initial loading by enabling `CanUseViewFilter` property.

### Functionality Customization

*Loading the Text Filters for the column having Number or Date value as underlying type*

If you want to use the Text Filters for the column that has number or date value as underlying type, you need to set `FilterBehavior` property of the `GridColumn` as `StringTyped`. This loads the Text Filters instead of Number or Date Filters.

### XML

```

<syncfusion:GridColumn MappingName="OrderID"
FilterBehavior="StringTyped"/>

```

You can achieve this programmatically by using `FilterItemsPopulating` event also.

### C#

```
this.dataGrid.FilterItemsPopulating += dataGrid_FilterItemsPopulating;
void dataGrid_FilterItemsPopulating(object sender,
Syncfusion.UI.Xaml.Grid.GridFilterItemsPopulatingEventArgs e)
{
    if (e.Column.MappingName == "OrderID")
    e.FilterControl.AdvancedFilterType = AdvancedFilterType.TextFilter;
    e.FilterControl.SetColumnDataType(typeof(string));
    e.FilterControl.AscendingSortString =
    GridResourceWrapper.SortStringAscending;
    e.FilterControl.DescendingSortString =
    GridResourceWrapper.SortStringDescending;
}
```

#### Changing AdvancedFilter type while loading dynamic ItemsSource

By default, **TextFilters** will be loaded for the columns if **ItemsSource** is **dynamic**. If you want to load Number Filters or Date Filters based on column values, you need to use **ColumnMemberType** property.

#### C#

```
this.dataGrid.Columns["OrderID"].ColumnMemberType = typeof(double?);
```

You can achieve this by using **FilterItemsPopulating** event also. But in this case, **Nullable** type values will not be filtered in advanced filtering. So you need to set **ColumnMemberType**.

#### Customizing Excel like Filter ItemsSource

When you want to restrict some data from filtering, you need to customize the **GridFilterControl** **ItemsSource** by using **FilterItemsPopulated** event. Here, **FilterElement** which has **ActualValue** as 1005 is removed from **itemsSource**.

#### C#

```
this.dataGrid.FilterItemsPopulated += dataGrid_FilterItemsPopulated;
void dataGrid_FilterItemsPopulated(object sender,
GridFilterItemsPopulatedEventArgs e)
{
    if (e.Column.MappingName == "OrderID")
    {
        var itemsSource = e.ItemsSource as List<FilterElement>;
        //Get the FilterElement to Remove from itemsSource.
        var filterElement = itemsSource.FirstOrDefault(items =>
items.ActualValue.Equals(1005));
        //Remove the FilterElement from itemsSource.
        itemsSource.Remove(filterElement);
    }
}
```

Likewise, **FilterElement** also can be changed.

#### Customizing Filter predicates

If you want to customize the filter predicates, you need to use **FilterChanging** event. Here, **FilterValue** is changed according to some conditions.

#### C#

```
this.dataGrid.FilterChanging += dataGrid_FilterChanging;
```

```
void dataGrid_FilterChanging(object sender, GridFilterEventArgs e)
{
    if (e.FilterPredicates == null || e.Column.MappingName != "CustomerID" ||
        e.FilterPredicates.Count == 0)
        return;
    if (e.FilterPredicates[0].FilterValue.Equals("ALFKI"))
        e.FilterPredicates[0].FilterValue = "ANATR";
}
```

Appearance customization

[GridFilterControl](#) is derived from [ContentControl](#) and has its own structure. This structure is customized using the properties [SfDataGrid.FilterPopupStyle](#) and [SfDataGrid.FilterPopupTemplate](#) for all the columns in grid.

When you need to change the appearance of the GridFilterControl for a particular column, [GridColumn.FilterPopupStyle](#) and [GridColumn.FilterPopupTemplate](#) properties are used.

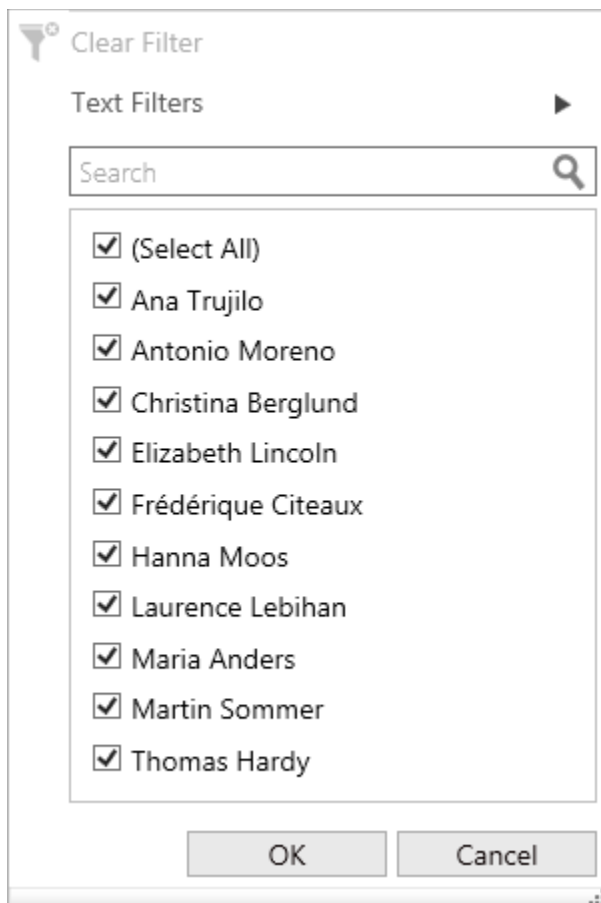
*Collapsing Sort Options in GridFilterControl*

Sort Options can be collapsed by setting [SortOptionVisibility](#) property in [GridFilterControl](#).

#### XML

```
<Style TargetType="syncfusion:GridFilterControl"
    x:Key="gridFilterControlStyle">
    <Setter Property="SortOptionVisibility" Value="Collapsed"/>
</Style>
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowFiltering="True"
    AutoGenerateColumns="True"
    FilterPopupStyle="{StaticResource gridFilterControlStyle}"
    ItemsSource="{Binding Orders}"/>
```





#### *Customizing Sort Options text*

Sort Options text can be customized by using [AscendingSortString](#) and [DescendingSortString](#) properties in [GridFilterControl](#).

#### **C#**

```
this.dataGrid.FilterItemsPopulating += DataGrid_FilterItemsPopulating;
void DataGrid_FilterItemsPopulating(object sender,
GridFilterItemsPopulatingEventArgs e)
{
    if (e.Column.MappingName=="CustomerName")
    {
        e.FilterControl.AscendingSortString = "Sort Ascending";
        e.FilterControl.DescendingSortString = "Sort Descending";
    }
}
```

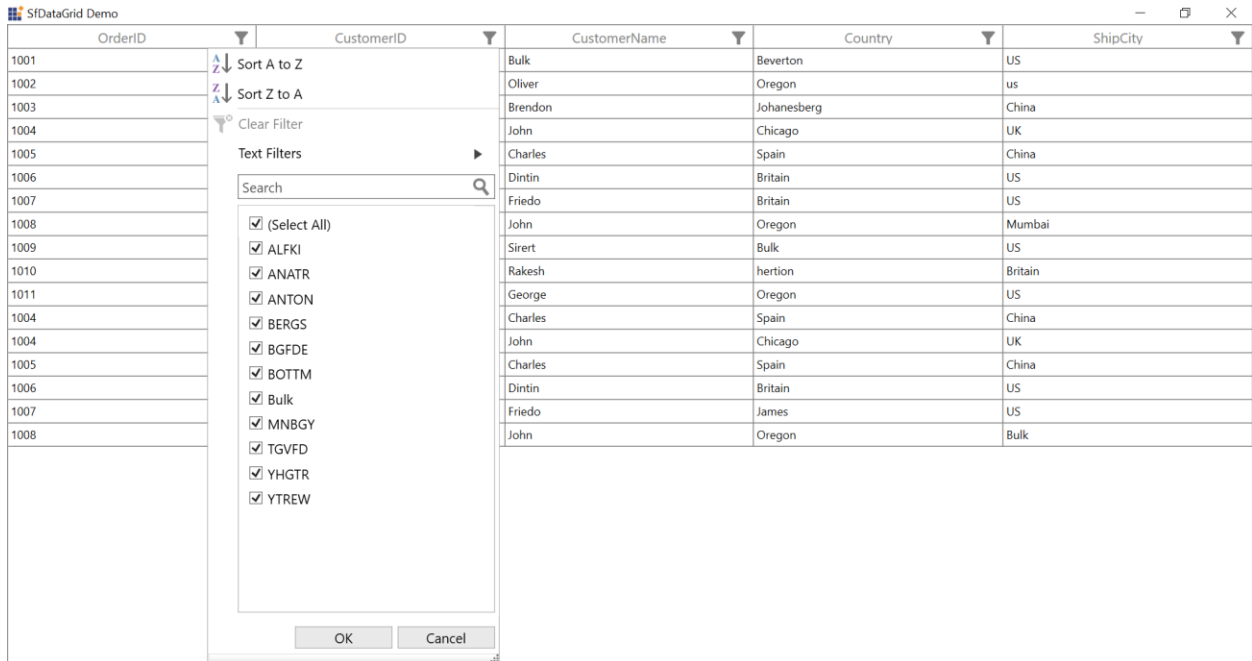
#### *Customize the FilterPopup size using GridFilterControl style*

You can customize the FilterPopup size using [FilterPopupHeight](#) property by writing style of TargetType as GridFilterControl.

#### **XML**

```
<Window.Resources>
<Style TargetType="Syncfusion:GridFilterControl">
<Setter Property="FontSize" Value="14" />
```

```
<Setter Property="FontWeight" Value="Normal" />
<Setter Property="FilterPopupHeight" Value="632"/>
</Style>
</Window.Resources>
```



#### Changing filter icon style after applying filters

You can change the filter icon style by editing the [FilterToggleButton](#) style. In [FilterToggleButton](#) style, you can see [Filtered](#) and [UnFiltered](#) VisualStates. In that, you can change [PathFillColor](#) for [FilterToggleButton](#).

#### XML

```
<Style TargetType="syncfusion:FilterToggleButton">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="syncfusion:FilterToggleButton">
        <Grid SnapsToDevicePixels="True">
          <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
              <VisualState x:Name="Normal" />
              <VisualState x:Name="MouseOver" />
              <VisualState x:Name="Pressed" />
              <VisualState x:Name="Disabled" />
            </VisualStateGroup>
            <VisualStateGroup x:Name="FilterStates">
              <VisualState x:Name="Filtered">
                <Storyboard>
                  <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="PART_FilterToggleButtonIndicator"
                    Storyboard.TargetProperty="Data">
                    <DiscreteObjectKeyFrame KeyTime="0">
                      <DiscreteObjectKeyFrame.Value>
```

```

<Geometry>M2.1299944,9.9798575L55.945994,9.9798575 35.197562,34.081179
35.197562,
62.672859 23.428433,55.942383 23.428433,33.52121z M1.3001332,0L56.635813,
0C57.355887,0,57.935946,0.5891428,57.935946,1.3080959L57.935946,
2.8258877C57.935946,3.5448422,57.355887,4.133985,56.635813,4.133985L1.300133
2,
4.133985C0.58005941,4.133985,-2.3841858E-07,3.5448422,0,2.8258877L0,
1.3080959C-2.3841858E-07,0.5891428,0.58005941,0,1.3001332,0z
</Geometry>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
<ColorAnimation Duration="0:0:0:1"
Storyboard.TargetName="PathFillColor"
Storyboard.TargetProperty="Color"
To="Red" />
</Storyboard>
</VisualState>
<VisualState x:Name="UnFiltered">
<Storyboard>
<ObjectAnimationUsingKeyFrames
Storyboard.TargetName="PART_FilterToggleButtonIndicator"
Storyboard.TargetProperty="Data">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Geometry>F1M-2124.61,-1263.65L-2131.54,-1263.72 -2145.51,-1263.84 -2152.41,
-1263.9C-2155.99,-1263.93,-2157.48,-1262.16,-2155.7,-1259.96L-2152.05,
-1255.43C-2150.28,-1253.23,-2147.38,-1249.62,-2145.61,-1247.42L-2143.25,
-1244.5 -2143.25,-1230.24C-2143.25,-1229.23,-2142.43,-1228.42,-2141.42,
-1228.42L-2135.64,-1228.42C-2134.63,-1228.42,-2133.81,-1229.23,-2133.81,
-1230.24L-2133.81,-1244.78 -2131.7,-1247.3C-2129.89,-1249.47,-2126.93,-
1253.02,
-2125.12,-1255.18L-2121.39,-1259.65C-2119.57,-1261.82,-2121.02,-1263.62,-
2124.61,-1263.65z
</Geometry>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
<ColorAnimation Duration="0:0:0:1"
Storyboard.TargetName="PathFillColor"
Storyboard.TargetProperty="Color"
To="Gray" />
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border Width="{TemplateBinding Width}"
Height="{TemplateBinding Height}"
Background="{TemplateBinding Background}"
SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}">
<Path Name="PART_FilterToggleButtonIndicator"
Margin="3"
Data="F1M-2124.61,-1263.65L-2131.54,-1263.72 -2145.51,-1263.84 -2152.41,
-1263.9C-2155.99,-1263.93,-2157.48,-1262.16,-2155.7,
-1259.96L-2152.05,-1255.43C-2150.28,-1253.23,-2147.38,
-1249.62,-2145.61,-1247.42L-2143.25,-1244.5 -2143.25,
-1230.24C-2143.25,-1229.23,-2142.43,-1228.42,-2141.42,

```

```
-1228.42L-2135.64,-1228.42C-2134.63,-1228.42,-2133.81,  
-1229.23,-2133.81,-1230.24L-2133.81,-1244.78 -2131.7,  
-1247.3C-2129.89,-1249.47,-2126.93,-1253.02,-2125.12,  
-1255.18L-2121.39,-1259.65C-2119.57,-1261.82,-2121.02,  
-1263.62,-2124.61,-1263.65z"  
SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}"  
Stretch="Fill">  
<Path.Fill>  
<SolidColorBrush x:Name="PathFillColor"  
Color="Gray" />  
</Path.Fill>  
</Path>  
</Border>  
</Grid>  
</ControlTemplate>  
</Setter.Value>  
</Setter>  
</Style>
```

When you apply above style to [FilterToggleButton](#), FilterIcon changes from Default to Gray and to Red when filtering is applied. When you clear it, it changes from Red to Gray and to default style.

See Also

[How to display values with underscore in check boxes of the filter control?](#)

[How to serialize the filtered values based on FilterMode of the column](#)

[How to apply search and filter for one column in SfDataGrid?](#)

[How to customize the Filtering and Sorting icons in the SfDataGrid ?](#)

[How to change the Filter Predicate showing in CheckBoxFilter UI ?](#)

[How to show filter status message in SfDataGrid?](#)

[How to localize the filter values in GridCheckBoxColumn ?](#)

[How to load NumberFilters in AdvanceFilters using Dynamic Collection?](#)

[How to search and select record in SfDataGrid?](#)

[How to skip the frozen row data from filtering in the SfDataGrid?](#)

[How to filter the records based on display text in the SfDataGrid?](#)

[How to change the position of FilterToggleButton and SortIcon in header cell of SfDataGrid?](#)

[How to Save and Reload the filters in SfDataGrid?](#)

[How to Customize the Excel like Filtering Items Source in SfDataGrid?](#)

[How to clear the filtering for all columns using HeaderComponentMenu?](#)

[How to change the FilterToggleButton color while filtering?](#)

[How to access the filtered records from SfDataGrid?](#)

[Filter Row in WPF DataGrid \(SfDataGrid\)](#)

SfDataGrid provides built-in row (called FilterRow) to filter the records. You can enable the FilterRow by specifying the position where it should be displayed by setting [SfDataGrid.FilterRowPosition](#) property.


**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    FilterRowPosition="FixedTop"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}"/>
```

**C#**

```
this.dataGrid.FilterRowPosition = FilterRowPosition.FixedTop;
```

OrderID	CustomerID	CustomerName	Country	ShipCity
▼	▼		▼	
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1011	ALFKI	Maria Anders	Germany	Berlin
1012	ANATR	Ana Trujillo	Mexico	México D.F.



You can get the row index of FilterRow by using the [SfDataGrid.GetFilterRowIndex](#) method.

**C#**

```
int filterRowIndex = this.dataGrid.GetFilterRowIndex();
```

You can check whether the specified row index is FilterRow index, by using [SfDataGrid.IsFilterRowIndex](#) helper method.

**C#**

```
bool isFilterRowIndex = this.dataGrid.IsFilterRowIndex(1);
```

**Note:** The above helper methods are available in [Syncfusion.UI.Xaml.Grid.Helpers](#) namespace

**Built-in Editors**

By default, FilterRow loads the editors based on underlying property type to filter the data easily. You can change the default editors by using [GridColumn.FilterRowEditorType](#) property.

**XML**

```
<syncfusion:GridTextColumn MappingName="CustomerName"
    FilterRowEditorType="MultiSelectComboBox"/>
```

**C#**

```
this.dataGrid.Columns[2].FilterRowEditorType = "MultiSelectComboBox";
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	<input type="checkbox"/> Select All	Germany	Berlin
1002	ANATR	<input checked="" type="checkbox"/> Ana Trujilo	Mexico	México D.F.
1003	ANTON	<input checked="" type="checkbox"/> Antonio Moreno	Mexico	México D.F.
1004	AROUT	<input checked="" type="checkbox"/> Christina Berglund	UK	London
1005	BERGS	<input type="checkbox"/> Elizabeth Lincoln	Sweden	Luleå
1006	BLAUS	<input type="checkbox"/> Frédérique Citeaux	Germany	Mannheim
1007	BLONP	<input type="checkbox"/> Hanna Moos	France	Strasbourg
1008	BOLID	<input type="checkbox"/> Laurence Lebihan	Spain	Madrid
1009	BONAP	<input type="checkbox"/> Laurence Lebihan	France	Marseille
1010	BOTTM	OK Cancel	Canada	Tsawassen
1011	ALFKI	Maria Anders	Germany	Berlin
1012	ANATR	Ana Trujilo	Mexico	México D.F.

Below are the built-in FilterRow editor types supported in SfDataGrid.

FilterRowEditor Type	Editor Control	Renderer	Description
TextBox	TextBox	<a href="#">GridFilterRowTextBoxRenderer</a>	Used for filtering the string values.
Numeric	DoubleTextBox	<a href="#">GridFilterRowNumericRenderer</a>	Used for filtering the numeric values.
ComboBox	ComboBoxAdv	<a href="#">GridFilterRowComboBoxRenderer</a>	Used for filtering the specific value from the drop down.
MultiSelectComboBox	ComboBoxAdv	<a href="#">GridFilterRowMultiSelectRenderer</a>	Used for filtering the multiple values from the drop down.
CheckBox	CheckBox	<a href="#">GridFilterRowCheckBoxRenderer</a>	Used for filtering the Boolean values.
DateTime	DateTimeEdit	<a href="#">GridFilterRowDateTimeRenderer</a>	Used for filtering the DateTime values.

### Filter options

Based on the editor type, FilterRowCell displays the filter conditions in dropdown where you can easily switch between the conditions to filter the data. You can disable filter options by setting [GridColumn.FilterRowOptionsVisibility](#) property.

### XML

```
<syncfusion:GridNumericColumn MappingName="OrderID"
    FilterRowOptionsVisibility="Collapsed"
    FilterRowEditorType="Numeric"/>
```

**C#**

```
this.dataGrid.Columns[0].FilterRowOptionsVisibility =
System.Windows.Visibility.Collapsed;
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1011	ALFKI	Maria Anders	Germany	Berlin
1012	ANATR	Ana Trujilo	Mexico	México D.F.

Below are the filter conditions supported by different filter row editors in SfDataGrid.

Numeric Editor	Text Box Editor	DateTime Editor	CheckBox Editor	ComboBox, MultiSelectComboBox Editor
When integer, double, short, decimal, byte or long are bound to the <code>[GridColumn](http://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.GridColumn.html)</code> , the Numeric editor type are loaded in <code>[FilterRowCell](https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.RowFilter.GridFilterRowCellRenderer-2.html#SyncfusionUIXamlGridRowFilterGridFilterRowCellRenderer2_FilterRowCell)</code> .	When string value is bound to the <code>{{'GridColumn' mark down if}}</code> or the items is dynamic,	When DateTime type is bounded to the <code>[GridColumn](http://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.GridColumn.html)</code> , then DateTime editor is loaded in <code>[FilterRowCell](https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.RowFilter.GridFilterRowCellRenderer-2.html#SyncfusionUIXamlGridRowFilterGridFilterRowCellRenderer2_FilterRowCell)</code> .	When Boolean type is bounded to the <code>{{'GridColumn' mark down if}}</code> , then CheckBox editor is	If we need the ComboBox and MultiSelectComboBox we have to set the <a href="#">FilterRowEditorType</a> .

	then Text Box edito r type are loade d in {{'Filt erRo wCell'   mark down ify }}.		loade d in {{'Filt erRo wCell'   mark down ify }}.	
<p>The default filter condition is Equals, the below filter conditions are available in numeric filter.</p> <ol style="list-style-type: none"> <li>1. Equals</li> <li>2. Does Not Equal</li> <li>3. Null</li> <li>4. Not Null</li> <li>5. Less Than</li> <li>6. Less Than or Equal</li> <li>7. Greater Than</li> <li>8. Greater Than or Equal</li> </ol>	<p>The default filter condition is Begins With, the below filter conditions are available in text filter.</p> <ol style="list-style-type: none"> <li>1.</li> <li>2.</li> </ol>	<p>The default filter condition is Equals, the below filter conditions are available in date time filter.</p> <ol style="list-style-type: none"> <li>1. Equals</li> <li>2. Does Not Equal</li> <li>3. Null</li> <li>4. Not Null</li> <li>5. Before</li> <li>6. Before or Equal</li> <li>7. After</li> <li>8. After or Equal</li> </ol>	<p>Always equals or not equals filter condition will be applied for selected items count for filtering the items.</p>	



	3.			
	4.			
	5.			
	6.			
	7.			

	8.			
	9.			
	10			
	11			
	12			

--	--	--	--	--

You can change the default FilterRow condition for a corresponding column by using [GridColumn.FilterRowCondition](#) property.

#### XML

```
<syncfusion:GridNumericColumn MappingName="OrderID"
    FilterRowCondition="LessThanOrEqual"
    FilterRowEditorType="Numeric"/>
```

#### C#

```
this.dataGrid.Columns[0].FilterRowCondition =
    FilterRowCondition.LessThanOrEqual;
```

OrderID	CustomerID	CustomerName	Country	ShipCity
▼	▼		▼	▼
Equals	ALFKI	Maria Anders	Germany	Berlin
Does Not Equal	ANATR	Ana Trujilo	Mexico	México D.F.
Less Than	ANTON	Antonio Moreno	Mexico	México D.F.
Less Than or Equal	AROUT	Thomas Hardy	UK	London
Greater Than	BERGS	Christina Berglund	Sweden	Luleå
Greater Than or Equal	BLAUS	Hanna Moos	Germany	Mannheim
Null	BLONP	Frédérique Citeaux	France	Strasbourg
Not Null				
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1011	ALFKI	Maria Anders	Germany	Berlin
1012	ANATR	Ana Trujilo	Mexico	México D.F.

#### Filtering null values

You can enable or disable filtering of null values by setting [GridColumn.AllowBlankFilters](#) property. The default value is `true`.

When null value filtering is enabled, the filter options loaded with two additional options ("Null" and "Not Null") to filter the null values. ComboBox and MultiSelectComboBox editors, loads with "Blanks" item in drop down to filter the null values.

#### XML

```
<syncfusion:GridNumericColumn MappingName="OrderID"
    AllowBlankFilters="False">
```

```
FilterRowEditorType="Numeric"/>
```

**C#**

```
this.dataGrid.Columns[0].AllowBlankFilters = false;
```

OrderID	CustomerID	CustomerName	Country	ShipCity
	ALFKI	Maria Anders	Germany	Berlin
	ANATR	Ana Trujilo	Mexico	México D.F.
	ANTON	Antonio Moreno	Mexico	México D.F.
	AROUT	Thomas Hardy	UK	London
	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1011	ALFKI	Maria Anders	Germany	Berlin
1012	ANATR		Mexico	México D.F.

**XML**

```
<syncfusion:GridTextColumn MappingName="CustomerName"
AllowBlankFilters="True"
FilterRowEditorType="MultiSelectComboBox"/>
```

**C#**

```
this.dataGrid.Columns[2].AllowBlankFilters = true;
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	<input type="checkbox"/> Select All	Germany	Berlin
1002	ANATR	<input checked="" type="checkbox"/> (Blanks)	Mexico	México D.F.
1003	ANTON	<input type="checkbox"/> Ana Trujilo	Mexico	México D.F.
1004	AROUT	<input type="checkbox"/> Antonio Moreno	UK	London
1005	BERGS	<input type="checkbox"/> Christina Berglund	Sweden	Luleå
1006	BLAUS	<input type="checkbox"/> Elizabeth Lincoln	Germany	Mannheim
1007	BLONP	<input type="checkbox"/> Frédérique Citeaux	France	Strasbourg
1008	BOLID	<input type="checkbox"/> Hanna Moos	Spain	Madrid
1009	BONAP	<input type="checkbox"/> Maria Anders	France	Marseille
1010	BOTTM	OK Cancel	Canada	Tsawassen
1011	ALFKI		Germany	Berlin
1012	ANATR		Mexico	México D.F.

### Instant Filtering

By default, filters are applied to the columns when moving to other cells or pressing enter key. You can apply filter when typing or selecting in editor itself by setting [GridColumn.ImmediateUpdateColumnFilter](#) as `true`.

### XML

```
<syncfusion:GridTextColumn MappingName="CustomerName"
    FilterRowEditorType="MultiSelectComboBox"
    ImmediateUpdateColumnFilter="True"/>
```

### C#

```
this.dataGrid.Columns[2].ImmediateUpdateColumnFilter = true;
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	<input type="checkbox"/> Select All	Germany	Berlin
1002	ANATR	<input type="checkbox"/> Ana Trujilo	Mexico	México D.F.
1003	ANTON	<input type="checkbox"/> Antonio Moreno	Mexico	México D.F.
1004	AROUT	<input type="checkbox"/> Christina Berglund	UK	London
1005	BERGS	<input type="checkbox"/> Elizabeth Lincoln	Sweden	Luleå
1006	BLAUS	<input type="checkbox"/> Frédérique Citeaux	Germany	Mannheim
1007	BLONP	<input type="checkbox"/> Hanna Moos	France	Strasbourg
1008	BOLID	<input type="checkbox"/> Laurence Lebihan	Spain	Madrid
1009	BONAP	<input type="checkbox"/> Maria Anders	France	Marseille
1010	BOTTM		Canada	Tsawassen
1011	ALFKI	Maria Anders	Germany	Berlin
1012	ANATR	Ana Trujilo	Mexico	México D.F.

### Disable filtering for a particular FilterRowCell

By default, you can filter the records by editing filter row cell. You can disable this editing by using [CurrentCellBeginEdit](#) event.

### C#

```
this.dataGrid.CurrentCellBeginEdit += dataGrid_CurrentCellBeginEdit;
void dataGrid_CurrentCellBeginEdit(object sender,
    CurrentCellBeginEditEventArgs args)
{
    //Cancel the editing for filter row cell in OrderID Column
    if (args.Column.MappingName == "OrderID" &&
        dataGrid.IsFilterRowIndex(args.RowColumnIndex.RowIndex))
        args.Cancel = true;
}
```

You can collapse the FilterOption button using `FilterRowOptionsVisibility` property.

### XML

```
<syncfusion:GridNumericColumn MappingName="OrderID"
```

```
FilterRowOptionsVisibility="Collapsed"/>
```

## Styling

### Filter row style

You can customize the style of filter row by writing style of TargetType [FilterRowControl](#).

### XML

```
<Window.Resources>
<Style TargetType="syncfusion:FilterRowControl">
<Setter Property="Background" Value="BlanchedAlmond"/>
</Style>
</Window.Resources>
```

OrderID	CustomerID	CustomerName	Country	ShipCity
▼	▼		▼	▼
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1011	ALFKI	Maria Anders	Germany	Berlin
1012	ANATR	Ana Trujilo	Mexico	México D.F.

### Filter row - cell style

You can customize the style of filter row cell by writing style of TargetType [GridFilterRowCell](#).

### XML

```
<local:FilterRowCellStyleConverter x:Key="filterRowCellStyleConverter"/>
<Style TargetType="syncfusion:GridFilterRowCell">
<Setter Property="Background" Value="{Binding RelativeSource={RelativeSource
Self}, Converter={StaticResource filterRowCellStyleConverter}}"/>
</Style>
```

### C#

```
public class FilterRowCellStyleConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        var filterRowCell = value as GridFilterRowCell;
        if (filterRowCell == null) return null;
        if (filterRowCell.DataColumn.GridColumn.MappingName == "OrderID" ||
            filterRowCell.DataColumn.GridColumn.MappingName == "CustomerName")
```

```

return new SolidColorBrush(Colors.LightBlue);
return new SolidColorBrush(Colors.Honeydew);
}
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
return value;
}
}

```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1011	ALFKI	Maria Anders	Germany	Berlin
1012	ANATR	Ana Trujilo	Mexico	México D.F.

### Customizing filter row cell

You can customize the filter row cell by overriding the [GridFilterRowCell](#). You have to override the GetGridCell method in [RowGenerator](#) to load the customized [GridFilterRowCell](#).

### C#

```

public class GridFilterRowCellExt : GridFilterRowCell
{
public GridFilterRowCellExt()
: base()
{ }
}
public class CustomRowGenerator : RowGenerator
{
public CustomRowGenerator(SfDataGrid dataGrid)
: base(dataGrid)
{
}
/// <summary>
/// Return the Custom FilterRowCell
/// </summary>
/// <typeparam name="T"></typeparam>
/// <returns>GridCell</returns>
protected override GridCell GetGridCell<T>()
{
//If the Cell is FilterRowCell return custom FilterRowCell
if (typeof(T) == typeof(GridFilterRowCell))

```

```

return new GridFilterRowCellExt();
return base.GetGridCell<GridCell>();
}
}
public MainWindow()
{
InitializeComponent();
this.dataGrid.RowGenerator = new CustomRowGenerator(this.dataGrid);
}

```

#### Customizing filter options for particular columns

By default, TextBox editor will display the string related conditions like Begins With, Does Not Begin With, Ends With, Does Not End With, Contains, Does Not Contain, Empty, Not Empty filter row conditions in drop down. The below code shows how to display the custom filter row conditions in TextBox editor by overriding the [OpenFilterOptionPopup](#) method in a [GridFilterRowCell](#) class.

#### C#

```

this.dataGrid.RowGenerator = new CustomRowGenerator(this.dataGrid);
public class GridFilterRowCellExt : GridFilterRowCell
{
public GridFilterRowCellExt()
: base()
{ }
/// <summary>
/// Opens the FilterOptionPopup with the FilterOptionList.
/// </summary>
public override void OpenFilterOptionPopup()
{
base.OpenFilterOptionPopup();
if (this.DataColumn.GridColumn.MappingName != "CustomerID")
return;
var list = this.OptionsList();
if (list.Count > 0)
this.FilterOptionsList.ItemsSource = list;
}
/// <summary>
/// Populates the FilterOption list which will loaded in FilterOptionPopup
for ShipAddress.
/// </summary>
/// <returns></returns>
private new ObservableCollection<string> OptionsList()
{
var list = new ObservableCollection<string>();
list.Add("Contains");
list.Add("Does not contain");
list.Add("Match");
list.Add("Does not match");
list.Add("Like");
list.Add("Not Like");
return list;
}
}
public class CustomRowGenerator : RowGenerator
{
public CustomRowGenerator(SfDataGrid dataGrid)

```



```

: base(dataGrid)
{
}
/// <summary>
/// Return the Custom FilterRowCell
/// </summary>
/// <typeparam name="T"></typeparam>
/// <returns>GridCell</returns>
protected override GridCell GetGridCell<T>()
{
    //If the Cell is FilterRowCell return custom FilterRowCell
    if (typeof(T) == typeof(GridFilterRowCell))
    return new GridFilterRowCellExt();
    return base.GetGridCell<GridCell>();
}
}

```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	A	Maria Anders	Germany	Berlin
1002	A	Ana Trujilo	Mexico	México D.F.
1003	A	Antonio Moreno	Mexico	México D.F.
1004	A	Thomas Hardy	UK	London
1005	B	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1011	ALFKI	Maria Anders	Germany	Berlin
1012	ANATR	Ana Trujilo	Mexico	México D.F.

### Customizing Filter row editors

#### Customizing the filter row renderer

SfDataGrid allows you to customize the filter row renderer behavior by overriding the corresponding renderer associated with the filter row cell. Each renderer have a set of virtual methods for handling the filter row behaviors. You can also create new renderers instead of overriding the existing renderer.

You can customize the default TextBox editor behavior by overriding `GridFilterRowTextBoxRenderer` class and add the custom renderer to [FilterRowCellRenderers](#).

#### XML

```

<syncfusion:GridTextColumn MappingName="CustomerName"
    FilterRowEditorType="TextBoxExt"/>

```

#### C#

```

public class GridFilterRowTextBoxRendererExt : GridFilterRowTextBoxRenderer
{
    public GridFilterRowTextBoxRendererExt()
    {
    }
}

```

```

: base()
{
}
}

public MainWindow()
{
InitializeComponent();
this.dataGrid.FilterRowCellRenderers.Add("TextBoxExt", new
GridFilterRowTextBoxRendererExt());
}

```

#### Filter based on numeric interval by using the multi select combobox filter

By default you can filter the multiple data in the column by using MultiSelectComboBox filter editor type, the below code shows how to filter the data based on range of numeric values by overriding the [ProcessMultipleFilters](#) method in [GridFilterRowComboBoxRenderer](#) class.

#### XML

```

<syncfusion:GridNumericColumn MappingName="OrderID"
FilterRowOptionsVisibility="Collapsed"
NumberDecimalDigits="0"
FilterRowEditorType="ComboBoxExt"/>

```

#### C#

```

this.dataGrid.FilterRowCellRenderers.Add("ComboBoxExt", new
GridFilterRowComboBoxRendererExt());

public class GridFilterRowComboBoxRendererExt :
GridFilterRowComboBoxRenderer, INotifyPropertyChanged
{
private List<string> numericComboBoxItems;
public GridFilterRowComboBoxRendererExt()
: base()
{
SetNumericComboBoxItemsList();
}

/// <summary>
/// Generate the Items for NumericComboBox
/// </summary>
/// <returns></returns>
public void SetNumericComboBoxItemsList()
{
numericComboBoxItems = new List<string>();
numericComboBoxItems.Add("Between 1001 and 1004");
numericComboBoxItems.Add("Between 1005 and 1009");
numericComboBoxItems.Add("Between 1010 and 1014");
numericComboBoxItems.Add("Between 1015 and 1020");
numericComboBoxItems.Add(">1020");
}

/// <summary>
/// InitializeEditBinding based on our item, set the SelectedItem and set
the ItemSource.
/// </summary>
/// <param name="uiElement">Corresponding UIElement</param>
/// <param name="dataColumn">Corresponding Column</param>

```

```

protected override void
InitializeEditBinding(Syncfusion.Windows.Tools.Controls.ComboBoxAdv
uiElement, DataColumnBase dataColumn)
{
    ObservableCollection<object> selItems = new ObservableCollection<object>();
    //Generate the items for FilterRow
    uiElement.ItemsSource = numericComboBoxItems;
    if (dataColumn.GridColumn.FilteredFrom == FilteredFrom.FilterRow &&
        dataColumn.GridColumn.FilterPredicates.Count > 0)
    {
        if (numericComboBoxItems != null)
        {
            numericComboBoxItems.ForEach(element =>
            {
                //Check if the filter is already applied or not, if applied means again add
                the filter
                bool needToAdd = false;
                switch (element)
                {
                    case "Between 1001 and 1004":
                        needToAdd = this.NeedToAdd(dataColumn.GridColumn.FilterPredicates, "1001");
                        break;
                    case "Between 1005 and 1009":
                        needToAdd = this.NeedToAdd(dataColumn.GridColumn.FilterPredicates, "1005");
                        break;
                    case "Between 1010 and 1014":
                        needToAdd = this.NeedToAdd(dataColumn.GridColumn.FilterPredicates, "1010");
                        break;
                    case "Between 1015 and 1020":
                        needToAdd = this.NeedToAdd(dataColumn.GridColumn.FilterPredicates, "1015");
                        break;
                    case ">1020":
                        needToAdd = this.NeedToAdd(dataColumn.GridColumn.FilterPredicates, "1020");
                        break;
                }
                if (needToAdd)
                    selItems.Add(element);
            });
        }
        if (selItems.Count > 0)
            uiElement.SelectedItems = selItems;
        else if (uiElement.SelectedItems != null)
            uiElement.SelectedItems = null;
        uiElement.AllowMultiSelect = true;
        uiElement.AllowSelectAll = true;
        uiElement.EnableOKCancel = true;
        uiElement.IsEditable = false;
    }
    /// <summary>
    /// Check whether the column having a FilterPredicate or not
    /// </summary>
    /// <param name="filterPredicate">FilterPredicates for a column</param>
    /// <param name="filterValue">FilterValue for a column</param>
    /// <returns></returns>
    private bool NeedToAdd(ObservableCollection<FilterPredicate>
filterPredicate, string filterValue)

```

```
{
    bool needToAdd = false;
    foreach (var item in filterPredicate)
    {
        if ((item as FilterPredicate).FilterValue.ToString() == filterValue)
        {
            needToAdd = true;
            break;
        }
    }
    return needToAdd;
}

/// <summary>
/// Generate the FilterPredicates and apply the filter for a corresponding
/// column
/// </summary>
/// <param name="filterValues">Corresponding Filter Value</param>
/// <param name="totalItems">Corresponding Filter Items</param>
public override void ProcessMultipleFilters(List<object> filterValues,
List<object> totalItems)
{
    var selectedItems = filterValues.Cast<string>().ToList();
    var total = totalItems.Cast<string>().ToList();
    if (selectedItems == null || total == null || filterValues == null)
        return;
    if (selectedItems.Count == total.Count)
    {
        this.ApplyFilters(null, string.Empty);
        this.IsValueChanged = false;
        return;
    }
    var filterPredicates = new List<FilterPredicate>();
    if (filterValues.Count > 0)
    {
        selectedItems.ForEach(item =>
        {
            switch (item)
            {
                case "Between 1001 and 1004":
                    filterPredicates.Add(GetFilterPredicates((int)1001, FilterType.GreaterThan,
                    PredicateType.OrElse));
                    filterPredicates.Add(GetFilterPredicates((int)1004, FilterType.LessThan,
                    PredicateType.And));
                    break;
                case "Between 1005 and 1009":
                    filterPredicates.Add(GetFilterPredicates((int)1005, FilterType.GreaterThan,
                    PredicateType.OrElse));
                    filterPredicates.Add(GetFilterPredicates((int)1009, FilterType.LessThan,
                    PredicateType.And));
                    break;
                case "Between 1010 and 1014":
                    filterPredicates.Add(GetFilterPredicates((int)1010, FilterType.GreaterThan,
                    PredicateType.OrElse));
                    filterPredicates.Add(GetFilterPredicates((int)1014, FilterType.LessThan,
                    PredicateType.And));
                    break;
                case "Between 1015 and 1020":
```

```

filterPredicates.Add(GetFilterPredicates((int)1015, FilterType.GreaterThan,
PredicateType.OrElse));
filterPredicates.Add(GetFilterPredicates((int)1020, FilterType.LessThan,
PredicateType.And));
break;
case ">1020":
filterPredicates.Add(GetFilterPredicates((int)1020, FilterType.GreaterThan,
PredicateType.Or));
break;
}
});
}
string _filterText = string.Empty;
//Creates the FilterText
if (filterPredicates.Count > 0)
{
var selectItems = ((IList)filterValues).Cast<string>().ToList();
for (int i = 0; i < selectItems.Count; i++)
{
_filterText += selectItems[i];
if (i != selectItems.Count - 1)
_filterText += " - ";
}
}
if (filterPredicates != null)
this.ApplyFilters(filterPredicates, _filterText);
this.IsValueChanged = false;
}
/// <summary>
/// Generate the Filter Predicates for Strongly Typed
/// </summary>
/// <param name="value"></param>
/// <param name="filterType"></param>
/// <param name="predType"></param>
/// <returns></returns>
private FilterPredicate GetFilterPredicates(object value, FilterType
filterType, PredicateType predType)
{
return new FilterPredicate()
{
FilterBehavior = FilterBehavior.StronglyTyped,
FilterType = filterType,
FilterValue = value,
IsCaseSensitive = false,
PredicateType = predType
};
}
/// <summary>
/// Generate the Filter Predicates for String Typed
/// </summary>
/// <param name="value"></param>
/// <param name="filterType"></param>
/// <param name="predType"></param>
/// <returns></returns>
private FilterPredicate GetStringFilterPredicates(object value, FilterType
filterType, PredicateType predType)
{

```

```

return new FilterPredicate()
{
    FilterBehavior = FilterBehavior.StringTyped,
    FilterType = filterType,
    FilterValue = value,
    IsCaseSensitive = false,
    PredicateType = predType
};
}
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(String prop)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(prop));
    }
}
}

```

CustomerID	OrderID	CustomerName	Country	ShipCity
ALFKI	<input type="checkbox"/> Select All	Maria Anders	Germany	Berlin
ANATR	<input type="checkbox"/> Between 1001 and 1004	Ana Trujilo	Mexico	México D.F.
ANTON	<input type="checkbox"/> Between 1005 and 1009	Antonio Moreno	Mexico	México D.F.
AROUT	<input type="checkbox"/> Between 1010 and 1014	Thomas Hardy	UK	London
BERGS	<input type="checkbox"/> Between 1015 and 1020	Christina Berglund	Sweden	Luleå
BLAUS	<input type="checkbox"/> >1020	Hanna Moos	Germany	Mannheim
BLONP	OK Cancel	Frédérique Citeaux	France	Strasbourg
BOLID		Martin Sommer	Spain	Madrid
BONAP	1009	Laurence Lebihan	France	Marseille
BOTTM	1010	Elizabeth Lincoln	Canada	Tsawassen
ALFKI	1011	Maria Anders	Germany	Berlin
ANATR	1012	Ana Trujilo	Mexico	México D.F.
ANTON	1013	Antonio Moreno	Mexico	México D.F.
AROUT	1014	Thomas Hardy	UK	London
BERGS	1015	Christina Berglund	Sweden	Luleå
BLAUS	1016	Hanna Moos	Germany	Mannheim
BLONP	1017	Frédérique Citeaux	France	Strasbourg
BOLID	1018	Martin Sommer	Spain	Madrid
BONAP	1019	Laurence Lebihan	France	Marseille
BOTTM	1020	Elizabeth Lincoln	Canada	Tsawassen

#### *Numeric filter row conditions for string typed column*

By default, TextBox filter is loaded when underlying property type is string which is bound to a column. The below code shows how to apply the numeric filter row conditions for that particular column using converters in [ValueBinding](#) property.

You can filter the decimal value in a filter row cell by overriding the [OnInitializeEditElement](#) method in [GridFilterRowNumericRenderer](#) class.

**XML**

```
<syncfusion:GridTextColumn MappingName="Freight"
    UseBindingValue="True"
    ValueBinding="{Binding Path=Freight, Converter={StaticResource
        stringToNumericConverter}}"/>
```

**C#**

```
public class StringToNumericConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        return double.Parse(value.ToString());
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
//Remove the existing renderer
if (dataGrid.FilterRowCellRenderers.ContainsKey("Numeric"))
    dataGrid.FilterRowCellRenderers.Remove("Numeric");
//Add the new custom renderer
dataGrid.FilterRowCellRenderers.Add("Numeric", new
    GridFilterRowNumericRendererExt());
public class GridFilterRowNumericRendererExt : GridFilterRowNumericRenderer
{
    public override void OnInitializeEditElement(DataColumnBase dataColumn,
        DoubleTextBox uiElement, object dataContext)
    {
        base.OnInitializeEditElement(dataColumn, uiElement, dataContext);
        //Set the NumericDecimalDigits
        uiElement.NumberDecimalDigits = 2;
    }
}
```

## Customizing GridFilterRowMultiSelectRenderer

By default, in SfDataGrid ComboBox is loaded while enter into edit mode in FilterRow but you can customize the [GridFilterRowMultiSelectRenderer]

(<https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.RowFilter.GridFilterRowMultiSelectRenderer.html>) for display the combobox while FilterRow loading itself.

**XML**

```
<Syncfusion:GridTextColumn MappingName="OrderID" HeaderText="OrderID"
    FilterRowEditorType="MultiSelectComboBox"/>
```

**C#**

```
sfgrid.FilterRowCellRenderers.Remove("MultiSelectComboBox");
sfgrid.FilterRowCellRenderers.Add("MultiSelectComboBox", new
    GridMultiSelectComboBoxRendererExt());
```

```
public class GridMultiSelectComboBoxRendererExt:
GridFilterRowMultiSelectRenderer
{
    public GridMultiSelectComboBoxRendererExt():base()
    {
        SupportsRenderOptimization = false;
        IsEditable = false;
    }
    protected override void
    OnWireEditUIElement(Syncfusion.Windows.Tools.Controls.ComboBoxAdv uiElement)
    {
        base.OnWireEditUIElement(uiElement);
        uiElement.PreviewMouseDown += uiElement_PreviewMouseDown;
    }
    void uiElement_PreviewMouseDown(object sender, MouseButtonEventArgs e)
    {
        if (this.HasCurrentCellState && this.CurrentCellRendererElement == sender)
            return;
        else
        {
            var position = e.GetPosition(DataGrid.GetVisualContainer());
            var rowColumnIndex =
            DataGrid.GetVisualContainer().PointToCellRowColumnIndex(position, false);
            DataGrid.SelectionController.MoveCurrentCell(rowColumnIndex, true);
        }
    }
    protected override void
    OnUnwireEditUIElement(Syncfusion.Windows.Tools.Controls.ComboBoxAdv
    uiElement)
    {
        base.OnUnwireEditUIElement(uiElement);
        uiElement.PreviewMouseDown -= uiElement_PreviewMouseDown;
    }
}
```



SfDataGrid Demo

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Bulk	Beverton	US
1002	ANATR	Oliver	Oregon	us
1003	ANTON	Brendon	Johanesberg	China
1004	YHGT	John	Chicago	UK
1005	BERGS	Charles	Spain	China
1006	TGVFD	Dintin	Britain	US
1007	YTREW	Friedo	Britain	US
1008	MNBGY	John	Oregon	Mumbai
1009	BGFDE	Sirert	Bulk	US
1010	BOTTM	Rakesh	hertion	Britain
1011	TGVFD	George	Oregon	US
1004	BERGS	Charles	Spain	China
1004	Bulk	John	Chicago	UK
1005	BERGS	Charles	Spain	China
1006	TGVFD	Dintin	Britain	US
1007	YTREW	Friedo	James	US
1008	MNBGY	John	Oregon	Bulk

You can get the sample from [here](#).

See Also

[How to load the symbols in FilterRow and perform actions based on that ?](#)

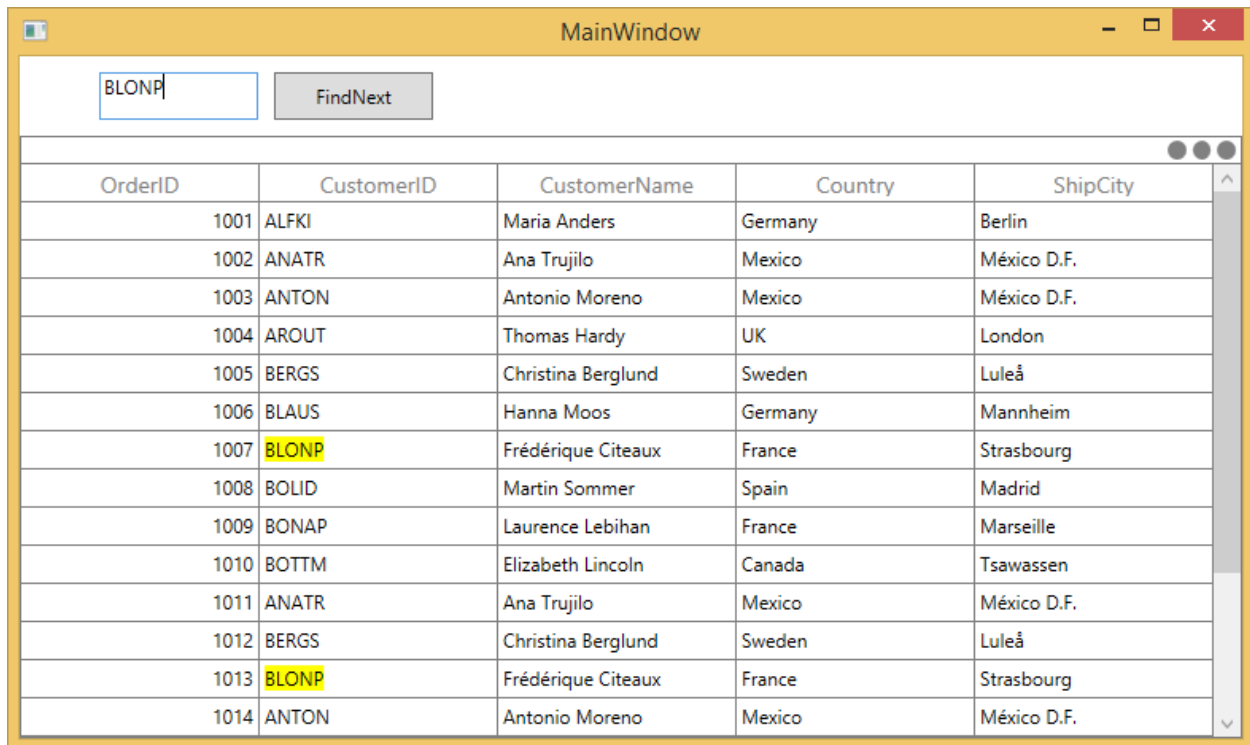
[How to show filter status message in SfDataGrid?](#)

Search in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) control allows you to search the data displayed in the SfDataGrid. You can search the data by using [SearchHelper.Search](#) method.

**C#**

```
this.dataGrid.SearchHelper.Search (TextBox.Text) ;
```



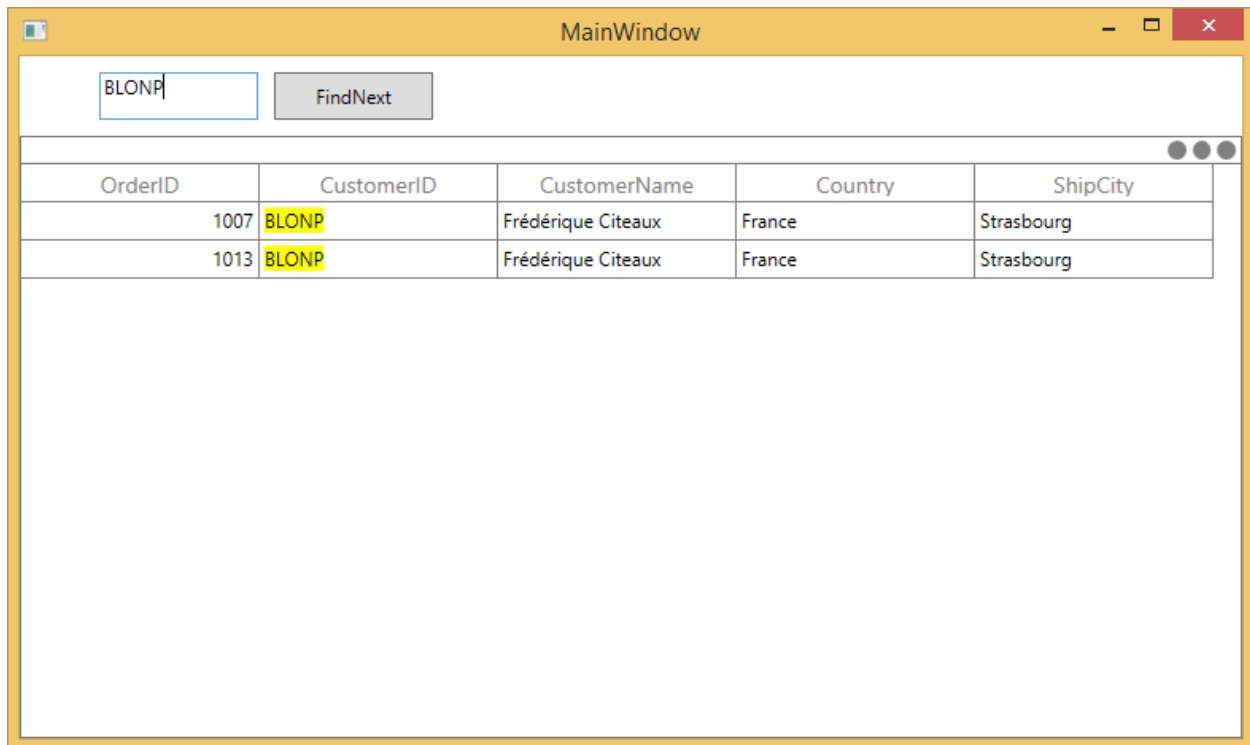
OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1011	ANATR	Ana Trujillo	Mexico	México D.F.
1012	BERGS	Christina Berglund	Sweden	Luleå
1013	BLONP	Frédérique Citeaux	France	Strasbourg
1014	ANTON	Antonio Moreno	Mexico	México D.F.

### Filtering

You can enable filter based on search by setting [SearchHelper.AllowFiltering](#) property to true.

### C#

```
this.dataGrid.SearchHelper.AllowFiltering = true;
this.dataGrid.SearchHelper.Search (TextBox.Text) ;
```



You can search the data with the case-sensitivity by setting [SearchHelper.AllowCaseSensitiveSearch](#) property.

#### C#

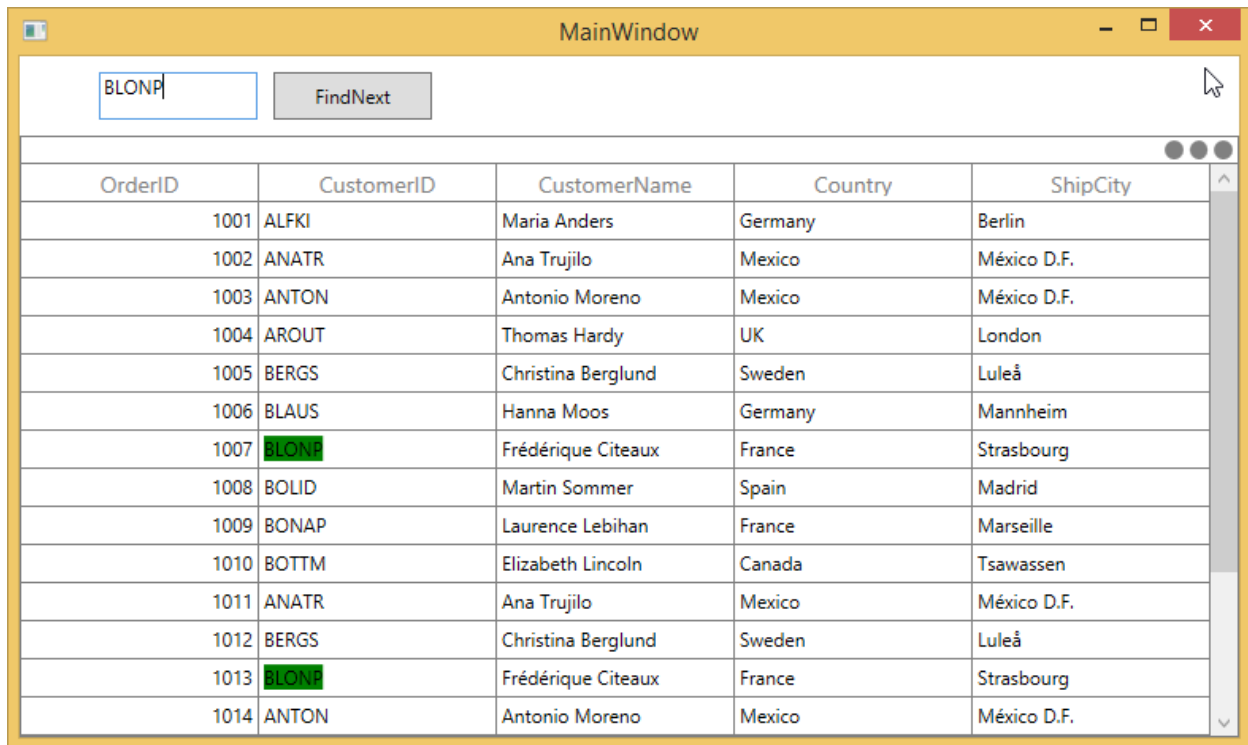
```
this.dataGrid.SearchHelper.AllowCaseSensitiveSearch = true;
```

#### *Changing Search Highlight Background*

In WPF DataGrid (SfDataGrid), you can change the search text highlighting color by setting [SearchHelper.SearchBrush](#) property.

#### C#

```
this.dataGrid.SearchHelper.SearchBrush = Brushes.Green;  
this.dataGrid.SearchHelper.Search(TextBox.Text);
```

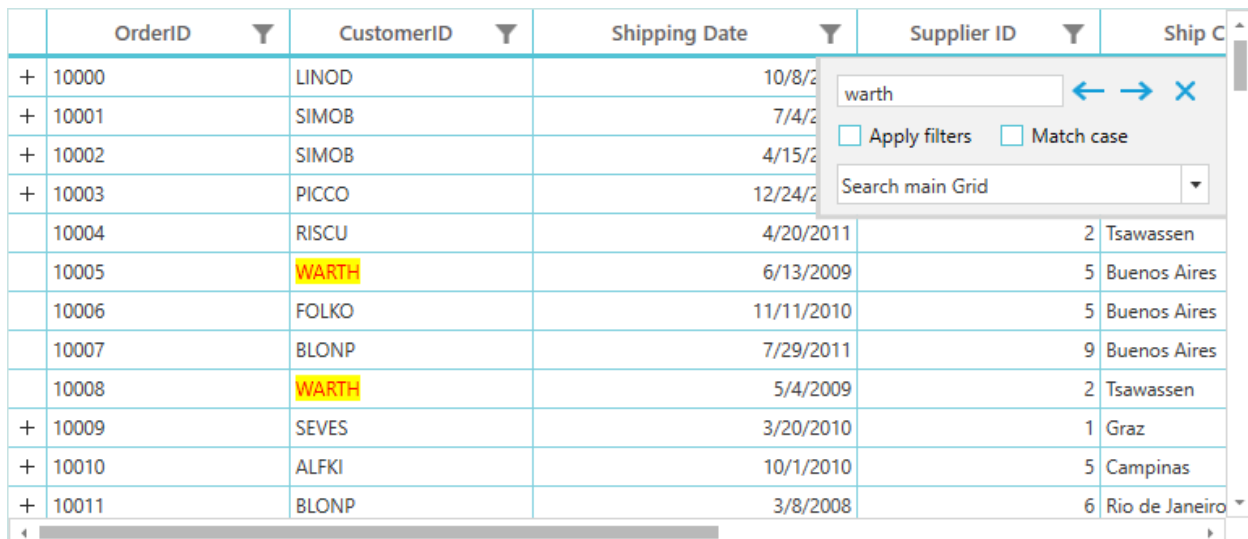


#### Changing foreground for search highlight

In WPF DataGrid (SfDataGrid), you can change the foreground color for search text by setting the [SearchHelper.SearchForegroundBrush](#) property.

#### C#

```
this.dataGrid.SearchHelper.SearchForegroundBrush = Brushes.Red;
```



#### Navigating cells based on search text

You can navigate to the cells contains the SearchText using [SearchHelper.FindNext](#) and [SearchHelper.FindPrevious](#) methods.

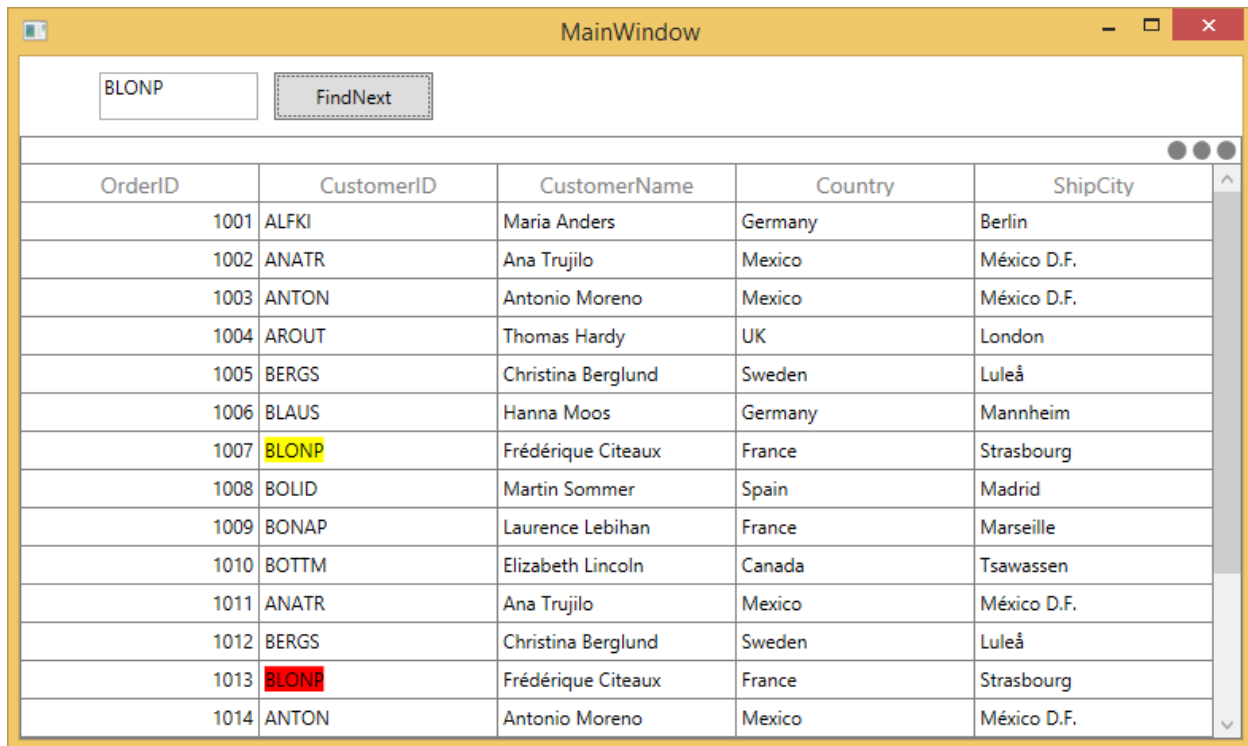
**C#**

```
this.dataGrid.SearchHelper.FindNext("SearchText");  
this.dataGrid.SearchHelper.FindPrevious("SearchText");
```

You can highlight the currently navigated search text using [SearchHelper.SearchHighlightBrush](#) property.

**C#**

```
this.dataGrid.SearchHelper.SearchHighlightBrush = Brushes.Red;  
this.dataGrid.SearchHelper.FindNext("TextBox.Text");
```



You can highlight the foreground color of current navigated search text by using the [SearchHelper.SearchForegroundHighlightBrush](#) property.

**C#**

```
this.dataGrid.SearchHelper.SearchForegroundHighlightBrush = Brushes.Red;
```

	OrderID	CustomerID	Shipping Date	Supplier ID	Ship C
+	10000	FOLIG	10/8/2		
+	10001	VAFFE	7/4/2		
+	10002	SIMOB	4/15/2		
+	10003	VAFFE	12/24/2		
	10004	FURIB	4/20/2011	6	Bruxelles
	10005	WELLI	6/13/2009	5	Buenos Aires
	10006	FOLKO	11/11/2010	8	Montréal
	10007	FOLKO	7/29/2011	1	Bruxelles
	10008	FOLIG	5/4/2009	8	Buenos Aires
+	10009	PICCO	3/20/2010	8	Resende
+	10010	RISCU	10/1/2010	3	Bruxelles
+	10011	FOLIG	3/8/2008	6	Buenos Aires

### Move CurrentCell when FindNext and FindPrevious

You can move the current cell along with FindNext and FindPrevious operation using [MoveCurrentCell](#) method in selection controller.

#### C#

```
this.dataGrid.SearchHelper.FindNext("BLONP");
this.dataGrid.SelectionController.MoveCurrentCell(this.dataGrid.SearchHelper.CurrentRowIndex);
```

### Clear Search

You can clear the search by calling the [SearchHelper.ClearSearch](#) method.

#### C#

```
this.dataGrid.SearchHelper.ClearSearch();
```

### Search operation on Master-Details View

Master-details view allows you to search the data by using [SearchHelper.Search](#) method in the [ViewDefinition.DataGrid](#).

#### C#

```
(this.datagrid.DetailsViewDefinition[0] as
GridViewDefinition).DataGrid.SearchHelper.Search(TextBox.Text);
```

	OrderID	CustomerID	Shipping Date	Supplier ID	Ship City	Ship Country
+	10000	LINOD	10/8/2011	7	Graz	Austria
-	10001	VAFEE	7/4/2008	5	Rio de Janeiro	Brazil
	OrderID	ProductID	Unit Price	Quantity	Discount	CustomerID
	10001	23	45	76	0	BLONP
	10001	45	67	23	5	FOLIG
	10001	45	42	16	3	ALFKI
	10001	23	95	15	2	RISCU
-	10002	LINOD	4/15/2009	5	Bruxelles	Belgium
	OrderID	ProductID	Unit Price	Quantity	Discount	CustomerID
	10002	7	70	6	4	FRANS
	10002	2	30	5	2	RISCU
+	10003	MEREP	12/24/2008	2	Bruxelles	Belgium
	10004	PICCO	4/20/2011	8	Campinas	Brazil

#### Navigating cells based on search text in DetailsViewDataGrid

You can navigate to the cells contains the SearchText using [SearchHelper.FindNext](#) and [SearchHelper.FindPrevious](#) methods in [ViewDefinition.DataGrid](#).

#### C#

```
(this.datagrid.DetailsViewDefinition[0] as
GridViewDefinition).DataGrid.SearchHelper.FindNext(SearchBox.Text);
(this.datagrid.DetailsViewDefinition[0] as
GridViewDefinition).DataGrid.SearchHelper.FindPrevious(SearchBox.Text);
```

You can get the sample from [here](#).

**Note:** It is not possible to Navigate with the two DataGrid at a time.

#### Search customization

WPF DataGrid (SfDataGrid) process the search operations in [SearchHelper](#) class. You can change the default search behaviors by overriding [SearchHelper](#) class and set to [SfDataGrid.SearchHelper](#).

#### C#

```
this.datagrid.SearchHelper = new SearchHelperExt(this.datagrid);
public class SearchHelperExt : SearchHelper
{
    public SearchHelperExt(SfDataGrid datagrid)
    : base(datagrid)
    {
    }
}
```

#### Search only selected columns

You can search only selected columns by overriding [SearchCell](#) method of [SearchHelper](#). In the [SearchCell](#) method, based on [MappingName](#) you can skip the columns that you don't want to search.

In the below code, except [Quantity](#) column other columns are gets excluded from search.

**C#**

```

this.sfgrid.SearchHelper = new SearchHelperExt(this.sfgrid);
this.sfgrid.SearchHelper.Search("5");
public class SearchHelperExt : SearchHelper
{
    public SearchHelperExt(SfDataGrid datagrid)
    : base(datagrid)
    {
    }
    protected override bool SearchCell(DataColumnBase column, object record,
    bool ApplySearchHighlightBrush)
    {
        if (column.GridColumn.MappingName == "Quantity")
        return base.SearchCell(column, record, ApplySearchHighlightBrush);
        return false;
    }
}

```

Order ID	Customer ID	Product Name	Order Date	Quantity	Unit Price
10000	FRANS	Alice Mutton	5/10/1991	4	27.00
10001	MEREP	NuNuCa Nuß-Nougat-Crem	5/13/1991	30	9.80
10001	MEREP	Boston Crab Meat	5/13/1991	40	12.80
10001	MEREP	Raclette Courdavault	5/13/1991	8	38.50
10001	MEREP	Wimmers gute Semmelknöl	5/13/1991	15	23.00
10002	FOLKO	Gorgonzola Telino	5/14/1991	35	8.00
10002	FOLKO	Chartreuse verte	5/14/1991	18	12.60
10002	FOLKO	Fløtemysost	5/14/1991	15	15.00
10003	SIMOB	Carnarvon Tigers	5/15/1991	12	43.70
10004	VAFFE	Thüringer Rostbratwurst	5/16/1991	35	86.00
10004	VAFFE	Veggie-spread	5/16/1991	6	30.70
10005	WARTH	Tarte au sucre	5/20/1991	6	34.00
10006	FRANS	Konbu	5/21/1991	10	4.20
10006	FRANS	Valkoinen suklaa	5/21/1991	4	11.30
10007	MORGK	Queso Manchego La Pastor	5/22/1991	4	26.00
10007	MORGK	Perth Pasties	5/22/1991	30	22.90
10007	MORGK	Veggie-spread	5/22/1991	20	30.70

Select the record based on the SearchText

You can select the records which contains the search text by using `GetSearchedRecord` method.

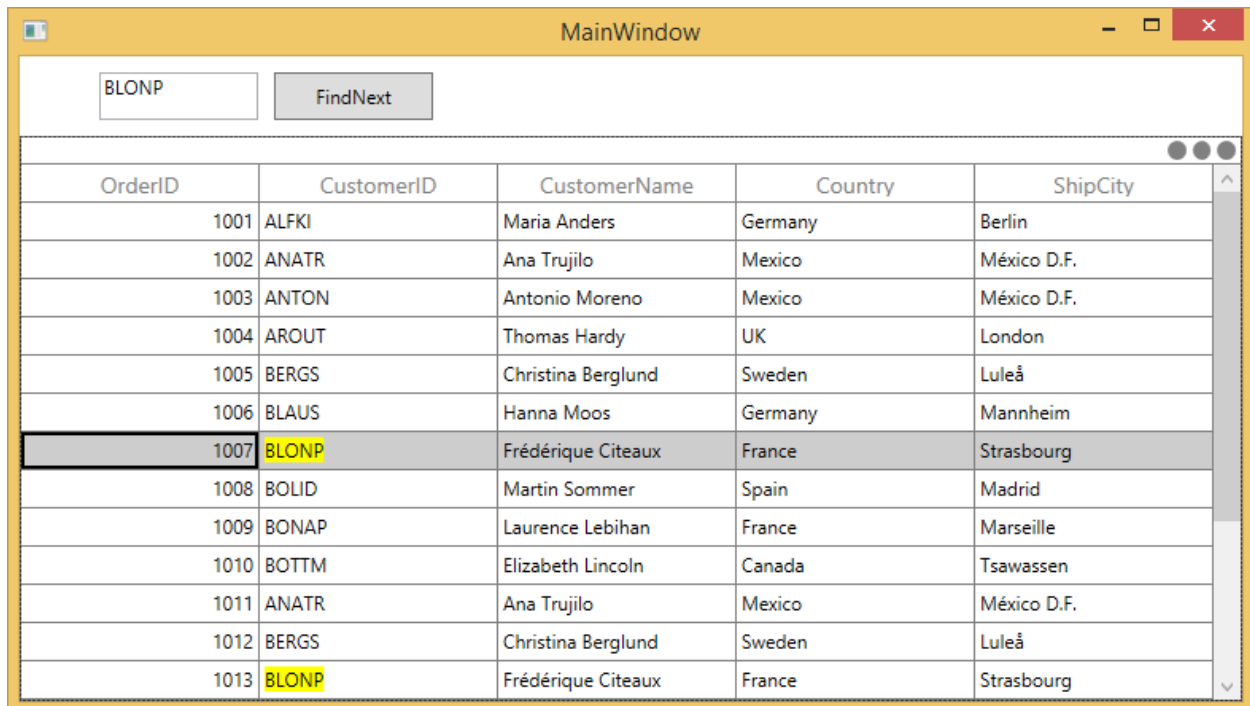
**C#**

```

this.dataGrid.SelectedItems.Clear();
this.dataGrid.SearchHelper.Search("SearchText");
var list = this.dataGrid.SearchHelper.GetSearchRecords();
int recordIndex =
this.dataGrid.ResolveToRecordIndex(this.dataGrid.ResolveToRowIndex(list[0].Record));
this.dataGrid.SelectedIndex = recordIndex;

```





#### Search with the GridComboBoxColumn

You can search the data in SfDataGrid with all the GridColumns which loads TextBlock as display element. To perform the search operation in the [GridComboBoxColumn](#) you need to customize the GridComboBoxColumn. As it loads the ContentControl in display mode.

#### C#

```
public class ComboBoxColumnExt : GridComboBoxColumn
{
    public ComboBoxColumnExt()
    {
        SetCellType("ComboBoxExt");
    }
    protected override Freezable CreateInstanceCore()
    {
        return new ComboBoxColumnExt();
    }
}
```

You can change the display element of each column by creating new renderer for the particular column and assign to corresponding cell type in SfDataGrid.CellRenderers.

#### C#

```
public class ComboBoxRendererExt : GridVirtualizingCellRenderer<TextBlock,
    ComboBox>
{
    public ComboBoxRendererExt()
    {
    }
}
```

```
public override object GetControlValue()
{
    if (!HasCurrentCellState)
        return null;
    return CurrentCellRendererElement.GetValue(IsInEditing ?
        Selector.SelectedValueProperty : TextBlock.TextProperty);
}
// Creates the binding to the Edit-element
private void InitializeEditBinding(ComboBox uiElement, GridColumn column)
{
    var comboBoxColumn = (GridColumn)column;
    var source = comboBoxColumn.ValueBinding as Binding;
    // Creates the bind element to the edit-element.
    var bind = new Binding
    {
        Mode = BindingMode.TwoWay,
        Path = source.Path,
        StringFormat = source.StringFormat,
        TargetNullValue = source.TargetNullValue,
        UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
        ValidatesOnExceptions = source.ValidatesOnExceptions,
        AsyncState = source.AsyncState,
        BindingGroupName = source.BindingGroupName,
        IsAsync = source.IsAsync,
        NotifyOnSourceUpdated = source.NotifyOnSourceUpdated,
        NotifyOnTargetUpdated = source.NotifyOnTargetUpdated,
        UpdateSourceExceptionFilter = source.UpdateSourceExceptionFilter,
        XPath = source.XPath
    };
    uiElement.SetBinding(ComboBox.SelectedValueProperty, bind);
    // Binding the ItemSource to the GridComboBox.
    var itemsSourceBinding = new Binding { Path = new
        PropertyPath("ItemsSource"), Mode = BindingMode.TwoWay, Source =
        comboBoxColumn };
    uiElement.SetBinding(ComboBox.ItemsSourceProperty, itemsSourceBinding);
    var displayMemberBinding = new Binding { Path = new
        PropertyPath("DisplayMemberPath"), Mode = BindingMode.TwoWay, Source =
        comboBoxColumn };
    uiElement.SetBinding(ComboBox.DisplayMemberPathProperty,
        displayMemberBinding);
    var selectedValuePathBinding = new Binding { Path = new
        PropertyPath("SelectedValuePath"), Mode = BindingMode.TwoWay, Source =
        comboBoxColumn };
    uiElement.SetBinding(ComboBox.SelectedValuePathProperty,
        selectedValuePathBinding);
    var staysOpenOnEditBinding = new Binding { Path = new
        PropertyPath("StaysOpenOnEdit"), Mode = BindingMode.TwoWay, Source =
        comboBoxColumn };
    uiElement.SetBinding(ComboBox.StaysOpenOnEditProperty,
        staysOpenOnEditBinding);
    var isEditableBinding = new Binding { Path = new PropertyPath("IsEditable"),
        Mode = BindingMode.TwoWay, Source = comboBoxColumn };
    uiElement.SetBinding(ComboBox.IsEditableProperty, isEditableBinding);
    var itemTemplateBinding = new Binding { Path = new
        PropertyPath("ItemTemplate"), Mode = BindingMode.TwoWay, Source =
        comboBoxColumn };
    uiElement.SetBinding(ComboBox.ItemTemplateProperty, itemTemplateBinding);
}
```

```

}
// This will be invoke when the editing is triggered.
public override void OnInitializeEditElement(DataColumnBase dataColumn,
ComboBox uiElement, object dataContext)
{
GridColumn column = dataColumn.GridColumn;
InitializeEditBinding(uiElement, column);
var textAlignment = new Binding { Path = new PropertyPath("TextAlignment"),
Mode = BindingMode.OneWay, Source = column, Converter = new
TextAlignmentToHorizontalAlignmentConverter() };
uiElement.SetBinding(Control.HorizontalContentAlignmentProperty,
textAlignment);
var verticalAlignment = new Binding { Path = new
PropertyPath("VerticalAlignment"), Mode = BindingMode.TwoWay, Source =
column };
uiElement.SetBinding(Control.VerticalContentAlignmentProperty,
verticalAlignment);
}
// Display Element initialized for required properties
public override void
OnInitializeDisplayElement(Syncfusion.UI.Xaml.Grid.DataColumnBase
dataColumn, TextBlock uiElement, object dataContext)
{
var column = dataColumn.GridColumn;
var gridColumn = column as GridComboBoxColumn;
uiElement.SetBinding(TextBlock.TextProperty, column.DisplayBinding);
var textAlignment = new Binding { Path = new PropertyPath("TextAlignment"),
Mode = BindingMode.OneWay, Source = column, Converter = new
TextAlignmentToHorizontalAlignmentConverter() };
uiElement.SetBinding(Control.HorizontalContentAlignmentProperty, textAlignment);
var verticalAlignment = new Binding { Path = new
PropertyPath("VerticalAlignment"), Mode = BindingMode.TwoWay, Source =
column };
uiElement.SetBinding(Control.VerticalAlignmentProperty, verticalAlignment);
uiElement.Margin = new Thickness(3,0,1,0);
}
// This Display Binding has to be set with the TextBlock.TextProperty.
private static void SetDisplayBinding(TextBlock element, GridColumn column,
object dataContext)
{
var customColumn = (ComboBoxColumnExt)column;
var binding = new Binding
{
Path = new PropertyPath(customColumn.MappingName),
Mode = BindingMode.TwoWay,
UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
};
element.SetBinding(TextBlock.TextProperty, binding);
}
}

```

See Also

[How to apply search and filter for one column in SfDataGrid?](#)

[How to filter the records with searching when underlying items source is DataTable in SfDataGrid?](#)

[How to perform incremental search ?](#)

## Editing in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) provides support for editing and it can be enabled or disabled by setting [SfDataGrid.AllowEditing](#) property.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowEditing="True"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}" />
```

**C#**

```
dataGrid.AllowEditing = true;
```

You can enable or disable editing for particular column by setting [GridColumn.AllowEditing](#) property.

**XML**

```
<syncfusion:GridTextColumn AllowEditing="True" MappingName="OrderID" />
```

**C#**

```
dataGrid.Columns["OrderID"].AllowEditing = true;
```

**Note:** [GridColumn.AllowEditing](#) takes higher priority than [SfDataGrid.AllowEditing](#).



OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

**Note:** It is mandatory to set the [NavigationMode](#) to Cell to enable [CurrentCell](#) navigation and editing.

## Edit mode

You can enter into edit mode by pressing F2 key or clicking (touch also supported) the cell. You can allow users to edit the cell in single click (OnTap) or double click (OnDoubleTap) by setting by [EditTrigger](#) property.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowEditing="True"
    EditTrigger="OnTap"
    ItemsSource="{Binding Orders}" />
```

### C#

```
dataGrid.EditTrigger = EditTrigger.OnTap;
```

### Edit cursor placement

When the cell enters into edit mode, cursor is placed based on [EditorSelectionBehavior](#) property.

- **SelectAll** – selects the text of edit element loaded inside cell.
- **MoveLast** – places the cursor at the last of edit element loaded inside cell.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowEditing="True"
    EditorSelectionBehavior="SelectAll"
    ItemsSource="{Binding Orders}" />
```

### C#

```
dataGrid.EditorSelectionBehavior = EditorSelectionBehavior.SelectAll;
```

### Retain editing on lost focus

The editing of current cell will be ended by default while the focus is moving from DataGrid to another control. You can set the [LostFocusBehavior](#) property to `LostFocusBehavior.Default` if you want to retain the editing of the current cell even when focus is moved to another control.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowEditing="True"
    LostFocusBehavior="Default"
    ItemsSource="{Binding Orders}" />
```

### C#

```
dataGrid.LostFocusBehavior = LostFocusBehavior.Default;
```

### Working with IEditableObject interface

WPF DataGrid (SfDataGrid) supports to commit and roll back the changes in row level when underlying data object implements [IEditableObject](#) interface.

The editing changes in a row will be committed only when user move to next row or pressing enter key in [EndEdit](#). Also when user press Esc key, then the changes made in a row will be reverted in [CancelEdit](#).

**IEditableObject** has the following methods to capture editing,

- [BeginEdit](#) – Gets called to begin edit on underlying data object when cell in a row get into edit mode.
- [CancelEdit](#) – Gets called when user press the `Esc` key to discard the changes in a row since last `BeginEdit` call.
- [EndEdit](#) – Gets called when user move to the next row or press `Enter` key to commit changes in underlying data object since last `BeginEdit` call.

In the below code snippet explains the simple implementation of **IEditableObject**.

**C#**

```
public class Employee : NotificationObject, IEditableObject
{
    private string _Name;
    private int _ContactID;
    private string _Title;
    private DateTime _BirthDate;
    private string _Gender;
    private double _SickLeaveHours;
    private double _Salary;
    protected Dictionary<string, object> BackUp()
    {
        var dict = new Dictionary<string, object>();
        var itemProperties = this.GetType().GetTypeInfo().DeclaredProperties;
        foreach (var pDescriptor in itemProperties)
        {
            if (pDescriptor.CanWrite)
                dict.Add(pDescriptor.Name, pDescriptor.GetValue(this));
        }
        return dict;
    }
    public string Name
    {
        get { return this._Name; }
        set
        {
            this._Name = value;
            this.RaisePropertyChanged("Name");
        }
    }
    public string Title
    {
        get { return this._Title; }
        set
        {
            this._Title = value;
            this.RaisePropertyChanged("Title");
        }
    }
    public int ContactID
    {
        get { return this._ContactID; }
        set
```

```
{
    this._ContactID = value;
    this.RaisePropertyChanged("ContactID");
}
}
public DateTime BirthDate
{
    get { return this._BirthDate; }
    set
    {
        this._BirthDate = value;
        this.RaisePropertyChanged("BirthDate");
    }
}
public string Gender
{
    get { return this._Gender; }
    set
    {
        this._Gender = value;
        this.RaisePropertyChanged("Gender");
    }
}
public double SickLeaveHours
{
    get { return this._SickLeaveHours; }
    set
    {
        this._SickLeaveHours = value;
        this.RaisePropertyChanged("SickLeaveHours");
    }
}
public double Salary
{
    get { return this._Salary; }
    set
    {
        this._Salary = value;
        this.RaisePropertyChanged("Salary");
    }
}
private int _EmployeeID;
public int EmployeeID
{
    get { return this._EmployeeID; }
    set
    {
        this._EmployeeID = value;
        this.RaisePropertyChanged("EmployeeID");
    }
}
private Dictionary<string, object> storedValues;
public void BeginEdit()
{
    this.storedValues = this.BackUp();
}
public void CancelEdit()
```

```

{
    if (this.storedValues == null)
        return;
    foreach (var item in this.storedValues)
    {
        var itemProperties = this.GetType().GetTypeInfo().DeclaredProperties;
        var pDesc = itemProperties.FirstOrDefault(p => p.Name == item.Key);
        if (pDesc != null)
            pDesc.SetValue(this, item.Value);
    }
}

public void EndEdit()
{
    if (this.storedValues != null)
    {
        this.storedValues.Clear();
        this.storedValues = null;
    }
    Debug.WriteLine("End Edit Called");
}
}

public class NotificationObject : INotifyPropertyChanged
{
    public void RaisePropertyChanged(string propName)
    {
        if (this.PropertyChanged != null)
            this.PropertyChanged(this, new PropertyChangedEventArgs(propName));
    }
    public event PropertyChangedEventHandler PropertyChanged;
}

```

### Edit events

SfDataGrid triggers the following events during editing.

#### *CurrentCellBeginEdit Event*

[CurrentCellBeginEdit](#) event occurs when the [CurrentCell](#) enter into edit mode.

[CurrentCellBeginEditEventArgs](#) has following members which provides information for [CurrentCellBeginEdit](#) event.

- [Cancel](#) : When set to `true`, the event is canceled and the [CurrentCell](#) does not enter into the edit mode.
- [RowColumnIndex](#) : Gets the current row column index of the DataGrid.
- [Column](#) : Gets the Grid Column of the SfDataGrid.

### C#

```

this.dataGrid.CurrentCellBeginEdit += dataGrid_CurrentCellBeginEdit;
void dataGrid_CurrentCellBeginEdit(object sender,
Syncfusion.UI.Xaml.Grid.CurrentCellBeginEditEventArgs args)
{
}

```



*CurrentCellEndEdit Event*

[CurrentCellEndEdit](#) event occurs when the [CurrentCell](#) exits the edit mode. [CurrentCellEndEditEventArgs](#) has following members which provides information for [CurrentCellEndEdit](#) event.

- [RowColumnIndex](#) : Gets the value for the current row column index.

**C#**

```
this.dataGrid.CurrentCellEndEdit += dataGrid_CurrentCellEndEdit;  
void dataGrid_CurrentCellEndEdit(object sender,  
Syncfusion.UI.Xaml.Grid.CurrentCellEndEditEventArgs args)  
{  
}
```

*CurrentCellValueChanged Event*

[CurrentCellValueChanged](#) event occurs whenever a value changes in GridColumn's that supports editing. [CurrentCellValueChangedEventArgs](#) has following members which provides information for [CurrentCellValueChanged](#) event.

- [Column](#) : Gets the Grid Column of the SfDataGrid.
- [RowColumnIndex](#) : Gets the value of the current RowColumnIndex.

**C#**

```
this.dataGrid.CurrentCellValueChanged += dataGrid_CurrentCellValueChanged;  
void dataGrid_CurrentCellValueChanged(object sender,  
Syncfusion.UI.Xaml.Grid.CurrentCellValueChangedEventArgs args)  
{  
}
```

---

**Note:** [GridComboBoxColumn](#) and [GridMultiColumnDropDownList](#), you have to use the [CurrentCellDropDownSelectionChanged](#) event.

---

*Combobox column selectionchanged event*

[CurrentCellDropDownSelectionChanged](#) event occurs whenever the [SelectedItem](#) of [GridMultiColumnDropDownList](#) and [GridComboBoxColumn](#) column changed.

[CurrentCellDropDownSelectionChangedEventArgs](#) has following members which provides information for [CurrentCellDropDownSelectionChanged](#) event.

- [RowColumnIndex](#) – Gets the RowColumnIndex of the corresponding item that were selected from the drop-down control.
- [SelectedIndex](#) – Gets the index of the corresponding item that were selected from the drop-down control.
- [SelectedItem](#) – Gets the data item that were selected from the drop-down control.

**C#**

```
this.dataGrid.CurrentCellDropDownSelectionChanged +=  
dataGrid_CurrentCellDropDownSelectionChanged;
```

```
void dataGrid_CurrentCellDropDownSelectionChanged(object sender,
CurrentCellDropDownSelectionChangedEventArgs args)
{
}
```

Programmatically edit the cell

#### *BeginEdit*

WPF DataGrid (SfDataGrid) allows you to edit the cell programmatically by calling the [BeginEdit](#) method. Initially the [CurrentCell](#) need to set before calling the [BeginEdit](#) method when the CurrentCell value is null.

#### **C#**

```
//Add this namespace to access the RowColumnIndex structure type in
SfDataGrid
using Syncfusion.UI.Xaml.ScrollAxis;
this.dataGrid.Loaded += dataGrid_Loaded;
void dataGrid_Loaded(object sender, RoutedEventArgs e)
{
    RowColumnIndex rowColumnIndex = new RowColumnIndex(3, 2);
    this.dataGrid.MoveCurrentCell(rowColumnIndex);
    this.dataGrid.SelectionController.CurrentCellManager.BeginEdit();
}
```

#### *EndEdit*

You can call the [EndEdit](#) method to programmatically end edit.

#### **C#**

```
//Add this namespace to access the RowColumnIndex structure type in
SfDataGrid
using Syncfusion.UI.Xaml.ScrollAxis;
this.dataGrid.Loaded += dataGrid_Loaded;
void dataGrid_Loaded(object sender, RoutedEventArgs e)
{
    RowColumnIndex rowColumnIndex = new RowColumnIndex(3, 2);
    this.dataGrid.MoveCurrentCell(rowColumnIndex);
    this.dataGrid.SelectionController.CurrentCellManager.EndEdit();
}
```

#### *CancelEdit*

You can use the [CurrentCellBeginEdit](#) event to cancel the editing operation for the corresponding cell.

#### **C#**

```
//Add this namespace to access the RowColumnIndex structure type in
SfDataGrid
using Syncfusion.UI.Xaml.ScrollAxis;
this.dataGrid.CurrentCellBeginEdit += dataGrid_CurrentCellBeginEdit;
void dataGrid_CurrentCellBeginEdit(object sender,
Syncfusion.UI.Xaml.Grid.CurrentCellBeginEditEventArgs args)
{
    var recordIndex =
    this.dataGrid.ResolveToRecordIndex(args.RowColumnIndex.RowIndex);
}
```

```

var columnIndex =
this.dataGrid.ResolveToGridVisibleColumnIndex(args.RowColumnIndex.ColumnIndex);
var mappingName = this.dataGrid.Columns[columnIndex].MappingName;
var record = this.dataGrid.View.Records.GetItemAt(recordIndex);
var cellValue =
this.dataGrid.View.GetPropertyAccessProvider().GetValue(record,
mappingName);
if (args.RowColumnIndex == new RowColumnIndex(3, 2))
args.Cancel = true;
}

```

### Cell click events

WPF DataGrid provides **CellTapped** and **CellDoubleTapped** events to handle cell click actions.

#### Cell tapped event

WPF DataGrid **CellTapped** event occurs when the user clicks or touches a cell in DataGrid with [GridCellTappedEventArgs](#). **CellTapped** event does not occur for the non-selectable cells. The **GridCellTappedEventArgs** has following members which provides information for **CellTapped** event.

- [Column](#) - Gets the GridColumn of the tapped cell.
- [Record](#) - Gets the data context of the tapped cell.
- [RowColumnIndex](#) - Gets the RowColumnIndex of the tapped cell.
- [ChangedButton](#) - Get the MouseButton associated with the event.
- [OriginalSender](#) - Gets the original reporting source that raised the event.

### XML

```

<Syncfusion:SfDataGrid x:Name="dataGrid"
CellTapped="datagrid_CellTapped"
ItemsSource="{Binding OrderInfoCollection }">
</Syncfusion:SfDataGrid>

```

### C#

```

this.dataGrid.CellTapped += Datagrid_CellTapped;
private void Datagrid_CellTapped(object sender, GridCellTappedEventArgs e)
{
    //You can do your own logic here.
}

```

#### Cell double tapped event

**CellDoubleTapped** event occurs when the user double clicks or double taps a cell in DataGrid with [GridCellDoubleTappedEventArgs](#). **CellDoubleTapped** event does not occur for non-selectable cells. **GridCellDoubleTappedEventArgs** has following members which provides information for **CellDoubleTapped** event.

- [Column](#) - Gets the GridColumn of the double tapped cell.
- [Record](#) - Gets the data context of the double tapped cell.
- [RowColumnIndex](#) - Gets the RowColumnIndex of the double tapped cell.

- [ChangedButton](#) - Gets the MouseButton associated with the event.
- [OriginalSender](#) - Gets the original reporting source that raised the event.

**XML**

```
<Syncfusion:SfDataGrid x:Name="dataGrid"
CellDoubleTapped="datagrid_CellDoubleTapped"
ItemsSource="{Binding OrderInfoCollection }">
</Syncfusion:SfDataGrid>
```

**C#**

```
this.dataGrid.CellDoubleTapped += Datagrid_CellDoubleTapped;
private void Datagrid_CellDoubleTapped(object sender,
GridCellDoubleTappedEventArgs e)
{
//you can do your own logic here.
}
```

**Mouse and Keyboard operations for UIElement inside Template**

You can directly load edit element using `GridTemplateColumn.CellTemplate` property. In this case, you can provide focus and control (keyboard and mouse) to the UIElement inside CellTemplate in the below ways,

*Providing focus to the control inside the Template*

You can focus to the particular UIElement loaded inside template when cell gets activated by setting [FocusedManager.FocusedElement](#) attached property.

**XML**

```
<syncfusion:GridTemplateColumn HeaderText="Customer ID"
MappingName="CustomerID" >
<syncfusion:GridTemplateColumn.CellTemplate>
<DataTemplate>
<TextBlock syncfusion:FocusManagerHelper.FocusedElement="True"
FontStyle="Italic"
FontWeight="SemiBold"
Padding="2,0"
Text="{Binding CustomerID}" />
</DataTemplate>
</syncfusion:GridTemplateColumn.CellTemplate>
</syncfusion:GridTemplateColumn>
```

*Providing keyboard control to UIElement inside CellTemplate*

You can allow UIElement loaded inside CellTemplate to handle keyboard interaction by setting [FocusManagerHelper.WantsKeyInput](#) attached property to `GridColumn`.

**XML**

```
<syncfusion:GridTemplateColumn MappingName="ProductId"
syncfusion:FocusManagerHelper.WantsKeyInput="True">
<syncfusion:GridTemplateColumn.CellTemplate>
<DataTemplate>
<Grid>
```

```
<TextBox x:Name="text" Text="{Binding ProductId}"/>
</Grid>
</DataTemplate>
</syncfusion:GridTemplateColumn.CellTemplate>
</syncfusion:GridTemplateColumn>
```

**Note:** Enter and Tab keys are always handled by SfDataGrid only.

*Providing mouse control to UIElement inside Template*

You can allow UIElement loaded inside template to handle mouse interaction in required cases by setting [VisualContainer.WantsMouseInput](#) attached property to GridColumn.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTemplateColumn MappingName="ProductName">
<syncfusion:GridTemplateColumn.CellTemplate>
<DataTemplate>
<ComboBox ItemsSource="{Binding ComboItems, Source={StaticResource
viewModel}}">
syncfusion:VisualContainer.WantsMouseInput="True" />
</DataTemplate>
</syncfusion:GridTemplateColumn.CellTemplate>
</syncfusion:GridTemplateColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

How to

*How to track edited cells in WPF DataGrid?*

You can change the foreground color of edited cells through the [CellStyleSelector](#) to track edited cells.

Please follow the below steps to highlight the edited cells.

1. Add new property EditedColumns in data object to maintain edited column MappingName.
2. Add the MappingName of the column to EditedColumns, in CurrentCellValueChanged event to keep track of edited columns in data object.

#### C#

```
this.dataGrid.CurrentCellValueChanged+=dataGrid_CurrentCellValueChanged;
private void dataGrid_CurrentCellValueChanged(object sender,
CurrentCellValueChangedEventArgs args)
{
if (!(args.Record as
OrderInfo).EditedColumns.Contains(args.Column.MappingName))
(args.Record as OrderInfo).EditedColumns.Add(args.Column.MappingName);
//updates the current row index
this.dataGrid.UpdateDataRow(args.RowColumnIndex.RowIndex);
}
```

3. Create a style of TargetType `GridCell` and change the Foreground using `CellStyleSelector` based on `EditedColumns` property in data object.

#### XML

```
<Application.Resources>
<Style x:Key="cellStyle" TargetType="syncfusion:GridCell">
<Setter Property="Foreground" Value="DarkOrange" />
</Style>
</Application.Resources>
<Window.Resources>
<local:CellStyleSelector x:Key="cellStyleSelector" />
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
CellStyleSelector="{StaticResource cellStyleSelector}"
AllowEditing="True"
ItemsSource="{Binding Path=OrdersDetails}"
ShowRowHeader="True">
</syncfusion:SfDataGrid>
```

#### C#

```
public class CellStyleSelector : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var gridCell = container as GridCell;
        if (gridCell.ColumnBase == null || gridCell.ColumnBase.GridColumn == null)
            base.SelectStyle(item, container);
        var record = item as OrderInfo;
        if
            (record.EditedColumns.Contains(gridCell.ColumnBase.GridColumn.MappingName))
            return App.Current.Resources["cellStyle"] as Style;
        return base.SelectStyle(item, container);
    }
}
```

Customer ID	Shipping Date	Employee ID	Ship City	Ship Country	Freight
SIMOB	10/8/2011	9	Resende	Brazil	380.42
WELLI	7/4/2008	4	Bruxelles	Belgium	649.96
RISCU	4/15/2009	9	Buenos Aires	Argentina	221.67
SEVES	12/24/2008	2	Rio de Janeiro	Canada	220.87
FURIB	4/20/2011	3	Bruxelles	Belgium	231.21
> WARTH	6/16/2009	2	Bruxelles	Belgium	646.09
WARTH	11/11/2010	9	Bruxelles	Belgium	189.04
SIMOB	7/29/2011	1	Montréal	Canada	845.05
ALFKI	5/4/2009	2	Resende	Brazil	876.74
WARTH	3/20/2010	4	Rio de Janeiro	Brazil	112.85
FOLKO	10/1/2010	9	Buenos Aires	Argentina	482.2
FURIB	3/8/2008	6	Montréal	Canada	238.62
SEVES	3/10/2008	6	Tsawassen	Canada	688.33
RISCU	8/30/2008	4	Rio de Janeiro	Brazil	191.62
SEVES	6/21/2010	6	Rio de Janeiro	Brazil	95.89
FURIB	5/20/2008	5	Graz	Austria	59.03
VAFFE	10/9/2011	1	Resende	Brazil	461.99
FURIB	5/19/2011	4	Bruxelles	Belgium	840.94
FRANS	10/22/2008	1	Bruxelles	Belgium	817.32

Allow editing when pressing minus key

SfDataGrid does not allow the cell to get into the edit mode while pressing the Minus key or any special character. You can overcome this behavior by customizing the SfDataGrid class, and overriding its `OnTextInput()` method.

**C#**

```
public class SfDataGridExt : SfDataGrid
{
    public SfDataGridExt()
    : base()
    {
    }
    protected override void OnTextInput(TextCompositionEventArgs e)
    {
        if (!SelectionController.CurrentCellManager.HasCurrentCell)
        {
            base.OnTextInput(e);
            return;
        }
        //Get the Current Row and Column index from the CurrentCellManager
        var rowColumnIndex =
            SelectionController.CurrentCellManager.CurrentRowColumnIndex;
        RowGenerator rowGenerator = this.RowGenerator;
        //Get the row from the Row index
        var dataRow = rowGenerator.Items.FirstOrDefault(item => item.RowIndex ==
            rowColumnIndex.RowIndex);
        //Check whether the dataRow is null or not and the type as DataRow
        if (dataRow != null && dataRow is DataRow)
        {
            //Get the column from the VisibleColumn collection based on the column index
            var dataColumn = dataRow.VisibleColumns.FirstOrDefault(column =>
                column.ColumnIndex == rowColumnIndex.ColumnIndex);
```

```
//Convert the input text to char type
char text;
char.TryParse(e.Text, out text);
//Skip if the column is GridTemplateColumn and the column is not already in
editing
//Allow Editing only pressed letters digits and Minus sign key
if (dataColumn != null && !(dataColumn.GridColumn is GridTemplateColumn) &&
!dataColumn.IsEditing && SelectionController.CurrentCellManager.BeginEdit()
&& (e.Text.Equals("-") || char.IsLetterOrDigit(text)))
dataColumn.Renderer.PreviewTextInput(e);
}
base.OnTextInput(e);
}
}
```

See Also

[How to change row background based on RowState.Modified when underlying itemsSource is DataTable in SfDataGrid?](#)

[How to show different controls in same column of SfDataGrid?](#)

[How to edit GridHyperLinkColumn?](#)

[How to use the editing related events in GridCheckBoxColumn?](#)

[How to skip editing for Read-Only columns in AddNewRow?](#)

[How to change the cell value of selectedcells when end edit?](#)

[How to show the Number Keyboard when editing GridNumericColumn?](#)

[How to validate the AddNewRow value based on already existing records?](#)

[How to change the CheckBox value for all SelectedItems when any selected CheckBox value changed?](#)

[How to fire RowValidating event for GridCheckBoxColumn in SfDataGrid](#)

[How to create ReadOnly UnboundRows?](#)

[How to load null value to the GridDateTimeColumn when AllowInlineEditing is set to true?](#)

[How to move the CurrentCell to the first column of the AddNewRow when the Tab key is pressed from the last column and its position is at the Bottom of the SfDataGrid?](#)

[How to customize edit mode behavior of GridCell in SfDataGrid?](#)

[How to change the Enter key behavior in SfDataGrid?](#)

[How to change the Enter key behavior to insert line break when the CurrentCell is in the edit mode?](#)

[How to edit SfDataGrid Template column by single tap?](#)

[How to set the Copy and Paste option of the Grid by using ContextMenu and SfRibbon?](#)

[How to hide the rows based on condition in SfDataGrid?](#)

[How to disable Edit mode for cells in SfDataGrid with different background for those disabled cells?](#)

[How to focus a particular UIElement inside DataTemplate after calling CurrentCell.BeginEdit\(\) or when entering edit mode?](#)



[How to change the same values in all records when the ComboBox column value is changed?](#)

[How to disable the edit mode of AddNewRow in SfDataGrid when AllowEditing is set as False?](#)

[How to get the parent grid while editing the child grid?](#)

[How to handle keyboard and mouse interactions for GridTemplateColumn?](#)

## Data Validation in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) (SfDataGrid) allows you to validate the data and display hints in case of validation is not passed. In case of invalid data, error icon is displayed at the top right corner of [GridCell](#). When mouse over the error icon, error information will be displayed in tooltip.

### Built-in validations

Built-in validations through [IDataErrorInfo](#), [INotifyDataErrorInfo](#) and Data annotation attributes, can be enabled by setting [SfDataGrid.GridValidationMode](#) or [GridColumn.GridValidationMode](#) properties.

[GridColumn.GridValidationMode](#) takes priority than [SfDataGrid.GridValidationMode](#).

- [GridValidation.InEdit](#) - display error icon & tips and also doesn't allows the users to commit the invalid data without allowing users to edit other cells.
- [GridValidation.InView](#) - displays error icons and tips alone.
- [GridValidation.None](#) - disables built-in validation support.

### Built-in validation using [IDataErrorInfo](#) / [INotifyDataErrorInfo](#)

WPF DataGrid (SfDataGrid) provides support to validate the data based on [IDataErrorInfo](#) / [INotifyDataErrorInfo](#).

#### Using [IDataErrorInfo](#)

You can validate the data by inheriting the [IDataErrorInfo](#) interface in model class.

### C#

```
public class OrderInfo : IDataErrorInfo
{
    private string country;
    public string Country
    {
        get { return country; }
        set { country = value; }
    }
    [Display(AutoGenerateField = false)]
    public string Error
    {
        get
        {
            return string.Empty;
        }
    }
    public string this[string columnName]
    {
        get
        {
            if (!columnName.Equals("Country"))
            {
                return string.Empty;
            }
        }
    }
}
```

```

if (this.Country.Contains("Germany") || this.Country.Contains("UK"))
return "Delivery not available for the country " + this.Country;
return string.Empty;
}
}
}

```

Enable built-in validation support by setting [SfDataGrid.GridValidationMode](#) or [GridColumn.GridValidationMode](#) property to `InEdit` or `InView`.

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
AllowEditing="True"
GridValidationMode="InView"/>

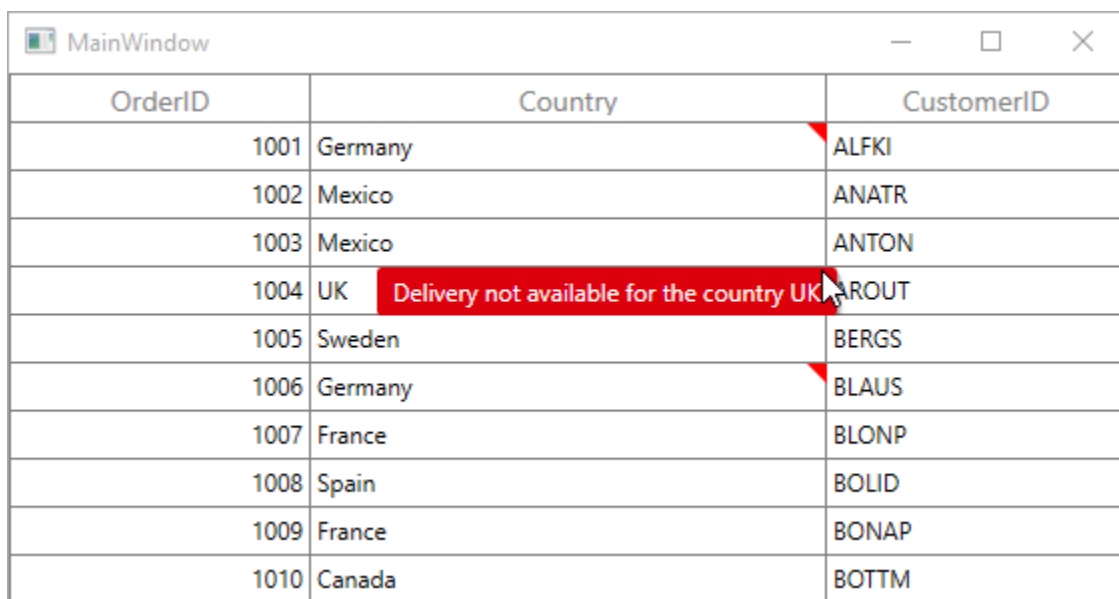
```

### C#

```

this.dataGrid.GridValidationMode = GridValidationMode.InView;

```



OrderID	Country	CustomerID
1001	Germany	ALFKI
1002	Mexico	ANATR
1003	Mexico	ANTON
1004	UK	AROUT
1005	Sweden	BERGS
1006	Germany	BLAUS
1007	France	BLONP
1008	Spain	BOLID
1009	France	BONAP
1010	Canada	BOTTM

### [INotifyDataErrorInfo](#)

You can validate the data by inheriting the [INotifyDataErrorInfo](#) interface in model class.

### C#

```

public class OrderInfo : INotifyDataErrorInfo
{
    private List<string> errors = new List<string>();
    private string shippingCity;
    public string ShipCity
    {
        get { return shippingCity; }
        set { shippingCity = value; }
    }
}

```

```

public System.Collections.IEnumerable GetErrors(string propertyName)
{
    if (!propertyName.Equals("ShipCity"))
        return null;
    if (this.ShipCity.Contains("Mexico D.F. "))
        errors.Add("Delivery not available for the city " + this.ShipCity);
    return errors;
}

[Display(AutoGenerateField = false)]
public bool HasErrors
{
    get
    {
        return false;
    }
}

public event EventHandler<DataErrorsChangedEventArgs> ErrorsChanged;
}

```

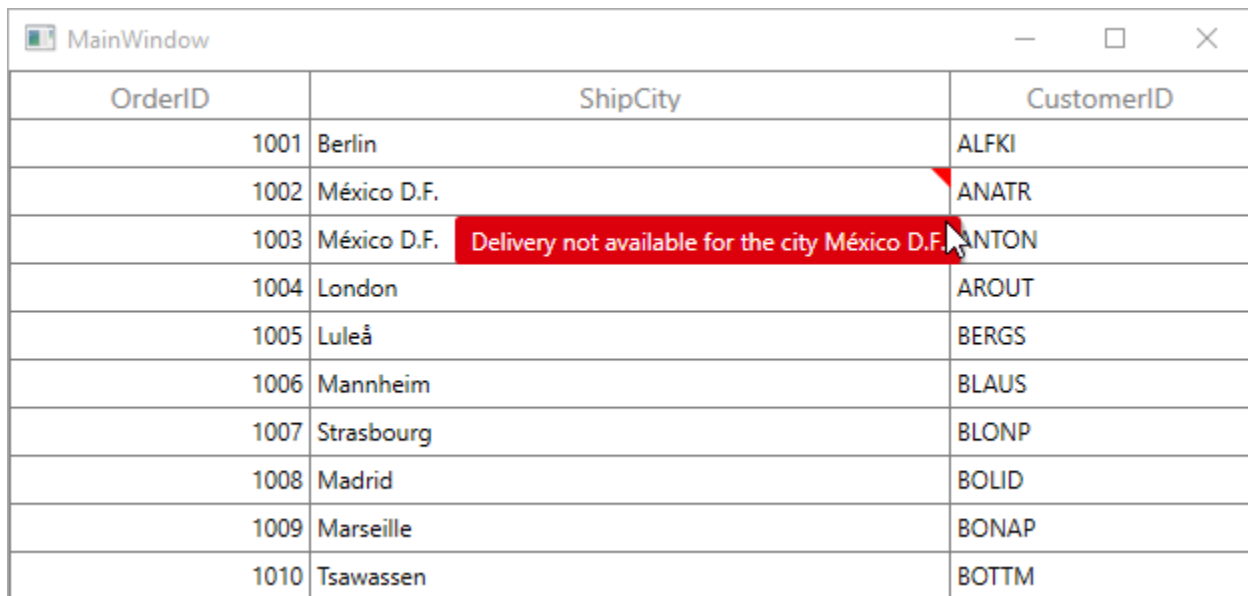
Enable built-in validation support by setting [SfDataGrid.GridValidationMode](#) or [GridColumn.GridValidationMode](#) property to `InEdit` or `InView`.

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
    ItemsSource="{Binding Orders}"
    AllowEditing="True"
    GridValidationMode="InView"/>

```



OrderID	ShipCity	CustomerID
1001	Berlin	ALFKI
1002	México D.F.	ANATR
1003	México D.F.	ANTON
1004	London	AROUT
1005	Luleå	BERGS
1006	Mannheim	BLAUS
1007	Strasbourg	BLONP
1008	Madrid	BOLID
1009	Marseille	BONAP
1010	Tsawassen	BOTTM

Built-in validation using Data Annotation

You can validate the data using **data annotation attributes** by setting [SfDataGrid.GridValidationMode](#) or [GridColumn.GridValidationMode](#) property to `InEdit` or `InView`.

### Using different annotations

The numeric type like int, double, decimal properties can be validated using [Range attributes](#).

#### C#

```
private int orderID;
[Range(1001, 1005, ErrorMessage = "OrderID between 1001 and 1005 alone
processed")]
public int OrderID
{
    get { return orderID; }
    set { orderID = value; }
}
private decimal price;
[Range(typeof(decimal), "12", "20")]
public decimal Price
{
    get { return price; }
    set { price = value; }
}
```

The string type property can be validated using [Required](#), [String Length attributes](#)

#### C#

```
private string shippingCity;
[Required]
public string ShipCity
{
    get { return shippingCity; }
    set { shippingCity = value; }
}
private string customerName;
[StringLength(17)]
public string CustomerName
{
    get { return customerName; }
    set { customerName = value; }
}
```

The data that has heterogeneous type (combination of number, special character) can be validated using [RegularExpressions](#).

#### C#

```
[RegularExpressionAttribute(@"^[a-zA-Z]{1,40}$", ErrorMessage="Numbers and
special characters not allowed")]
public string CustomerID
{
    get { return customerId; }
    set { customerId = value; }
}
```

### Cell validation

You can validate the cells using [CurrentCellValidating](#) event when the cell is edited.

[CurrentCellValidating](#) event occurs when the edited cells tries to commit the data or lose the focus.

DataGrid will not allow user to edit other cells if validation failed.

[CurrentCellValidatingEventArgs](#) provides information to [CurrentCellValidating](#) event for validating the cell. [CurrentCellValidatingEventArgs.OriginalSender](#) returns the DataGrid fired this event for DetailsView.

[CurrentCellValidatingEventArgs.NewValue](#) returns the edited value and you can set the validation status using [CurrentCellValidatingEventArgs.IsValid](#) property.

#### C#

```
this.dataGrid.CurrentCellValidating += dataGrid_CurrentCellValidating;
void dataGrid_CurrentCellValidating(object sender,
CurrentCellValidatingEventArgs args)
{
    if (args.NewValue.ToString().Equals("1004"))
    {
        args.IsValid = false;
        args.ErrorMessage = "OrderID 1004 cannot be passed";
    }
}
```

[SfDataGrid.CurrentCellValidated](#) event triggered when the cell has finished validating with valid data.

#### C#

```
this.dataGrid.CurrentCellValidated += dataGrid_CurrentCellValidated;
void dataGrid_CurrentCellValidated(object sender,
CurrentCellValidatedEventArgs args)
{
}
```

### Row validation

You can validate the row using [RowValidating](#) event when the cell is edited. The [RowValidating](#) event occurs when the edited cells tries to commit the row data or lose the focus. DataGrid will not allow user to edit other rows if validation failed.

[RowValidatingEventArgs](#) provides information to [RowValidating](#) event for validating row.

[RowValidatingEventArgs.OriginalSender](#) returns the DataGrid fired this event for DetailsView.

[RowValidatingEventArgs.RowData](#) returns the edited value and you can set the validation status using [RowValidatingEventArgs.IsValid](#) property.

#### C#

```
this.dataGrid.RowValidating += dataGrid_RowValidating;
void dataGrid_RowValidating(object sender, RowValidatingEventArgs args)
{
    var data =
args.RowData.GetType().GetProperty("CustomerID").GetValue(args.RowData);
    if (data.ToString().Equals("AROUT"))
    {

```

```
args.IsValid = false;
args.ErrorMessages.Add("CustomerID", "Customer AROUT cannot be passed");
}
}
```

[SfDataGrid.RowValidated](#) event triggered when the row has finished validating with valid row data.

### C#

```
this.dataGrid.RowValidated += dataGrid_RowValidated;
void dataGrid_RowValidated(object sender, RowValidatedEventArgs args)
{
}
```

### Data validation error icon customization

You can customize the error icon by editing `GridCell` style.

#### *Change the shape of error icon*

You can change the validation error template shape of the GridCell by changing the `Data` property of the path in the `PART_InValidCellBorder` of GridCell.

### XML

```
<Style TargetType="{x:Type syncfusion:GridCell}">
  <Setter Property="Background" Value="Transparent" />
  <Setter Property="BorderBrush" Value="Gray" />
  <Setter Property="BorderThickness" Value="0,0,1,1" />
  <Setter Property="Padding" Value="0,0,0,0" />
  <Setter Property="FocusVisualStyle" Value="{x:Null}" />
  <Setter Property="IsTabStop" Value="False" />
  <Setter Property="VerticalContentAlignment" Value="Center"/>
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type syncfusion:GridCell}">
        <Grid SnapsToDevicePixels="True">
          <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="IndicationStates">
              <VisualState x:Name="HasError">
                <Storyboard>
                  <DoubleAnimationUsingKeyFrames
                    Storyboard.TargetName="PART_InValidCellBorder"
                    Storyboard.TargetProperty="Width">
                    <EasingDoubleKeyFrame KeyTime="0" Value="0" />
                    <EasingDoubleKeyFrame KeyTime="0" Value="10" />
                  </DoubleAnimationUsingKeyFrames>
                  <ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
                    Storyboard.TargetName="PART_InValidCellBorder"
                    Storyboard.TargetProperty="(UIElement.Visibility)">
                    <DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
                    Visibility.Visible}"/>
                  </ObjectAnimationUsingKeyFrames>
                </Storyboard>
              </VisualState>
              <VisualState x:Name="NoError">
                <Storyboard BeginTime="0">

```

```

<DoubleAnimationUsingKeyFrames
Storyboard.TargetName="PART_InValidCellBorder"
Storyboard.TargetProperty="Width">
<EasingDoubleKeyFrame KeyTime="0" Value="1" />
<EasingDoubleKeyFrame KeyTime="0" Value="0" />
</DoubleAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_InValidCellBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Collapsed}"/>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="BorderStates">
<VisualState x:Name="NormalCell"/>
<VisualState x:Name="FrozenColumnCell">
<Storyboard BeginTime="0">
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_GridCellBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0"
Value="0,0,1,1"/>
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="FooterColumnCell">
<Storyboard BeginTime="0">
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_GridCellBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0"
Value="1,0,1,1"/>
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="BeforeFooterColumnCell">
<Storyboard BeginTime="0">
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_GridCellBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0"
Value="0,0,0,1"/>
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border Background="{TemplateBinding CellSelectionBrush}"
SnapsToDevicePixels="True"
Visibility="{TemplateBinding SelectionBorderVisibility}" />
<Border x:Name="PART_GridCellBorder"
Background="{TemplateBinding Background}"

```

```

BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
SnapsToDevicePixels="True">
<Grid>
<ContentPresenter Margin="{TemplateBinding Padding}" />
</Grid>
</Border>
<Border Background="Transparent"
BorderBrush="{TemplateBinding CurrentCellBorderBrush}"
BorderThickness="{TemplateBinding CurrentCellBorderThickness}"
IsHitTestVisible="False"
SnapsToDevicePixels="True"
Margin="0,0,1,1"
Visibility="{TemplateBinding CurrentCellBorderVisibility}" />
<Border x:Name="PART_InvalidCellBorder"
Width="10"
Height="10"
HorizontalAlignment="Right"
Visibility="Collapsed"
VerticalAlignment="Top"
SnapsToDevicePixels="True">
<ToolTipService.ToolTip>
<ToolTip Background="#FFDB00C"
Placement="Right"
PlacementRectangle="20,0,0,0"
Tag="{TemplateBinding ErrorMessage}"
Template="{StaticResource ValidationToolTipTemplate}" />
</ToolTipService.ToolTip>
<Path Data="M15.396557,23.044006C14.220558,23.044006 13.268559,23.886993
13.268559,24.927994 13.268559,25.975006 14.220558,26.817001
15.396557,26.817001 16.572557,26.817001 17.523547,25.975006
17.523547,24.927994 17.523547,23.886993 16.572557,23.044006
15.396557,23.044006z M15.467541,5.1819992C15.447552,5.1819992
15.436566,5.1829987 15.436566,5.1829987 13.118533,5.5049973
13.055545,7.3330002 13.055545,7.3330002L13.055545,9.2929993
13.626531,16.539001C13.983558,18.357002 14.243538,19.020004
14.243538,19.020004 15.275555,19.975006 16.203567,19.25 16.203567,19.25
16.976548,18.565994 17.028552,16.962997 17.028552,16.962997
17.956563,9.2929993 17.696553,7.1029968 17.696553,7.1029968
17.608571,5.2839966 15.823561,5.1849976 15.490551,5.1819992
15.481549,5.1819992 15.473553,5.1819992 15.467541,5.1819992z
M15.56355,0C15.56355,0 21.710574,4.1259995 31.581613,2.8030014
31.581613,2.8030014 33.634629,26.556992 15.56355,32 15.56355,32 -
0.10249132,27.548004 0.00050565118,2.9670029 0.0005058694,2.9670029
10.72555,3.6309967 15.56355,0z"
Fill="Red"
SnapsToDevicePixels="True"
Stretch="Fill" />
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```



OrderID	ShipCity	CustomerID	CustomerName	Country
1001	Berlin	ALFKI	Maria Anders	Germany
1002	México D.F.	ANATR	Ana Trujilo	Mexico
1003	México D.F.	ANTON	Antonio Moreno	Mexico
1004	London	AROUT	Thomas Hardy	UK
1005	Luleå	BERGS	Christina Berglund	Sweden
1006	Mannheim	BLAUS	Hanna Moos	Germany
1007		BLONP	Frédérique Citeaux	France
1008	Madrid	BOLID	Martin Sommer	Spain
1009	Marseille	BONAP	Laurence Lebihan	France
1010	Tsawassen	BOTTM	Elizabeth Lincoln	Canada

*Change the color of error icon*

You can change the validation error template color of the `GridCell` by changing the `Fill` property of the path in the `PART_InValidCellBorder` of `GridCell`.

#### XML

```
<Style TargetType="{x:Type syncfusion:GridCell}">
  <Setter Property="Background" Value="Transparent" />
  <Setter Property="BorderBrush" Value="Gray" />
  <Setter Property="BorderThickness" Value="0,0,1,1" />
  <Setter Property="Padding" Value="0,0,0,0" />
  <Setter Property="FocusVisualStyle" Value="{x:Null}" />
  <Setter Property="IsTabStop" Value="False" />
  <Setter Property="VerticalContentAlignment" Value="Center"/>
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type syncfusion:GridCell}">
        <Grid SnapsToDevicePixels="True">
          <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="IndicationStates">
              <VisualState x:Name="HasError">
                <Storyboard>
                  <DoubleAnimationUsingKeyFrames
                    Storyboard.TargetName="PART_InValidCellBorder"
                    Storyboard.TargetProperty="Width">
                    <EasingDoubleKeyFrame KeyTime="0" Value="0" />
                    <EasingDoubleKeyFrame KeyTime="0" Value="10" />
                  </DoubleAnimationUsingKeyFrames>
                  <ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
                    Storyboard.TargetName="PART_InValidCellBorder"
                    Storyboard.TargetProperty="(UIElement.Visibility)">
                    <DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
                    Visibility.Visible}"/>
                  </ObjectAnimationUsingKeyFrames>
                </Storyboard>
              </VisualState>
            </VisualStateGroup>
          </VisualStateManager.VisualStateGroups>
        </Grid>
      </Setter.Value>
    </Setter>
  </Setter>
</Style>
```

```

<VisualState x:Name="NoError">
  <Storyboard BeginTime="0">
    <DoubleAnimationUsingKeyFrames
      Storyboard.TargetName="PART_InValidCellBorder"
      Storyboard.TargetProperty="Width">
      <EasingDoubleKeyFrame KeyTime="0" Value="1" />
      <EasingDoubleKeyFrame KeyTime="0" Value="0" />
    </DoubleAnimationUsingKeyFrames>
    <ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
      Storyboard.TargetName="PART_InValidCellBorder"
      Storyboard.TargetProperty="(UIElement.Visibility)">
      <DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
        Visibility.Collapsed}"/>
    </ObjectAnimationUsingKeyFrames>
  </Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="BorderStates">
  <VisualState x:Name="NormalCell"/>
  <VisualState x:Name="FrozenColumnCell">
    <Storyboard BeginTime="0">
      <ThicknessAnimationUsingKeyFrames BeginTime="0"
        Duration="1"
        Storyboard.TargetName="PART_GridCellBorder"
        Storyboard.TargetProperty="BorderThickness">
        <EasingThicknessKeyFrame KeyTime="0"
          Value="0,0,1,1"/>
      </ThicknessAnimationUsingKeyFrames>
    </Storyboard>
  </VisualState>
  <VisualState x:Name="FooterColumnCell">
    <Storyboard BeginTime="0">
      <ThicknessAnimationUsingKeyFrames BeginTime="0"
        Duration="1"
        Storyboard.TargetName="PART_GridCellBorder"
        Storyboard.TargetProperty="BorderThickness">
        <EasingThicknessKeyFrame KeyTime="0"
          Value="1,0,1,1"/>
      </ThicknessAnimationUsingKeyFrames>
    </Storyboard>
  </VisualState>
  <VisualState x:Name="BeforeFooterColumnCell">
    <Storyboard BeginTime="0">
      <ThicknessAnimationUsingKeyFrames BeginTime="0"
        Duration="1"
        Storyboard.TargetName="PART_GridCellBorder"
        Storyboard.TargetProperty="BorderThickness">
        <EasingThicknessKeyFrame KeyTime="0"
          Value="0,0,0,1"/>
      </ThicknessAnimationUsingKeyFrames>
    </Storyboard>
  </VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border Background="{TemplateBinding CellSelectionBrush}"
  SnapsToDevicePixels="True"
  Visibility="{TemplateBinding SelectionBorderVisibility}" />

```

```
<Border x:Name="PART_GridCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
SnapsToDevicePixels="True">
<Grid>
<ContentPresenter Margin="{TemplateBinding Padding}" />
</Grid>
</Border>
<Border Background="Transparent"
BorderBrush="{TemplateBinding CurrentCellBorderBrush}"
BorderThickness="{TemplateBinding CurrentCellBorderThickness}"
IsHitTestVisible="False"
SnapsToDevicePixels="True"
Margin="0,0,1,1"
Visibility="{TemplateBinding CurrentCellBorderVisibility}" />
<Border x:Name="PART_InvalidCellBorder"
Width="10"
Height="10"
HorizontalAlignment="Right"
Visibility="Collapsed"
VerticalAlignment="Top"
SnapsToDevicePixels="True">
<ToolTipService.ToolTip>
<ToolTip Background="#FFDB00C"
Placement="Right"
PlacementRectangle="20,0,0,0"
Tag="{TemplateBinding ErrorMessage}"
Template="{StaticResource ValidationToolTipTemplate}" />
</ToolTipService.ToolTip>
<Path Data="M0.5,0.5 L12.652698,0.5 12.652698,12.068006 z"
Fill="Orange"
SnapsToDevicePixels="True"
Stretch="Fill" />
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
```

OrderID	ShipCity	CustomerID	CustomerName	Country
1001	Berlin	ALFKI	Maria Anders	Germany
1002	México D.F.	ANATR	Ana Trujilo	Mexico
1003	México D.F.	ANTON	Antonio Moreno	Mexico
1004	London	AROUT	Thomas Hardy	UK
1005	Luleå	BERGS	Christina Berglund	Sweden
1006	Mannheim	BLAUS	Hanna Moos	Germany
1007		BLONP	Frédérique Citeaux	France
1008	Madrid	BOLID	Martin Sommer	Spain
1009	Marseille	BONAP	Laurence Lebihan	France
1010	Tsawassen	BOTTM	Elizabeth Lincoln	Canada

*Change the cursor over error icon*

You can change the validation error template cursor of the `GridCell` by changing the `Cursor` property of the path codes in the `PART_InValidCellBorder` of `GridCell`.

#### XML

```
<Style TargetType="{x:Type syncfusion:GridCell}">
  <Setter Property="Background" Value="Transparent" />
  <Setter Property="BorderBrush" Value="Gray" />
  <Setter Property="BorderThickness" Value="0,0,1,1" />
  <Setter Property="Padding" Value="0,0,0,0" />
  <Setter Property="FocusVisualStyle" Value="{x:Null}" />
  <Setter Property="IsTabStop" Value="False" />
  <Setter Property="VerticalContentAlignment" Value="Center"/>
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type syncfusion:GridCell}">
        <Grid SnapsToDevicePixels="True">
          <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="IndicationStates">
              <VisualState x:Name="HasError">
                <Storyboard>
                  <DoubleAnimationUsingKeyFrames
                    Storyboard.TargetName="PART_InValidCellBorder"
                    Storyboard.TargetProperty="Width">
                    <EasingDoubleKeyFrame KeyTime="0" Value="0" />
                    <EasingDoubleKeyFrame KeyTime="0" Value="10" />
                  </DoubleAnimationUsingKeyFrames>
                  <ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
                    Storyboard.TargetName="PART_InValidCellBorder"
                    Storyboard.TargetProperty="(UIElement.Visibility)">
                    <DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
                      Visibility.Visible}"/>
                  </ObjectAnimationUsingKeyFrames>
                </Storyboard>
              </VisualState>
            </VisualStateGroup>
          </VisualStateManager.VisualStateGroups>
        </Grid>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

```

<VisualState x:Name="NoError">
  <Storyboard BeginTime="0">
    <DoubleAnimationUsingKeyFrames
      Storyboard.TargetName="PART_InValidCellBorder"
      Storyboard.TargetProperty="Width">
      <EasingDoubleKeyFrame KeyTime="0" Value="1" />
      <EasingDoubleKeyFrame KeyTime="0" Value="0" />
    </DoubleAnimationUsingKeyFrames>
    <ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
      Storyboard.TargetName="PART_InValidCellBorder"
      Storyboard.TargetProperty="(UIElement.Visibility)">
      <DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
        Visibility.Collapsed}"/>
    </ObjectAnimationUsingKeyFrames>
  </Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="BorderStates">
  <VisualState x:Name="NormalCell"/>
  <VisualState x:Name="FrozenColumnCell">
    <Storyboard BeginTime="0">
      <ThicknessAnimationUsingKeyFrames BeginTime="0"
        Duration="1"
        Storyboard.TargetName="PART_GridCellBorder"
        Storyboard.TargetProperty="BorderThickness">
        <EasingThicknessKeyFrame KeyTime="0"
          Value="0,0,1,1"/>
      </ThicknessAnimationUsingKeyFrames>
    </Storyboard>
  </VisualState>
  <VisualState x:Name="FooterColumnCell">
    <Storyboard BeginTime="0">
      <ThicknessAnimationUsingKeyFrames BeginTime="0"
        Duration="1"
        Storyboard.TargetName="PART_GridCellBorder"
        Storyboard.TargetProperty="BorderThickness">
        <EasingThicknessKeyFrame KeyTime="0"
          Value="1,0,1,1"/>
      </ThicknessAnimationUsingKeyFrames>
    </Storyboard>
  </VisualState>
  <VisualState x:Name="BeforeFooterColumnCell">
    <Storyboard BeginTime="0">
      <ThicknessAnimationUsingKeyFrames BeginTime="0"
        Duration="1"
        Storyboard.TargetName="PART_GridCellBorder"
        Storyboard.TargetProperty="BorderThickness">
        <EasingThicknessKeyFrame KeyTime="0"
          Value="0,0,0,1"/>
      </ThicknessAnimationUsingKeyFrames>
    </Storyboard>
  </VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border Background="{TemplateBinding CellSelectionBrush}"
  SnapsToDevicePixels="True"
  Visibility="{TemplateBinding SelectionBorderVisibility}" />

```

```
<Border x:Name="PART_GridCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
SnapsToDevicePixels="True">
<Grid>
<ContentPresenter Margin="{TemplateBinding Padding}" />
</Grid>
</Border>
<Border Background="Transparent"
BorderBrush="{TemplateBinding CurrentCellBorderBrush}"
BorderThickness="{TemplateBinding CurrentCellBorderThickness}"
IsHitTestVisible="False"
SnapsToDevicePixels="True"
Margin="0,0,1,1"
Visibility="{TemplateBinding CurrentCellBorderVisibility}" />
<Border x:Name="PART_InvalidCellBorder"
Width="10"
Height="10"
HorizontalAlignment="Right"
Visibility="Collapsed"
VerticalAlignment="Top"
SnapsToDevicePixels="True">
<ToolTipService.ToolTip>
<ToolTip Background="#FFDB00C"
Placement="Right"
PlacementRectangle="20,0,0,0"
Tag="{TemplateBinding ErrorMessage}"
Template="{StaticResource ValidationToolTipTemplate}" />
</ToolTipService.ToolTip>
<Path Data="M0.5,0.5 L12.652698,0.5 12.652698,12.068006 z"
Fill="Red"
SnapsToDevicePixels="True"
Cursor="Hand"
Stretch="Fill" />
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
```

OrderID	ShipCity	CustomerID	CustomerName	Country
1001	Berlin	ALFKI	Maria Anders	Germany
1002	México D.F.	ANATR	Ana Trujilo	Mexico
1003	México D.F.	ANTON	Antonio Moreno	Mexico
1004	London	AROUT	Thomas Hardy	UK
1005	Luleå			Sweden
1006	Mannheim			Germany
1007		BLONP	Frédérique Citeaux	France
1008	Madrid	BOLID	Martin Sommer	Spain
1009	Marseille	BONAP	Laurence Lebihan	France
1010	Tsawassen	BOTTM	Elizabeth Lincoln	Canada

### Data validation error tip (help tip) customization

You can customize the error tip by editing the style of `ValidationToolTipTemplate`. Get the style of `ValidationToolTipTemplate` by editing the `GridCell` style.

#### *Change the background and foreground color of error tip*

You can change the error tip background color by setting `Background` property of the border in `ValidationToolTipTemplate`. The error tip foreground color can be changed by setting `Foreground` property of the `TextBlock` in `ValidationToolTipTemplate`.

### XML

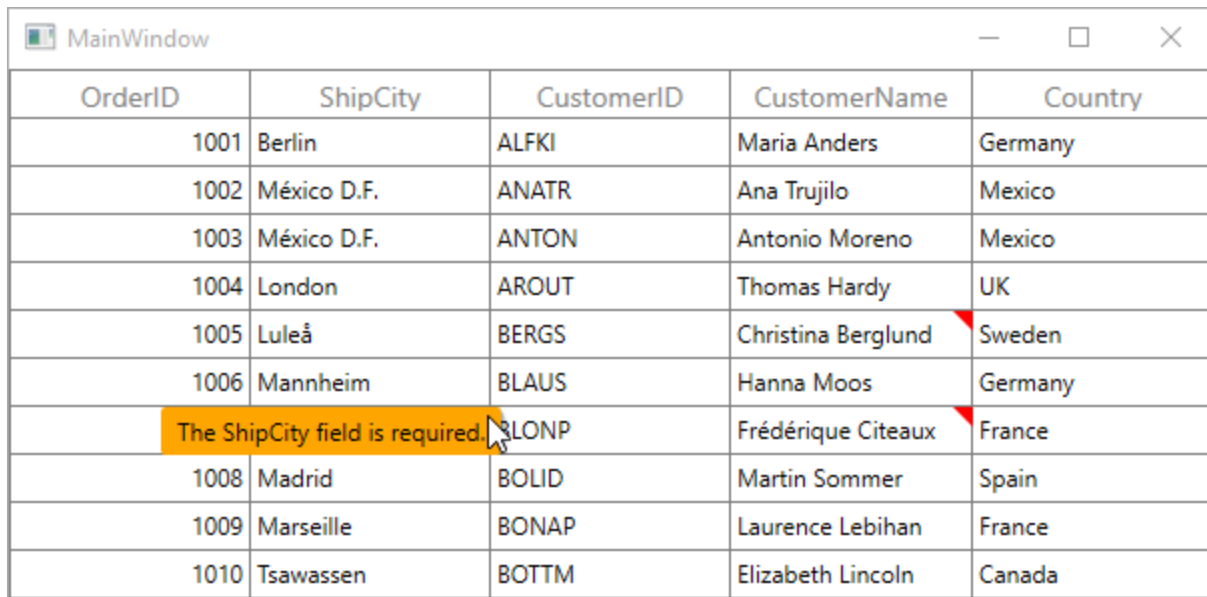
```
<ControlTemplate x:Key="ValidationToolTipTemplate">
  <Grid x:Name="Root"
    Margin="5,0"
    Opacity="0"
    RenderTransformOrigin="0,0">
    <Grid.RenderTransform>
      <TranslateTransform x:Name="xForm" X="-25" />
    </Grid.RenderTransform>
    <VisualStateManager.VisualStateGroups>
      <VisualStateGroup x:Name="OpenStates">
        <VisualStateGroup.Transitions>
          <VisualTransition GeneratedDuration="0" />
          <VisualTransition GeneratedDuration="0:0:0.2" To="Open">
            <Storyboard>
              <DoubleAnimation Duration="0:0:0.2"
                Storyboard.TargetName="xForm"
                Storyboard.TargetProperty="X"
                To="0">
                <DoubleAnimation.EasingFunction>
                  <BackEase Amplitude=".3" EasingMode="EaseOut" />
                </DoubleAnimation.EasingFunction>
              </DoubleAnimation>
              <DoubleAnimation Duration="0:0:0.2"
                Storyboard.TargetName="Root"
                Storyboard.TargetProperty="Opacity">
```

```

To="1" />
</Storyboard>
</VisualTransition>
</VisualStateGroup.Transitions>
<VisualState x:Name="Closed">
<Storyboard>
<DoubleAnimation Duration="0"
Storyboard.TargetName="Root"
Storyboard.TargetProperty="Opacity"
To="0" />
</Storyboard>
</VisualState>
<VisualState x:Name="Open">
<Storyboard>
<DoubleAnimation Duration="0"
Storyboard.TargetName="xForm"
Storyboard.TargetProperty="X"
To="0" />
<DoubleAnimation Duration="0"
Storyboard.TargetName="Root"
Storyboard.TargetProperty="Opacity"
To="1" />
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border Margin="4,4,-4,-4"
Background="#052A2E31"
CornerRadius="5" />
<Border Margin="3,3,-3,-3"
Background="#152A2E31"
CornerRadius="4" />
<Border Margin="2,2,-2,-2"
Background="#252A2E31"
CornerRadius="3" />
<Border Margin="1,1,-1,-1"
Background="#352A2E31"
CornerRadius="2" />
<Border Background="Orange" CornerRadius="2" />
<Border CornerRadius="2">
<TextBlock MaxWidth="250"
Margin="8,4,8,4"
Foreground="Black"
Text="{TemplateBinding Tag}"
TextWrapping="Wrap"
UseLayoutRounding="false" />
</Border>
</Grid>
</ControlTemplate>

```





OrderID	ShipCity	CustomerID	CustomerName	Country
1001	Berlin	ALFKI	Maria Anders	Germany
1002	México D.F.	ANATR	Ana Trujilo	Mexico
1003	México D.F.	ANTON	Antonio Moreno	Mexico
1004	London	AROUT	Thomas Hardy	UK
1005	Luleå	BERGS	Christina Berglund	Sweden
1006	Mannheim	BLAUS	Hanna Moos	Germany
	The ShipCity field is required.	SLONP	Frédérique Citeaux	France
1008	Madrid	BOLID	Martin Sommer	Spain
1009	Marseille	BONAP	Laurence Lebihan	France
1010	Tsawassen	BOTTM	Elizabeth Lincoln	Canada

Showing error details in RowHeader

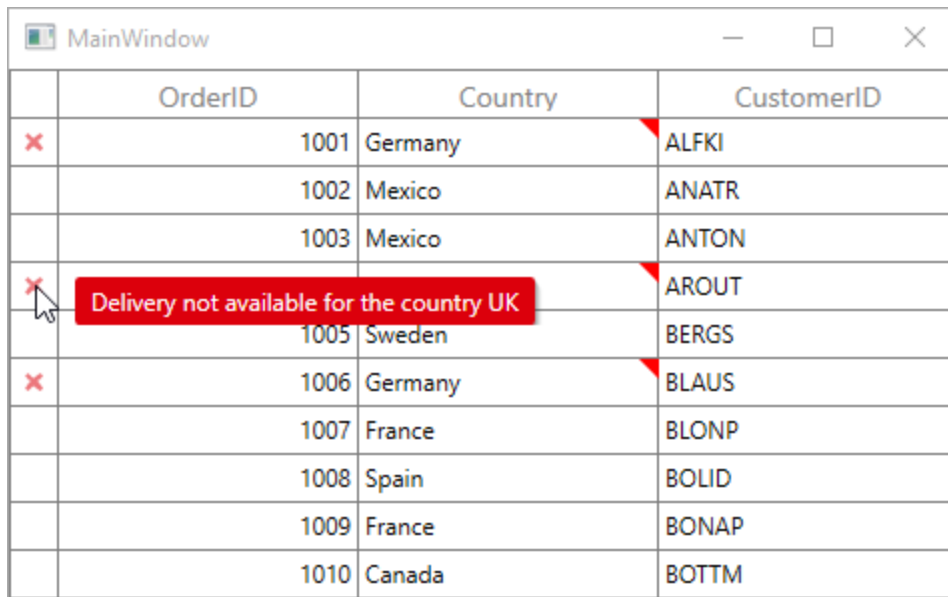
WPF DataGrid (SfDataGrid) support to show the error icon in [GridRowHeaderCell](#) based on [IDataErrorInfo.Error](#) or [INotifyDataErrorInfo.HasErrors](#) property.

*Using IDataErrorInfo*

You can show the error information in row header by setting [IDataErrorInfo.Error](#). `IDataErrorInfo.Error` will be displayed as error message in tooltip.

## C#

```
[Display(AutoGenerateField = false)]
public string Error
{
    get
    {
        if (this.Country.Contains("Germany") || this.Country.Contains("UK"))
            return "Delivery not available for the country " + this.Country;
        return string.Empty;
    }
}
```



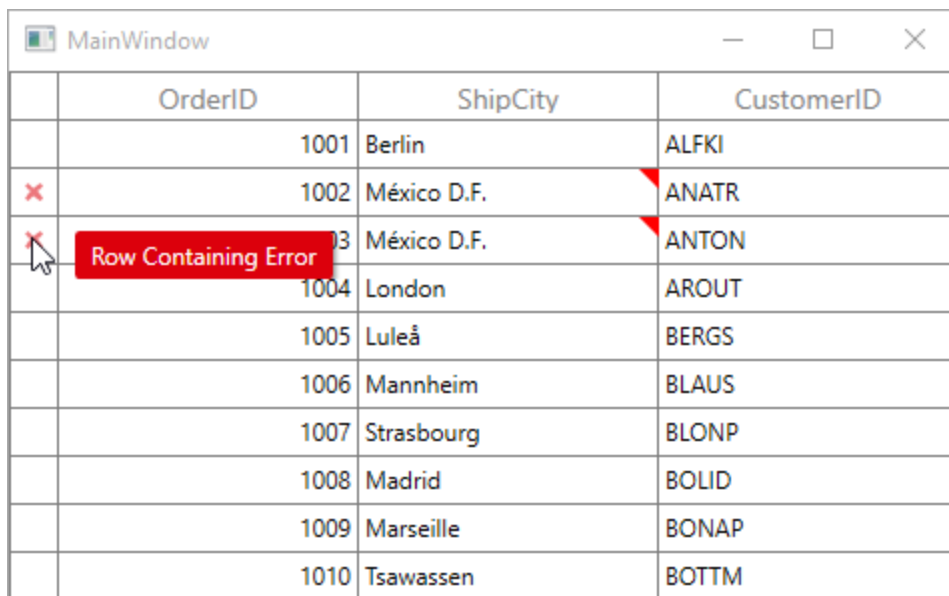
	OrderID	Country	CustomerID
✖	1001	Germany	ALFKI
	1002	Mexico	ANATR
	1003	Mexico	ANTON
✖			AROUT
	1005	Sweden	BERGS
✖	1006	Germany	BLAUS
	1007	France	BLONP
	1008	Spain	BOLID
	1009	France	BONAP
	1010	Canada	BOTTM

#### Using INotifyDataErrorInfo

You can show the error information in row header by setting [INotifyDataErrorInfo.HasErrors](#). By default error message **Row Containing Error** will be displayed. You can change this by changing **RowErrorMessage** in the **resx** file.

#### C#

```
[Display(AutoGenerateField = false)]
public bool HasErrors
{
    get
    {
        if (this.ShipCity.Contains("Mexico D.F. "))
            return true;
        return false;
    }
}
```



	OrderID	ShipCity	CustomerID
	1001	Berlin	ALFKI
✗	1002	México D.F.	ANATR
✗	1003	México D.F.	ANTON
	1004	London	AROUT
	1005	Luleå	BERGS
	1006	Mannheim	BLAUS
	1007	Strasbourg	BLONP
	1008	Madrid	BOLID
	1009	Marseille	BONAP
	1010	Tsawassen	BOTTM

### Data validation with Master-details view

Master-Details View allows you to [validate](#) the bound data is valid or not.

You can do both built-in and custom validation of data in `DetailsViewDataGrid`.

#### Built-in validations

You can validate the bound data based on [IDataErrorInfo](#) / [INotifyDataErrorInfo](#) or [Data Annotation](#) Attributes by setting [GridValidationMode](#) property of [ViewDefinition.DataGrid](#).

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    NavigationMode="Cell"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.DetailsViewDefinition>
    <syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
    <syncfusion:GridViewDefinition.DataGrid>
    <syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
        GridValidationMode="InView"
        AutoGenerateColumns="True">
    </syncfusion:SfDataGrid>
    </syncfusion:GridViewDefinition.DataGrid>
    </syncfusion:GridViewDefinition>
    </syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>
```

When the relation is auto-generated, the data can be validated by setting `GridValidationMode` property to `AutoGeneratingRelations.GridViewDefinition.DataGrid` in [AutoGeneratingRelations](#) event handler.

#### C#

```
dataGrid.AutoGenerateRelations = true;
dataGrid.AutoGeneratingRelations += dataGrid_AutoGeneratingRelations;
```

```

void dataGrid_AutoGeneratingRelations(object sender,
Syncfusion.UI.Xaml.Grid.AutoGeneratingRelationsArgs e)
{
e.GridViewDefinition.DataGrid.GridValidationMode =
GridValidationMode.InView;
}

```

	OrderID	CustomerID	CustomerName	Country	ShipCity
+	1001	ALFKI	Maria Anders	Germany	Berlin
-	1002	ANATR	Ana Trujillo	Mexico	México D.F.

	OrderID	ProductName	DateOfMonth
		The ProductName field is required.	5/7/2014
	1002	Bike1	5/20/2012
	1002	Bike3	5/27/2015

+	1003	ANTON	Antonio Moreno	Mexico	México D.F.
+	1004	AROUT	Thomas Hardy	UK	London
+	1005	BERGS	Christina Berglun	Sweden	Luleå
+	1006	BLAUS	Hanna Moos	Germany	Mannheim
+	1007	BLONP	Frédérique Citea	France	Strasbourg
+	1008	BOLID	Martin Sommer	Spain	Madrid

#### Custom validation through events

Master-Details View support to validate the cells and rows using [CurrentCellValidating](#) and [RowValidating](#) events.

#### Cell Validation

You can validate the cells using [CurrentCellValidating](#) event of [ViewDefinition.DataGrid](#) when the cell is edited. [CurrentCellValidating](#) event occurs when the edited cells tries to commit the data or lose the focus.

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
NavigationMode="Cell"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AllowEditing="True"
CurrentCellValidating="FirstLevelNestedGrid_CurrentCellValidating"
AutoGenerateColumns="True">
</syncfusion:SfDataGrid>
</syncfusion:GridViewDefinition.DataGrid>

```

```

</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>

```

<br/>

### C#

```

this.FirstLevelNestedGrid.CurrentCellValidating
+=FirstLevelNestedGrid_CurrentCellValidating;
private void FirstLevelNestedGrid_CurrentCellValidating(object sender,
CurrentCellValidatingEventArgs args)
{
    if(args.NewValue.ToString().Equals("Bike2"))
    {
        args.IsValid = false;
        args.ErrorMessage = "Order ID 1002 not ordered Bike2";
    }
}

```

[CurrentCellValidated](#) event [ViewDefinition.DataGrid](#) triggered when the cell has finished validating with valid data

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
    NavigationMode="Cell"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.DetailsViewDefinition>
    <syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
    <syncfusion:GridViewDefinition.DataGrid>
    <syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
        AllowEditing="True"
        CurrentCellValidated="FirstLevelNestedGrid_CurrentCellValidated"
        AutoGenerateColumns="True">
    </syncfusion:SfDataGrid>
    </syncfusion:GridViewDefinition.DataGrid>
    </syncfusion:GridViewDefinition>
    </syncfusion:SfDataGrid.DetailsViewDefinition>
    </syncfusion:SfDataGrid>

```

<br/>

### C#

```

this.FirstLevelNestedGrid.CurrentCellValidated
+=FirstLevelNestedGrid_CurrentCellValidated;
private void FirstLevelNestedGrid_CurrentCellValidated(object sender,
CurrentCellValidatedEventArgs args)
{
}

```

When the relation is auto-generated, you can wire the `CurrentCellValidating` and `CurrentCellValidated` events for `AutoGeneratingRelations.GridViewDefinition.DataGrid` in [AutoGeneratingRelations](#) event handler.

### C#

```
dataGrid.AutoGenerateRelations = true;
dataGrid.AutoGeneratingRelations += dataGrid_AutoGeneratingRelations;
void dataGrid_AutoGeneratingRelations(object sender,
Syncfusion.UI.Xaml.Grid.AutoGeneratingRelationsArgs e)
{
    e.GridViewDefinition.DataGrid.CurrentCellValidating +=
    FirstLevelNestedGrid_CurrentCellValidating;
    e.GridViewDefinition.DataGrid.CurrentCellValidated +=
    FirstLevelNestedGrid_CurrentCellValidated;
}
```

### Row Validation

You can validate the row using `RowValidating` event of `ViewDefinition.DataGrid` when the cell is edited.

The `RowValidating` event occurs when edited cells tries to commit the row data or lose the focus.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    NavigationMode="Cell"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.DetailsViewDefinition>
    <syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
    <syncfusion:GridViewDefinition.DataGrid>
    <syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
        AllowEditing="True"
        RowValidating="FirstLevelNestedGrid_RowValidating"
        AutoGenerateColumns="True">
    </syncfusion:SfDataGrid>
    </syncfusion:GridViewDefinition.DataGrid>
    </syncfusion:GridViewDefinition>
    </syncfusion:SfDataGrid.DetailsViewDefinition>
    </syncfusion:SfDataGrid>
```

<br/>

### C#

```
this.FirstLevelNestedGrid.RowValidating
+= FirstLevelNestedGrid_RowValidating;
private void FirstLevelNestedGrid_RowValidating(object sender,
RowValidatingEventArgs args)
{
    var data =
    args.RowData.GetType().GetProperty("ProductName").GetValue(args.RowData);
    if (data.ToString().Equals("Bike1"))
    {
        args.IsValid = false;
    }
}
```

```
args.ErrorMessages.Add("ProductName", "Product Bike 1 cannot be order for
OrderID 1002");
}
}
```

[RowValidated](#) of [ViewDefinition.DataGrid](#) event triggered when the row has finished validating with valid row data.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    NavigationMode="Cell"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.DetailsViewDefinition>
    <syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
    <syncfusion:GridViewDefinition.DataGrid>
    <syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
        AllowEditing="True"
        RowValidated="FirstLevelNestedGrid_RowValidated"
        AutoGenerateColumns="True">
    </syncfusion:SfDataGrid>
    </syncfusion:GridViewDefinition.DataGrid>
    </syncfusion:GridViewDefinition>
    </syncfusion:SfDataGrid.DetailsViewDefinition>
    </syncfusion:SfDataGrid>
```

<br/>

#### C#

```
this.FirstLevelNestedGrid.RowValidated +=FirstLevelNestedGrid_RowValidated;
private void FirstLevelNestedGrid_RowValidated(object sender,
    RowValidatedEventArgs args)
{
}
```

When the relation is auto-generated, you can wire the [RowValidating](#) and [RowValidated](#) events for [AutoGeneratingRelations.GridViewDefinition.DataGrid](#) in [AutoGeneratingRelations](#) event handler.

#### C#

```
dataGrid.AutoGenerateRelations = true;
dataGrid.AutoGeneratingRelations +=dataGrid_AutoGeneratingRelations;
void dataGrid_AutoGeneratingRelations(object sender,
    Syncfusion.UI.Xaml.Grid.AutoGeneratingRelationsArgs e)
{
    e.GridViewDefinition.DataGrid.RowValidating +=
    FirstLevelNestedGrid_RowValidating;
    e.GridViewDefinition.DataGrid.RowValidated +=
    FirstLevelNestedGrid_RowValidated;
}
```

### Data validation with checkbox column

SfDataGrid doesn't support to validate the [GridCheckBoxColumn](#) through validating events.

You can validate the check box column value by setting [ValidationHelper.IsCurrentCellValidated](#) and [ValidationHelper.IsCurrentRowValidated](#) static properties by calling [SetCurrentRowValidated](#) and [SetCurrentCellValidated](#) methods from [ValidationHelper](#).

### C#

```
using Syncfusion.UI.Xaml.Grid.Helpers;
this.dataGrid.CurrentCellValueChanged += dataGrid_CurrentCellValueChanged;
void dataGrid_CurrentCellValueChanged(object sender,
CurrentCellValueChangedEventArgs args)
{
    int columnIndex =
    this.dataGrid.ResolveToGridVisibleColumnIndex(args.RowColumnIndex.ColumnIndex);
    //We are enabling the RowValidating, CellValidating event if the changes
    happen in GridCheckBoxColumn
    if (this.dataGrid.Columns[columnIndex].CellType == "CheckBox")
    {
        this.dataGrid.GetValidationHelper().SetCurrentRowValidated(false);
        this.dataGrid.GetValidationHelper().SetCurrentCellValidated(false);
    }
}
```

OrderID	Delivered	Country
1001	<input type="checkbox"/>	Germany
1002	<input checked="" type="checkbox"/>	Mexico
1003	<input type="checkbox"/>	Mexico
1004	<input type="checkbox"/>	UK
1005	<input checked="" type="checkbox"/>	Sweden
1006	<input type="checkbox"/>	Germany
1007	<input type="checkbox"/>	France
1008	<input checked="" type="checkbox"/>	Spain
1009	<input type="checkbox"/>	France
1010	<input type="checkbox"/>	Canada

### Show validation errors when using UseDrawing

By default, validation is not supported while enabling the [UseDrawing](#) property since the cell content were drawn instead of loading the UIElement. However, SfDataGrid provides an option to achieve the validation by adding the validation template.

Please refer the below code example for further details about achieving Validation when using [UseDrawing](#) property.

### XML

```
<Window.Resources>
<ControlTemplate x:Key="ValidationToolTipTemplate">
<Grid x:Name="Root"
Margin="5,0"
```



```
Opacity="0"
RenderTransformOrigin="0,0">
<Grid.RenderTransform>
<TranslateTransform x:Name="xform" X="-25" />
</Grid.RenderTransform>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup Name="OpenStates">
<VisualStateGroup.Transitions>
<VisualTransition GeneratedDuration="0" />
<VisualTransition GeneratedDuration="0:0:0.2" To="Open">
<Storyboard>
<DoubleAnimation Duration="0:0:0.2"
Storyboard.TargetName="xform"
Storyboard.TargetProperty="X"
To="0">
<DoubleAnimation.EasingFunction>
<BackEase Amplitude=".3" EasingMode="EaseOut" />
</DoubleAnimation.EasingFunction>
</DoubleAnimation>
<DoubleAnimation Duration="0:0:0.2"
Storyboard.TargetName="Root"
Storyboard.TargetProperty="Opacity"
To="1" />
</Storyboard>
</VisualTransition>
</VisualStateGroup.Transitions>
<VisualState x:Name="Closed">
<Storyboard>
<DoubleAnimation Duration="0"
Storyboard.TargetName="Root"
Storyboard.TargetProperty="Opacity"
To="0" />
</Storyboard>
</VisualState>
<VisualState x:Name="Open">
<Storyboard>
<DoubleAnimation Duration="0"
Storyboard.TargetName="xform"
Storyboard.TargetProperty="X"
To="0" />
<DoubleAnimation Duration="0"
Storyboard.TargetName="Root"
Storyboard.TargetProperty="Opacity"
To="1" />
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border Margin="4,4,-4,-4"
Background="#052A2E31"
CornerRadius="5" />
<Border Margin="3,3,-3,-3"
Background="#152A2E31"
CornerRadius="4" />
<Border Margin="2,2,-2,-2"
Background="#252A2E31"
CornerRadius="3" />
```

```

<Border Margin="1,1,-1,-1"
Background="#352A2E31"
CornerRadius="2" />
<Border Background="#FFDC000C" CornerRadius="2" />
<Border CornerRadius="2">
<TextBlock MaxWidth="250"
Margin="8,4,8,4"
Foreground="White"
Text="{TemplateBinding Tag}"
TextWrapping="Wrap"
UseLayoutRounding="false" />
</Border>
</Grid>
</ControlTemplate>
<ControlTemplate x:Key="GridCellControlTemplate"
TargetType="Syncfusion:GridCell">
<Grid SnapsToDevicePixels="True">
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="IndicationStates">
<VisualState x:Name="NoError">
<Storyboard BeginTime="0">
<!--<DoubleAnimationUsingKeyFrames
Storyboard.TargetName="PART_InValidCellBorder"
Storyboard.TargetProperty="Width">
<EasingDoubleKeyFrame KeyTime="0" Value="1" />
<EasingDoubleKeyFrame KeyTime="0" Value="0" />
</DoubleAnimationUsingKeyFrames>-->
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_InValidCellBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Collapsed}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="HasError">
<Storyboard>
<!--<DoubleAnimationUsingKeyFrames
Storyboard.TargetName="PART_InValidCellBorder"
Storyboard.TargetProperty="Width">
<EasingDoubleKeyFrame KeyTime="0" Value="0" />
<EasingDoubleKeyFrame KeyTime="0" Value="10" />
</DoubleAnimationUsingKeyFrames>-->
<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="PART_InValidCellBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="00:00:00" Value="{x:Static
Visibility.Visible}" />
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="BorderStates">
<VisualState x:Name="NormalCell" />
<VisualState x:Name="FrozenColumnCell"/>
<VisualState x:Name="FooterColumnCell">
<Storyboard BeginTime="0">

```

```

<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_GridCellBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0" Value="1,0,1,1" />
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="BeforeFooterColumnCell">
<Storyboard BeginTime="0">
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_GridCellBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0" Value="0,0,0,1" />
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border Background="{TemplateBinding CellSelectionBrush}"
SnapsToDevicePixels="True"
Visibility="{TemplateBinding SelectionBorderVisibility}" />
<Border x:Name="PART_GridCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
SnapsToDevicePixels="True">
<ContentPresenter Margin="{TemplateBinding Padding}"
Opacity="{TemplateBinding AnimationOpacity}" />
</Border>
<Border Margin="0,0,1,1"
Background="Transparent"
BorderBrush="{TemplateBinding CurrentCellBorderBrush}"
BorderThickness="{TemplateBinding CurrentCellBorderThickness}"
IsHitTestVisible="False"
SnapsToDevicePixels="True"
Visibility="{TemplateBinding CurrentCellBorderVisibility}" />
<Border x:Name="PART_InvalidCellBorder"
Width="10"
Height="10"
HorizontalAlignment="Right"
VerticalAlignment="Top"
SnapsToDevicePixels="True"
Visibility="Collapsed">
<ToolTipService.ToolTip>
<ToolTip Background="#FFDB000C"
Placement="Right"
PlacementRectangle="20,0,0,0"
Tag="{TemplateBinding ErrorMessage}"
Template="{StaticResource ValidationToolTipTemplate}" />
</ToolTipService.ToolTip>
<Path Data="M0.5,0.5 L12.652698,0.5 12.652698,12.068006 z"
Fill="Red"
SnapsToDevicePixels="True"
Stretch="Fill" />
</Border>

```

```

</Grid>
</ControlTemplate>
<Style TargetType="Syncfusion:GridHeaderCellControl">
<Setter Property="BorderThickness" Value="0.4"/>
</Style>
<Style TargetType="Syncfusion:GridCell">
<Setter Property="BorderThickness" Value="0.4"/>
<Setter Property="Template" Value="{StaticResource
GridCellControlTemplate}"/>
</Style>
</Window.Resources>

```

You can download a working demo for the above customization from [here](#).

### Limitations

1. Non editable columns will not support custom validation except [GridCheckBoxColumn](#).
2. `CurrentCellValidating` event will not triggered for [GridTemplateColumn](#) and [GridUnboundColumn](#).

See Also

[How to navigate to the error cells in datagrid via button click?](#)

[How to validate the AddNewRow value based on already existing records?](#)

[How to show the validation tooltip without hovering the red indicator in cell?](#)

[How to fire RowValidating event for GridCheckBoxColumn in SfDataGrid](#)

[How to remove the top-right corner error mark from the GridCell by pressing Esc key when validated by handling the CurrentCellValidating event?](#)

[How to change the validation error template color?](#)

[How to wire the RowValidating event after pasted the content to datagrid?](#)

### CRUD Operations in WPF DataGrid (SfDataGrid)

DataGrid listens and responds to the CRUD operations such as add, delete and data update (property change) at runtime. Also, it supports [editing](#), [add new row](#), [delete row](#) by pressing Delete key.

### Managing data updates

DataGrid manages the sorting, filtering, grouping and summaries during data updates based on [SfDataGrid.LiveDataUpdateMode](#) property.

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}"
LiveDataUpdateMode="AllowDataShaping" />

```

### C#

```

this.dataGrid.LiveDataUpdateMode = LiveDataUpdateMode.AllowDataShaping;

```

*LiveDataUpdateMode – Default*

Grid operations / Data Manipulation operations	Add	Remove / delete	Property change
Sorting	Record added at last	Updated	Sort order not updated
Grouping	Updated	Updated	Groups not refreshed based on change
Filtering	Updated	Updated	Filter not refreshed based on change
Summaries	Not updated	Not updated	Not updated

*LiveDataUpdateMode – AllowSummaryUpdate*

Grid operations/ Data Manipulation operations	Add	Remove / delete	Property change
Sorting	Record added at last	Updated	Sort order not updated
Grouping	Updated	Updated	Groups not refreshed based on change
Filtering	Updated	Updated	Filter not refreshed based on change
Summaries	Updated	Updated	Updated

*LiveDataUpdateMode – AllowDataShaping*

Grid operations/ Data Manipulation operations	Add	Remove / delete	Property change
Sorting	Updated	Updated	Updated
Grouping	Updated	Updated	Updated
Filtering	Updated	Updated	Updated
Summaries	Updated	Updated	Updated

*Limitations*

- **AllowDataShaping** and **AllowSummaryUpdate** is not supported when you are binding with dynamic data objects.
- Complex and indexer properties doesn't support **LiveDataUpdateMode - AllowDataShaping** and **AllowSummaryUpdate**.
- **LiveDataUpdateMode** is not supported when **DataTable** is **ItemsSource**.

### Add new rows

DataGrid provides built-in row (called AddNewRow) that allows user to add new records to underlying collection. Built-in add new row can be enabled or disabled by setting [SfDataGrid.AddNewRowPosition](#) property. **AddNewRowPosition** also denotes the position of add new row in DataGrid.

When you start editing in AddNewRow, the SfDataGrid control creates an instance for the underlying data object and adds it to underlying collection when editing completed.

**Note:** The underlying data object must be defined with default constructor.


### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AddNewRowPosition="Top"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}" />
```

### C#

```
this.dataGrid.AddNewRowPosition = AddNewRowPosition.Top;
```

**AddNewRow**



OrderID	CustomerID	CustomerName
Click here to add a new row		
1001.00	ALFKI	Maria Anders
1002.00	ANATR	Ana Trujilo
1003.00	ANTON	Antonio Moreno
1004.00	AROUT	Thomas Hardy
1005.00	BERGS	Christina Berglund
1006.00	BLAUS	Hanna Moos
1007.00	BLONP	Frédérique Citeaux
1008.00	BOLID	Martin Sommer
1009.00	BONAP	Laurence Lebihan
1010.00	BOTTM	Elizabeth Lincoln

You can get the row index of AddNewRow using [GridAddNewRowController.GetAddNewRowIndex](#) method.

### C#

```
var addNewRowController=this.dataGrid.GetAddNewRowController();
int addNewRowIndex = addNewRowController.GetAddNewRowIndex();
```

You can check whether the specified row index is AddNewRow index, by using [SfDataGrid.IsAddNewIndex](#) helper method.

**C#**

```
bool isAddNewRowIndex = this.dataGrid.IsAddNewIndex(1);
```

*Changing the AddNewRow default text in DataGrid*

You can change the default static string of AddNewRow in datagrid by using the [SfDataGrid.AddNewRowText](#) property. The `AddNewRowText` property has higher priority than the text that is localized in resx file.

**XML**

```
<Syncfusion:SfDataGrid x:Name="dataGrid"
AddNewRowPosition="Top"
AddNewRowText="Click here to add new row in datagrid"
ItemsSource="{Binding Employees}" />
```

**C#**

```
this.dataGrid.AddNewRowPosition = AddNewRowPosition.Top;
this.dataGrid.AddNewRowText = "Click here to add new row in datagrid";
```

Employee ID	Order ID	City
Click here to add new row in datagrid		
1	1001	Berlin
2	1002	Mexico D.F.
3	1003	London
4	1004	BERGS
5	1005	Mannheim
6	1006	Berlin
7	1007	Mexico D.F.
8	1008	London
9	1009	BERGS
10	1010	Mannheim

*Customize the newly added row position*

SfDataGrid adds new data item from AddNewRow at the end of collection. When data operations (sorting, grouping) performed, the new item added based on data operations. You can customize the newly added data item position by setting [SfDataGrid.NewItemPlaceholderPosition](#).

**XML**

```
<Syncfusion:SfDataGrid x:Name="datagrid"
AddNewRowPosition="Top"
NewItemPlaceholderPosition="AtBeginning"
ItemsSource="{Binding OrderInfoCollection }">
```

**C#**

```
this.datagrid.AddNewRowPosition = AddNewRowPosition.Top;
this.datagrid.NewItemPlaceholderPosition =
NewItemPlaceholderPosition.AtBeginning;
```

#### Initializing default values for AddNewRow

SfDataGrid allows you to set the default values for AddNewRow while initiating, through [AddNewRowInitiatingEventArgs.NewObject](#) property in [SfDataGrid.AddNewRowInitiating](#) event.

#### C#

```
this.dataGrid.AddNewRowInitiating += dataGrid_AddNewRowInitiating;
void dataGrid_AddNewRowInitiating(object sender,
AddNewRowInitiatingEventArgs args)
{
    var data = args.NewObject as OrderInfo;
    data.OrderID = 101;
}
```

OrderID	CustomerID	CustomerName
101.00		
1001.00	ALFKI	Maria Anders
1002.00	ANATR	Ana Trujilo
1003.00	ANTON	Antonio Moreno
1004.00	AROUT	Thomas Hardy
1005.00	BERGS	Christina Berglund
1006.00	BLAUS	Hanna Moos
1007.00	BLONP	Frédérique Citeaux
1008.00	BOLID	Martin Sommer
1009.00	BONAP	Laurence Lebihan
1010.00	BOTTM	Elizabeth Lincoln

#### Working with complex properties in AddNewRow

SfDataGrid control does not initiate values for complex properties defined in the data object. Hence, you need to initiate the default values for the complex properties externally by using the [SfDataGrid.AddNewRowInitiating](#) event.

#### XML

```
<syncfusion:SfDataGrid AddNewRowInitiating="SfDataGrid_AddNewRowInitiating"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="OrderID" />
<syncfusion:GridTextColumn MappingName="Customer.CustomerID" />
<syncfusion:GridTextColumn MappingName="ShipCity" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```



**C#**

```
this.dataGrid.AddNewRowInitiating += dataGrid_AddNewRowInitiating;
private void SfDataGrid_AddNewRowInitiating(object sender,
Syncfusion.UI.Xaml.Grid.AddNewRowInitiatingEventArgs args)
{
    var data = args.NewObject as OrderInfo;
    data.Customer = new Customer();
}
```

*Add row programmatically*

You can commit or cancel the new record in AddNewRow by pressing the Enter and Esc key respectively. AddNewRow operations can be performed programmatically by using [GridAddNewRowController.CommitAddNew](#) and [GridAddNewRowController.CancelAddNew](#) methods at runtime.

*Cancel AddNewRow***C#**

```
using Syncfusion.UI.Xaml.Grid.Helpers;
//Check whether the data is newly added
if (this.dataGrid.View.IsAddingNew)
{
    //Which end edit the current cell. By passing false, it revert the entered value.
    if
    (this.dataGrid.SelectionController.CurrentCellManager.CurrentCell.IsEditing)
        this.dataGrid.SelectionController.CurrentCellManager.EndEdit(true);
    var addNewRowController = this.dataGrid.GetAddNewRowController();
    addNewRowController.CancelAddNew();
}
```

*Commit AddNewRow***C#**

```
RowColumnIndex rowColumnIndex = new RowColumnIndex();
if (this.dataGrid.View.IsAddingNew)
{
    if
    (this.dataGrid.SelectionController.CurrentCellManager.CurrentCell.IsEditing)
        this.dataGrid.SelectionController.CurrentCellManager.EndEdit(true);
    rowColumnIndex =
    this.dataGrid.SelectionController.CurrentCellManager.CurrentRowColumnIndex;
    //Process the commit operation in AddNewRow.
    var addNewRowController = this.dataGrid.GetAddNewRowController();
    addNewRowController.CommitAddNew();
    //Gets the row index of AddNewRow
    rowColumnIndex.RowIndex = addNewRowController.GetAddNewRowIndex();
    this.dataGrid.SelectedItems.Clear();
    //If the AddNewRowPosition is Top need to move the current cell to next row
    if (this.dataGrid.AddNewRowPosition == AddNewRowPosition.Top)
        rowColumnIndex.RowIndex = rowColumnIndex.RowIndex + 1;
```

```
//Which retains the current cell border in the row after canceling AddNewRow
as you press ESC key operation.
this.dataGrid.MoveCurrentCell(rowColumnIndex);
}
```

#### Validating AddNewRow

You can validate the data in AddNewRow like other data rows through [built-in validation](#) or [custom validation](#). Here, AddNewRow is validated using [RowValidating](#) event by setting `RowValidatingEventArgs.IsValid` to `false` which doesn't allow users to commit the AddNewRow until the validation gets succeeded.

#### C#

```
this.dataGrid.RowValidating += dataGrid_RowValidating;
void dataGrid_RowValidating(object sender, RowValidatingEventArgs args)
{
    if(this.dataGrid.IsAddNewIndex(args.RowIndex))
    {
        var data = args.RowData as OrderInfo;
        if (data.OrderID >= 1010)
        {
            args.IsValid = false;
            args.ErrorMessages.Add("OrderID", "OrderID should not exceed 1010.");
        }
    }
}
```

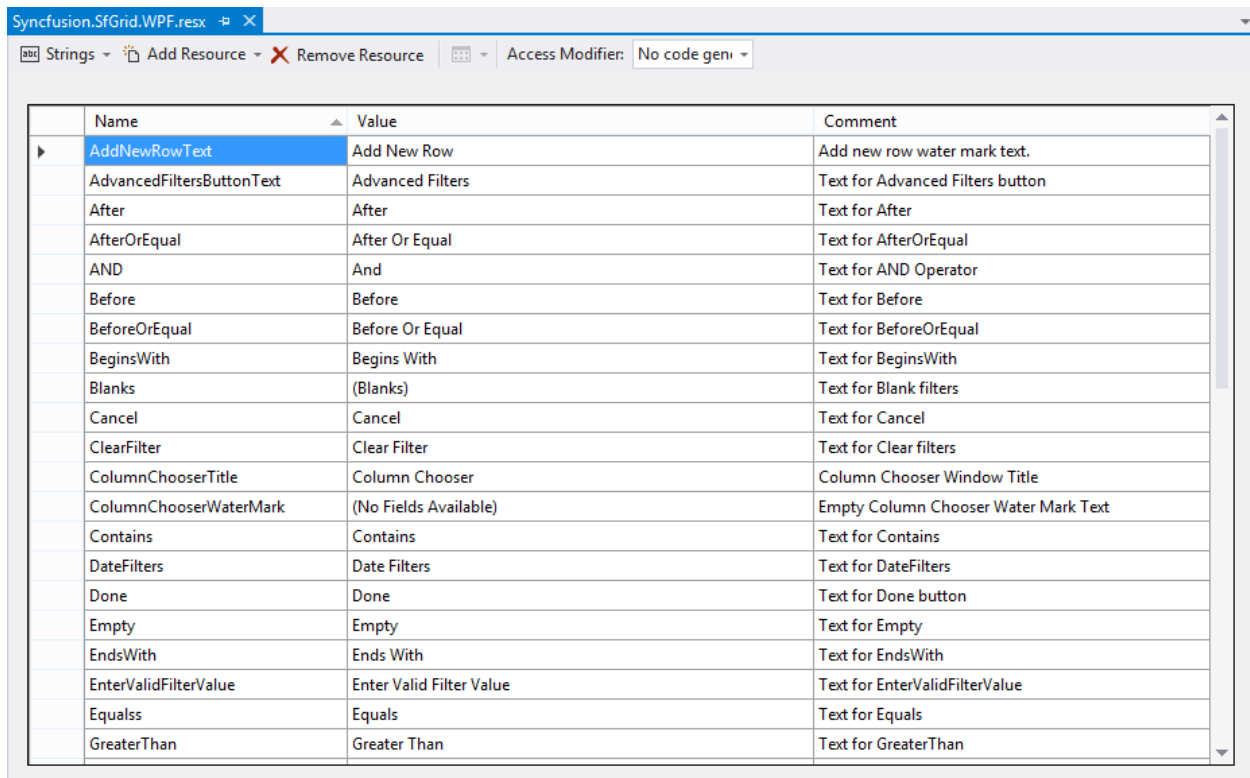
CustomerID	CustomerName	OrderID
ZENKI	Anderson	1011
ALFKI	Maria Anders	1001
ANATR	Ana Trujilo	1002
ANTON	Antonio Moreno	1003
AROUT	Thomas Hardy	1004
BERGS	Christina Berglund	1005
BLAUS	Hanna Moos	1006
BLONP	Frédérique Citeaux	1007
BOLID	Martin Sommer	1008
BONAP	Laurence Lebihan	1009
BOTTM	Elizabeth Lincoln	1010

Similarly, you can validate the cells in AddNewRow by using the [CurrentCellValidating](#) event.

#### Customizing AddNewRow text using default resource file

SfDataGrid enables you to customize the watermark text of AddNewRow by changing value of `AddNewRowText` in Resource Designer. For more information, you can refer [Editing default culture resource](#) section.

To customize the `AddNewRowText`, add the default `Syncfusion.SfDataGrid.WPF.resx` file in **Resources** folder and then customize the value of `AddNewRowText`. Refer [here](#) to learn more about localization.



CustomerID	CustomerName	OrderID
Add New Row		
ALFKI	Maria Anders	1001
ANATR	Ana Trujilo	1002
ANTON	Antonio Moreno	1003
AROUT	Thomas Hardy	1004
BERGS	Christina Berglund	1005
BLAUS	Hanna Moos	1006
BLONP	Frédérique Citeaux	1007
BOLID	Martin Sommer	1008
BONAP	Laurence Lebihan	1009
BOTTM	Elizabeth Lincoln	1010

#### Customizing AddNewRow text using style

You can customize the watermark text of AddNewRow by editing the style of `AddNewRowControl` and change the content of `PART_AddNewRowTextBorder`'s `ContentPresenter`.

#### XML

```
<Application.Resources>
<Style TargetType="syncfusion:AddNewRowControl">
<Setter Property="Foreground" Value="White" />
<Setter Property="BorderBrush" Value="Gray" />
<Setter Property="BorderThickness" Value="0" />

```

```

<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:AddNewRowControl">
<Grid>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="AddNewRowStates">
<VisualState x:Name="Normal" />
<VisualState x:Name="Edit">
<Storyboard>
<ObjectAnimationUsingKeyFrames
Storyboard.TargetName="PART_AddNewRowTextBorder"
Storyboard.TargetProperty="(UIElement.Visibility)">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Visibility>Collapsed</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="BorderStates">
<VisualState x:Name="NormalRow" />
<VisualState x:Name="FooterRow">
<Storyboard BeginTime="0">
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_AddNewRowBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0" Value="0, 1, 0, 0" />
</ThicknessAnimationUsingKeyFrames>
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_AddNewRowBorder"
Storyboard.TargetProperty="Margin">
<EasingThicknessKeyFrame KeyTime="0" Value="0, -1, 0, 0" />
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Rectangle x:Name="PART_CurrentFocusRow"
Margin="2,2,0,2"
HorizontalAlignment="Right"
Stroke="DarkGray"
StrokeDashArray="2,2"
StrokeThickness="1"
Visibility="{TemplateBinding CurrentFocusRowVisibility}" />
<Border Background="{TemplateBinding RowSelectionBrush}"
Clip="{TemplateBinding SelectionBorderClipRect}"
Visibility="{TemplateBinding SelectionBorderVisibility}" />
<Border x:Name="PART_AddNewRowBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}">
<ContentPresenter />
</Border>

```

```

<Border x:Name="PART_AddNewRowTextBorder"
Background="#FFE5E5E5"
BorderBrush="Transparent"
BorderThickness="0,0,1,1"
Clip="{TemplateBinding TextBorderClip}"
IsHitTestVisible="False">
<ContentPresenter Margin="{TemplateBinding TextMargin}"
HorizontalAlignment="Left"
VerticalAlignment="Center"
Content="Add New Row" />
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Application.Resources>

```

CustomerID	CustomerName	OrderID
AddNewRow text		
ALFKI	Maria Anders	1001
ANATR	Ana Trujilo	1002
ANTON	Antonio Moreno	1003
AROUT	Thomas Hardy	1004
BERGS	Christina Berglund	1005
BLAUS	Hanna Moos	1006
BLONP	Frédérique Citeaux	1007
BOLID	Martin Sommer	1008
BONAP	Laurence Lebihan	1009
BOTTM	Elizabeth Lincoln	1010

#### AddNewRow support in Master-Details View

You can enable the AddNewRow in `DetailsViewDataGrid` by specifying the position to [SfDataGrid.AddNewRowPosition](#) property in [ViewDefinition.DataGrid](#).

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="firstLevelNestedGrid"
AddNewRowPosition="Top"
AutoGenerateColumns="True" />
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>

```

```
</syncfusion:SfDataGrid.DetailsViewDefinition>  
</syncfusion:SfDataGrid>
```

### C#

```
this.firstLevelNestedGrid.AddNewRowPosition = AddNewRowPosition.Top;
```

Similarly, you can wire [AddNewRowInitiating](#) event for `ViewDefinition.DataGrid`.

### C#

```
this.FirstLevelNestedGrid.AddNewRowInitiating +=  
FirstLevelNestedGrid_AddNewRowInitiating;  
void FirstLevelNestedGrid_AddNewRowInitiating(object sender,  
AddNewRowInitiatingEventArgs args)  
{  
}
```

For auto-generated relation (when the [AutoGenerateRelations](#) is set to `true`), the `AddNewRow` can be enabled by specifying the position to `AddNewRowPosition` property in [AutoGeneratingRelations](#) event.

### C#

```
this.dataGrid.AutoGeneratingRelations+=dataGrid_AutoGeneratingRelations;  
void dataGrid_AutoGeneratingRelations(object sender,  
Syncfusion.UI.Xaml.Grid.AutoGeneratingRelationsArgs e)  
{  
e.GridViewDefinition.DataGrid.AddNewRowPosition = AddNewRowPosition.Top;  
}
```

In the same way, you can wire [AddNewRowInitiating](#) event in the [AutoGeneratingRelations](#) event.

### C#

```
this.dataGrid.AutoGeneratingRelations+=dataGrid_AutoGeneratingRelations;  
void dataGrid_AutoGeneratingRelations(object sender,  
Syncfusion.UI.Xaml.Grid.AutoGeneratingRelationsArgs e)  
{  
e.GridViewDefinition.DataGrid.AddNewRowInitiating +=  
DataGrid_AddNewRowInitiating;  
}  
void DataGrid_AddNewRowInitiating(object sender,  
AddNewRowInitiatingEventArgs args)  
{  
}
```

### *Changing the AddNewRow default text in details view grid*

You can change the default static string of `AddNewRow` in details view grid by using the [SfDataGrid.AddNewRowText](#) property in `ViewDefinition.DataGrid`. The `AddNewRowText` property has higher priority than the text that is localized in resx file.

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Employees}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="firstLevelNestedGrid"
AddNewRowPosition="Top"
AddNewRowText="Click here to add new row in child grid"
AutoGenerateColumns="True" />
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>

```

**C#**

```

this.firstLevelNestedGrid.AddNewRowPosition = AddNewRowPosition.Top;
this.firstLevelNestedGrid.AddNewRowText = "Click here to add new row in
child grid";

```

	Employee ID	Order ID	City																		
—	1	1001	Berlin																		
<table><tr><td>Order ID</td><td>Sales ID</td><td>Product Name</td></tr><tr><td colspan="3">Click here to add new row in child grid</td></tr><tr><td>1001</td><td>A00001</td><td>Bike</td></tr><tr><td>1001</td><td>A00002</td><td>Cycle</td></tr><tr><td>1001</td><td>A00003</td><td>Cycle</td></tr><tr><td>1001</td><td>A00004</td><td>Car</td></tr></table>				Order ID	Sales ID	Product Name	Click here to add new row in child grid			1001	A00001	Bike	1001	A00002	Cycle	1001	A00003	Cycle	1001	A00004	Car
Order ID	Sales ID	Product Name																			
Click here to add new row in child grid																					
1001	A00001	Bike																			
1001	A00002	Cycle																			
1001	A00003	Cycle																			
1001	A00004	Car																			
—	2	1002	Mexico D.F.																		
<table><tr><td>Order ID</td><td>Sales ID</td><td>Product Name</td></tr><tr><td colspan="3">Click here to add new row in child grid</td></tr><tr><td>1002</td><td>A00004</td><td>Car</td></tr><tr><td>1002</td><td>A00002</td><td>Bike1</td></tr><tr><td>1002</td><td>A00004</td><td>Car</td></tr></table>				Order ID	Sales ID	Product Name	Click here to add new row in child grid			1002	A00004	Car	1002	A00002	Bike1	1002	A00004	Car			
Order ID	Sales ID	Product Name																			
Click here to add new row in child grid																					
1002	A00004	Car																			
1002	A00002	Bike1																			
1002	A00004	Car																			
+	3	1003	London																		

**Delete row**

DataGrid provides built-in support to delete the selected records in user interface (UI) by pressing Delete key. You can enable the deleting support by setting the `SfDataGrid.AllowDeleting` property to true. AllowDeleting is only supported when SelectionUnit is Row.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AllowDeleting="True"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}" />
```

### C#

```
this.dataGrid.AllowDeleting = true;
```

You can delete record directly in underlying collection also using `Remove ()` or `RemoveAt (int index)`.

### C#

```
(dataGrid.DataContext as ViewModel).Orders.Remove(dataGrid.CurrentItem as
OrderInfo);
// OR
(dataGrid.DataContext as ViewModel).Orders.RemoveAt(2);
```

### Events

#### RecordDeleting

[RecordDeleting](#) event occurs when the record is being deleted from SfDataGrid. The [RecordDeletingEventArgs](#) provides information to [RecordDeleting](#) event for deleting the record and it contains the following members.

- [Cancel](#) - Gets or sets a value indicating whether the event should be canceled.
- [OriginalSender](#) - Gets the original sender from where the event is raised.
- [Items](#) - Gets or sets the items to be removed from the source collection.

You can cancel the delete operation through [Cancel](#) property in [RecordDeleting](#) event.

### C#

```
this.dataGrid.RecordDeleting += dataGrid_RecordDeleting;
void dataGrid_RecordDeleting(object sender, RecordDeletingEventArgs args)
{
    var item = args.Items[0] as OrderInfo;
    if (item.OrderID == 1005)
    {
        args.Cancel = true;
    }
}
```

#### RecordDeleted

[RecordDeleted](#) event occurs after the record is deleted. The [RecordDeletedEventArgs](#) of [RecordDeleted](#) event contains the following members.

- [Items](#) - Gets the records that were removed from the source collection.
- [SelectedIndex](#) - Gets or sets the selected index for the SfDataGrid control.



### Handling selection after deleting the record from SfDataGrid

You can handle the selection after remove the records through [SelectedIndex](#) property of [RecordDeleted](#) event.

#### C#

```
this.dataGrid.RecordDeleted += dataGrid_RecordDeleted;
void dataGrid_RecordDeleted(object sender, RecordDeletedEventArgs args)
{
    args.SelectedIndex = -1;
}
```

### Deleting cell value in display mode

By default, the cell content can be cleared in edit mode by pressing Delete or Backspace key. It is also possible to delete the cell when it's not in edit mode by handling the Delete key operation in the [ProcessKeyDown](#) method of [GridSelectionController](#) or [GridCellSelectionController](#). Based on type of [SelectionUnit](#), override right selection controller.

#### C#

```
this.dataGrid.SelectionController = new
GridSelectionControllerExt(dataGrid);
public class GridSelectionControllerExt : GridSelectionController
{
    public GridSelectionControllerExt(SfDataGrid dataGrid) : base(dataGrid)
    {
    }
    protected override void ProcessKeyDown(KeyEventArgs args)
    {
        //Customizes the Delete key operation.
        if (args.Key == Key.Delete)
        {
            //Gets the cell value of current column.
            var record = this.DataGrid.CurrentItem;
            var columnIndex = this.CurrentCellManager.CurrentCell.ColumnIndex;
            var columnIndex =
                this.DataGrid.ResolveToGridVisibleColumnIndex(currentColumnIndex);
            var mappingName = this.DataGrid.Columns[columnIndex].MappingName;
            var cellVal =
                this.DataGrid.View.GetPropertyAccessProvider().GetValue(record,
                mappingName);
            //Returns the cell value when the current column's cell is not set to null.
            if (cellVal != null)
            {
                PropertyDescriptorExtensions.SetValue(this.DataGrid.View.GetItemProperties()
                , record, null, mappingName);
            }
        }
        else
        {
            base.ProcessKeyDown(args);
        }
    }
}
```

### *Conditionally deleting records when pressing Delete key*

You can cancel the record deletion by using the [RecordDeletingEventArgs.Cancel](#) of `RecordDeleting` event. You can skip certain records when deleting more than one record by removing items from `RecordDeletingEventArgs.Items`.

#### **C#**

```
this.dataGrid.RecordDeleting += dataGrid_RecordDeleting;
void dataGrid_RecordDeleting(object sender, RecordDeletingEventArgs args)
{
    foreach(var item in args.Items)
    {
        if((item as OrderInfo).OrderID==1001)
        {
            args.Cancel = true;
        }
    }
}
```

### Selection in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) (SfDataGrid) allows you to select one or more rows or cells. For selecting specific row or group of rows you have to set [SelectionUnit](#) as [Row](#) and for selecting a specific cell or group of cells you have to set [SelectionUnit](#) as [Cell](#) or [Any](#). In [SelectionUnit.Any](#) option you can select the row by clicking on row header.

#### *Current Cell Navigation*

Keyboard navigation through the cells and rows is determined based on the [NavigationMode](#) property. [NavigationMode.Cell](#) allows you to navigate between the cells in a row as well as between rows. [NavigationMode.Row](#) allows you to navigate only between rows. It is not possible to set [NavigationMode.Row](#) when cell selection is enabled ([SelectionUnit](#) is [Cell](#) or [Any](#)).

#### *Selection Modes*

The [SelectionUnit](#) and [SelectionMode](#) properties together define the behavior of selection in SfDataGrid. If the [SelectionMode](#) is [Single](#), you can able to select single row or cell, and if the [SelectionMode](#) is [Extended](#) or [Multiple](#), you can able to select multiple rows or cell, and if you want to disable the selection you need to set [SelectionMode](#) as [None](#),

#### **XML**

```
<Syncfusion:SfDataGrid x:Name="dataGrid"
    SelectionUnit="Row"
    NavigationMode="Cell"
    SelectionMode="Single"
    ItemsSource="{Binding Orders}">
```

OrderID	CustomerID	CustomerName	Country	
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

#### Disable selection for rows and columns

You can disable selection and navigation on particular column by setting [GridColumn.AllowFocus](#) property. You can disable selection on particular row or cell or column by handling [CurrentCellActivating](#) event.

**Note:** It is not possible to select header rows, table summary rows, unbound rows which are above the table summary row when it's placed in top and the unbound rows which are below table summary rows when it's placed in bottom of SfDataGrid.

#### Multiple Row or Cell Selection

The WPF DataGrid (SfDataGrid) allows you to select multiple rows or cells by setting [SelectionMode](#) property as [Extended](#) or [Multiple](#), where you can select multiple rows or cells by dragging the mouse on SfDataGrid and also using the key modifiers.

While using [Extended](#), you can select multiple rows or cells by pressing the key modifiers Ctrl and Shift.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    SelectionUnit="Cell"
    NavigationMode="Cell"
    SelectionMode="Extended"
    ItemsSource="{Binding Orders}">
```

OrderID	CustomerID	CustomerName	Country	
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D
1003	ANTON	Antonio Moreno	Mexico	México D
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannhei
1007	BLONP	Frédérique Citeaux	France	Strasbou
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawasse

**Note:** When the [SelectionMode](#) as [Multiple](#), you can select or deselect multiple rows and cells by clicking the respective cell or row. Also in multiple selection pressing navigation keys will move only the current cell and you can select or deselect by pressing space key.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    SelectionUnit="Cell"
    NavigationMode="Cell"
    SelectionMode="Multiple"
    ItemsSource="{Binding Orders}">
```

OrderID	CustomerID	CustomerName	Country	
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D
1003	ANTON	Antonio Moreno	Mexico	México D
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannhei
1007	BLONP	Frédérique Citeaux	France	Strasbou
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawasse

#### CheckBox column selection

SfDataGrid allows you to select or deselect rows by interacting with the check box in a column. All the rows in a datagrid can be selected by interacting with an intuitive check box in the column header. Refer to [GridCheckBoxSelectorColumn](#) documentation for more information.

### Get Selected Rows and Cells

The [SelectedItem](#) property returns the data object of the selected row and the [SelectedIndex](#) property returns the index of the [SelectedItem](#) in SfDataGrid. [SelectedItem](#) denotes the first selected row in multiple selection.

The [CurrentItem](#) returns the data object that currently has the focus and the [CurrentColumn](#) denotes the [GridColumn](#) that currently has the focus. The [CurrentCellInfo](#) returns an instance [GridCellInfo](#) which contains the information about the cell that currently has the focus.

### Row Selection

You can get all the selected records through [SelectedItems](#) property and you can also get all selected rows information through [SfDataGrid.SelectionController.SelectedRows](#) which is the collection of [GridRowInfo](#).

### Cell Selection

You can get all selected cells information through [SfDataGrid.SelectionController.SelectedCells](#) property which is the collection of [GridSelectedCellsInfo](#).

You can get the selected cells as [GridCellInfo](#) collection by using [GetSelectedCells](#) method.

### C#

```
List<GridCellInfo> selectedCells = this.dataGrid.GetSelectedCells();
```

### CurrentItem vs SelectedItem

Both [SelectedItem](#) and [CurrentItem](#) returns the same data object when there is single cell or row is selected in SfDataGrid. When you have selected more than one rows or cells, the record that had been selected initially is maintained in [SelectedItem](#) and the record that currently have focus is maintained in [CurrentItem](#).

### Programmatic selection

#### Process selection using properties

You can select a single row or cell by setting [SelectedItem](#) property or [SelectedIndex](#) property.

### C#

```
var recordIndex = this.dataGrid.ResolveToRecordIndex(5);  
this.dataGrid.SelectedIndex = recordIndex;
```

### C#

```
var record = this.dataGrid.GetRecordAtRowIndex(6);  
this.dataGrid.SelectedItem = record;
```

In Row selection, you can select multiple rows by adding data objects to [SelectedItems](#) property.

### C#

```
var viewModel = this.dataGrid.DataContext as ViewModel;  
foreach (var order in viewModel.Orders)  
{  
    if (order.Country == "Mexico")  
        this.dataGrid.SelectedItems.Add(order);  
}
```

OrderID	CustomerID	CustomerName	Country	
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D
1003	ANTON	Antonio Moreno	Mexico	México D
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannhei
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawasse

*Process selection using methods*

You can select range of rows through [SelectRows](#) method in row selection.

**C#**

```
this.dataGrid.SelectRows(3, 7);
```

OrderID	CustomerID	CustomerName	Country	
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D
1003	ANTON	Antonio Moreno	Mexico	México D
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannhei
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawasse

You can select a specific cell by using the [SelectCell](#) method in cell selection.

**C#**

```
var record = this.dataGrid.GetRecordAtRowIndex(3);
var column = this.dataGrid.Columns[1];
this.dataGrid.SelectCell(record, column);
```

OrderID	CustomerID	CustomerName	Country	
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

You can select a range of cells through [SelectCells](#) method in cell selection.

### C#

```

public MainWindow()
{
    InitializeComponent();
    this.dataGrid.Loaded += dataGrid_Loaded;
    this.dataGrid.SelectionController = new
    GridSelectionControllerExt(this.dataGrid);
}
private void dataGrid_Loaded(object sender, RoutedEventArgs e)
{
    var firstRecord = this.dataGrid.GetRecordAtRowIndex(3);
    var lastRecord = this.dataGrid.GetRecordAtRowIndex(7);
    var firstColumn = this.dataGrid.Columns[1];
    var lastColumn = this.dataGrid.Columns[3];
    this.dataGrid.SelectCells(firstRecord, firstColumn, lastRecord, lastColumn);
    (this.dataGrid.SelectionController as
    GridSelectionControllerExt).rowColumnIndex = new RowColumnIndex(7, 3);
    //Updates the PressedRowIndex value in the
    GridBaseSelectionController.
    (this.dataGrid.SelectionController as
    GridSelectionControllerExt).UpdatePressedIndex();
}
public class GridSelectionControllerExt : GridCellSelectionController
{
    //Overrides the GridCellSelectionController class to update the
    PressedRowIndex.
    public GridSelectionControllerExt(SfDataGrid datagrid)
    : base(datagrid)
    {
    }
    private RowColumnIndex rowColumnIndex;
    public RowColumnIndex RowColumnIndex
    {
        get { return rowColumnIndex; }
        set { rowColumnIndex = value; }
    }
}

```

```

}
//Updates the PressedRowColumnIndex to maintain the ShiftSelection.
public void UpdatePressedIndex()
{
    this.isInShiftSelection = true;
    this.PressedRowColumnIndex = RowColumnIndex;
}
}

```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1011	ANATR	Ana Trujillo	Mexico	México D.F.
1012	BERGS	Christina Berglund	Sweden	Luleå
1013	BLONP	Frédérique Citeaux	France	Strasbourg
1014	ANTON	Antonio Moreno	Mexico	México D.F.

You can select all the rows or cells using [SelectAll](#) method.

### C#

```
this.dataGrid.SelectAll();
```

### Process Current Cell

When you set the [CurrentItem](#) to particular record, the [CurrentCell](#) will be moved to corresponding record when the [SelectionMode](#) is [Multiple](#) or [Extended](#) and the selection will added to the particular record item when the [SelectionMode](#) is [Single](#).

### C#

```

var viewModel = this.dataGrid.DataContext as ViewModel;
this.dataGrid.CurrentItem = viewModel.Orders.FirstOrDefault(order =>
order.Country == "Spain");

```

You can move the [CurrentCell](#) to a particular rowColumnIndex by using the [MoveCurrentCell](#) method.

### C#

```
this.dataGrid.MoveCurrentCell(new RowColumnIndex(3,2), false);
```



### Clear Selection

You can clear the selection by using the [ClearSelection](#) method. In Row Selection you can also remove the selection by setting null to [SelectedItem](#) or clearing the [SelectedItems](#) property.

#### C#

```
this.dataGrid.SelectionController.ClearSelections(true);
```

You can clear selection on group of cells by using the [UnSelectCells](#) method in cell selection.

#### C#

```
var firstRecord = this.dataGrid.GetRecordAtRowIndex(3);
var lastRecord = this.dataGrid.GetRecordAtRowIndex(7);
var firstColumn = this.dataGrid.Columns[1];
var lastColumn = this.dataGrid.Columns[3];
this.dataGrid.UnSelectCells(firstRecord, firstColumn, lastRecord,
lastColumn);
```

You can clear the selection on particular cell by using the [UnSelectCell](#) method in cell selection.

#### C#

```
var removeRecord = this.dataGrid.GetRecordAtRowIndex(5);
var removeColumn = this.dataGrid.Columns[2];
this.dataGrid.UnSelectCell(removeRecord, removeColumn);
```

OrderID	CustomerID	CustomerName	Country	
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

### Selection in Master-Details View

Master-Details View provides support to select one or more rows or cells in [DetailsViewDataGrid](#). You can't able to maintain the selection in both ParentDataGrid and [DetailsViewDataGrid](#). Selection will be maintained either in ParentDataGrid or in [DetailsViewDataGrid](#).

	OrderID	EmployeeID	ContactID	BirthDate	SickLeaveHours
+	Click here to add a new row				
- 1	1001	(98)-9678	5/3/1991	24	
>	OrderID	CustomerID	UnitPrice	Quantity	Unbound Column1
	1002	702	\$14.00	4	2004
	1006	706	\$18.00	8	2012
	> 1010	710	\$22.00	12	2020
	1014	714	\$26.00	16	2028
	1018	718	\$30.00	20	2036
	1022	722	\$34.00	24	2044
- 2	1002	(78)-9878	5/4/1991	36	
	OrderID	CustomerID	UnitPrice	Quantity	Unbound Column1
	1026	726	\$38.00	29	2052
	1030	730	\$42.00	33	2060
	1034	734	\$46.00	37	2068
	1038	738	\$50.00	41	2076
	1042	742	\$54.00	45	2084
	1046	746	\$58.00	49	2092

#### Getting SelectedDetailsViewDataGrid

You can get the currently selected [DetailsViewDataGrid](#) by using the [SelectedDetailsViewGrid](#) property of parent DataGrid.

#### C#

```
var detailsViewDataGrid = this.dataGrid.SelectedDetailsViewGrid;
```

For accessing nested level [SelectedDetailsViewGrid](#) ,

#### C#

```
var detailsViewDataGrid =
this.dataGrid.SelectedDetailsViewGrid.SelectedDetailsViewGrid;
```

#### Getting SelectedItem from DetailsViewDataGrid

You can get the selected record of [DetailsViewDataGrid](#) by using the [SelectedItem](#) property.

#### C#

```
var detailsViewDataGrid = this.datagrid.SelectedDetailsViewGrid;
var SelectedItem = detailsViewDataGrid.SelectedItem;
```

You can get the [SelectedItem](#) while it's changed using [SelectionChanged](#) event of [ViewDefinition.DataGrid](#).

#### XML

```
<syncfusion:GridViewDefinition RelationalColumn="OrderDetails">
  <syncfusion:GridViewDefinition.DataGrid>
    <syncfusion:SfDataGrid x:Name="FirstDetailsViewGrid"
      SelectionChanged="FirstDetailsViewGrid_SelectionChanged">
```

```

</syncfusion:SfDataGrid>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>

```

**C#**

```

FirstDetailsViewGrid.SelectionChanged+=FirstDetailsViewGrid_SelectionChanged
;
private void FirstDetailsViewGrid_SelectionChanged(object sender,
Syncfusion.UI.Xaml.Grid.GridSelectionChangedEventArgs e)
{
var SelectedItem = (e.OriginalSender as DetailsViewDataGrid).SelectedItem;
}

```

**Note:** You can get the `SelectedIndex` and `SelectedItems` also in `SelectionChanged` event.

You can refer [here](#) to wire the events for `ViewDefinition.DataGrid` based on `AutoPopulateRelations` for different levels.

*Get the CurrentItem of DetailsViewDataGrid*

You can get the current record of the [DetailsViewDataGrid](#) by using the [CurrentItem](#) property.

**C#**

```

var detailsViewDataGrid = this.datagrid.SelectedDetailsViewGrid;
var CurrentItem = detailsViewDataGrid.CurrentItem;

```

You can get the `CurrentItem` while it's changed using [SelectionChanged](#) event of `ViewDefinition.DataGrid`.

**XML**

```

<syncfusion:GridViewDefinition RelationalColumn="OrderDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstDetailsViewGrid"
SelectionChanged="FirstDetailsViewGrid_SelectionChanged">
</syncfusion:SfDataGrid>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>

```

**C#**

```

FirstDetailsViewGrid.SelectionChanged+=FirstDetailsViewGrid_SelectionChanged
;
private void FirstDetailsViewGrid_SelectionChanged(object sender,
Syncfusion.UI.Xaml.Grid.GridSelectionChangedEventArgs e)
{
var SelectedItem = (e.OriginalSender as DetailsViewDataGrid).CurrentItem;
}

```

You can refer [here](#) to wire the events from `ViewDefinition.DataGrid` based on `AutoPopulateRelations` for different levels.

*Get CurrentCell of DetailsViewDataGrid*

You can get the [CurrentCell](#) of [DetailsViewDataGrid](#) by using the [SelectedDetailsViewGrid](#) property. You can use different events of [ViewDefinition.DataGrid](#) like [CurrentCellBeginEdit](#), [CurrentCellActivated](#) to get the [CurrentCell](#).

**C#**

```
var currentCell =
this.dataGrid.SelectedDetailsViewGrid.SelectionController.CurrentCellManager
.CurrentCell;
```

You can get the [CurrentCell](#) using [CurrentCellBeginEdit](#) event [ViewDefinition.DataGrid](#).

**XML**

```
<syncfusion:GridViewDefinition RelationalColumn="OrderDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
CurrentCellBeginEdit=" FirstLevelNestedGrid_CurrentCellBeginEdit">
</syncfusion:SfDataGrid>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
```

**C#**

```
this.FirstLevelNestedGrid.CurrentCellBeginEdit +=
FirstLevelNestedGrid_CurrentCellBeginEdit;
void FirstLevelNestedGrid_CurrentCellBeginEdit(object sender,
CurrentCellBeginEditEventArgs args)
{
var detailsViewDataGrid = args.OriginalSender as DetailsViewDataGrid;
var currentCell =
detailsViewDataGrid.SelectionController.CurrentCellManager.CurrentCell;
}
```

You can refer [here](#) to wire the events from the [ViewDefinition.DataGrid](#) based on [AutoPopulateRelations](#) for different levels.

*Programmatic Selection in DetailsViewDataGrid*

You can select data objects while loading [DetailsViewDataGrid](#) using [DetailsViewLoading](#) event.

**XML**

```
<syncfusion:SfDataGrid Name="dataGrid"
DetailsViewLoading="dataGrid_DetailsViewLoading" >
</syncfusion:SfDataGrid>
```

**C#**

```
this.dataGrid.DetailsViewLoading += dataGrid_DetailsViewLoading;
void dataGrid_DetailsViewLoading(object sender,
DetailsViewLoadingAndUnloadingEventArgs e)
{
```

```
var record =  
e.DetailsViewDataGrid.GetRecordAtRowIndex(e.DetailsViewDataGrid.GetFirstData  
RowIndex());  
e.DetailsViewDataGrid.SelectedItem = record;  
}
```

#### *Getting the parent of DetailsViewDataGrid*

You can get the immediate parent of `DetailsViewDataGrid` through [GetParentDataGrid](#) helper method.

#### **C#**

```
using Syncfusion.UI.Xaml.Grid.Helpers;  
var parentDataGrid =  
this.dataGrid.SelectedDetailsViewGrid.GetParentDataGrid();
```

You can get the top level DataGrid for the `DetailsViewDataGrid` through the [GetTopLevelParentDataGrid](#) helper method.

#### **C#**

```
using Syncfusion.UI.Xaml.Grid.Helpers;  
var dataGrid = this.detailsViewDataGrid.GetTopLevelParentDataGrid();
```

#### *Getting the DetailsViewDataGrid based on index*

You can get the [DetailsViewDataGrid](#) based on row index through [GetDetailsViewGrid](#) helper method.

#### **C#**

```
var detailsViewDataGrid = this.dataGrid.GetDetailsViewGrid(2);
```

You can also get the `DetailsViewDataGrid` based on the record index and relational column name using `GetDetailsViewGrid` method.

#### **C#**

```
var detailsViewDataGrid = this.dataGrid.GetDetailsViewGrid(0,  
"ProductDetails");
```

#### *Programmatically expanding and scrolling DetailsViewDataGrid*

You can expand the `DetailsViewDataGrid` programmatically by calling `ExpandDetailsViewAt` method by passing the record index.

#### **C#**

```
int parentRowIndex = 2;  
var recordIndex = this.dataGrid.ResolveToRecordIndex(parentRowIndex);  
var record = this.dataGrid.View.Records[recordIndex];  
if (!record.IsExpanded)  
this.dataGrid.ExpandDetailsViewAt(recordIndex);
```

If the expanded `DetailsViewDataGrid` is not in view, then you can scroll using [DetailsViewManager.BringIntoView](#) method.

**C#**

```

int recordIndex = 20;
int index = 0;
int parentRowIndex = 25;
datagrid.ExpandDetailsViewAt(recordIndex);
foreach (var def in this.dataGrid.DetailsViewDefinition)
{
    if (def.RelationalColumn == "ProductDetails")
    {
        index = this.dataGrid.DetailsViewDefinition.IndexOf(def);
        index = parentRowIndex + index + 1;
    }
}
//Get the Details view based upon the recordIndex and Column name
SfDataGrid detailsViewDataGrid = datagrid.GetDetailsViewGrid(recordIndex,
"OrderDetails");
//Get the DetailsViewManager using Reflection
var propertyInfo = dataGrid.GetType().GetField("DetailsViewManager",
System.Reflection.BindingFlags.Instance |
System.Reflection.BindingFlags.NonPublic);
DetailsViewManager detailsViewManager = propertyInfo.GetValue(dataGrid) as
DetailsViewManager;
if (detailsViewDataGrid == null)
{
    detailsViewManager.BringIntoView(index);
    detailsViewDataGrid =
this.dataGrid.GetDetailsViewGrid(this.dataGrid.ResolveToRecordIndex(recordIndex), " OrderDetails ");
}

```

SfDataGrid Demo

	OrderID	CustomerID	Shipping Date	EmployeeID	Ship City	
+	10012	PICCO	3/10/2008	1	Tsawassen	C
+	10013	WARTH	8/30/2008	9	Campinas	E
+	10014	FOLIG	6/21/2010	9	Bruxelles	E
+	10015	WARTH	5/20/2008	4	Bruxelles	E
+	10016	LINOD	10/9/2011	1	Rio de Janeiro	E
+	10017	MEREP	5/19/2011	1	Graz	A
+	10018	PICCO	10/22/2008	1	Tsawassen	C
+	10019	MEREP	3/22/2011	7	Buenos Aires	A
+	10020	FURIB	3/29/2010	8	Resende	E
+	10021	MEREP	6/3/2009	5	Graz	A
-	10022	RISCU	6/5/2009	8	Resende	E

OrderID	ProductID	Unit Price	Quantity	Discount	
10022	59	80	2	0	SIMOB
10022	60	111	2	7	FOLIG
10022	47	35	2	0	FRANS

Select Child Record

You can get the sample from [here](#).

*Programmatically select the records in DetailsViewDataGrid which is not in view*

Scrolling to the DetailsViewDataGrid

You can expand the [DetailsViewDataGrid](#) by using [ExpandDetailsViewAt](#) helper method. If the [DetailsViewDataGrid](#) is already expanded, you can use [ScrollInView](#) method to bring it into view. You can also use [DetailsViewManager.BringIntoView](#) method to get the [DetailsViewDataGrid](#) into view.

**C#**

```
//Find DetailsViewDataRow index based on relational column
int index = 0;
int parentRowIndex = 25;
foreach (var def in this.dataGrid.DetailsViewDefinition)
{
    if (def.RelationalColumn == "ProductDetails")
    {
        index = this.dataGrid.DetailsViewDefinition.IndexOf(def);
        index = parentRowIndex + index + 1;
    }
}
//Get the DetailsViewManager using Reflection
var propertyInfo = dataGrid.GetType().GetField("DetailsViewManager",
System.Reflection.BindingFlags.Instance |
System.Reflection.BindingFlags.NonPublic);
DetailsViewManager detailsViewManager = propertyInfo.GetValue(dataGrid) as
DetailsViewManager;
var rowColumnIndex = new RowColumnIndex(index, 1);
//Get the DetailsViewDataGrid by passing the corresponding row index and
relation name
var detailsViewDataGrid =
this.dataGrid.GetDetailsViewGrid(this.dataGrid.ResolveToRecordIndex(parentRo
wIndex), "ProductDetails");
//If the DetailsViewDataGrid is not in view, then you can call the
BringIntoView method.
if (detailsViewDataGrid == null)
{
    detailsViewManager.BringIntoView(index);
    detailsViewDataGrid =
this.dataGrid.GetDetailsViewGrid(this.dataGrid.ResolveToRecordIndex(parentRo
wIndex), "ProductDetails");
}
else
{
    //If the DetailsViewDataGrid is already expanded, bring that into view
    dataGrid.ScrollInView(rowColumnIndex);
}
```

Select the record in the DetailsViewDataGrid

You can select the record of the [DetailsViewDataGrid](#) programmatically at run time by setting the corresponding child grid record index to the [SfDataGrid.SelectedIndex](#) property.

**C#**

```
detailsViewDataGrid.SelectedIndex = childIndex;
```

You can get the sample from [here](#).

#### *Customizing the SelectionController for DetailsViewDataGrid*

The [DetailsViewDataGrid](#) process the selection operations in selection controller. Below are the built-in selection controllers,

- [GridSelectionController](#) – Process selection operations when selection unit as row.
- [GridCellSelectionController](#) – Process selection operations when selection unit as cell or Any.

You can customize the default row selection behavior by overriding `GridSelectionController` class and set it to `DetailsViewDataGrid.SelectionController`.

The below code-snippet explains you about the customization of `GridSelectionController` class.

#### **C#**

```
this.dataGrid.DetailsViewLoading += dataGrid_DetailsViewLoading;
void dataGrid_DetailsViewLoading(object sender,
DetailsViewLoadingAndUnloadingEventArgs e)
{
    if (!(e.DetailsViewDataGrid.SelectionController is
GridSelectionControllerExt))
    e.DetailsViewDataGrid.SelectionController = new GridSelectionControllerExt
(e.DetailsViewDataGrid);
}
public class GridSelectionControllerExt : GridSelectionController
{
    public GridSelectionControllerExt(SfDataGrid datagrid)
    : base(datagrid)
    {
    }
}
```

#### Scrolling Rows or Columns

##### *Automatic scrolling on Drag Selection*

SfDataGrid will scrolls rows and columns automatically when you try to perform the drag selection like in excel. You can enable or disable AutoScrolling by setting the [AutoScroller.AutoScrolling](#) property.

#### **C#**

```
this.dataGrid.AutoScroller.AutoScrolling = AutoScrollOrientation.Both;
```

##### *Scroll to particular RowColumnIndex*

You can scroll programmatically to particular cell using [ScrollInView](#) method by passing row and column index.

#### **C#**

```
int rowIndex = this.dataGrid.GetLastDataRowIndex();
int columnIndex = this.dataGrid.GetLastColumnIndex();
this.dataGrid.ScrollInView(new RowColumnIndex(rowIndex, columnIndex));
```



*Scroll to SelectedItem*

You can scroll programmatically to the `SelectedItem` using the `ScrollInView` method.

**C#**

```
using Syncfusion.UI.Xaml.Grid.Helpers;
var rowIndex = this.datagrid.ResolveToRowIndex(this.datagrid.SelectedItem);
var columnIndex = this.datagrid.ResolveToStartColumnIndex();
this.datagrid.ScrollInView(new RowColumnIndex(rowIndex, columnIndex));
```

## Mouse and Keyboard Behaviors

*Keyboard Behavior*

Key or KeyCombinations	Description
DownArrow	Moves CurrentCell directly below the active current cell. If the CurrentCell is in last row, pressing DownArrow does nothing.
UpArrow	Moves the CurrentCell directly above the active current cell. If the CurrentCell is in first row, pressing UpArrow does nothing.
LeftArrow	Moves the current cell to previous to the active current cell. If the CurrentCell is in first cell, pressing LeftArrow does nothing. If the focused row is group header, the group will be collapsed when it is in expanded state.
RightArrow	Moves the current cell to next to the active current cell. If the CurrentCell is in last cell, pressing RightArrow does nothing. If the focused row is group header, the group will be expanded when it is in collapsed state.
Home / Ctrl + LeftArrow	Moves the current cell to the first cell of the current row.
End / Ctrl + RightArrow	Moves the current cell to the last cell of the current row.
PageDown	The SfDataGrid will be scrolled to next set of rows that are not displayed in view, including the row that are partially displayed and the current cell is set to last row.
PageUp	The SfDataGrid will be scrolled to previous set of rows that are not displayed in view, including the row that are partially displayed and the current cell is set to the first row.
Tab	Moves the current cell to next to the active current cell. If the active current cell is the last cell of the current row, the focus will be moved to first cell of the row next to the current row. If the active current cell is the last cell of the last row, the focus will be moved to next control in the tab order of the parent container.
Shift + Tab	Moves the current cell to previous to the active current cell. If the active current cell is the first cell of the current row, the current cell will be moved to last cell of the row previous to the current row. If the active current cell is the first cell of the first row, the focus will be moved to previous control in the tab order of the parent container.

Ctrl + DownArrow	Moves the current cell to the current column of the last row.
Ctrl + UpArrow	Moves the current cell to the current column of the first row.
Ctrl + Home	Moves the current cell to the first cell of the first row.
Ctrl + End	Moves the current cell to the last cell of the last row.
Enter	If the active current cell is in edit mode, the changes will committed and moves the current cell to below the active current cell. If the active current cell is in last row, commits changes only and retains in the same cell.
Ctrl + Enter	Commits only the changes when the current cell in edit mode and retains the focus in same cell.
F2	If the <a href="#">DataGrid.AllowEditing</a> property is true and the <a href="#">GridColumn.AllowEditing</a> property is true for the current column, the current cell enters into edit mode.
Esc	If the current cell is in edit mode, reverts the changes that had been done in the current cell. If the underlying source implements the <a href="#">IEditableObject</a> , on pressing of <code>Esc</code> key for the second time will cancel the edit mode for entire row.
Delete	If the current cell is not in edit mode, the current row will be deleted.
Ctrl + A	All rows or cells will be selected.

**Note:** When the [NavigationMode](#) is in [Row](#), the RightArrow and LeftArrow only work for grouping headers and the following keys Tab, Shift + Tab, Delete, Home, End will not work.

#### Shift Key Combinations

When the [SelectionMode](#) is set to [Extended](#), you can select multiple rows or cells using the navigation keys along with the Shift key. Before navigation starts, the current cell will be marked as a pressed cell and the selection will be done in all rows or cells between the pressed cell and current cell.

Key Combinations
Shift + DownArrow
Shift + UpArrow
Shift + RightArrow
Shift + LeftArrow
Shift + Home
Shift + End
Shift + PageDown

Shift + PageUp
Shift + Ctrl + DownArrow
Shift + Ctrl + UpArrow
Shift + Ctrl + RightArrow
Shift + Ctrl + LeftArrow
Shift + Ctrl + Home
Shift + Ctrl + End
Shift + Ctrl + PageDown
Shift + Ctrl + PageUp

### Mouse Behavior

You can enable/disable the selection when the mouse button is in pressed state by setting the [AllowSelectionOnPointerPressed](#) property.

When the [SelectionMode](#) is set to [Extended](#), you can select multiple rows or cells by clicking on any cell along with ctrl and Shift key. When you click a cell along with Ctrl key, you can select or deselect the particular row or cell. When you click a cell along with Shift key, you can select the range rows or cells from the pressed cell to the current cell.

### Customizing mouse and keyboard behaviors

You can customize mouse and keyboard behaviors by overriding the selection controller. You can refer the section [Customizing Selection Behaviors](#) to override the selection controller.

### Events Processed on Selection

#### CurrentCellActivating Event

The [CurrentCellActivating](#) event will occurs before moving the current cell to particular cell. [CurrentCellActivatingEventArgs](#) has following members which provides information for [CurrentCellActivatingEvent](#).

- [ActivationTrigger](#) – Returns the reason for moving the current cell.
- [CurrentRowColumnIndex](#) – [RowColumnIndex](#) of the cell where the current cell need to move.
- [PreviousRowColumnIndex](#) – [RowColumnIndex](#) of the cell from where the current cell was move.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    CurrentCellActivating="dataGrid_CurrentCellActivating"
    ItemsSource="{Binding Orders}">
```

### C#

```
this.dataGrid.CurrentCellActivating += dataGrid_CurrentCellActivating;
```

```
void dataGrid_CurrentCellActivating(object sender,
CurrentCellActivatingEventArgs args)
{
}
```

You can cancel the current cell moving process within this event by setting [GridCurrentCellActivatingEventArgs.Cancel](#) as true.

### C#

```
void dataGrid_CurrentCellActivating(object sender,
CurrentCellActivatingEventArgs args)
{
    var provider = this.dataGrid.View.GetPropertyAccessProvider();
    var record =
    this.dataGrid.GetRecordAtRowIndex(args.CurrentRowColumnIndex.RowIndex);
    if (record == null)
    return;
    var column =
    this.dataGrid.Columns[this.dataGrid.ResolveToGridVisibleColumnIndex(args.Cur
rentRowColumnIndex.ColumnIndex)];
    var cellValue = provider.GetValue(record, column.MappingName);
    if (cellValue.ToString() == "1001")
    args.Cancel = true;
}
```

### CurrentCellActivated Event

The [CurrentCellActivated](#) event will occur once the current cell is moved to corresponding cell. [CurrentCellActivatedEventArgs](#) has following members which provides information for [CurrentCellActivated](#) event.

- [ActivationTrigger](#) – Returns the reason of the current cell movement.
- [CurrentRowColumnIndex](#) – RowColumnIndex of the cell where the current cell was moved.
- [PreviousRowColumnIndex](#) – RowColumnIndex of the cell from where the current cell has been moved.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
CurrentCellActivated="dataGrid_CurrentCellActivated"
ItemsSource="{Binding Orders}">
```

### C#

```
this.dataGrid.CurrentCellActivated += dataGrid_CurrentCellActivated;
void dataGrid_CurrentCellActivated(object sender,
CurrentCellActivatedEventArgs args)
{
}
```

*SelectionChanging Event*

[SelectionChanging](#) event occurs before processing the selection to particular row or cell. This event will be triggered only to the keyboard and mouse interactions. [GridSelectionChangingEventArgs](#) has the following members which provides the information for [SelectionChanging](#) event.

- [AddedItems](#) – Collection of [GridRowInfo](#) or [GridCellInfo](#) where the selection going to process.
- [RemovedItems](#) – Collection of [GridRowInfo](#) or [GridCellInfo](#) where the selection going to remove.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
SelectionChanging="dataGrid_SelectionChanging"
ItemsSource="{Binding Orders}">
```

**C#**

```
this.dataGrid.SelectionChanging += dataGrid_SelectionChanging;
void dataGrid_SelectionChanging(object sender,
GridSelectionChangingEventArgs e)
{
}
```

You can cancel the selection process within this event by setting [GridSelectionChangingEventArgs.Cancel](#) property as true.

**C#**

```
private void Datagrid_SelectionChanging(object sender,
GridSelectionChangingEventArgs e)
{
var unBoundRow = e.AddedItems.Where(rowInfo => (rowInfo as
GridRowInfo).IsUnBoundRow).ToList();
if (unBoundRow.Count() > 0)
e.Cancel = true;
}
```

*SelectionChanged Event*

The [SelectionChanged](#) event will occurs once the selection process has been completed for particular row or cell in SfDataGrid. [GridSelectionChangedEventArgs](#) has following members which provides information for [SelectionChanged](#) event.

- [AddedItems](#) – Collection of [GridRowInfo](#) or [GridCellInfo](#) where the selection has been processed.
- [RemovedItems](#) – Collection of [GridRowInfo](#) or [GridCellInfo](#) from where the selection has been removed.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
SelectionChanged="dataGrid_SelectionChanged">
```

```
ItemsSource="{Binding Orders}">
```

**C#**

```
this.dataGrid.SelectionChanged += dataGrid_SelectionChanged;
void dataGrid_SelectionChanged(object sender, GridSelectionChangedEventArgs e)
{
}
```

## Appearance

*Changing Selection Background and Foreground*

You can change the selection background and foreground using [RowSelectionBrush](#), [GroupRowSelectionBrush](#) and [SelectionForegroundBrush](#) properties. The [RowSelectionBrush](#) is only applied to the rows other than summary rows and the [GroupRowSelectionBrush](#) is applied for caption summary and group summary rows.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AllowGrouping="True"
ShowGroupDropArea="True"
RowSelectionBrush="#FFDFF3F4"
GroupRowSelectionBrush="#FFC8E3E3"
SelectionForegroundBrush="DarkBlue"
SelectionMode="Extended"
ItemsSource="{Binding Orders}">
```

OrderID	CustomerID	CustomerName	Country
Country : Canada - 1 Items			
1010	BOTTM	Elizabeth Lincoln	Canada
Country : France - 2 Items			
1007	BLONP	Frédérique Citeaux	France
1009	BONAP	Laurence Lebihan	France
Country : Germany - 2 Items			
1001	ALFKI	Maria Anders	Germany

*Changing Current Cell Border Style*

You can change the current cell border thickness and border color using [CurrentCellBorderThickness](#) and [CurrentCellBorderBrush](#) property.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
```

```
CurrentCellBorderBrush="Red"
CurrentCellBorderThickness="1.6"
ItemsSource="{Binding Orders}">
```

OrderID	CustomerID	CustomerName	Country	
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

#### Customizing Row Selection Border

You can customize the row selection by editing the control template of corresponding row controls.

- Data Row / Add New Row – [VirtualizingCellsControl](#)
- CaptionSummary Row – [CaptionSummaryRowControl](#)
- GroupSummary Row – [GroupSummaryRowControl](#)
- UnBound Row – [UnBoundRowControl](#)
- Filter Row – [FilterRowControl](#)

#### XML

```
<Style TargetType="{x:Type syncfusion:VirtualizingCellsControl}">
<Setter Property="Background" Value="Transparent" />
<Setter Property="BorderBrush" Value="Gray" />
<Setter Property="BorderThickness" Value="0" />
<Setter Property="IsTabStop" Value="False" />
<Setter Property="FocusVisualStyle" Value="{x:Null}" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type syncfusion:VirtualizingCellsControl}">
<Grid>
<Border x:Name="PART_RowBorder"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}" />
<Rectangle x:Name="PART_CurrentFocusRow"
Margin="{TemplateBinding CurrentFocusBorderMargin}"
Stroke="DarkGray"
StrokeDashArray="2,2"
StrokeThickness="1"
Visibility="{TemplateBinding CurrentFocusRowVisibility}" />
<Border Background="{TemplateBinding RowHoverBackgroundBrush}"
```

```

BorderBrush="{TemplateBinding RowHoverBackgroundBrush}"
BorderThickness="{TemplateBinding RowHighlightBorderThickness}"
Clip="{TemplateBinding HighlightBorderClipRect}"
SnapsToDevicePixels="True"
Visibility="{TemplateBinding HighlightSelectionBorderVisiblity}" />
<Rectangle Clip="{TemplateBinding RowBackgroundClip}"
Fill="{TemplateBinding Background}" />
<!-- Adding new border to show border for whole selected row -->
<Border x:Name="PART_SelectionBorder"
BorderBrush="Red"
BorderThickness="1.5,1.5,1.5,2.5"
Opacity="0.75"
Background="{TemplateBinding RowSelectionBrush}"
Clip="{TemplateBinding SelectionBorderClipRect}"
Visibility="{TemplateBinding SelectionBorderVisiblity}" />
<Border BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}">
<ContentPresenter />
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

OrderID	CustomerID	CustomerName	Country	
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

### Customizing Cell Selection

You can customize the cell selection by editing the control template of the corresponding cell control.

- **DataRow / AddNewRow** – [GridCell](#)
- **UnBound Row** – [GridUnBoundRowCell](#)
- **Filter Row** – [GridFilterRowCell](#)

### XML

```

<Style TargetType="{x:Type syncfusion:GridCell}">

```



```

<Setter Property="Background" Value="Transparent" />
<Setter Property="BorderBrush" Value="Gray" />
<Setter Property="BorderThickness" Value="0,0,1,1" />
<Setter Property="Padding" Value="0,0,0,0" />
<Setter Property="FocusVisualStyle" Value="{x:Null}" />
<Setter Property="IsTabStop" Value="False" />
<Setter Property="VerticalContentAlignment" Value="Center"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type syncfusion:GridCell}">
<Grid SnapsToDevicePixels="True">
<Border Background="{TemplateBinding CellSelectionBrush}"
SnapsToDevicePixels="True"
Visibility="{TemplateBinding SelectionBorderVisibility}" />
<Border x:Name="PART_GridCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
SnapsToDevicePixels="True">
<Grid>
<ContentPresenter Margin="{TemplateBinding Padding}" />
</Grid>
</Border>
<Border Background="Transparent"
BorderBrush="Red"
BorderThickness="0.5"
IsHitTestVisible="False"
SnapsToDevicePixels="True"
Margin="0,0,1,1"
Visibility="{TemplateBinding CurrentCellBorderVisibility}" />
<!--Adding new border to show inner border to the CurrentCellBorder -->
<Border Background="Transparent"
BorderBrush="Red"
BorderThickness="0.5"
IsHitTestVisible="False"
SnapsToDevicePixels="True"
Margin="2,2,3,3"
Visibility="{TemplateBinding CurrentCellBorderVisibility}" />
<Border x:Name="PART_InvalidCellBorder"
Width="10"
Height="10"
HorizontalAlignment="Right"
Visibility="Collapsed"
VerticalAlignment="Top"
SnapsToDevicePixels="True">
<ToolTipService.ToolTip>
<ToolTip Background="#FFDB00C"
Placement="Right"
PlacementRectangle="20,0,0,0"
Tag="{TemplateBinding ErrorMessage}"
Template="{StaticResource ValidationToolTipTemplate}" />
</ToolTipService.ToolTip>
<Path Data="M0.5,0.5 L12.652698,0.5 12.652698,12.068006 z"
Fill="Red"
SnapsToDevicePixels="True"
Stretch="Fill" />
</Border>

```

```

</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

OrderID	CustomerID	CustomerName	Country	
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D
1003	ANTON	Antonio Moreno	Mexico	México D
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannhei
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawasse

### Binding Selection Properties

You can bind the selection properties like [SelectedItem](#), [SelectedIndex](#) and [CurrentItem](#) to the properties in **ViewModel** directly.

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
SelectedItem="{Binding DataGridSelectedItem, Mode=TwoWay}"
CurrentItem="{Binding DataGridCurrentItem, Mode=TwoWay}"
SelectedIndex="{Binding DataGridSelectedIndex, Mode=TwoWay}"
ItemsSource="{Binding Orders}">

```

In **DetailsView**, it is not possible to bind selection properties directly with **ViewModel**. You can use [Behavior](#) to achieve this requirement.

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Orders}">
<interactivity:Interaction.Behaviors>
<!-- Binding code behind properties to SelectionBehavior dependency
properties -->
<local:SelectionBehavior AttachedDetailsViewSelectedItem="{Binding
DetailsViewSelectedItem, Mode=TwoWay}"
AttachedDetailsViewSelectedIndex="{Binding
DetailsViewSelectedIndex, Mode=TwoWay}"
AttachedDetailsViewCurrentItem="{Binding DetailsViewCurrentItem,
Mode=TwoWay}"
</interactivity:Interaction.Behaviors>
<syncfusion:GridViewDefinition RelationalColumn="OrdersDetailsFirstNested">

```

```
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstDetailsViewGrid"
AutoGenerateColumns="True"/>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid>
```

**C#**

```
public class SelectionBehavior : Behavior<SfDataGrid>
{
    public object AttachedDetailsViewSelectedItem
    {
        get { return (object)GetValue(AttachedDetailsViewSelectedItemProperty); }
        set { SetValue(AttachedDetailsViewSelectedItemProperty, value); }
    }
    public static readonly DependencyProperty
AttachedDetailsViewSelectedItemProperty =
DependencyProperty.Register("AttachedDetailsViewSelectedItem",
typeof(object), typeof(SelectionBehavior), new PropertyMetadata(null));
    public object AttachedDetailsViewSelectedIndex
    {
        get { return (object)GetValue(AttachedDetailsViewSelectedIndexProperty); }
        set { SetValue(AttachedDetailsViewSelectedIndexProperty, value); }
    }
    public static readonly DependencyProperty
AttachedDetailsViewSelectedIndexProperty =
DependencyProperty.Register("AttachedDetailsViewSelectedIndex",
typeof(object), typeof(SelectionBehavior), new PropertyMetadata(null));
    public object AttachedDetailsViewCurrentItem
    {
        get { return (object)GetValue(AttachedDetailsViewCurrentItemProperty); }
        set { SetValue(AttachedDetailsViewCurrentItemProperty, value); }
    }
    public static readonly DependencyProperty
AttachedDetailsViewCurrentItemProperty =
DependencyProperty.Register("AttachedDetailsViewCurrentItem",
typeof(object), typeof(SelectionBehavior), new PropertyMetadata(null));
    protected override void OnAttached()
    {
        base.OnAttached();
        this.AssociatedObject.Loaded += AssociatedObject_Loaded;
    }
    void AssociatedObject_Loaded(object sender, System.Windows.RoutedEventArgs e)
    {
        (this.AssociatedObject.DetailsViewDefinition[0] as
GridViewDefinition).DataGrid.SelectionChanged +=
DetailsViewDataGridSelectionChanged;
    }
    void DetailsViewDataGridSelectionChanged(object sender,
GridSelectionChangedEventArgs e)
    {
        this.AttachedDetailsViewSelectedItem = (e.OriginalSender as
SfDataGrid).SelectedItem;
    }
}
```

```

this.AttachedDetailsViewSelectedIndex = (e.OriginalSender as
SfDataGrid).SelectedIndex;
this.AttachedDetailsViewCurrentItem = (e.OriginalSender as
SfDataGrid).CurrentItem;
}
protected override void OnDetaching()
{
base.OnDetaching();
this.AssociatedObject.Loaded -= AssociatedObject_Loaded;
}
}

```

You can get the sample from [here](#)

### Customizing Selection Behaviors

The SfDataGrid process the selection operations in selection controller. Below are the built-in selection controllers,

- [GridSelectionController](#) – Process selection operations when selection unit as row.
- [GridCellSelectionController](#) – process selection operations when selection unit as cell or Any.

You can customize the default row selection behaviors by overriding `GridSelectionController` class and set it to `SfDataGrid.SelectionController`.

#### C#

```

this.dataGrid.SelectionController = new
GridSelectionControllerExt(this.dataGrid);
public class GridSelectionControllerExt:GridSelectionController
{
public GridSelectionControllerExt(SfDataGrid dataGrid):base(dataGrid)
{
}
}
}

```

### Changing Enter Key Behavior

By default, while pressing Enter key the current cell will be moved to next focused cell in the same column. You can change the behavior by overriding the corresponding selection controllers based on [SelectionUnit](#).

You can change the Enter key behavior by overriding `ProcessKeyDown` method in selection controller. In this method you have to create new `KeyEventArgs` which refers the Tab key and processes the Tab key action.

#### C#

```

public class GridSelectionControllerExt:GridSelectionController
{
public GridSelectionControllerExt(SfDataGrid dataGrid):base(dataGrid)
{
}
}
protected override void ProcessKeyDown(KeyEventArgs args)
{
}

```

```

if (args.Key == Key.Enter)
{
    //Creates new KeyEventArgs to refer the Tab key.
    KeyEventArgs arguments = new KeyEventArgs(args.KeyboardDevice,
args.InputSource, args.Timestamp, Key.Tab) { RoutedEvent = args.RoutedEvent
};
    //Base ProcessKeyDown is invoked to process Tab key operations.
    base.ProcessKeyDown(arguments);
    args.Handled = arguments.Handled;
    return;
}
base.ProcessKeyDown(args);
}
}

```

#### Selecting all rows in a group when expanding

You can select all the rows in the group which is expanding through mouse click. To achieve this, you have to set [SelectionMode](#) as [Extended](#) or [Multiple](#) and also need to override [HandlePointerOperations](#) method in selection controller.

#### C#

```

public class GridSelectionControllerExt:GridSelectionController
{
    public GridSelectionControllerExt(SfDataGrid dataGrid)
    :base(dataGrid)
    {
    }
    public override void HandlePointerOperations(GridPointerEventArgs args,
Syncfusion.UI.Xaml.ScrollAxis.RowColumnIndex rowColumnIndex)
    {
        base.HandlePointerOperations(args, rowColumnIndex);
        if (args.Operation == PointerOperation.Released)
        {
            if (this.DataGrid.View.TopLevelGroup != null)
            {
                //Get the group from the DisplayElements by resolving record index of
corresponding row index
                var group =
this.DataGrid.View.TopLevelGroup.DisplayElements[this.DataGrid.ResolveToRecordIndex(rowColumnIndex.RowIndex)];
                if (group != null && group is Group)
                SelectGroupRows(group as Group);
            }
        }
    }
    private void SelectGroupRows(Group group)
    {
        if (group == null || !group.IsExpanded)
        return;
        //Check whether the group contains inner level groups.
        if (group.Groups == null)
        {
            //Get the corresponding start index of record by getting it from
DisplayElements .

```

```

var startIndex =
this.DataGrid.View.TopLevelGroup.DisplayElements.IndexOf(group as Group);
//Resolve the recordIndex to RowIndex.
var startRowIndex = this.DataGrid.ResolveToRowIndex(startIndex);
//Gets the count of rows in the group.
var count = group.ItemsCount + this.DataGrid.GroupSummaryRows.Count;
//Select the rows from corresponding start and end row index
this.DataGrid.SelectionController.SelectRows(startRowIndex, startRowIndex +
count);
}
else
{
foreach (var gr in group.Groups)
{
//Called recursively, to traverse it inner level of group.
SelectGroupRows(gr);
var startIndex =
this.DataGrid.View.TopLevelGroup.DisplayElements.IndexOf(group as Group);
var startRowIndex = this.DataGrid.ResolveToRowIndex(startIndex);
//Get the corresponding end index of the group by getting it from
DisplayElements using the inner level group.
var endIndex = this.DataGrid.View.TopLevelGroup.DisplayElements.IndexOf(gr
as Group);
var endRowIndex = this.DataGrid.ResolveToRowIndex(endIndex);
this.DataGrid.SelectionController.SelectRows(startRowIndex, endRowIndex);
}
}
}
}

```

#### Selecting the column when clicking header

You can select entire column on clicking column header by handling [MouseLeftButtonUp](#) event of SfDataGrid. You have to set [SelectionUnit](#) as [Cell](#) or [Any](#) and [SelectionMode](#) as [Extended](#) or [Multiple](#) to achieve this behavior.

By default the sorting operation will be performed while clicking on column header where you can disable this action by setting [AllowSorting](#) as false or [SortClickAction](#) as [DoubleClick](#).

#### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
SelectionUnit="Cell"
SortClickAction="DoubleClick"
SelectionMode="Extended"
ItemsSource="{Binding Orders}">

```

#### C#

```

this.dataGrid.MouseLeftButtonUp += dataGrid_MouseLeftButtonUp;
private void dataGrid_MouseLeftButtonUp(object sender, MouseButtonEventArgs
e)
{
if (e.ClickCount != 1)
return;
var visualContainer = this.dataGrid.GetVisualContainer();

```

```
//Getting row and column index from the current pointer position.
var rowColumnIndex =
visualContainer.PointToCellRowIndex(e.GetPosition(visualContainer));
var columnIndex =
this.dataGrid.ResolveToGridVisibleColumnIndex(rowColumnIndex.ColumnIndex);
//Return if rowColumnIndex does not refer the HeaderCell
if (this.dataGrid.GetHeaderIndex() != rowColumnIndex.RowIndex || columnIndex
< 0)
return;
//Gets the first record of the DataGrid.
var firstRowData =
this.dataGrid.GetRecordAtRowIndex(dataGrid.GetFirstDataRowIndex());
//Gets the last record of the DataGrid.
var lastRowData =
this.dataGrid.GetRecordAtRowIndex(dataGrid.GetLastDataRowIndex());
if (firstRowData == null || lastRowData == null)
return;
//Gets the column of the pointer position.
var column = this.dataGrid.Columns[columnIndex];
//Selects the entire column
this.dataGrid.SelectCells(firstRowData, column, lastRowData, column);
}
```

You can get the sample from [here](#).

EmployeeName	EmployeeDesignation	EmployeeSalary	EmployeeArea	EmployeeGender
Robert	Analysts	10000	Torino	Male
	SoftwareEngineer	15000	Montreal	Male
Nancy	Manager	27000	Bracke	Male
Andrew	SalesRepresentative	20500	Kobenhavn	Male
	Vicepresident	10000	Arhus	Male
Steven	SalesRepresentative	10000	Oulu	Male
Michael	salesmanager	10000	Torino	Male
	Insidesalescoordinator	10000	Lisboa	Male
Anne	SoftwareEngineer	13500	London	Male
Aburt	Businessmanager	16000	Geneve	Male
Tim	Mailclerk	10000	Malaysia	Male
Caroline	Receptionist	19000	Caracas	Male
Justin	Marketingdirector	17000	Lilly	Male
Xavier	Marketingassociate	12000	Kobenhavn	Male
	AdvertetisingSpecialist	15000	Hork	Male
Antony	SoftwareEngineer	43000	Bern	Male
Aburt	Businessmanager	10000	Geneve	Male
Tim	Mailclerk	10000	Malaysia	Male

#### *Avoid CaptionSummaryRow selection on Grouping*

While grouping any column, by default the first CaptionSummaryRow will be selected when the [CurrentItem](#) is null. You can change this action by overriding the [ProcessOnGroupChanged](#) method in selection controller.

#### **C#**

```
public class GridSelectionControllerExt : GridSelectionController
{
    public GridSelectionControllerExt(SfDataGrid dataGrid) : base(dataGrid)
    {
    }
    protected override void ProcessOnGroupChanged(GridGroupingEventArgs args)
    {
        base.ProcessOnGroupChanged(args);
        var removedItems = new List<object>();
        //Refresh the selection only in record rows.
        this.RefreshSelectedItems(ref removedItems);
        //Updates the current cell and current row.
        this.UpdateCurrentRowIndex();
    }
}
```

See Also

[How to disable the row selection while clicking on the checkbox column](#)

[How to copy active cell value alone when SelectionUnit as Row](#)

[How to improve the performance while selecting rows using Shift key ?](#)

[How to scroll and select record programmatically ?](#)

[How to prevent the selection while pressing rightclick ?](#)

[How to get the SelectedItem of the DetailsView?](#)

[How to add a new row continuously in AddNewRow without moving the selection to next row?](#)

[How to change the CheckBoxColumn values based on row selection?](#)

[How to bind SelectedItem and CurrentItem in DetailsViewDataGrid](#)

[How to select the rows based on a CellValue?](#)

[How to change the cell value of selectedcells when end edit?](#)

[How to clear selection while grouping/ungrouping?](#)

[How to search and select record in SfDataGrid?](#)

[How to read cell values from SelectedItems?](#)

[How to add a new record in specific DetailsViewDataGrid ?](#)

[How to move selection to newly added record using AddNewRow?](#)

[How to paste the data by custom column order instead of the first column in the SfDataGrid when SelectionUnit is a Row?](#)

[How to remove the top-right corner error mark from the GridCell by pressing Esc key when validated by handling the CurrentCellValidating event?](#)

[How to programmatically select the records of the Master-DetailsView at run time?](#)

[How to accomplish RecordNavigationBar in the SfDataGrid like Syncfusion Windows Forms DataBoundGrid?](#)

[How to select the entire column in SfDataGrid ?](#)



[How to move the CurrentCell to the first column of the AddNewRow when the Tab key is pressed from the last column and its position is at the Bottom of the SfDataGrid?](#)

[How to navigate the current cell within the selected ranges as in Excel, while pressing Enter or Tab key in SfDataGrid?](#)

[How to add selection to data rows of each group on expanding its CaptionSummaryRow?](#)

[How to change the Enter key behavior in SfDataGrid?](#)

[How to change the Enter key behavior to insert line break when the CurrentCell is in the edit mode?](#)

[How to show the selection of row/cell when setting the background for SfDataGrid GridCell?](#)

[How to set Border for the Selected Rows?](#)

[How to bind the SelectedItems property of SfDataGrid to ViewModel property?](#)

[How to get information from the selected cells when using cell selection?](#)

[How to avoid selection while grouping and ungrouping in SfDataGrid?](#)

[How to change the background and foreground for the selected row or cell?](#)

[How to set current cell on particular row when DataGrid loaded?](#)

[How to select the multiple rows on the SfDataGrid?](#)

Clipboard Operations in WPF DataGrid (SfDataGrid)

WPF DataGrid (SfDataGrid) provide support for the clipboard operations such as cut, copy and paste the data within control and between other applications such as Notepad, Excel. Clipboard operations copy and paste is enabled by default. You can copy selected records/cells from SfDataGrid by pressing Ctrl+C and also can paste the content from [Clipboard](#) to SfDataGrid by pressing Ctrl+V.

---

**Note:** Clipboard operations is not supported for the summary rows, add new row and unbound rows.

---

Copy to Clipboard in DataGrid

Copy operation works based on [GridCopyOption](#) property.

GridCopyOption provides the following options,

- [None](#) – Disables copy in SfDataGrid.
- [CopyData](#) – Enabled copy in SfDataGrid.
- [IncludeHeaders](#) – Column header also copied along with data.
- [IncludeFormat](#) – Copies the display text with format instead of actual value.
- [IncludeHiddenColumn](#) – Hidden column also copied to clipboard.

You have to use [IncludeHeaders](#), [IncludeFormat](#), [IncludeHiddenColumn](#) options along with [CopyData](#) option.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    SelectionUnit="Row"
    SelectionMode="Single"
    GridCopyOption="CopyData, IncludeHeaders"/>
```

```
ItemsSource="{Binding Orders}"/>
```

**C#**

```
this.dataGrid.GridCopyOption = GridCopyOption.CopyData |
GridCopyOption.IncludeHeaders;
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
				heims
				bourg
				d
				eille
				ssen

**Note:** IncludeHiddenColumn is not supported when SelectionUnit is [Cell](#).

## Paste from Clipboard in DataGrid

Paste operation works based on [GridPasteOption](#) property.

[GridPasteOption](#) provides the following options,

- [None](#) – Disable paste in SfDataGrid.
- [PasteData](#) – Enabled paste in SfDataGrid and when an incompatible value is pasted into a record/cell, the pasting operation is skipped for that particular record/cell.
- [ExcludeFirstLine](#) – This can be used when pasting data copied with [IncludeHeader](#) copy option.
- [IncludeHiddenColumn](#) – Paste the values in hidden columns also.

You have to use [ExcludeFirstLine](#), [IncludeHiddenColumn](#) options along with [PasteData](#) option.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
SelectionUnit="Row"
SelectionMode="Single"
GridPasteOption="PasteData,ExcludeFirstLine"
ItemsSource="{Binding Orders}"/>
```

**C#**

```
this.dataGrid.GridPasteOption = GridPasteOption.PasteData |
GridPasteOption.ExcludeFirstLine;
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

Cut to Clipboard in DataGrid

Cut operation works based on [GridCopyOption](#) property.

**GridCopyOption** provides the following options,

- [None](#) – Disables cut in SfDataGrid.
- [CutData](#) – Enabled cut in SfDataGrid.
- [IncludeHeaders](#) – Column header also copied along with data.
- [IncludeFormat](#) – Cut the display text with format instead of actual value.
- [IncludeHiddenColumn](#) – Hidden column also cut to clipboard.

You have to use **IncludeHeaders**, **IncludeFormat**, **IncludeHiddenColumn** options along with **CutData** option.

#### **XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    SelectionUnit="Row"
    SelectionMode="Single"
    GridCopyOption="CutData, IncludeHeaders"
    ItemsSource="{Binding Orders}"/>
```

#### **C#**

```
this.dataGrid.GridCopyOption = GridCopyOption.CutData |
    GridCopyOption.IncludeHeaders;
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
0				
1004	AROUT	Thomas Hardy	UK	London
1005				
1006				
1007				
1008				
1009				
1010				

**Note:** IncludeHiddenColumn is not supported when SelectionUnit is [Cell](#).

## Events

### GridCopyContent

[GridCopyContent](#) event occurs when copy/cut the cells in SfDataGrid. [GridCopyPasteEventArgs](#) provides information for [GridCopyContent](#) event. You can cancel copy operation by handling this event.

### C#

```
this.dataGrid.GridCopyContent += dataGrid_GridCopyContent;
void dataGrid_GridCopyContent(object sender, GridCopyPasteEventArgs e)
{
    if (((e.OriginalSender as SfDataGrid).SelectedItem as OrderInfo).OrderID == 1004)
    e.Handled = true;
}
```

### GridPasteContent

[GridPasteContent](#) event occurs when paste the clipboard value into SfDataGrid.

[GridCopyPasteEventArgs](#) provides information for [GridPasteContent](#) event. You can cancel paste operation by handling this event.

### C#

```
this.dataGrid.GridPasteContent+=dataGrid_GridPasteContent;
void dataGrid_GridPasteContent(object sender, GridCopyPasteEventArgs e)
{
    if (((e.OriginalSender as SfDataGrid).SelectedItem as OrderInfo).OrderID == 1010)
    e.Handled = true;
}
```

### CopyGridCellContent

[CopyGridCellContent](#) event occurs when cell being copy/cut. [GridCopyPasteCellEventArgs](#) provides information for [CopyGridCellContent](#) event, which has following members,

- [ClipboardValue](#) - Returns cell value.
- [Column](#) – Returns corresponding GridColumn of a cell.
- [RowData](#) – Returns corresponding RowData of a cell.
- [OriginalSender](#) – Returns the SfDataGrid.

You can change the text copied to clipboard by changing the `ClipboardValue`.

### C#

```
this.dataGrid.CopyGridCellContent += dataGrid_CopyGridCellContent;
void dataGrid_CopyGridCellContent(object sender, GridCopyPasteCellEventArgs e)
{
}
```

The below code example change the clipboard value as 100 instead of cell value 1003 in SfDataGrid.

### C#

```
void dataGrid_CopyGridCellContent(object sender, GridCopyPasteCellEventArgs e)
{
    if (e.Column.MappingName == "OrderID" && (e.RowData as OrderInfo).OrderID == 1003)
        e.ClipBoardValue = 100;
}
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
				London
				Luleå
				Mannheim
				Strasbourg
				Madrid
				Marseille
				Sawassen

The below code example handled the copy operation when `MappingName` of a Column is Country.

### C#

```
void dataGrid_CopyGridCellContent(object sender, GridCopyPasteCellEventArgs e)
{
    if (e.Column.MappingName == "Country")
        e.Handled = true;
}
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	CHOPD	Henrik Lehmann	Germany	Leipzig
1007	DELCO	Michael Blythe	USA	Long Beach
1008	DUMPE	Michael Blythe	USA	Long Beach
1009	FRANR	Michael Blythe	USA	Long Beach
1010	FRANR	Michael Blythe	USA	Long Beach

### *PasteGridCellContent*

[PasteGridCellContent](#) event occurs when cell being paste. [GridCopyPasteCellEventArgs](#) provides information for [PasteGridCellContent](#) event, which has following members.

- [ClipboardValue](#) - Returns clipboard value of a particular cell.
- [Column](#) – Returns corresponding GridColumn of a cell.
- [RowData](#) – Returns corresponding RowData of a cell.
- [OriginalSender](#) – Returns the SfDataGrid.

You can change the text paste to SfDataGrid by changing the [ClipboardValue](#).

### **C#**

```
this.dataGrid.PasteGridCellContent += dataGrid_PasteGridCellContent;
void dataGrid_PasteGridCellContent(object sender, GridCopyPasteCellEventArgs e)
{
}
```

The below code example change the clipboard value as Test instead of clipboard value BOLID.

### **C#**

```
void dataGrid_PasteGridCellContent(object sender, GridCopyPasteCellEventArgs e)
{
    if (e.Column.MappingName == "CustomerID" && (e.RowData as OrderInfo).CustomerID == "BERGS")
        e.ClipBoardValue = "Test";
}
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001				
1002				
1003				
1004				
1008	Test	Martin Sommer	Spain	Madrid
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

The below code example handled the paste operation when MappingName of Column is OrderID

### C#

```
void dataGrid_PasteGridCellContent(object sender, GridCopyPasteCellEventArgs e)
{
    if (e.Column.MappingName == "OrderID")
        e.Handled = true;
}
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALF			
1002	AN			
1003	AN			
1004	ARC			
1005	BER			
1006	BONAP	Laurence Lebihan	France	Marseille
1007	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

### Handling Programmatically

#### Programmatically Copy to Clipboard in WPF DataGrid

Copy the selected records/cells in SfDataGrid by using [Copy](#) method in [GridCopyPaste](#) of SfDataGrid.

### C#

```
this.dataGrid.GridCopyPaste.Copy();
```

Copy a record by selecting the record using [MoveCurrentCell](#) method and [Copy](#) method in [GridCopyPaste](#) of SfDataGrid.

**C#**

```
RowColumnIndex rowColumnIndex = new RowColumnIndex();  
rowColumnIndex.RowIndex = 2;  
rowColumnIndex.ColumnIndex = 2;  
this.dataGrid.SelectionController.MoveCurrentCell(rowColumnIndex);  
this.dataGrid.GridCopyPaste.Copy();
```

Copy the multiple records by selecting group of records using [SelectRows](#) method and [Copy](#) method in [GridCopyPaste](#) of SfDataGrid.

**C#**

```
this.dataGrid.SelectionController.SelectRows(1, 7);  
this.dataGrid.GridCopyPaste.Copy();
```

Copy the multiple cells by selecting group of cells using [SelectCells](#) method and [Copy](#) method in [GridCopyPaste](#) of SfDataGrid.

**C#**

```
this.dataGrid.SelectCells(this.dataGrid.GetRowGenerator().Items[2].RowData,  
this.dataGrid.Columns[1], this.dataGrid.GetRowGenerator().Items[5].RowData,  
this.dataGrid.Columns[3]);  
this.dataGrid.GridCopyPaste.Copy();
```

*Copy rows without selecting in WPF DataGrid*

You can copy the records without selection by using [CopyRowsToClipboard](#) method in [GridCopyPaste](#) of SfDataGrid.

**C#**

```
this.dataGrid.GridCopyPaste.CopyRowsToClipboard(2, 4);
```

*Programmatically Cut Data to Clipboard in WPF DataGrid*

Cut the selected records/cells in SfDataGrid by using [Cut](#) method in [GridCopyPaste](#) of SfDataGrid.

**C#**

```
this.dataGrid.GridCopyPaste.Cut();
```

Cut the entire record in SfDataGrid by selecting whole SfDataGrid using [SelectAll](#) method and [Cut](#) method in [GridCopyPaste](#) of SfDataGrid.

**C#**

```
this.dataGrid.SelectionController.SelectAll();  
this.dataGrid.GridCopyPaste.Cut();
```

Cut the entire column in SfDataGrid by using [SelectCells](#) method and [Cut](#) method in [GridCopyPaste](#) of SfDataGrid.

**C#**



```
var firstRowData =  
this.dataGrid.GetRecordAtRowIndex(dataGrid.GetFirstRowIndex());  
var lastRowData =  
this.dataGrid.GetRecordAtRowIndex(dataGrid.GetLastRowIndex());  
this.dataGrid.SelectCells(firstRowData, this.dataGrid.Columns[2],  
lastRowData, this.dataGrid.Columns[2]);  
this.dataGrid.GridCopyPaste.Cut();
```

#### *Programmatically Paste in DataGrid*

Paste the clipboard value into SfDataGrid by using [Paste](#) method in [GridCopyPaste](#) of SfDataGrid.

#### **C#**

```
this.dataGrid.GridCopyPaste.Paste();
```

Paste the clipboard value into selected record by selecting the record using [MoveCurrentCell](#) method and [Paste](#) method in [GridCopyPaste](#) of SfDataGrid.

#### **C#**

```
RowColumnIndex rowColumnIndex = new RowColumnIndex();  
rowColumnIndex.RowIndex = 1;  
rowColumnIndex.ColumnIndex = 1;  
this.dataGrid.SelectionController.MoveCurrentCell(rowColumnIndex);  
this.dataGrid.GridCopyPaste.Paste();
```

#### *Customizing Copy Paste Behavior in WPF DataGrid*

WPF DataGrid (SfDataGrid) process the clipboard operations in [GridCutCopyPaste](#) class. You can customize the default copy paste behaviors by overriding [GridCutCopyPaste](#) class and set it to [SfDataGrid.GridCopyPaste](#).

#### **C#**

```
public class CustomCopyPaste : GridCutCopyPaste  
{  
    public CustomCopyPaste(SfDataGrid sfGrid) : base(sfGrid)  
    {  
    }  
}
```

#### **C#**

```
public MainWindow()  
{  
    InitializeComponent();  
    this.dataGrid.GridCopyPaste = new CustomCopyPaste(this.dataGrid);  
}
```

#### *Paste a cell into many cells in WPF DataGrid*

By default, you can copy one cell and paste it into another cell when Cell Selection is enabled in SfDataGrid. The below code shows how to copy one cell and paste it into all the selected cells by overriding [PasteToCell](#) method in [GridCutCopyPaste](#) class.

**C#**

```

public class CustomCopyPaste : GridCutCopyPaste
{
    public CustomCopyPaste(SfDataGrid dataGrid) : base(dataGrid)
    {
    }
    protected override void PasteToCell(object record, GridColumn column, object value)
    {
        var text = Clipboard.GetText();
        string[] clipBoardText = Regex.Split(text, @"\r\n");
        clipBoardText = Regex.Split(clipBoardText[0], @"\t");
        //Get the clipBoardText and check if the clipBoardText is more than one cell
        //means call the base.
        if (clipBoardText.Count() > 1)
        {
            base.PasteToCell(record, column, value);
        }
        //Get the selectedCells for paste the copied cell
        var selectedCells = this.dataGrid.GetSelectedCells();
        int selectedCellsCount = selectedCells.Count;
        for (int i = 0; i < selectedCellsCount; i++)
        {
            record = selectedCells[i].RowData;
            column = selectedCells[i].Column;
            //Call PasteToCell method with particular record of selectedCells,
            //Column of selected records and rowData
            if (record != null && column != null)
            base.PasteToCell(record, column, value);
        }
    }
}

```

*Paste a record into many rows in WPF DataGrid*

By default, you can able to copy one row and paste it into another row when Row Selection is enabled in SfDataGrid. The below code shows how to copy one row and paste it into all selected rows by overriding the [PasteToRow](#) method in the [GridCutCopyPaste](#) class.

**C#**

```

public class CustomCopyPaste : GridCutCopyPaste
{
    public CustomCopyPaste(SfDataGrid dataGrid) : base(dataGrid)
    {
    }
    protected override void PasteToRow(object clipBoardContent, object selectedRecords)
    {
        var text = Clipboard.GetText();
        string[] clipBoardText = Regex.Split(text, @"\r\n");
        //Get the clipBoardText and check if the clipBoardText is more than one row
        //means call the base.
        if (clipBoardText.Count() > 1)
        {
            base.PasteToRow(clipBoardContent, selectedRecords);
        }
    }
}

```

```

return;
}
var selectedRecord = this.dataGrid.SelectedItems;
for (int i = 0; i < selectedRecord.Count; i++)
{
    //Get the selected records for paste the copied row.
    selectedRecords = selectedRecord[i];
    //Call the PasteToRow method with clipBoardContent and selectedRecords
    base.PasteToRow(clipBoardContent, selectedRecords);
}
}
}

```

#### Select pasted records in WPF DataGrid

By default after pasting the clipboard value to SfDataGrid selection is maintains in previously selected records as it is. The below code shows select the pasted records after the Paste operation, by overriding the [PasteToRows](#) and [PasteToRow](#) methods in [GridCutCopyPaste](#) class. This code is applicable when [SelectionUnit](#) is [Row](#).

#### C#

```

public class CustomCopyPaste : GridCutCopyPaste
{
    public CustomCopyPaste(SfDataGrid dataGrid) : base(dataGrid)
    {
    }

    //Creating the new list for add the selected records
    public List<object> selectedItem = new List<object>();
    protected override void PasteToRows(object clipBoardRows)
    {
        base.PasteToRows(clipBoardRows);
        //Using the SelectionController apply the selection for Pasted records
        this.dataGrid.SelectionController.HandleGridOperations(new
        GridOperationsHandlerArgs(GridOperation.Paste, selectedItem));
    }
    protected override void PasteToRow(object clipBoardContent, object
    selectedRecords)
    {
        //Added the selected record to list
        selectedItem.Add(selectedRecords);
        base.PasteToRow(clipBoardContent, selectedRecords);
    }
}

```

#### Create new records while pasting in WPF DataGrid

By default while paste the clipboard value to SfDataGrid, it changes the values of the already existing records. The below code example shows how to add the copied records as new rows in SfDataGrid by overriding the [PasteToRows](#) method in [GridCutCopyPaste](#) class.

#### C#

```

public class CustomCopyPaste : GridCutCopyPaste
{
    public CustomCopyPaste(SfDataGrid dataGrid)
    : base(dataGrid)
    {
    }
}

```

```
{
}
protected override void PasteToRows(object clipBoardRows)
{
    var copiedRecord = (string[])clipBoardRows;
    int copiedRecordsCount = copiedRecord.Count();
    //Based on the clipboard count added the new record for paste
    if (copiedRecordsCount > 0)
    {
        //Get the viewModel for adding the record
        var record = this.dataGrid.DataContext as ViewModel;
        for (int i = 0; i < copiedRecordsCount; i++)
        {
            //Create the new instance for Model, for adding the new record
            OrderInfo entity = new OrderInfo();
            for (int j = 0; j < this.dataGrid.Columns.Count; j++)
            {
                string[] values = Regex.Split(copiedRecord[i], @"\t");
                //Adding the new record by using PasteToCell method by passing the
                //created data, particular column, and clipboard value
                this.PasteToCell(entity, this.dataGrid.Columns[j], values[j]);
            }
            //Added the pasted record in collection
            record.Orders.Add(entity);
        }
    }
}
```

See Also

[How to copy active cell value alone when SelectionUnit as Row](#)

[How to paste the copied row in AddNewRow?](#)

[How to paste the empty string while using Cell selection?](#)

[How to paste the data by custom column order instead of the first column in the SfDataGrid when SelectionUnit is a Row?](#)

[How to add the copied rows as new rows in the SfDataGrid while pasting?](#)

[How to copy one row and paste it into all the selected rows like Excel in the SfDataGrid?](#)

[How to copy one cell and paste it into all the selected cells in the SfDataGrid like the Excel?](#)

[How to copy the column and paste it as a new column by ContextMenu in SfDataGrid?](#)

Master Details View in WPF DataGrid (SfDataGrid)

DataGrid provides support to represent the hierarchical data in the form of nested tables using Master-Details View. You can expand or collapse the nested tables ([DetailsViewDataGrid](#)) by using an expander in a row or programmatically. The number of tables nested with relations is unlimited.

	EmployeeID	OrderID	City
-	1	1001	Berlin
	OrderID	SalesID	ProductName
-	1001	A00001	Bike1
	OrderID	ProductName	
	1001	Bike1	
	1001	Bike2	
+	1001	A00002	Bike1
	OrderID	Quantity	
	1001	10	
	1001	10	
-	2	1002	México D.F.
	OrderID	SalesID	ProductName
+	1002	A00003	Cycle
	OrderID	Quantity	
	1002	20	
	1002	20	
+	3	1003	London

## Generating Master-Details view from IEnumerable

Master-Details View's relation can be generated for the properties of type [IEnumerable](#) in the underlying data object contain.

Follow the below steps to generate the Master-Details View for IEnumerable.

- Create the data model with relations (Here, relations are `IEnumerable` type properties)
- Defining relations
- Auto-generating relations
- Manually defining relations

## Create the data model with relations

Create an `Employee` class with `Sales` and `Orders` property of type [ObservableCollection](#) to form the relations. The `Sales` and `Orders` properties are defined as `ObservableCollection<SalesInfo>` and `ObservableCollection<OrderInfo>` respectively.

## C#

```
public class SalesInfo : INotifyPropertyChanged
{
    private int    orderId;
```

```
private string _salesID;
private string _productName;
private ObservableCollection<ProductInfo> products;
public int OrderID
{
    get { return _orderID; }
    set
    {
        _orderID = value;
        OnPropertyChanged("OrderID");
    }
}
public string SalesID
{
    get { return _salesID; }
    set
    {
        _salesID = value;
        OnPropertyChanged("SalesID");
    }
}
public string ProductName
{
    get { return _productName; }
    set
    {
        _productName = value;
        OnPropertyChanged("ProductName");
    }
}
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(String name)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(name));
    }
}
public class OrderInfo : INotifyPropertyChanged
{
    private int orderId;
    private int _quantity;
    public int OrderID
    {
        get { return orderId; }
        set
        {
            orderId = value;
            OnPropertyChanged("OrderID");
        }
    }
    public int Quantity
    {
        get { return _quantity; }
        set
        {

```

```
_quantity = value;
OnPropertyChanged("Quantity");
}
}
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(String name)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(name));
    }
}
}
public class Employee : INotifyPropertyChanged
{
    private int _EmployeeID;
    private int _orderId;
    private string _city;
    private ObservableCollection<SalesInfo> _sales;
    private ObservableCollection<OrderInfo> _orders;
    public int EmployeeID
    {
        get { return this._EmployeeID; }
        set
        {
            this._EmployeeID = value;
            OnPropertyChanged("EmployeeID");
        }
    }
    public int OrderID
    {
        get { return this._orderId; }
        set
        {
            this._orderId = value;
            OnPropertyChanged("OrderID");
        }
    }
    public string City
    {
        get { return _city; }
        set
        {
            _city = value;
            OnPropertyChanged("City");
        }
    }
    public ObservableCollection<SalesInfo> Sales
    {
        get { return _sales; }
        set
        {
            _sales = value;
            OnPropertyChanged("Sales");
        }
    }
    public ObservableCollection<OrderInfo> Orders
```

```

{
    get { return _orders; }
    set
    {
        _orders = value;
        OnPropertyChanged("Orders");
    }
}

public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(String name)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(name));
    }
}
}

```

Create a **ViewModel** class with **Employees** property and it is initialized with several data objects in the constructor. Similarly, the **Sales** and **Orders** property are also initialized.

#### C#

```

public class ViewModel
{
    ObservableCollection<Employee> _employees;
    public ObservableCollection<Employee> Employees
    {
        get { return _employees; }
        set { _employees = value; }
    }

    public ViewModel()
    {
        this.GenerateOrders();
        this.GenerateSales();
        _employees = GetEmployeesDetails();
    }

    public ObservableCollection<Employee> GetEmployeesDetails()
    {
        var employees = new ObservableCollection<Employee>();
        employees.Add(new Employee() { EmployeeID = 1, OrderID = 1001, City = "Berlin", Orders = GetOrders(1001), Sales = GetSales(1001) });
        employees.Add(new Employee() { EmployeeID = 2, OrderID = 1002, City = "Mexico D.F.", Orders = GetOrders(1002), Sales = GetSales(1002) });
        employees.Add(new Employee() { EmployeeID = 3, OrderID = 1003, City = "London", Orders = GetOrders(1002), Sales = GetSales(1003) });
        employees.Add(new Employee() { EmployeeID = 4, OrderID = 1004, City = "BERGS", Orders = GetOrders(1002), Sales = GetSales(1004) });
        employees.Add(new Employee() { EmployeeID = 5, OrderID = 1005, City = "Mannheim", Orders = GetOrders(1002), Sales = GetSales(1005) });
        return employees;
    }

    //Orders collection is initialized here.
    ObservableCollection<OrderInfo> Orders = new
    ObservableCollection<OrderInfo>();
    public void GenerateOrders()

```



```

{
Orders.Add(new OrderInfo() { OrderID = 1001, Quantity = 10 });
Orders.Add(new OrderInfo() { OrderID = 1001, Quantity = 10 });
Orders.Add(new OrderInfo() { OrderID = 1002, Quantity = 20 });
Orders.Add(new OrderInfo() { OrderID = 1002, Quantity = 20 });
Orders.Add(new OrderInfo() { OrderID = 1003, Quantity = 50 });
Orders.Add(new OrderInfo() { OrderID = 1004, Quantity = 70 });
Orders.Add(new OrderInfo() { OrderID = 1005, Quantity = 20 });
Orders.Add(new OrderInfo() { OrderID = 1005, Quantity = 20 });
}
private ObservableCollection<OrderInfo> GetOrders(int orderID)
{
ObservableCollection<OrderInfo> orders = new
ObservableCollection<OrderInfo>();
foreach (var order in Orders)
if (order.OrderID == orderID)
orders.Add(order);
return orders;
}
//Sales collection is initialized here.
ObservableCollection<SalesInfo> Sales = new
ObservableCollection<SalesInfo>();
public void GenerateSales()
{
Sales.Add(new SalesInfo() { OrderID = 1001, SalesID = "A00001", ProductName
= "Bike1" });
Sales.Add(new SalesInfo() { OrderID = 1001, SalesID = "A00002", ProductName
= "Bike1" });
Sales.Add(new SalesInfo() { OrderID = 1002, SalesID = "A00003", ProductName
= "Cycle" });
Sales.Add(new SalesInfo() { OrderID = 1003, SalesID = "A00004", ProductName
= "Car" });
}
private ObservableCollection<SalesInfo> GetSales(int orderID)
{
ObservableCollection<SalesInfo> sales = new
ObservableCollection<SalesInfo>();
foreach (var sale in Sales)
if (sale.OrderID == orderID)
sales.Add(sale);
return sales;
}
}

```

### Defining relations in DataGrid

#### Auto-generating relations

SfDataGrid will automatically generate relations and inner relations for the `IEnumerable` property types in the data object. This can be enabled by setting [SfDataGrid.AutoGenerateRelations](#) to `true`.

Bind the collection created in the previous step to [SfDataGrid.ItemsSource](#) and set the [SfDataGrid.AutoGenerateRelations](#) to `true`.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
```

```
AutoGenerateColumns="True"
AutoGenerateRelations="True"
ItemsSource="{Binding Employees}" />
```

### C#

```
dataGrid.AutoGenerateRelations = true;
```

When relations are auto-generated, you can handle the [SfDataGrid.AutoGeneratingRelations](#) event to customize or cancel the [GridViewDefinition](#) before they are added to the [SfDataGrid.DetailsViewDefinition](#).

Here, two relations are created from **Sales** and **Orders** collection property.

	EmployeeID	OrderID	City
—	1	1001	Berlin
		OrderID	SalesID
		1001	A00001
		1001	A00002
		OrderID	Quantity
		1001	10
		1001	10
+	2	1002	México D.F.
+	3	1003	London
+	4	1004	BERGS
+	5	1005	Mannheim

### Manually defining Relations

You can define the Master-Details View's relation manually using [SfDataGrid.DetailsViewDefinition](#), when the [SfDataGrid.AutoGenerateRelations](#) is **false**.

To define Master-Details View relations, create [GridViewDefinition](#) and set the name of **IEnumerable** type property (from data object) to [ViewDefinition.RelationalColumn](#). Then, add the [GridViewDefinition](#) to the [SfDataGrid.DetailsViewDefinition](#).

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Employees}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<!-- FirstLevelNestedGrid1 is created here -->
<syncfusion:GridViewDefinition RelationalColumn="Sales">
```

```

<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid1"
AutoGenerateColumns="True"/>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
<!-- FirstLevelNestedGrid2 is created here -->
<syncfusion:GridViewDefinition RelationalColumn="Orders">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid2"
AutoGenerateColumns="True"/>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>

```

**C#**

```

dataGrid.AutoGenerateRelations = false;
var gridViewDefinition1 = new GridViewDefinition();
gridViewDefinition1.RelationalColumn = "Sales";
gridViewDefinition1.DataGrid = new SfDataGrid() { Name =
"FirstLevelNestedGrid1", AutoGenerateColumns = true };
var gridViewDefinition2 = new GridViewDefinition();
gridViewDefinition2.RelationalColumn = "Orders";
gridViewDefinition2.DataGrid = new SfDataGrid() { Name =
"FirstLevelNestedGrid2", AutoGenerateColumns = true };
dataGrid.DetailsViewDefinition.Add(gridViewDefinition1);
dataGrid.DetailsViewDefinition.Add(gridViewDefinition2);

```

	EmployeeID	OrderID	City
-	1	1001	Berlin
		OrderID	SalesID
		1001	A00001
		1001	A00002
		OrderID	Quantity
		1001	10
		1001	10
+	2	1002	México D.F.
+	3	1003	London
+	4	1004	BERGS
+	5	1005	Mannheim

In the same way, you can define relations for first level nested grids by defining relations to the [ViewDefinition.DataGrid](#) of first level nested grid.

## XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Employees}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<!-- FirstLevelNestedGrid is created here -->
<syncfusion:GridViewDefinition RelationalColumn="Orders">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False">
<!-- SecondLevelNestedGrid is created here -->
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="Products">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="SecondLevelNestedGrid"
AutoGenerateColumns="True" />
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>
```

## C#

```
dataGrid.AutoGenerateRelations = false;
// GridViewDefinition for parent DataGrid
var gridViewDefinition1 = new GridViewDefinition();
gridViewDefinition1.RelationalColumn = "Sales";
var firstLevelNestedGrid = new SfDataGrid() { Name = "FirstLevelNestedGrid",
AutoGenerateColumns = true };
firstLevelNestedGrid.AutoGenerateRelations = false;
// GridViewDefinition for FirstLevelNestedGrid
var gridViewDefinition = new GridViewDefinition();
gridViewDefinition.RelationalColumn = "Products";
gridViewDefinition.DataGrid = new SfDataGrid() { Name =
"SecondLevelNestedGrid", AutoGenerateColumns = true };
firstLevelNestedGrid.DetailsViewDefinition.Add(gridViewDefinition);
gridViewDefinition1.DataGrid = firstLevelNestedGrid;
dataGrid.DetailsViewDefinition.Add(gridViewDefinition1);
```

	EmployeeID	OrderID	City
-	1	1001	Berlin
	OrderID	SalesID	ProductName
-	1001	A00001	Bike1
	OrderID	ProductName	
	1001	Bike1	
	1001	Bike2	
+	1001	A00002	Bike1
-	2	1002	México D.F.
	OrderID	SalesID	ProductName
+	1002	A00003	Cycle
+	3	1003	London
+	4	1004	BERGS
+	5	1005	Mannheim

## Generating Master-Details view from DataTable

Master-Details View's relation can be generated for [DataTable](#), when [DataRelation](#) is defined between two tables in underlying [DataSet](#).

Follow the below steps to generate the Master-Details View's relation for DataTable,

- Create the [DataTable](#) with relations.
- Defining relations
- Auto-generating relations
- Manually defining relations

### Create the DataTable with relations

Create **ViewModel** class and define the **Suppliers** property of type **DataTable**. Then, add the data relations between **Suppliers** and **Products** tables in the **DataSet** based on **SupplierID** column.

## C#

```
public class ViewModel
{
    private DataTable _suppliers;
    public DataTable Suppliers
    {
        get { return _suppliers; }
        set { _suppliers = value; }
    }
    public ViewModel()
    {
        _suppliers = GetDataTable();
    }
}
```

```

public DataTable GetDataTable()
{
    DataSet ds = new DataSet();
    string connectionString = string.Format(@"Data Source = {0}",
        ("Northwind.sdf"));
    using (SqlCeConnection con = new SqlCeConnection(connectionString))
    {
        con.Open();
        SqlCeDataAdapter sda = new SqlCeDataAdapter("SELECT * FROM Suppliers", con);
        sda.Fill(ds, "Suppliers");
    }
    using (SqlCeConnection con1 = new SqlCeConnection(connectionString))
    {
        con1.Open();
        SqlCeDataAdapter sda1 = new SqlCeDataAdapter("SELECT * FROM Products",
            con1);
        sda1.Fill(ds, "Products");
    }
    //Both tables have Supplier ID as common to make relation
    ds.Relations.Add(new DataRelation("Supplier_Product",
        ds.Tables[0].Columns["Supplier ID"], ds.Tables[1].Columns["Supplier ID"]));
    if (ds.Tables.Count > 0)
    return ds.Tables[0];
    else
    return null;
}
}

```

### Defining relations in DataGrid

#### Auto-generating relations

SfDataGrid will automatically generate relations and inner relations based on relations defined in [DataSet](#). This can be enabled by setting [SfDataGrid.AutoGenerateRelations](#) to `true`.

Bind the `Suppliers` table created in the previous step to [SfDataGrid.ItemsSource](#) and set the [SfDataGrid.AutoGenerateRelations](#) to `true`.

#### XML

```

<syncfusion:SfDataGrid x:Name="datagrid"
    AutoGenerateColumns="True"
    AutoGenerateRelations="True"
    ItemsSource="{Binding Suppliers}" />

```

#### C#

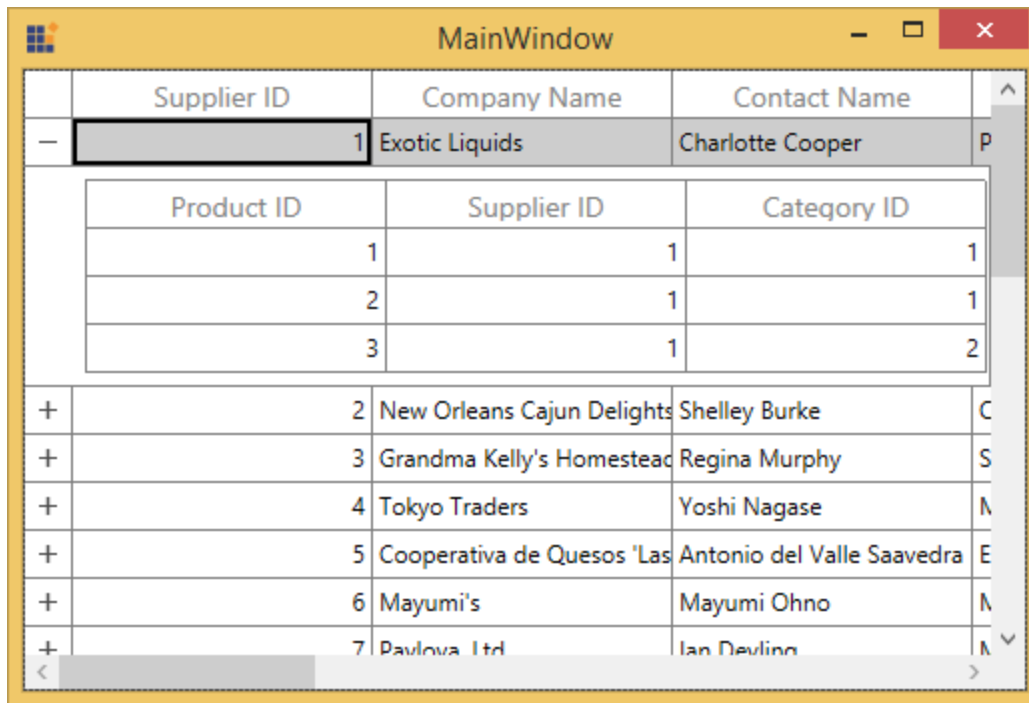
```

datagrid.AutoGenerateRelations = true;

```

When relations are auto-generated, you can handle the [SfDataGrid.AutoGeneratingRelations](#) event to customize or cancel the [GridViewDefinition](#) before they are added to the [SfDataGrid.DetailsViewDefinition](#).

Here, Master-Details View relation is auto generated based on the `Supplier_Product` relation.



### Manually defining Relations

You can define the Master-Details View's relation manually using [SfDataGrid.DetailsViewDefinition](#), when the [SfDataGrid.AutoGenerateRelations](#) is false.

To define Master-Details View relations, create [GridViewDefinition](#) and set the relation name Supplier\_Product to [ViewDefinition.RelationalColumn](#). Then, the [GridViewDefinition](#) is added to the [SfDataGrid.DetailsViewDefinition](#) collection of parent DataGrid.

### XML

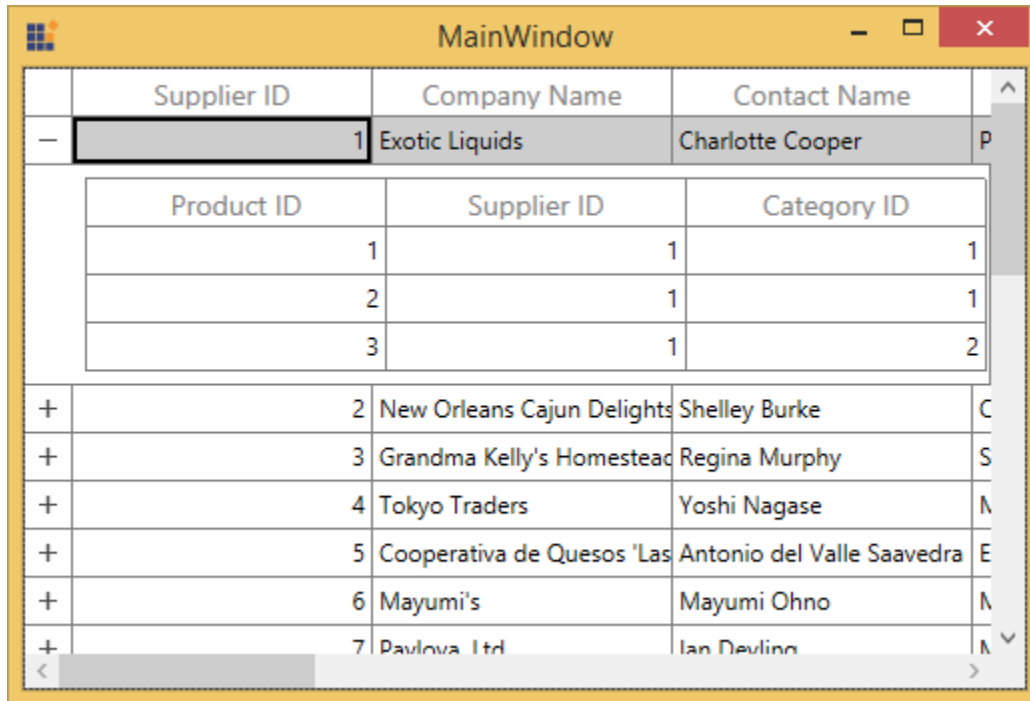
```
<syncfusion:SfDataGrid Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Suppliers}">
  <syncfusion:SfDataGrid.DetailsViewDefinition>
    <syncfusion:GridViewDefinition RelationalColumn="Supplier_Product">
      <syncfusion:GridViewDefinition.DataGrid>
        <syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AutoGenerateColumns="True" />
      </syncfusion:GridViewDefinition.DataGrid>
    </syncfusion:GridViewDefinition>
  </syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>
```

### C#

```
dataGrid.AutoGenerateRelations = false;
// GridViewDefinition for DataGrid
var gridViewDefinition = new GridViewDefinition();
gridViewDefinition.RelationalColumn = "Supplier_Product";
gridViewDefinition.DataGrid = new SfDataGrid() { Name =
"FirstLevelNestedGrid", AutoGenerateColumns = true };

```

```
this.dataGrid.DetailsViewDefinition.Add(gridViewDefinition);
```



Populating Master-Details view through events

You can load `ItemsSource` for [DetailsViewDataGrid](#) asynchronously by handling [SfDataGrid.DetailsViewExpanding](#). You can set `ItemsSource` in on-demand when expanding record through [GridDetailsViewExpandingEventArgs.DetailsViewItemsSource](#) property in the [SfDataGrid.DetailsViewExpanding](#) event handler.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    AutoGenerateRelations="True"
    ItemsSource="{Binding Orders}" />
```

#### C#

```
this.dataGrid.DetailsViewExpanding += dataGrid_DetailsViewExpanding;
void dataGrid_DetailsViewExpanding(object sender,
    Syncfusion.UI.Xaml.Grid.GridDetailsViewExpandingEventArgs e)
{
    e.DetailsViewItemsSource.Clear();
    var itemsSource = GetItemSource();
    e.DetailsViewItemsSource.Add("ProductDetails", Products);
}
private ObservableCollection<ProductInfo> GetItemSource()
{
    Products = new ObservableCollection<ProductInfo>();
    Products.Add(new ProductInfo() { OrderID = 1001, ProductName = "Bike" });
    Products.Add(new ProductInfo() { OrderID = 1002, ProductName = "Car" });
    Products.Add(new ProductInfo() { OrderID = 1003, ProductName = "Bike1" });
}
```



```
return Products;
}
```

**Note:** This event will be triggered only when underlying data object contains relations. Otherwise, you have to define dummy relation to notify DataGrid to fire this event.

In the below code snippet, `AutoGenerateRelations` set to false and also relation is defined with some name to `RelationalColumn`. For example, `ProductDetails` is the dummy relational column and underlying data object does not contain the `IEnumerable` type property with name `ProductDetails`.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AutoGenerateColumns="True" />
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>
```

Now the `ItemsSource` for [DetailsViewDataGrid](#) can be supplied through [DetailsViewExpanding](#) event as mentioned above.

### Loading DetailsViewItemsSource asynchronously

While populating Master-Details view through events, if the data to be loaded is downloaded from an external source or being read from a file, you may get a time delay. In such case, the `DetailsViewExpanding` event will be executed before the I/O processes get completes.

In this case, you can use [async](#) and [await](#) to load the data with a time delay and hold the event from executing before the data gets loaded from an external source gets completed.

### C#

```
this.dataGrid.DetailsViewExpanding += dataGrid_DetailsViewExpanding;
private async Task<ObservableCollection<ProductInfo>> GetItemSource()
{
var products = new ObservableCollection<ProductInfo>();
await Schedule(() =>
{
products.Add(new ProductInfo() { OrderID = 1001, ProductName = "Bike" });
products.Add(new ProductInfo() { OrderID = 1002, ProductName = "Car" });
products.Add(new ProductInfo() { OrderID = 1003, ProductName = "Bike1" });
}, 2000);
return products;
}
public async Task<bool> Schedule(Action _onCompletion, int durationMS)
{
DispatcherTimer timer = new DispatcherTimer();
timer.Interval = TimeSpan.FromMilliseconds(durationMS);
```

```
//Task that causes time delay
timer.Tick += timer_Tick;
_onCompletion();
timer.Stop();
return true;
}
async void dataGrid_DetailsViewExpanding(object sender,
Syncfusion.UI.Xaml.Grid.GridDetailsViewExpandingEventArgs e)
{
e.DetailsViewItemsSource.Clear();
var itemsSource = await GetItemSource();
e.DetailsViewItemsSource.Add("ProductDetails", itemsSource);
}
```

The declaration of `await` with `GetItemSource` method hold the further process of adding the `DetailsViewItemsSource` until the items are assigned to the `ItemsSource`. Here the timer is invoked with 2 seconds delay in asynchronous `Schedule` method call in `GetItemSource` method while adding the `DetailsViewDataGrid` items.

The `DetailsViewExpanding` method runs synchronously until it reaches its first `await` expression. After `await` is reached, it is suspended until the awaited task is complete.

Defining properties for `DetailsViewDataGrid`

You can set properties like `AllowEditing`, `AllowFiltering` and `AllowSorting` for `DetailsViewDataGrid` by using the `GridViewDefinition.DataGrid` property.

*When `AutoGenerateRelations` is false*

For manually defined relation, the properties can be directly set to the `ViewDefinition.DataGrid`.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AllowEditing="True"
AllowFiltering="True"
AllowResizingColumns="True"
AllowSorting="True"
AutoGenerateColumns="False" />
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>
```

### C#

```
FirstLevelNestedGrid.AllowEditing = true;
FirstLevelNestedGrid.AllowFiltering = true;
FirstLevelNestedGrid.AllowResizingColumns = true;
FirstLevelNestedGrid.AllowSorting = true;
```

For two levels of nesting,

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Employees}">
  <syncfusion:SfDataGrid.DetailsViewDefinition>
    <!-- FirstLevelNestedGrid is created here -->
    <syncfusion:GridViewDefinition RelationalColumn="Sales">
      <syncfusion:GridViewDefinition.DataGrid>
        <syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False">
          <syncfusion:SfDataGrid.DetailsViewDefinition>
            <!-- SecondLevelNestedGrid is created here -->
            <syncfusion:GridViewDefinition
RelationalColumn="Products">
              <syncfusion:GridViewDefinition.DataGrid>
                <syncfusion:SfDataGrid
x:Name="SecondLevelNestedGrid"
AllowEditing="True"
AllowFiltering="True"
AutoGenerateColumns="True" />
              </syncfusion:GridViewDefinition.DataGrid>
            </syncfusion:GridViewDefinition>
          </syncfusion:SfDataGrid.DetailsViewDefinition>
        </syncfusion:SfDataGrid>
      </syncfusion:GridViewDefinition.DataGrid>
    </syncfusion:GridViewDefinition>
  </syncfusion:SfDataGrid>
</syncfusion:SfDataGrid>
```

#### C#

```
SecondLevelNestedGrid.AllowEditing = true;
SecondLevelNestedGrid.AllowFiltering = true;
```

*When AutoGenerateRelations is true*

When the relation is auto-generated, you can get the [GridViewDefinition.DataGrid](#) in the [AutoGeneratingRelations](#) event handler to set the properties.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="True"
ItemsSource="{Binding Employees}"/>
```

#### C#

```
this.dataGrid.AutoGeneratingRelations += dataGrid_AutoGeneratingRelations;
```

```
void dataGrid_AutoGeneratingRelations(object sender,
Syncfusion.UI.Xaml.Grid.AutoGeneratingRelationsArgs e)
{
e.GridViewDefinition.DataGrid.AllowEditing = true;
e.GridViewDefinition.DataGrid.AllowFiltering = true;
e.GridViewDefinition.DataGrid.AllowSorting = true;
e.GridViewDefinition.DataGrid.AllowResizingColumns = true;
}
```

For two levels of nesting,

### C#

```
this.dataGrid.AutoGeneratingRelations += dataGrid_AutoGeneratingRelations;
void dataGrid_AutoGeneratingRelations(object sender,
AutoGeneratingRelationsArgs e)
{
e.GridViewDefinition.DataGrid.AutoGenerateRelations = true;
e.GridViewDefinition.DataGrid.AutoGeneratingRelations +=
FirstLevelNestedGrid_AutoGeneratingRelations;
}
Void FirstLevelNestedGrid_AutoGeneratingRelations(object sender,
AutoGeneratingRelationsArgs e)
{
e.GridViewDefinition.DataGrid.AutoGenerateColumns = true;
e.GridViewDefinition.DataGrid.AllowEditing = true;
e.GridViewDefinition.DataGrid.AllowFiltering = true;
}
```

**Note:** When you make any change in one [DetailsViewDataGrid](#), that change will be applied to all [DetailsViewDataGrid's](#) in the same level. For example, when you resize the first column in one [DetailsViewDataGrid](#), the same column width is applied to all [DetailsViewDataGrid's](#) at that level. This is applicable for features like filtering, sorting, grouping and re-ordering columns also.

	EmployeeID	OrderID	City
–	1	1001	Berlin
		OrderID	SalesID ▼ ProductName
+	1001	A00002	Bike1
+	1001	A00001	Bike1
		OrderID	Quantity
	1001		10
	1001		10
–	2	1002	México D.F.
		OrderID	SalesID ▼ ProductName
+	1002	A00003	Cycle
		OrderID	Quantity
	1002		20
	1002		20
+	3	1003	London
+	4	1004	BERGS
+	5	1005	Mannheim

Here, **SalesID** column is sorted in all DetailsViewDataGrid at the same level.

**Note:** [AllowFrozenGroupHeaders](#) , [FrozenRowCount](#), [FooterRowCount](#), [FooterColumnCount](#), [FrozenColumnCount](#) properties are not supported while using Master Details view.

Defining columns for DetailsViewDataGrid

The [ViewDefinition.DataGrid](#)'s columns can be generated either automatically or manually like parent DataGrid. You can refer [here](#) to know more about columns.

#### Auto-generating columns

You can auto-generate the [ViewDefinition.DataGrid](#)'s columns by setting the [GridViewDefinition.DataGrid.AutoGenerateColumns](#) to **true**. You can **cancel** or **customize** the column being created for [ViewDefinition.DataGrid](#) by handling [GridViewDefinition.DataGrid.AutoGeneratingColumn](#) event.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
```

```

AutoGenerateColumns="True"
AutoGeneratingColumn="FirstLevelNestedGrid_AutoGeneratingColumn" />
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>

```

**C#**

```

FirstLevelNestedGrid.AutoGeneratingColumn +=
FirstLevelNestedGrid_AutoGeneratingColumn;

```

When relation is auto generated, you can set properties and wire

[GridViewDefinition.DataGrid.AutoGeneratingColumn](#) event in [SfDataGrid.AutoGeneratingRelations](#) event handler.

**C#**

```

void dataGrid_AutoGeneratingRelations(object sender,
Syncfusion.UI.Xaml.Grid.AutoGeneratingRelationsArgs e)
{
e.GridViewDefinition.DataGrid.AutoGenerateColumns = true;
e.GridViewDefinition.DataGrid += FirstLevelNestedGrid_AutoGeneratingColumn;
}

```

*Manually defining columns*

You can directly define the columns to [ViewDefinition.DataGrid](#) when [AutoGenerateColumns](#) is false.

When relation is manually defined, you can define the columns directly to [ViewDefinition.DataGrid](#) in XAML or C#, by adding desired column to the [SfDataGrid.Columns](#) collection.

**XML**

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AutoGenerateColumns="False">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="OrderID" />
<syncfusion:GridTextColumn MappingName="ProductName" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>

```

When relation is auto generated, you can define the [ViewDefinition.DataGrid's](#) columns manually through the [SfDataGrid.AutoGeneratingRelations](#) event handler.

**C#**

```

this.dataGrid.AutoGeneratingRelations += dataGrid_AutoGeneratingRelations;
void dataGrid_AutoGeneratingRelations(object sender,
Syncfusion.UI.Xaml.Grid.AutoGeneratingRelationsArgs e)
{
    e.GridViewDefinition.DataGrid.AutoGenerateColumns = false;
    e.GridViewDefinition.DataGrid.Columns.Add(new GridTextColumn() { MappingName
= "OrderID" });
    e.GridViewDefinition.DataGrid.Columns.Add(new GridTextColumn() { MappingName
= "ProductName" });
}

```

*Creating Custom Column*

You can also define your own column type to [ViewDefinition.DataGrid](#) like parent DataGrid. For more information about creating custom column, refer the [Custom Column support](#).

After creating the custom column, add the customized renderer to [CellRenderers](#) collection of [DetailsViewDataGrid](#).

**C#**

```

this.dataGrid.DetailsViewLoading += dataGrid_DetailsViewLoading;
void dataGrid_DetailsViewLoading(object sender,
DetailsViewLoadingAndUnloadingEventArgs e)
{
    if (!e.DetailsViewDataGrid.CellRenderers.ContainsKey("DatePickerRenderer"))
    e.DetailsViewDataGrid.CellRenderers.Add("DatePickerRenderer", new
DatePickerRenderer());
}

```

Now, you can add the custom column to **Columns** collection of [ViewDefinition.DataGrid](#).

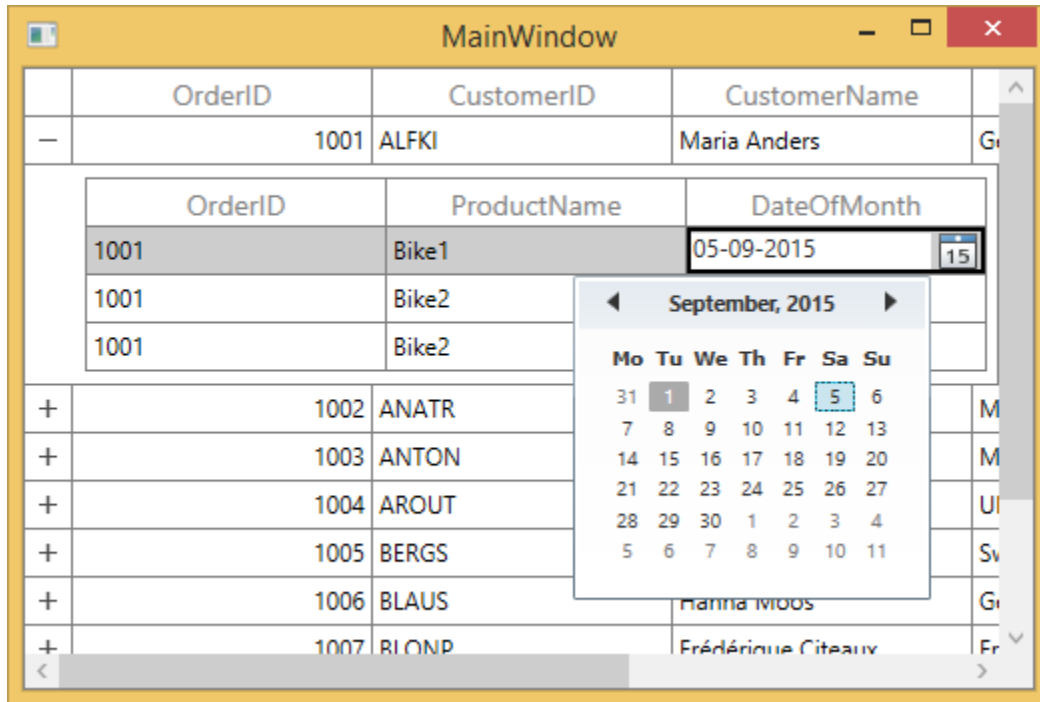
**XML**

```

<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.DetailsViewDefinition>
        <syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
            <syncfusion:GridViewDefinition.DataGrid>
                <syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AllowEditing="True"
AutoGenerateColumns="False">
                    <syncfusion:SfDataGrid.Columns>
                        <syncfusion:GridTextColumn MappingName="OrderID" />
                        <syncfusion:GridTextColumn MappingName="ProductName" />
                        <local:DatePickerColumn DateMapping="DateOfMonth"
DisplayBinding="{Binding DateOfMonth,
Converter={StaticResource converter}}"
MappingName="DateOfMonth" />
                    </syncfusion:SfDataGrid.Columns>
                </syncfusion:SfDataGrid>
            </syncfusion:GridViewDefinition.DataGrid>
        </syncfusion:GridViewDefinition>
    </syncfusion:SfDataGrid>

```

```
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>
```



#### Handling events for DetailsViewDataGrid

You can handle [DetailsViewDataGrid](#) events by wiring events to [ViewDefinition.DataGrid](#) where sender is [ViewDefinition.DataGrid](#). In another way, you can handle [DetailsViewDataGrid](#) events also through [ParentDataGrid](#) events by setting [NotifyEventsToParentDataGrid](#) property of [ViewDefinition.DataGrid](#). For more information refer [Listen DetailsViewDataGrid event from ParentDataGrid event handler](#) section.

#### *When AutoGenerateRelations is false*

For manually defined relation, the events can be wired from [ViewDefinition.DataGrid](#) directly in XAML or C#.

#### **XAML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AutoGenerateColumns="True"
CurrentCellBeginEdit="FirstLevelNestedGrid_CurrentCellBeginEdit"
FilterChanging="FirstLevelNestedGrid_FilterChanging"
SortColumnsChanging="FirstLevelNestedGrid_SortColumnsChanging" />
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
```



```
</syncfusion:SfDataGrid>
```

**C#**

```
FirstLevelNestedGrid.CurrentCellBeginEdit +=
FirstLevelNestedGrid_CurrentCellBeginEdit;
FirstLevelNestedGrid.SortColumnsChanging +=
FirstLevelNestedGrid_SortColumnsChanging;
FirstLevelNestedGrid.FilterChanging += FirstLevelNestedGrid_FilterChanging;
```

For second level nested grid,

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Employees}">
  <syncfusion:SfDataGrid.DetailsViewDefinition>
    <syncfusion:GridViewDefinition RelationalColumn="Sales">
      <syncfusion:GridViewDefinition.DataGrid>
        <syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False">
          <syncfusion:SfDataGrid.DetailsViewDefinition>
            <syncfusion:GridViewDefinition
RelationalColumn="Products">
              <syncfusion:GridViewDefinition.DataGrid>
                <syncfusion:SfDataGrid
x:Name="SecondLevelNestedGrid"
AllowFiltering="True"
AutoGenerateColumns="True"
CurrentCellBeginEdit="SecondLevelNestedGrid_CurrentCellBeginEdit"
FilterChanging="SecondLevelNestedGrid_FilterChanging"/>
              </syncfusion:GridViewDefinition.DataGrid>
            </syncfusion:GridViewDefinition>
          </syncfusion:SfDataGrid.DetailsViewDefinition>
        </syncfusion:SfDataGrid>
      </syncfusion:GridViewDefinition.DataGrid>
    </syncfusion:GridViewDefinition>
  </syncfusion:SfDataGrid>
</syncfusion:SfDataGrid>
```

**C#**

```
SecondLevelNestedGrid.CurrentCellBeginEdit+=
SecondLevelNestedGrid_CurrentCellBeginEdit;
SecondLevelNestedGrid.FilterChanging +=
SecondLevelNestedGrid_FilterChanging;
private void SecondLevelNestedGrid_CurrentCellBeginEdit(object sender,
CurrentCellBeginEditEventArgs args)
{
}
```

*When AutoGenerateRelations is true*

When the relation is auto-generated, you can get the [GridViewDefinition.DataGrid](#) in the [AutoGeneratingRelations](#) event handler to wire the events.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    AutoGenerateRelations="True"
    ItemsSource="{Binding Employees}"/>
```

#### C#

```
this.dataGrid.AutoGeneratingRelations += dataGrid_AutoGeneratingRelations;
void dataGrid_AutoGeneratingRelations(object sender,
Syncfusion.UI.Xaml.Grid.AutoGeneratingRelationsArgs e)
{
    e.GridViewDefinition.DataGrid.CurrentCellBeginEdit +=
    FirstLevelNestedGrid_CurrentCellBeginEdit;
    e.GridViewDefinition.DataGrid.SortColumnsChanging +=
    FirstLevelNestedGrid_SortColumnsChanging;
    e.GridViewDefinition.DataGrid.FilterChanging +=
    FirstLevelNestedGrid_FilterChanging;
}
void FirstLevelNestedGrid_CurrentCellBeginEdit(object sender,
CurrentCellBeginEditEventArgs args)
{
}
```

For second level nested grid,

#### C#

```
this.dataGrid.AutoGeneratingRelations += dataGrid_AutoGeneratingRelations;
void dataGrid_AutoGeneratingRelations(object sender,
AutoGeneratingRelationsArgs e)
{
    // FirstLevelNestedGrid
    e.GridViewDefinition.DataGrid.AutoGenerateRelations = true;
    e.GridViewDefinition.DataGrid.AutoGeneratingRelations +=
    FirstLevelNestedGrid_AutoGeneratingRelations;
}
void FirstLevelNestedGrid_AutoGeneratingRelations(object sender,
AutoGeneratingRelationsArgs e)
{
    // SecondLevelNestedGrid
    e.GridViewDefinition.DataGrid.CurrentCellBeginEdit +=
    SecondLevelNestedGrid_CurrentCellBeginEdit;
    e.GridViewDefinition.DataGrid.FilterChanging +=
    SecondLevelNestedGrid_FilterChanging;
}
```

*Listen DetailsViewDataGrid event in ParentDataGrid event handler*

You can listen `DetailsViewDataGrid` events in `ParentDataGrid` event handlers itself by setting [NotifyEventsToParentDataGrid](#) property of `ViewDefinition.DataGrid`. So, you don't have to listen events for each level as discussed above.

**XML**

```
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="OrderDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstDetailsViewGrid"
AllowEditing="True"
AutoGenerateColumns="True"
NotifyEventsToParentDataGrid="True">
</syncfusion:SfDataGrid>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
```

You can wire the events in `ParentDataGrid` and get the corresponding `DetailsViewDataGrid` in `ParentDataGrid EventArgs`.

**XML**

```
<syncfusion:SfDataGrid Name="datagrid"
ItemsSource="{Binding Source}"
AllowEditing="True"
RowValidating="Datagrid_RowValidating">
```

**C#**

```
this.datagrid.RowValidating += Datagrid_RowValidating;
private void Datagrid_RowValidating(object sender, RowValidatingEventArgs e)
{
    var detailsViewDataGrid = e.OriginalSender as DetailsViewDataGrid;
}
```

You can get the `SourceDataGrid` in `ParentDataGrid` events using [GetSourceDataGrid](#) helper method.

**C#**

```
using Syncfusion.UI.Xaml.Grid.Helpers;
var sourceDataGrid = (e.OriginalSender as
DetailsViewDataGrid).GetSourceDataGrid();
```

Refer [here](#) for get the `ParentDataGrid` using [GetParentDataGrid](#) helper method.

*Binding DetailsViewDataGrid event to command in ViewModel*

You can bind the `DetailsViewDataGrid` events using commands by setting [NotifyEventsToParentDataGrid](#) property of `ViewDefinition.DataGrid`. Using this property, listen the `DetailsViewDataGrid` events in `ParentDataGrid` event handler.

Bind the events using commands in `ViewModel` as like below.

### XML

```
<syncfusion:SfDataGrid Name="datagrid"
AutoGenerateColumns="True"
ItemsSource="{Binding Source}"
AllowEditing="True">
<i:Interaction.Triggers>
<i:EventTrigger EventName="RowValidating">
<i:InvokeCommandAction Command="{Binding Path=RowValidating}"/>
</i:EventTrigger>
</i:Interaction.Triggers>
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="OrderDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstDetailsViewGrid"
AllowEditing="True"
AutoGenerateColumns="True"
NotifyEventsToParentDataGrid="True">
</syncfusion:SfDataGrid>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>
```

### C#

```
public class ViewModel : INotifyPropertyChanged
{
    private ICommand _rowValidatingCommand;
    public ICommand RowValidating
    {
        get
        {
            return _rowValidatingCommand ?? (_rowValidatingCommand = new
            CommandHandler(() => RowValidatingEvent(), true));
        }
    }
    public void RowValidatingEvent()
    {
    }
}
public class CommandHandler : ICommand
{
    private Action _action;
    private bool _canExecute;
    public CommandHandler(Action action, bool canExecute)
    {
        _action = action;
        _canExecute = canExecute;
    }
    public bool CanExecute(object parameter)
    {
        return _canExecute;
    }
    public event EventHandler CanExecuteChanged;
    public void Execute(object parameter)
```

```
{
    _action();
}
```

#### *Getting the parent DataGrid while editing DetailsViewDataGrid*

You can get the corresponding parent DataGrid while editing [DetailsViewDataGrid](#) through its [CurrentCellBeginEdit](#) event handler.

#### **C#**

```
this.FirstLevelNestedGrid.CurrentCellBeginEdit +=
FirstLevelNestedGrid_CurrentCellBeginEdit;
void FirstLevelNestedGrid_CurrentCellBeginEdit(object sender,
CurrentCellBeginEditEventArgs args)
{
    var detailsViewDataGrid = args.OriginalSender as DetailsViewDataGrid;
    var parentDataGrid = detailsViewDataGrid.GetParentDataGrid();
}
```

Here, sender is [ViewDefinition.DataGrid](#). You can get the [DetailsViewDataGrid](#) which actually raises the event by using [OriginalSender](#).

#### Column sizing

SfDataGrid allows you to apply column sizer to [DetailsViewDataGrid](#) by setting the [GridViewDefinition.DataGrid.ColumnSizer](#) like parent DataGrid. For more information, refer the [Column Sizing](#) section.

#### *Disable resizing of last column in parent DataGrid*

By default, the [DetailsViewDataGrid](#) is clipped while resizing the last column of parent DataGrid if the parent DataGrid width is less than the [DetailsViewDataGrid](#) width.

You can disable the resizing of last column of parent DataGrid by setting [DetailsViewManager.DisableLastColumnResizing](#) attached property to `true`.

#### **XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AllowResizingColumns="True"
AutoGenerateColumns="True"
AutoGenerateRelations="True"
ItemsSource="{Binding Orders}"
syncfusion:DetailsViewManager.DisableLastColumnResizing="True" />
```

#### **C#**

```
DetailsViewManager.SetDisableLastColumnResizing(this.dataGrid, true);
```

#### *Resizing parent DataGrid and DetailsViewDataGrid simultaneously*

By default, [DetailsViewDataGrid](#) column width will not be adjusted while resizing the parent DataGrid's columns. You can adjust DetailsViewDataGrid's column width simultaneously while resizing parent DataGrid. This can be achieved by handling [DetailsViewLoading](#) and [ResizingColumns](#) events.

**Note:** It is applicable only when the parent and [DetailsViewDataGrid](#) having same number of columns.

The column width of [DetailsViewDataGrid](#) is set based on the parent DataGrid's column in [DetailsViewLoading](#) event.

### C#

```
dataGrid.DetailsViewLoading += dataGrid_DetailsViewLoading;
void dataGrid_DetailsViewLoading(object sender,
DetailsViewLoadingAndUnloadingEventArgs e)
{
    var parentGrid = e.OriginalSender is DetailsViewDataGrid ? (e.OriginalSender
as SfDataGrid) : sender as SfDataGrid;
    if (!CanResize(parentGrid))
        return;
    if (parentGrid.Columns.Count != e.DetailsViewDataGrid.Columns.Count)
        return;
    double width = 0;
    var detailsViewStartColumnIndex =
e.DetailsViewDataGrid.ResolveToStartColumnIndex();
    for (int i = 0; i < parentGrid.Columns.Count; i++)
    {
        width = i == 0 ? parentGrid.Columns[i].ActualWidth -
detailsViewStartColumnIndex * 24 : parentGrid.Columns[i].Width;
        if (e.DetailsViewDataGrid.Columns[i].Width != parentGrid.Columns[i].Width)
            e.DetailsViewDataGrid.Columns[i].Width = width;
    }
}
```

When the column is resized in parent DataGrid column, then the new [Width](#) is set to corresponding column of [DetailsViewDataGrid](#) based on the [ColumnIndex](#) argument in [ResizingColumns](#) event.

### C#

```
dataGrid.ResizingColumns += dataGrid_ResizingColumns;
void dataGrid_ResizingColumns(object sender, ResizingColumnsEventArgs e)
{
    var grid = sender as SfDataGrid;
    if (e.OriginalSender is DetailsViewDataGrid)
        grid = e.OriginalSender as SfDataGrid;
    if (grid.View == null)
        return;
    SetWidth(grid, e.ColumnIndex, e.Width);
}
private void SetWidth(SfDataGrid grid, int scrollColumnIndex, double width)
{
    if (grid.DetailsViewDefinition == null || !grid.DetailsViewDefinition.Any())
        return;
    if (!CanResize(grid))
        return;
    var columnIndex =
grid.HelperResolveToGridVisibleColumnIndex(scrollColumnIndex);
    if (columnIndex < 0)
        return;
    var parentStartColumnIndex = grid.HelperResolveToStartColumnIndex();
    var indentColumnsWidth = 0;
    foreach (var definition in grid.DetailsViewDefinition)
```

```

{
    var detailsViewDataGrid = (definition as GridViewDefinition).DataGrid;
    var startColumnIndex =
        detailsViewDataGrid.HelperResolveToStartColumnIndex();
    indentColumnsWidth = startColumnIndex * 24;
    var tempWidth = width - indentColumnsWidth < 0 ? 0 : width -
        indentColumnsWidth;
    detailsViewDataGrid.Columns[columnIndex].Width = scrollColumnIndex ==
        parentStartColumnIndex ? tempWidth : width;
    // If DetailsViewDataGrid has DetailsViewDefinition(nested levels),
    recursively set width upto all levels
    if (detailsViewDataGrid.DetailsViewDefinition != null &&
        detailsViewDataGrid.DetailsViewDefinition.Any())
        SetWidth(detailsViewDataGrid,
            detailsViewDataGrid.HelperResolveToScrollColumnIndex(columnIndex),
            detailsViewDataGrid.Columns[columnIndex].Width);
}
}

private bool CanResize(SfDataGrid dataGrid)
{
    if (dataGrid.DetailsViewDefinition == null &&
        !dataGrid.DetailsViewDefinition.Any())
        return true;
    foreach (var definition in dataGrid.DetailsViewDefinition)
    {
        var detailsViewGrid = (definition as GridViewDefinition).DataGrid;
        if (detailsViewGrid.DetailsViewDefinition == null &&
            !detailsViewGrid.DetailsViewDefinition.Any())
            return CanResize(detailsViewGrid);
        if (detailsViewGrid.Columns.Count != dataGrid.Columns.Count)
            return false;
    }
    return true;
}

```

The above `HelperResolveToStartColumnIndex`, `HelperResolveToGridVisibleColumnIndex`, `HelperResolveToScrollColumnIndex` helper methods are used to resolve row and column index in [ViewDefinition.DataGrid](#).

## C#

```

public static class GridHelperClass
{
    /// <summary>
    /// Resolves the start column index of the ViewDefinition.DataGrid.
    /// </summary>
    /// <param name="dataGrid">
    /// The ViewDefinition.DataGrid
    /// </param>
    /// <returns>
    /// Returns the start column index of the ViewDefinition.DataGrid.
    /// </returns>
    public static int HelperResolveToStartColumnIndex(this SfDataGrid dataGrid)
    {
        int startIndex = 0;
        if (dataGrid.ShowRowHeader)

```

```

startIndex += 1;
if (dataGrid.GroupColumnDescriptions != null &&
dataGrid.GroupColumnDescriptions.Any())
startIndex += dataGrid.GroupColumnDescriptions.Count;
if (dataGrid.DetailsViewDefinition != null &&
dataGrid.DetailsViewDefinition.Any())
startIndex += 1;
return startIndex;
}
/// <summary>
/// Resolves the visible column index for the specified column index in
ViewDefinition.DataGrid.
/// </summary>
/// <param name="dataGrid">
/// The ViewDefinition.DataGrid.
/// </param>
/// <param name="visibleColumnIndex">
/// The visibleColumnIndex.
/// </param>
/// <returns>
/// Returns the corresponding visible column index for the specified column
index.
/// </returns>
public static int HelperResolveToGridVisibleColumnIndex(this SfDataGrid
dataGrid, int visibleColumnIndex)
{
var indentColumnCount = (dataGrid.GroupColumnDescriptions != null ?
dataGrid.GroupColumnDescriptions.Count : 0) +
((dataGrid.DetailsViewDefinition != null &&
dataGrid.DetailsViewDefinition.Any()) ? 1 : 0);
int resolvedIndex = visibleColumnIndex - (indentColumnCount +
(dataGrid.ShowRowHeader ? 1 : 0));
return resolvedIndex;
}
/// <summary>
/// Resolves the scroll column index for the specified column index in
ViewDefinition.DataGrid.
/// </summary>
/// <param name="dataGrid">
/// The ViewDefinition.DataGrid.
/// </param>
/// <param name="gridColumnIndex">
/// The corresponding column index to get the scroll column index.
/// </param>
/// <returns>
/// Returns the scroll column index for the specified column index.
/// </returns>
public static int HelperResolveToScrollColumnIndex(this SfDataGrid dataGrid,
int gridColumnIndex)
{
var indentColumnCount = ((dataGrid.DetailsViewDefinition != null &&
dataGrid.DetailsViewDefinition.Any()) ? 1 : 0) +
(dataGrid.GroupColumnDescriptions != null ?
dataGrid.GroupColumnDescriptions.Count : 0);
return ((dataGrid.ShowRowHeader ? 1 : 0) + indentColumnCount) +
gridColumnIndex;
}

```



```
}
```

You can get the sample from [here](#).

**Note:** To display parent and [DetailsViewDataGrid](#) in the same line, set [DetailsViewPadding](#) as Zero.

### Selection

[DetailsViewDataGrid](#) allows you to select rows or cells based on the [SelectionUnit](#) property in its parent DataGrid.

#### *Getting the selected DetailsViewDataGrid*

You can get the currently selected [DetailsViewDataGrid](#) by using the [SelectedDetailsViewGrid](#) property of parent DataGrid.

#### C#

```
var detailsViewDataGrid = this.dataGrid.SelectedDetailsViewGrid;
```

For accessing nested level [SelectedDetailsViewGrid](#),

#### C#

```
var detailsViewDataGrid =  
this.dataGrid.SelectedDetailsViewGrid.SelectedDetailsViewGrid;
```

You can also get the selected [DetailsViewDataGrid](#) using [GetDataGrid](#) helper method which returns the DataGrid that contains the current cell.

#### C#

```
var detailsViewDataGrid = this.dataGrid.GetDataGrid();
```

#### *Getting the SelectedItem, SelectedItems and SelectedIndex of DetailsViewDataGrid*

You can access the selected record or records and selected record index of [DetailsViewDataGrid](#) by using [SelectedItem](#), [SelectedItems](#) and [SelectedIndex](#) properties directly.

#### C#

```
var detailsViewDataGrid = this.dataGrid.GetDetailsViewGrid(2);  
int selectedIndex = detailsViewDataGrid.SelectedIndex;  
var selectedItem = detailsViewDataGrid.SelectedItem;  
var selectedItems = detailsViewDataGrid.SelectedItems;
```

You can access [DetailsViewDataGrid](#)'s [SelectedItem](#), [SelectedItems](#) and [SelectedIndex](#) properties by using parent dataGrid's [SelectedDetailsViewGrid](#) property also.

#### C#

```
int selectedIndex = this.dataGrid.SelectedDetailsViewGrid.SelectedIndex;  
var selectedItem = this.dataGrid.SelectedDetailsViewGrid.SelectedItem;  
var selectedItems = this.dataGrid.SelectedDetailsViewGrid.SelectedItems;
```

#### Getting the CurrentCell of DetailsViewDataGrid

You can get the [CurrentCell](#) of [DetailsViewDataGrid](#) by using the [SelectedDetailsViewGrid](#) property of parent DataGrid or [CurrentCellBeginEdit](#) event of DetailsViewDataGrid.

**C#**

```
var currentCell =
this.dataGrid.SelectedDetailsViewGrid.SelectionController.CurrentCellManager
.CurrentCell;
this.FirstLevelNestedGrid.CurrentCellBeginEdit +=
FirstLevelNestedGrid_CurrentCellBeginEdit;
void FirstLevelNestedGrid_CurrentCellBeginEdit(object sender,
CurrentCellBeginEditEventArgs args)
{
var detailsViewDataGrid = args.OriginalSender as DetailsViewDataGrid;
var currentCell =
detailsViewDataGrid.SelectionController.CurrentCellManager.CurrentCell;
}
```

You can refer [here](#) to know about handling events for [DetailsViewDataGrid](#).

#### Getting the parent DataGrid

You can get the immediate parent DataGrid of corresponding [DetailsViewDataGrid](#) through [GetParentDataGrid](#) helper method.

**C#**

```
var parentDataGrid =
this.dataGrid.SelectedDetailsViewGrid.GetParentDataGrid();
```

#### Getting the DetailsViewDataGrid

You can get the [DetailsViewDataGrid](#) based on row index through [GetDetailsViewGrid](#) helper method.

**C#**

```
var detailsViewDataGrid = this.dataGrid.GetDetailsViewGrid(2);
```

You can also get the [DetailsViewDataGrid](#) based on the record index and relational column name using [GetDetailsViewGrid](#) method.

**C#**

```
var detailsViewDataGrid = this.dataGrid.GetDetailsViewGrid(0,
"ProductDetails");
```

#### Programmatic Selection in DetailsViewDataGrid

In [DetailsViewDataGrid](#), you can add or remove the selection programmatically like parent DataGrid.

You can get particular [DetailsViewDataGrid](#) by using [DetailsViewLoading](#) event or [GetDetailsViewGrid](#) method to process the selection operations.

#### Selecting records

You can select the particular record by using [SelectedItem](#) property.

**C#**

```
this.dataGrid.DetailsViewLoading += dataGrid_DetailsViewLoading;  
void dataGrid_DetailsViewLoading (object sender,  
DetailsViewLoadingAndUnloadingEventArgs e)  
{  
    var record = e.DetailsViewDataGrid.GetRecordAtRowIndex(1);  
    e.DetailsViewDataGrid.SelectedItem = record;  
}
```

Here, the record in first row is selected in `DetailsViewDataGrid`.

You can also select the particular record by using [SelectedIndex](#) property.

#### C#

```
this.dataGrid.DetailsViewLoading += dataGrid_DetailsViewLoading;  
void dataGrid_DetailsViewLoading (object sender,  
DetailsViewLoadingAndUnloadingEventArgs e)  
{  
    e.DetailsViewDataGrid.SelectedIndex = 1;  
}
```

Here, the record in first position is selected in `DetailsViewDataGrid`.

You can select multiple records by using [SelectedItems](#) property.

You can refer [here](#) to know about handling events for `DetailsViewDataGrid`.

#### Row selection

You can select multiple rows by using [SelectRows](#) method.

#### C#

```
var detailsViewDataGrid = this.dataGrid.GetDetailsViewGrid(2);  
detailsViewDataGrid.SelectionController.SelectRows(1, 2);
```

Here, first and second rows are selected in the `DetailsViewDataGrid` which is present in second row of the parent `DataGrid`.

#### Cell selection

You can select cells also by using [SelectCell](#) and [SelectedCells](#) method. You can refer the [Selection](#) section.

#### *Programmatically expand and bring DetailsViewDataGrid into view*

SfDataGrid allows you to bring the specified [DetailsViewDataGrid](#) in to view by using [DetailsViewManager.BringInToView](#) method.

Before bringing the `DetailsViewDataGrid` into view, you have to expand the corresponding parent record if it is not already expanded.

#### C#

```
// parent DataGrid row index  
int parentRowIndex = 25;  
var record =  
this.dataGrid.View.Records[this.dataGrid.ResolveToRecordIndex(parentRowIndex)];
```

```
//Get the DetailsViewManager using Reflection
var propertyInfo = dataGrid.GetType().GetField("DetailsViewManager",
System.Reflection.BindingFlags.Instance |
System.Reflection.BindingFlags.NonPublic);
DetailsViewManager detailsViewManager = propertyInfo.GetValue(dataGrid) as
DetailsViewManager;
// Expand DetailsView at specified record index
if (!record.IsExpanded)
this.dataGrid.ExpandDetailsViewAt(this.dataGrid.ResolveToRecordIndex(parentRowIndex));
```

If the **DetailsViewDataGrid** is already expanded, you can use [ScrollInView](#) method to bring it into view. Else, you have to use [BringIntoView](#) method also.

### C#

```
// find DetailsViewDataRow index based on relational column
int index = 0;
foreach (var def in this.dataGrid.DetailsViewDefinition)
{
    if (def.RelationalColumn == "ProductDetails")
    {
        index = this.dataGrid.DetailsViewDefinition.IndexOf(def);
        index = parentRowIndex + index + 1;
    }
}
var rowColumnIndex = new RowColumnIndex(index, 1);
// if the DetailsViewDataGrid is already expanded, bring that into view
dataGrid.ScrollInView(rowColumnIndex);
//Get the DetailsViewDataGrid by passing the corresponding row index and
relation name
var detailsViewDataGrid =
this.dataGrid.GetDetailsViewGrid(this.dataGrid.ResolveToRecordIndex(parentRowIndex), "ProductDetails");
//if the DetailsViewDataGrid is not already expanded, call BringIntoView
method
if (detailsViewDataGrid == null)
{
    detailsViewManager.BringIntoView(index);
}
```

You can get the sample from [here](#).

### Customizing Selection for DetailsViewDataGrid

You can also customize the selection behavior of **DetailsViewDataGrid** like the parent DataGrid. For more information about customizing selection behavior, you can refer [here](#).

Follow the steps mentioned in selection customization section to customize selection behavior of [DetailsViewDataGrid](#) and set the customized selection controller to [DetailsViewDataGrid.SelectionController](#) in [DetailsViewLoading](#) event.

### C#

```
public class CustomSelectionController:GridSelectionController
{
```

```

public CustomSelectionController(SfDataGrid dataGrid)
:base(dataGrid)
{
}
}
this.dataGrid.DetailsViewLoading += dataGrid_DetailsViewLoading;
void dataGrid_DetailsViewLoading(object sender,
DetailsViewLoadingAndUnloadingEventArgs e)
{
if (!(e.DetailsViewDataGrid.SelectionController is
CustomSelectionController))
e.DetailsViewDataGrid.SelectionController = new
CustomSelectionController(e.DetailsViewDataGrid);
}

```

**Note:** For customizing selection in second level nested grid, you can refer [here](#).

#### Appearance customization

The visual appearance of [DetailsViewDataGrid](#) can be customized like parent DataGrid through [Styling and Templates support](#) in SfDataGrid.

#### Changing Header appearance of DetailsViewDataGrid

You can customize the header appearance of [DetailsViewDataGrid](#), through [HeaderStyle](#) property of [DetailsViewDataGrid](#).

#### XML

```

<Window.Resources>
<Style x:Key="headerStyle" TargetType="syncfusion:GridHeaderCellControl">
<Setter Property="Background" Value="Red" />
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AutoGenerateColumns="False"
HeaderStyle="{StaticResource headerStyle}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="OrderID" />
<syncfusion:GridTextColumn MappingName="ProductName" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>

```

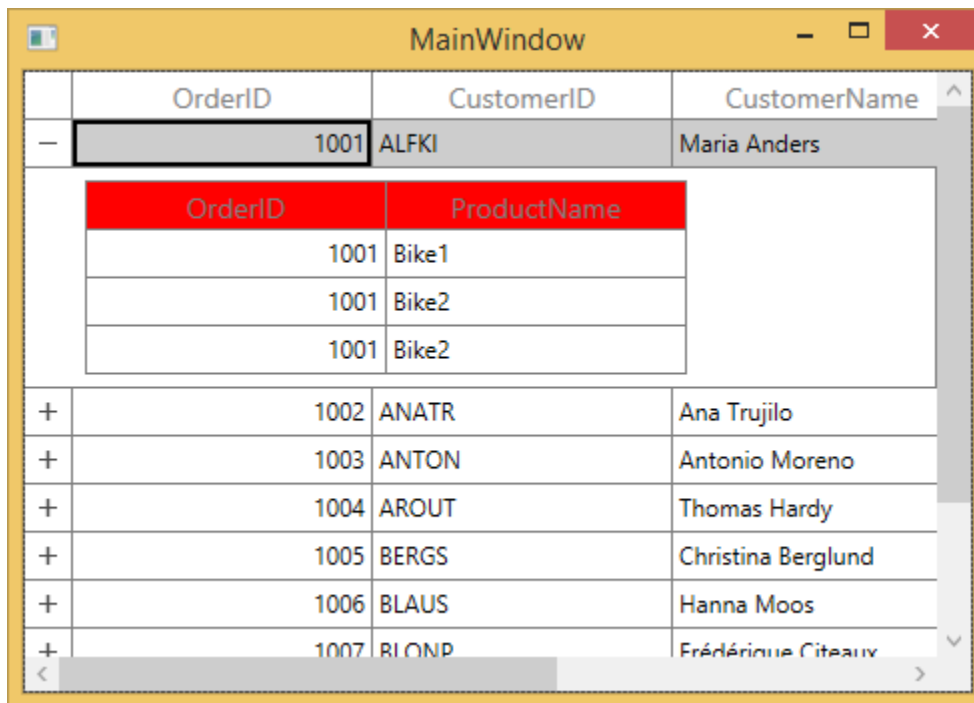
When the relation is auto-generated, you can assign the customized header style to [ViewDefinition.DataGrid](#) in [AutoGeneratingRelations](#) event.

**C#**

```

this.dataGrid.AutoGeneratingRelations += dataGrid_AutoGeneratingRelations;
void dataGrid_AutoGeneratingRelations(object sender,
Syncfusion.UI.Xaml.Grid.AutoGeneratingRelationsArgs e)
{
    e.GridViewDefinition.DataGrid.HeaderStyle = this.FindResource("headerStyle")
    as Style;
}

```



*Hiding header row of Master-Details View*

You can hide the header row of [DetailsViewDataGrid](#) by setting [HeaderRowHeight](#) property.

**XML**

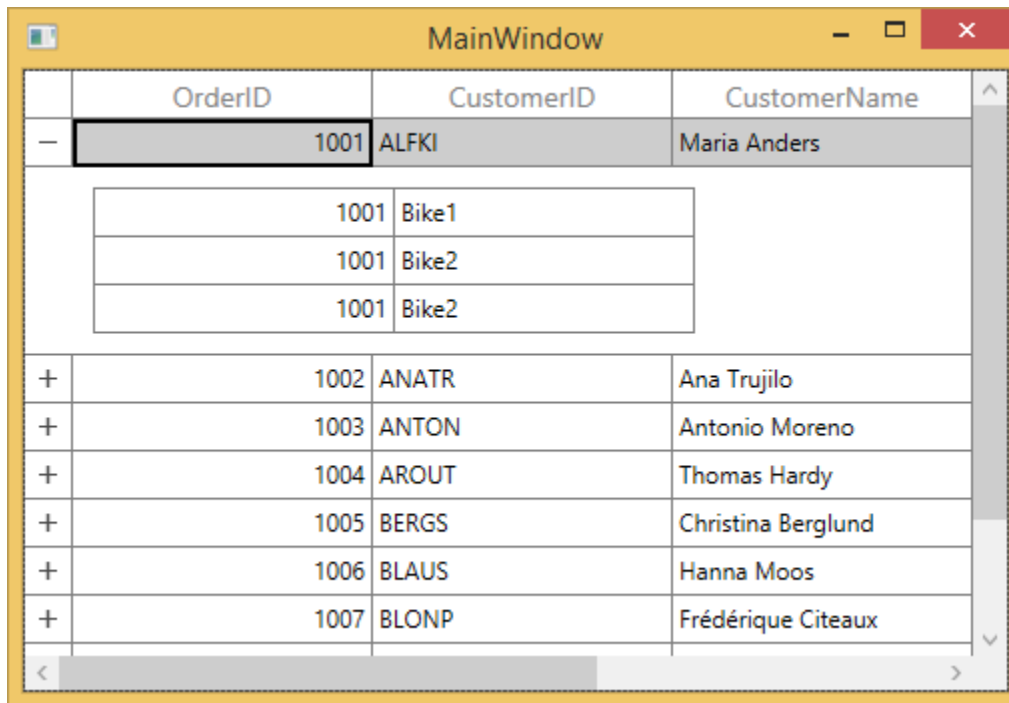
```

<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    AutoGenerateRelations="False"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.DetailsViewDefinition>
    <syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
    <syncfusion:GridViewDefinition.DataGrid>
    <syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
        AutoGenerateColumns="True"
        HeaderRowHeight="0" />
    </syncfusion:GridViewDefinition.DataGrid>
    </syncfusion:GridViewDefinition>
    </syncfusion:SfDataGrid.DetailsViewDefinition>
    </syncfusion:SfDataGrid>

```

**C#**

```
FirstLevelNestedGrid.HeaderRowHeight = 0;
```



#### Customizing padding of the DetailsViewDataGrid

The padding of [DetailsViewDataGrid](#) can be customized through the [DetailsViewPadding](#) property and it will be set to its corresponding parent DataGrid.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    AutoGenerateRelations="False"
    DetailsViewPadding="15"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.DetailsViewDefinition>
    <syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
    <syncfusion:GridViewDefinition.DataGrid>
    <syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
        AutoGenerateColumns="True" />
    </syncfusion:GridViewDefinition.DataGrid>
    </syncfusion:GridViewDefinition>
    </syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.DetailsViewPadding = new Thickness(15);
```

	OrderID	CustomerID	CustomerName								
—	1001	ALFKI	Maria Anders								
<table><tr><td>OrderID</td><td>ProductName</td></tr><tr><td>1001</td><td>Bike1</td></tr><tr><td>1001</td><td>Bike2</td></tr><tr><td>1001</td><td>Bike2</td></tr></table>				OrderID	ProductName	1001	Bike1	1001	Bike2	1001	Bike2
OrderID	ProductName										
1001	Bike1										
1001	Bike2										
1001	Bike2										
—	1002	ANATR	Ana Trujilo								
<table><tr><td>OrderID</td><td>ProductName</td></tr><tr><td>1002</td><td>Bike2</td></tr><tr><td>1002</td><td>Bike1</td></tr><tr><td>1002</td><td>Bike3</td></tr></table>				OrderID	ProductName	1002	Bike2	1002	Bike1	1002	Bike3
OrderID	ProductName										
1002	Bike2										
1002	Bike1										
1002	Bike3										
+	1003	ANTON	Antonio Moreno								
+	1004	AROUT	Thomas Hardy								
+	1005	BERGS	Christina Berglund								
+	1006	BLAUS	Hanna Moos								

**Note:** For customizing appearance for second level nested grid, you can refer [here](#).

#### Customize ExpanderColumn width

You can customize the width of ExpanderColumn in SfDataGrid by using [ExpanderColumnWidth](#) property as like below.

#### XML

```
<Syncfusion:SfDataGrid x:Name="dataGrid"
ExpanderColumnWidth="50"
AutoGenerateRelations="True"
ItemsSource="{Binding OrderInfoCollection }">
```

#### C#

```
this.dataGrid.ExpanderColumnWidth = 50;
```

#### Expanding and collapsing the DetailsViewDataGrid programmatically

SfDataGrid allows you to expand or collapse the [DetailsViewDataGrid](#) programmatically in different ways.

#### Expand or collapse all the DetailsViewDataGrid

You can expand or collapse all the [DetailsViewDataGrid](#) programmatically by using [ExpandAllDetailsView](#) and [CollapseAllDetailsView](#) methods.

#### C#



```
this.dataGrid.ExpandAllDetailsView();  
this.dataGrid.CollapseAllDetailsView();
```

#### *Expand DetailsViewDataGrid based on level*

You can expand all the [DetailsViewDataGrid](#) programmatically based on level using [ExpandAllDetailsView](#) method.

#### **C#**

```
this.dataGrid.ExpandAllDetailsView(2);
```

Here, all the DetailsViewDataGrids up to second level will be expanded.

#### *Expand or collapse Details View based on record index*

You can expand or collapse [DetailsViewDataGrid](#) based on the record index by using [ExpandDetailsViewAt](#) and [CollapseDetailsViewAt](#) methods.

#### **C#**

```
this.dataGrid.ExpandDetailsViewAt(0);  
this.dataGrid.CollapseDetailsViewAt(0);
```

Hiding expander when parent record's relation property has an empty collection or null

By default, the expander will be visible for all the data rows in parent DataGrid even if its [RelationalColumn](#) property has an empty collection or null.

You can hide the expander from the view when corresponding [RelationalColumn](#) property has an empty collection or null, by setting [HideEmptyGridViewDefinition](#) property as true.

#### **XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"  
AutoGenerateColumns="True"  
AutoGenerateRelations="True"  
HideEmptyGridViewDefinition="True"  
ItemsSource="{Binding Orders}" />
```

	OrderID	CustomerID	CustomerName
+	1001	ALFKI	Maria Anders
+	1002	ANATR	Ana Trujilo
+	1003	ANTON	Antonio Moreno
+	1004	AROUT	Thomas Hardy
+	1005	BERGS	Christina Berglund
+	1006	BLAUS	Hanna Moos
+	1007	BLONP	Frédérique Citeaux
+	1008	BOLID	Martin Sommer
	1009	BONAP	Laurence Lebihan
	1010	BOTTM	Elizabeth Lincoln

Hiding GridDetailsViewIndentCell in SfDataGrid

[GridDetailsViewIndentCell](#) is used to indicate the space between the expander and first column of the [DetailsViewDataGrid](#). You can hide the [GridDetailsViewIndentCell](#) by setting [SfDataGrid.ShowDetailsViewIndentCell](#) property to `False` for the respective parent grid.

	OrderID	EmployeeID	ContactID
- 1		1001	9896781
	OrderID	UnitPrice	Quantity
	1002	14	4
	1006	18	8
	1010	22	12
	1014	26	16
	1018	30	20
+ 2		1002	7898789
+ 3		1003	9896781

GridDetailsViewIndentCell

## XML

```
<syncfusion:SfDataGrid x:Name="dataGrid">
```

```

AutoGenerateColumns="True"
AutoGenerateRelations="True"
ShowDetailsViewIndentCell="False"
ItemsSource="{Binding Orders}" >
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="OrderDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AutoGenerateColumns="True"/>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>

```

**C#**

```
dataGrid.ShowDetailsViewIndentCell= False;
```

	OrderID	EmployeeID	ContactID
-	1	1001	9896781
	1002	14	4
	1006	18	8
	1010	22	12
	1014	26	16
	1018	30	20
+	2	1002	7898789
+	3	1003	9896781

### Hiding the details view expander icon based on child items count

By default, the state of expander icon is visible for all the data rows in parent DataGrid even if its `RelationalColumn` property has an empty collection or null.

You can customize hiding the details view expander icon by handling the `SfDataGrid.QueryDetailsViewExpanderState` event. This event occurs when expander icon is changed on expanding or collapsing the details view. You can hide the expander icon by setting the `ExpanderVisibility` property to `false` in the `SfDataGrid.QueryDetailsViewExpanderState` event based on condition.

**C#**

```

this.dataGrid.QueryDetailsViewExpanderState +=
DataGrid_QueryDetailsViewExpanderState;
private void DataGrid_QueryDetailsViewExpanderState(object sender,
Syncfusion.UI.Xaml.Grid.QueryDetailsViewExpanderStateEventArgs e)
{

```

```

var orderInfo = e.Record as OrderInfo;
if (orderInfo != null)
{
    if (orderInfo.OrderDetails.Count == 0)
    {
        e.ExpanderVisibility = false;
    }
}
}

```

The following screenshot illustrates how to hide the state of expander icon based on child items count.

	OrderID	CustomerID	Shipping Date	Supplier ID	Ship City
+	10000	BLONP	10/8/2011	5	Bruxelles
+	10001	LINOD	7/4/2008	3	Tsawassen
+	10002	RISCU	4/15/2009	4	Bruxelles
+	10003	BLONP	12/24/2008	8	Graz
+	10004	FRANS	4/20/2011	8	Montréal
+	10005	SIMOB	6/13/2009	9	Rio de Janeiro
+	10006	SIMOB	11/11/2010	6	Campinas
	10007	PICCO	7/29/2011	1	Bruxelles
	10008	WELLI	5/4/2009	1	Buenos Aires
+	10009	WARTH	3/20/2010	3	Tsawassen
+	10010	FOLKO	10/1/2010	4	Bruxelles

Expander icon is hidden

You can download the sample from the following link: [Sample](#).

Change `DetailsViewDataGrid ItemsSource` at runtime using `LiveDataUpdateMode` property. `ItemsSource` for `DetailsViewDataGrid` is populated from the `DataContext` of parent row based on `ViewDefinition.RelationalColumn`. `DetailsViewDataGrid` doesn't update its `ItemsSource` at runtime based on the property change, which is mapped the `DetailsViewDataGrid ItemsSource`. You can update the `ItemsSource` on the property change by setting `SfDataGrid.LiveDataUpdateMode` as `AllowChildViewUpdate`.

#### XML

```

<Syncfusion:SfDataGrid Name="dataGrid"
    AutoGenerateColumns="True"
    AutoGenerateRelations="True"
    ItemsSource="{Binding Source}"
    LiveDataUpdateMode="AllowChildViewUpdate,AllowDataShaping">

```

#### C#

```

this.dataGrid.LiveDataUpdateMode = LiveDataUpdateMode.AllowChildViewUpdate |
    LiveDataUpdateMode.AllowDataShaping;

```

You can get the sample from [here](#).

### Refreshing UI while adding records to relation property at run time

By default, the expander is hidden in row, when the [HideEmptyGridViewDefinition](#) is set to `true` and [RelationalColumn](#) property has an empty collection or null. You can refresh row to display expander when records are added to [RelationalColumn](#) property by calling [UpdateDataRow](#) method.

For example, if you try to add the new record in [ProductDetails](#) collection in the parent record having [OrderID](#) as 1009 and 1010 at run time, the new record is added but the expander is not shown. But it needs to be shown in UI since [RelationalColumn](#) property has record now. In this case, you need to refresh the particular data row to display expander by using [UpdateDataRow](#) helper method.

#### C#

```
var dataContext = DataContext as OrderInfoRepository;
var data = dataContext.Orders.Where(item => item.OrderID ==
1009).FirstOrDefault();
var newItem = new List<ProductInfo>();
newItem.Add(new ProductInfo() { OrderID = 1009, ProductName = "Bike" });
data.ProductDetails = newItem;
this.dataGrid.UpdateDataRow(dataGrid.ResolveToRowIndex(data));
```

### Handling Events

#### [DetailsViewLoading](#)

The [DetailsViewLoading](#) event is raised, when the [DetailsViewDataGrid](#) is being loaded in to the view (such as scrolling, window size changed and expanding the record using an expander or programmatically).

This event receives two arguments where sender as SfDataGrid and [DetailsViewLoadingAndUnloadingEventArgs](#) which contains the following member.

- [DetailsViewDataGrid](#) - Gets the [DetailsViewDataGrid](#) which is loaded into view. You can set the customized [Renderers](#), [SelectionController](#), [ResizingController](#), [GridColumnDragDropController](#), and [GridColumnSizer](#) to this. But it is not preferable to change the value of the public properties like [AllowFiltering](#), [AllowSorting](#), [SelectionUnit](#), [AllowDeleting](#), etc., here.

#### C#

```
this.dataGrid.DetailsViewLoading += dataGrid_DetailsViewLoading;
void dataGrid_DetailsViewLoading(object sender,
DetailsViewLoadingAndUnloadingEventArgs e)
{
if (!(e.DetailsViewDataGrid.SelectionController is
CustomSelectionController))
e.DetailsViewDataGrid.SelectionController = new
CustomSelectionController(e.DetailsViewDataGrid);
}
```

#### [DetailsViewUnLoading](#)

The [DetailsViewUnLoading](#) event is raised when the [DetailsViewDataGrid](#) is being unloaded from the view.

This event receives two arguments where sender as SfDataGrid and [DetailsViewLoadingAndUnloadingEventArgs](#) which contains the following member.

- [DetailsViewDataGrid](#) - Gets the [DetailsViewDataGrid](#) which was unloaded from the view (such as scrolling, window size changed, Sorting, Grouping, Filtering and collapsing the DetailsViewDataGrid using expander or programmatically).

#### C#

```
this.dataGrid.DetailsViewUnloading += dataGrid_DetailsViewUnloading;  
void dataGrid_DetailsViewUnloading(object sender,  
DetailsViewLoadingAndUnloadingEventArgs e)  
{  
}
```

#### *DetailsViewExpanding*

The [DetailsViewExpanding](#) event is raised when the [DetailsViewDataGrid](#) is being expanded by using an expander.

#### C#

```
this.dataGrid.DetailsViewExpanding += dataGrid_DetailsViewExpanding;  
void dataGrid_DetailsViewExpanding(object sender,  
Syncfusion.UI.Xaml.Grid.GridDetailsViewExpandingEventArgs e)  
{  
}
```

#### *DetailsViewExpanded*

The [DetailsViewExpanded](#) event is raised after the [DetailsViewDataGrid](#) is expanded by using an expander.

#### C#

```
this.dataGrid.DetailsViewExpanded += dataGrid_DetailsViewExpanded;  
void dataGrid_DetailsViewExpanded(object sender,  
GridDetailsViewExpandedEventArgs e)  
{  
}
```

#### *DetailsViewCollapsing*

The [DetailsViewCollapsing](#) event is raised when the [DetailsViewDataGrid](#) is being collapsed from the view by using an expander.

#### C#

```
this.dataGrid.DetailsViewCollapsing += dataGrid_DetailsViewCollapsing;  
void dataGrid_DetailsViewCollapsing(object sender,  
Syncfusion.UI.Xaml.Grid.GridDetailsViewCollapsingEventArgs e)  
{  
}
```

### *DetailsViewCollapsed*

The [DetailsViewCollapsed](#) event is raised after the [DetailsViewDataGrid](#) is collapsed by using an expander.

#### **C#**

```
this.dataGrid.DetailsViewCollapsed += dataGrid_DetailsViewCollapsed;
void dataGrid_DetailsViewCollapsed(object sender,
GridDetailsViewCollapsedEventArgs e)
{
}
```

### *Cancel expanding or collapsing operations through events*

You can cancel expanding operation while expanding the [DetailsViewDataGrid](#) by using [GridDetailsViewExpandingEventArgs.Cancel](#) property in the [DetailsViewExpanding](#) event handler.

#### **C#**

```
this.dataGrid.DetailsViewExpanding += dataGrid_DetailsViewExpanding;
void dataGrid_DetailsViewExpanding(object sender,
Syncfusion.UI.Xaml.Grid.GridDetailsViewExpandingEventArgs e)
{
    if ((e.Record as OrderInfo).OrderID == 1002)
        e.Cancel = true;
}
```

Similarly, the collapsing operation can be canceled through the [GridDetailsViewCollapsingEventArgs.Cancel](#) property in the [DetailsViewCollapsing](#) event handler.

#### **C#**

```
this.dataGrid.DetailsViewCollapsing += dataGrid_DetailsViewCollapsing;
void dataGrid_DetailsViewCollapsing(object sender,
Syncfusion.UI.Xaml.Grid.GridDetailsViewCollapsingEventArgs e)
{
    if ((e.Record as OrderInfo).OrderID == 1002)
        e.Cancel = true;
}
```

**Note:** To cancel expanding or collapsing operation in second level nested grid, you can refer [here](#).

### *Master-Details View limitations*

Following are the limitations of Master-Details View in SfDataGrid.

1. [DetailsViewDataGrid](#) does not have [GroupDropArea](#).
2. [DetailsViewDataGrid](#) does not support [AutoGenerateColumnsMode.ResetAll](#). Instead it works based on [Reset](#).
3. Master-Details View doesn't support Data Virtualization.
4. Master-Details View doesn't support [AllowFrozenGroupHeader](#).
5. Master-Details View doesn't support Freeze Pane.
6. Master-Details View doesn't support [AutoRowHeight](#).

7. For `DetailsViewDataGrid`, `SelectionMode`, `SelectionUnit`, `NavigationMode`, `DetailsViewPadding` properties are assigned from its parent grid only. So both parent `DataGrid` and `DetailsViewDataGrid` cannot have different values for these properties.

See Also

[How to maintain the DetailsView expanded state when Sorting and Grouping the DataGrid \(SfDataGrid\)?](#)

[How to set the background for selected DetailsViewGrid?](#)

[How to load the button command inside the Detailsview datagrid column and setting AncestorLevel?](#)

[How to export the DetailsView records in expanded state to Excel?](#)

[How to print the SfDataGrid with DetailsView?](#)

[How to get the SelectedItem of the DetailsView?](#)

[How to bind SelectedItem and CurrentItem in DetailsViewDataGrid](#)

[How to get the CurrentCell for DetailsViewDatagrid at runtime?](#)

[How to add a new record in specific DetailsViewDataGrid ?](#)

[How to hide the key mapping column in DetailsViewDataGrid?](#)

[How to resize the parent grid and DetailsViewDataGrid simultaneously?](#)

[How to create Custom Column in the DetailsViewDataGrid?](#)

[How to change the background color of the Header alone in DetailsView or Nested Grid?](#)

[How to prevent resizing the last column, when parent Grid width is less than the child Grid width?](#)

[How to change the ColumnSizer for the Nested grid?](#)

[How to get the parent grid while editing the child grid?](#)



[How to enable NestedGrid when I don't have relations in my datasource](#)

[How to hide the HeaderRow in Nested Grid?](#)

[Record Template View in WPF DataGrid \(SfDataGrid\)](#)

The `SfDataGrid` provides support to represent additional information of a row using [TemplateViewDefinition](#) that can be defined in datagrid. It allows you to load any WPF controls to [TemplateViewDefinition.RowTemplate](#) in order to display the additional information of a row. You can expand or collapse the row template view by using an expander in a row or programmatically.



Order ID	Customer ID	ProductName	Quantity	Freight	IsClosed
10000	FRANS	Alice Mutton	4	\$4.45	<input type="checkbox"/>
<div> <div>  <div> <b>Order Details</b>            Product Name : Alice Mutton            Quantity : 4            Unit Price : \$27.00            Discount : 0.00%            Freight : \$4.45         </div> </div> <div> <b>Shipping Details</b>            Order Date : 5/10/1991            Shipped Date : 5/15/1991            Ship Address : Via Monte Bianco 34            Ship Postal Code : 10100            Delivery Delay : 05:00         </div> </div>					
10001	MEREP	NuNuCa Nuß-Nougat-C	30	\$79.45	<input checked="" type="checkbox"/>
<div> <div>  <div> <b>Order Details</b>            Product Name : NuNuCa Nuß-Nougat-Creme            Quantity : 30            Unit Price : \$9.80            Discount : 15.00%         </div> </div> <div> <b>Shipping Details</b>            Order Date : 5/13/1991            Shipped Date : 5/23/1991            Ship Address : 43 rue St. Laurent            Ship Postal Code : H1J 1C3         </div> </div>					

### Defining row template

Template view can be generated for the master row by using the [RowTemplate](#) defined in the [TemplateViewDefinition](#).

Follow the below steps to define the row template,

- Create [TemplateViewDefinition](#).
- Define data template for the [TemplateViewDefinition.RowTemplate](#) property.
- Then add [TemplateViewDefinition](#) to the [SfDataGrid.DetailsViewDefinition](#).

You can bind the row data using the `Data.PropertyName` (where Data is the underlying object bound).

### XML

```
<Window.Resources>
<DataTemplate x:Key="DetailsViewTemplate">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="125"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="250"/>
<ColumnDefinition/>
<ColumnDefinition/>
</Grid.ColumnDefinitions>
<Image Grid.Row="0" Grid.Column="0" Grid.RowSpan="2" Margin="5" Height="150"
HorizontalAlignment="Left"
Source="{Binding Path=Data.CustomerID, Converter={StaticResource
ImageConverter}}"/>
<Label Grid.Column="1" Grid.Row="0" HorizontalAlignment="Left"
Margin="25,0,0,0" Content="Order Details" FontWeight="Bold" />
<StackPanel Orientation="Vertical" Grid.Column="1" Grid.Row="1"
HorizontalAlignment="Left" VerticalAlignment="Center">
<StackPanel Orientation="Horizontal">
<Label HorizontalAlignment="Left" VerticalAlignment="Center"
FontWeight="SemiBold"
Content="Product Name :"/>
<TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" Margin="5"
Text="{Binding Data.ProductName}"/>

```

```

</StackPanel>
<StackPanel Orientation="Horizontal">
<Label HorizontalAlignment="Left" VerticalAlignment="Center"
FontWeight="SemiBold" Content="Quantity :"/>
<TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" Margin="5"
Text="{Binding Data.Quantity}"/>
</StackPanel>
<StackPanel Orientation="Horizontal">
<Label HorizontalAlignment="Left" VerticalAlignment="Center"
FontWeight="SemiBold" Content="Unit Price :"/>
<TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" Margin="5"
Text="{Binding Data.UnitPrice, StringFormat='{0:C}'}"/>
</StackPanel>
<StackPanel Orientation="Horizontal">
<Label HorizontalAlignment="Left" VerticalAlignment="Center"
FontWeight="SemiBold" Content="Discount :"/>
<TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" Margin="5"
Text="{Binding Data.Discount, StringFormat='{0:P}'}"/>
</StackPanel>
<StackPanel Orientation="Horizontal">
<Label HorizontalAlignment="Left" VerticalAlignment="Center"
FontWeight="SemiBold" Content="Freight :"/>
<TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" Margin="5"
Text="{Binding Data.Freight, StringFormat='{0:c}'}"/>
</StackPanel>
</StackPanel>
<Label Grid.Column="2" Grid.Row="0" HorizontalAlignment="Left"
Margin="25,0,0,0"
Content="Shipping Details" FontWeight="Bold" />
<StackPanel Orientation="Vertical" Grid.Column="2" Grid.Row="1"
HorizontalAlignment="Left" VerticalAlignment="Center">
<StackPanel Orientation="Horizontal">
<Label HorizontalAlignment="Right" VerticalAlignment="Center"
FontWeight="SemiBold" Content="Order Date :"/>
<TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" Margin="5"
Text="{Binding Data.OrderDate, StringFormat=d}"/>
</StackPanel>
<StackPanel Orientation="Horizontal">
<Label HorizontalAlignment="Right" VerticalAlignment="Center"
FontWeight="SemiBold" Content="Shipped Date :"/>
<TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" Margin="5"
Text="{Binding Data.ShippedDate, StringFormat=d}"/>
</StackPanel>
<StackPanel Orientation="Horizontal">
<Label HorizontalAlignment="Right" VerticalAlignment="Center"
FontWeight="SemiBold" Content="Ship Address :"/>
<TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" Margin="5"
Text="{Binding Data.ShipAddress}"/>
</StackPanel>
<StackPanel Orientation="Horizontal">
<Label HorizontalAlignment="Right" VerticalAlignment="Center"
FontWeight="SemiBold" Content="Ship Postal Code :"/>
<TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" Margin="5"
Text="{Binding Data.ShipPostalCode, StringFormat=hh\\:mm}"/>
</StackPanel>
<StackPanel Orientation="Horizontal">
<Label HorizontalAlignment="Right" VerticalAlignment="Center"

```

```

FontWeight="SemiBold" Content="Delivery Delay :"/>
<TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" Margin="5"
Text="{Binding Data.DeliveryDelay, StringFormat=dd\\:hh}"/>
</StackPanel>
</StackPanel>
</Grid>
</DataTemplate>
</Window.Resources>
<syncfusion:SfDataGrid Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrderList}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:TemplateViewDefinition RowTemplate="{StaticResource
DetailsViewTemplate}"/>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>

```

### Defining RowTemplateSelector

[TemplateViewDefinition](#) provides support to choose different [DataTemplate](#) based on underlying data object using [TemplateViewDefinition.RowTemplateSelector](#) property.

### XML

```

<Application.Resources>
<DataTemplate x:Key="ProductInfo">
<Grid HorizontalAlignment="Left">
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Label Grid.Row="0" HorizontalAlignment="Center" Content="Prodcut Info"
FontWeight="Bold" />
<Grid Grid.Row="1" HorizontalAlignment="Left" VerticalAlignment="Center">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="120" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<TextBlock Grid.Row="0" Grid.Column="0" HorizontalAlignment="Left"
FontWeight="DemiBold" Text="Name" />
<TextBlock Grid.Row="0" Grid.Column="1" HorizontalAlignment="Left"
Text="{Binding Data.EnglishName}" />
<TextBlock Grid.Row="1" Grid.Column="0" HorizontalAlignment="Left"
FontWeight="DemiBold" Text="Quantity Per Unit" />
<TextBlock Grid.Row="1" Grid.Column="1" HorizontalAlignment="Left"
Text="{Binding Data.QuantityPerUnit}" />
<TextBlock Grid.Row="2" Grid.Column="0" HorizontalAlignment="Left"
FontWeight="DemiBold" Text="Unit Price" />
<TextBlock Grid.Row="2" Grid.Column="1" HorizontalAlignment="Left"
Text="{Binding Data.UnitPrice}" />
<TextBlock Grid.Row="3" Grid.Column="0" HorizontalAlignment="Left"

```

```

FontWeight="DemiBold" Text="Stock" />
<TextBlock Grid.Row="3" Grid.Column="1" HorizontalAlignment="Left"
Text="{Binding Data.UnitsInStock}" />
</Grid>
</Grid>
</DataTemplate>
<DataTemplate x:Key="SupplierInfo" >
<Grid HorizontalAlignment="Left">
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Label Grid.Row="0" HorizontalAlignment="Center" Content="Supplier Info"
FontWeight="Bold" />
<Grid HorizontalAlignment="Left" Grid.Row="1" >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="120" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<TextBlock Grid.Row="0" Grid.Column="0" HorizontalAlignment="Left"
FontWeight="DemiBold" Text="Company" />
<TextBlock Grid.Row="0" Grid.Column="1" HorizontalAlignment="Left"
Text="{Binding Data.Suppliers.CompanyName}" />
<TextBlock Grid.Row="1" Grid.Column="0" HorizontalAlignment="Left"
FontWeight="DemiBold" Text="Contact Person" />
<TextBlock Grid.Row="1" Grid.Column="1" HorizontalAlignment="Left"
Text="{Binding Data.Suppliers.ContactName}" />
<TextBlock Grid.Row="2" Grid.Column="0" HorizontalAlignment="Left"
FontWeight="DemiBold" Text="Title" />
<TextBlock Grid.Row="2" Grid.Column="1" HorizontalAlignment="Left"
Text="{Binding Data.Suppliers.ContactTitle, Mode=OneTime}" />
<TextBlock Grid.Row="3" Grid.Column="0" HorizontalAlignment="Left"
FontWeight="DemiBold" Text="Country" />
<TextBlock Grid.Row="3" Grid.Column="1" HorizontalAlignment="Left"
Text="{Binding Data.Suppliers.Country}" />
</Grid>
</Grid>
</DataTemplate>
</Application.Resources>
<Window.Resources>
<local:detailsViewTemplateSelector x:Key="EditTemplateSelector"/>
</Window.Resources>
<syncfusion:SfDataGrid Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrderList}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:TemplateViewDefinition RowTemplateSelector="{StaticResource
EditTemplateSelector}" />
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>

```

**C#**

```

public class detailsViewTemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
        var productId = ((item as RecordEntry).Data as Products).ProductID;
        if (productId % 2 == 0)
            return Application.Current.FindResource("SupplierInfo") as DataTemplate;
        else
            return Application.Current.FindResource("ProductInfo") as DataTemplate;
    }
}

```

	ProductID	SupplierID	ProductName	QuantityPerUnit	Discontinued
-	1	1	Chai	10 boxes x 20 bags	<input type="checkbox"/>
<b>Product Info</b> Name Dharamsala Tea Quantity Per Unit 10 boxes x 20 bags Unit Price 18 Stock 39					
-	2	1	Chang	24 - 12 oz bottles	<input type="checkbox"/>
<b>Supplier Info</b> Company Exotic Liquids Contact Person Charlotte Cooper Title Purchasing Manager Country UK					
+	3	1	Aniseed Syrup	12 - 550 ml bottles	<input type="checkbox"/>
+	4	2	Chef Anton's Cajun Seaso	48 - 6 oz jars	<input type="checkbox"/>
+	5	2	Chef Anton's Gumbo Mix	36 boxes	<input checked="" type="checkbox"/>
+	6	3	Grandma's Boysenberry S	12 - 8 oz jars	<input type="checkbox"/>
+	7	3	Uncle Bob's Organic Dried	12 - 1 lb pkgs.	<input type="checkbox"/>

## Height customization

*Height mode*

You can customize height of the row that contains [RowTemplate](#) by using the [TemplateViewDefinition.HeightMode](#) property. The height modes are as follows.

Height mode	Definition
Auto	Arranges template for the actual size as the [RowTemplate](https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.TemplateViewDefinition.html#SyncfusionUIXamlGridTemplateViewDefinition_RowTemplate) is measured.

Fixed	Arranges template for the specified height in <a href="#">TemplateViewDefinition.Height</a> .
View Port	Arranges template for the ViewPortHeight when the <code>{{'RowTemplate'  markdownify}}</code> actual height is greater than ViewPortHeight.

### Keyboard navigation support for DetailsViewTemplate

In the SfDataGrid, you can navigate from parent row to DetailsViewTemplate and vice-versa using Tab key by default. You can also restrict tab key navigation from parent to DetailsViewTemplate by setting the [TemplateViewDefinition.TemplateNavigationMode](#) property value to `ExcludeTemplateRow`.

### XML

```
<syncfusion:SfDataGrid Name="dataGrid"
    AutoGenerateColumns="False"
    ItemsSource="{Binding OrderList}">
    <syncfusion:SfDataGrid.DetailsViewDefinition>
    <syncfusion:TemplateViewDefinition RowTemplate="{StaticResource
    DetailsViewTemplate}" TemplateNavigationMode="ExcludeTemplateRow`"/>
    </syncfusion:SfDataGrid.DetailsViewDefinition>
    </Syncfusion.SfDataGrid>
```

**Note:** Except Tab key, other keys does not allow keyboard navigation from parent row to DetailsViewTemplate and vice-versa.

### Populating record template view using events

You can set the [RowTemplate](#) on-demand when expanding the record by using the [GridDetailsViewExpandingEventArgs.RowTemplate](#) property in [SfDataGrid.DetailsViewExpanding](#) event handler.

### XML

```
<syncfusion:SfDataGrid Name="dataGrid"
    AutoGenerateColumns="False"
    ItemsSource="{Binding OrderList}">
    <syncfusion:SfDataGrid.DetailsViewDefinition>
    <syncfusion:TemplateViewDefinition RowTemplate="{StaticResource
    DetailsViewTemplate}"/>
    </syncfusion:SfDataGrid.DetailsViewDefinition>
    </Syncfusion.SfDataGrid>
```

### C#

```
private DataTemplate GetDataTemplate()
{
    FrameworkElementFactory textBlock = new
    FrameworkElementFactory(typeof(TextBlock));
    Binding binding = new Binding();
    textBlock.SetBinding(TextBlock.TextProperty, binding);
    binding.Path = new PropertyPath("Data.ProductName");
    var dataTemplate = new DataTemplate() { VisualTree = textBlock };
    return dataTemplate;
}
```

```
}  
private void dataGrid_DetailsViewExpanding(object sender,  
Syncfusion.UI.Xaml.Grid.GridDetailsViewExpandingEventArgs e)  
{  
    e.RowTemplate = GetDataTemplate();  
}
```

### Expanding and collapsing row template programmatically

The SfDataGrid allows you to expand or collapse the [RowTemplate](#) programmatically in different ways.

#### *Expand or collapse all the row template*

You can expand or collapse all the [RowTemplate](#) by using the [ExpandAllDetailsView](#) and [CollapseAllDetailsView](#) methods.

#### **C#**

```
this.dataGrid.ExpandAllDetailsView();  
this.dataGrid.CollapseAllDetailsView();
```

#### *Expand or collapse row template based on record index*

You can expand or collapse the [RowTemplate](#) based on the record index by using the [ExpandDetailsViewAt](#) and [CollapseDetailsViewAt](#) methods.

#### **C#**

```
this.dataGrid.ExpandDetailsViewAt(0);  
this.dataGrid.CollapseDetailsViewAt(0);
```

### Handling events

#### *DetailsViewExpanding*

The [DetailsViewExpanding](#) event is raised when the [RowTemplate](#) is expanded by using an expander.

#### **C#**

```
this.dataGrid.DetailsViewExpanding += dataGrid_DetailsViewExpanding;  
void dataGrid_DetailsViewExpanding(object sender,  
Syncfusion.UI.Xaml.Grid.GridDetailsViewExpandingEventArgs e)  
{  
}
```

#### *DetailsViewExpanded*

The [DetailsViewExpanded](#) event is raised when the [RowTemplate](#) is expanded by using an expander.

#### **C#**

```
this.dataGrid.DetailsViewExpanded += dataGrid_DetailsViewExpanded;  
void dataGrid_DetailsViewExpanded(object sender,  
Syncfusion.UI.Xaml.Grid.GridDetailsViewExpandingEventArgs e)  
{  
}
```

#### *DetailsViewCollapsing*

The [DetailsViewCollapsing](#) event is raised when the [RowTemplate](#) is collapsed by using an expander.

**C#**

```
this.dataGrid.DetailsViewCollapsing += dataGrid_DetailsViewCollapsing;  
void dataGrid_DetailsViewCollapsing(object sender,  
Syncfusion.UI.Xaml.Grid.GridDetailsViewExpandingEventArgs e)  
{  
}
```

*DetailsViewCollapsed*

The [DetailsViewCollapsed](#) event is raised when the [RowTemplate](#) is collapsed by using an expander.

**C#**

```
this.dataGrid.DetailsViewCollapsed += dataGrid_DetailsViewCollapsed;  
void dataGrid_DetailsViewCollapsed(object sender,  
Syncfusion.UI.Xaml.Grid.GridDetailsViewExpandingEventArgs e)  
{  
}
```

*Cancel expanding or collapsing operations through events*

You can cancel the expanding operation when expanding the [RowTemplate](#) by using the [GridDetailsViewExpandingEventArgs.Cancel](#) property in the [DetailsViewExpanding](#) event handler.

**C#**

```
this.dataGrid.DetailsViewExpanding += dataGrid_DetailsViewExpanding;  
void dataGrid_DetailsViewExpanding(object sender,  
Syncfusion.UI.Xaml.Grid.GridDetailsViewExpandingEventArgs e)  
{  
    if ((e.Record as OrderInfo).OrderID == 1002)  
        e.Cancel = true;  
}
```

Similarly, the collapsing operation can also be canceled through the [GridDetailsViewCollapsingEventArgs.Cancel](#) property in [DetailsViewCollapsing](#) event handler.

**C#**

```
this.dataGrid.DetailsViewCollapsing += dataGrid_DetailsViewCollapsing;  
void dataGrid_DetailsViewCollapsing(object sender,  
Syncfusion.UI.Xaml.Grid.GridDetailsViewCollapsingEventArgs e)  
{  
    if ((e.Record as OrderInfo).OrderID == 1002)  
        e.Cancel = true;  
}
```

*Limitations*

Limitations are:

- Does not support both [DetailsViewTemplate](#) and [DetailsViewDataGrid](#) at same level.
- Does not support more than one [DetailsViewTemplate](#) in same level.



## Paging in WPF DataGrid (SfDataGrid)

SfDataGrid provides support to manipulate the data using SfDataPager control. You can refer [SfDataPager](#) control user guide for more information.

### Getting started

Follow the below steps to bind SfDataGrid with SfDataPager.

1. Create `IEnumerable` collection that you want to bind and set it to [SfDataPager.Source](#) property.
2. Set [SfDataPager.PageSize](#) property to specify the number of records to be displayed per page.
3. Bind [SfDataPager.PagedSource](#) to [SfDataGrid.ItemsSource](#) property. So whenever the page is changed, `PagedSource` will be update based on current page.

### XML

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<syncfusion:SfDataGrid x:Name="dataGrid"
Grid.Row="0"
ItemsSource="{Binding ElementName=dataPager, Path=PagedSource}" />
<syncfusion:SfDataPager x:Name="dataPager"
Grid.Row="1"
PageSize="5"
Source="{Binding Orders}" />
</Grid>
```

### C#

```
public class ViewModel
{
    private ObservableCollection<OrderInfo> _orders;
    public ObservableCollection<OrderInfo> Orders
    {
        get { return _orders; }
        set { _orders = value; }
    }
    public ViewModel()
    {
        _orders = new ObservableCollection<OrderInfo>();
        this.GenerateOrders();
    }
    private void GenerateOrders()
    {
        _orders.Add(new OrderInfo(1001, "Maria Anders", "Germany", "ALFKI",
"Berlin"));
        _orders.Add(new OrderInfo(1002, "Ana Trujillo", "Mexico", "ANATR", "Mexico
D.F.));
        _orders.Add(new OrderInfo(1003, "Antonio Moreno", "Mexico", "ANTON", "Mexico
D.F.));
        _orders.Add(new OrderInfo(1004, "Thomas Hardy", "UK", "AROUT", "London"));
        _orders.Add(new OrderInfo(1005, "Christina Berglund", "Sweden", "BERGS",
"Lula"));
    }
}
```

```
_orders.Add(new OrderInfo(1006, "Hanna Moos", "Germany", "BLAUS",  
"Mannheim"));  
_orders.Add(new OrderInfo(1007, "Frederique Citeaux", "France", "BLONP",  
"Strasbourg"));  
_orders.Add(new OrderInfo(1008, "Martin Sommer", "Spain", "BOLID",  
"Madrid"));  
_orders.Add(new OrderInfo(1009, "Laurence Lebihan", "France", "BONAP",  
"Marseille"));  
_orders.Add(new OrderInfo(1010, "Elizabeth Lincoln", "Canada", "BOTTM",  
"Tsawassen"));  
}  
}  
public class OrderInfo  
{  
    int orderID;  
    string customerId;  
    string country;  
    string customerName;  
    string shippingCity;  
    public int OrderID  
    {  
        get { return orderID; }  
        set { orderID = value; }  
    }  
    public string CustomerID  
    {  
        get { return customerId; }  
        set { customerId = value; }  
    }  
    public string CustomerName  
    {  
        get { return customerName; }  
        set { customerName = value; }  
    }  
    public string Country  
    {  
        get { return country; }  
        set { country = value; }  
    }  
    public string ShipCity  
    {  
        get { return shippingCity; }  
        set { shippingCity = value; }  
    }  
    public OrderInfo(int orderId, string customerName, string country, string  
customerId, string shipCity)  
    {  
        this.OrderID = orderId;  
        this.CustomerName = customerName;  
        this.Country = country;  
        this.CustomerID = customerId;  
        this.ShipCity = shipCity;  
    }  
}
```

TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund

### Limitations

1. SfDataPager doesn't accept `DataTable` as Source.
2. `AddNewRow` is not supported in SfDataPager.
3. `FilterRow` is not supported in SfDataPager.

### Load data in on-demand

SfDataPager allows you to load data for current page alone using `OnDemandPaging` instead of loading data for all pages.

Follow the below steps to load the `ItemsSource` for page in on-demand,

1. Set `SfDataPager.UseOnDemandPaging` as `true`.
2. Set `SfDataPager.PageCount` based on total number of records and `SfDataPager.PageSize` property.
3. Use `OnDemandLoading` event to load the `ItemsSource` for current page using `LoadDynamicItems` method.

`OnDemandLoading` event is raised when SfDataPager moves to another page and you can load the `ItemsSource` for corresponding page through `OnDemandLoading` event.

`OnDemandLoadingEventArgs` has the following members,

1. `StartIndex` - returns the start index based on `PageIndex` (Number of previous pages \* `PageSize`).
2. `PageSize` - denotes the number of records to be displayed in the page.

**Note:** Do not assign `SfDataPager.Source` property while using `OnDemandPaging`.

### XML

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<syncfusion:SfDataGrid x:Name="dataGrid"
Grid.Row="0"
ItemsSource="{Binding ElementName=dataPager,Path=PagedSource}"/>
<syncfusion:SfDataPager x:Name="dataPager"
PageCount="3"
PageSize="10"
UseOnDemandPaging="True"
Grid.Row="1"/>
</Grid>
```

### C#

```
private ObservableCollection<OrderInfo> source;
public MainWindow()
{
InitializeComponent();
dataPager.OnDemandLoading += dataPager_OnDemandLoading;
}
private void dataPager_OnDemandLoading(object sender,
Syncfusion.UI.Xaml.Controls.DataPager.OnDemandLoadingEventArgs args)
{
source = (this.DataContext as ViewModel).Orders;
dataPager.LoadDynamicItems(args.StartIndex,
source.Skip(args.StartIndex).Take(args.PageSize));
}
```

### Resetting cache

While navigating between the pages, records are loaded through `OnDemandLoading` event and the records of navigated pages will be maintained in cache. If you navigate to already navigated page, the records are loaded from cache instead of loading from `OnDemandLoading` event. You can clear the cache by using [PageCollectionView.ResetCache](#) method. Once this method is invoked, the `OnDemandLoading` event will be raised while navigating multiple times to the same page.

### XML

```
<syncfusion:SfDataPager x:Name="dataPager"
PageCount="3"
PageSize="10"
UseOnDemandPaging="True"
Grid.Row="1"/>
```

### C#

```
ObservableCollection<OrderInfo> source;
public MainWindow()
```

```

{
    InitializeComponent();
    dataPager.OnDemandLoading += dataPager_OnDemandLoading;
    source = (this.DataContext as ViewModel).Orders;
}
private void dataPager_OnDemandLoading(object sender,
    Syncfusion.UI.Xaml.Controls.DataPager.OnDemandLoadingEventArgs args)
{
    dataPager.LoadDynamicItems(args.StartIndex,
    source.Skip(args.StartIndex).Take(args.PageSize));
    //resetting cache for all pages.
    (dataPager.PagedSource as PagedCollectionView).ResetCache();
}

```

You can also clear the cache to particular page by specifying the **PageIndex** in [PageCollectionView.ResetCacheForPage](#) method.

### C#

```

(dataPager.PagedSource as
PagedCollectionView).ResetCacheForPage(this.dataPager.PageIndex);

```

### Loading data from database in on-demand

You can read the data from database in on-demand (here, records are retrieved from **Northwind** data provider) in **OnDemandLoading** event handler.

### XML

```

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<syncfusion:SfDataGrid x:Name="dataGrid"
Grid.Row="0"
AutoGenerateColumns="False"
ItemsSource="{Binding Path=PagedSource,
ElementName=dataPager}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="OrderID" TextAlignment="Left" />
<syncfusion:GridTextColumn MappingName="CustomerID" />
<syncfusion:GridTextColumn MappingName="EmployeeID" TextAlignment="Right" />
<syncfusion:GridTextColumn HeaderText="Ship Name" MappingName="ShipName" />
<syncfusion:GridTextColumn HeaderText="Ship Address"
MappingName="ShipAddress" />
<syncfusion:GridTextColumn HeaderText="Ship City" MappingName="ShipCity" />
<syncfusion:GridTextColumn HeaderText="Ship Country"
MappingName="ShipCountry" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
<syncfusion:SfDataPager x:Name="dataPager"
Grid.Row="1"
PageCount="5"
PageSize="10"
UseOnDemandPaging="True" />

```

```
</Grid>
```

**C#**

```
public partial class MainWindow : Window
{
    Northwind northWind;
    public MainWindow()
    {
        InitializeComponent();
        dataPager.OnDemandLoading += dataPager_OnDemandLoading;
        string connectionString = string.Format(@"Data Source = {0}",
            ("Northwind.sdf"));
        //northWind dataProvider connectivity.
        northWind = new Northwind(connectionString);
    }
    private void dataPager_OnDemandLoading(object sender,
        Syncfusion.UI.Xaml.Controls.DataPager.OnDemandLoadingEventArgs args)
    {
        dataPager.LoadDynamicItems(args.StartIndex,
            northWind.Orders.Skip(args.StartIndex).Take(args.PageSize).ToList());
    }
}
```

*Changing PageCount at run time while filtering*

You can change the [SfDataPager.PageCount](#) at runtime based on the records count in [OnDemandPaging](#).

Here, PageCount are modified by filtering the records in run time.

**XML**

```
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="250" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<StackPanel Grid.Column="1">
<TextBlock Width="250"
Margin="10"
FontSize="14"
Foreground="DarkBlue"
Text="Enter value to filter the ShipName column (Filter by contains)"
TextWrapping="Wrap" />
<TextBox Name="filterTextBox"
Width="150"
Margin="10" />
<Button Width="100"
Margin="10"
Click="FilterBtn_Click"
Content="Filter" />
</StackPanel>
<syncfusion:SfDataGrid x:Name="dataGrid">
```

```

Grid.Row="0"
AllowEditing="True"
AutoGenerateColumns="False"
ItemsSource="{Binding Path=PagedSource,
ElementName=dataPager}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn HeaderText="Ship Name" MappingName="ShipName" />
<syncfusion:GridTextColumn MappingName="OrderID" TextAlignment="Left" />
<syncfusion:GridTextColumn MappingName="CustomerID" />
<syncfusion:GridTextColumn MappingName="EmployeeID" TextAlignment="Right" />
<syncfusion:GridTextColumn HeaderText="Ship Address"
MappingName="ShipAddress" />
<syncfusion:GridTextColumn HeaderText="Ship City" MappingName="ShipCity" />
<syncfusion:GridTextColumn HeaderText="Ship Country"
MappingName="ShipCountry" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
<syncfusion:SfDataPager x:Name="dataPager"
Grid.Row="1"
PageCount="5"
PageSize="10"
UseOnDemandPaging="True" />
</Grid>

```

**C#**

```

public partial class MainWindow : Window
{
    List<Orders> source = new List<Orders>();
    Northwind northWind;
    public MainWindow()
    {
        InitializeComponent();
        dataPager.OnDemandLoading += dataPager_OnDemandLoading;
        string connectionString = string.Format(@"Data Source = {0}",
("Northwind.sdf"));
        //northwind dataprovider connectivity.
        northWind = new Northwind(connectionString);
        source = northWind.Orders.ToList();
    }
    private void dataPager_OnDemandLoading(object sender,
Syncfusion.UI.Xaml.Controls.DataPager.OnDemandLoadingEventArgs args)
    {
        dataPager.LoadDynamicItems(args.StartIndex,
source.Skip(args.StartIndex).Take(args.PageSize));
    }
    private List<Orders> ApplyFilter(Northwind NorthwindSource)
    {
        //records are filtered based on ShipName column
        return NorthwindSource.Orders.Where(item =>
item.ShipName.Contains(fitlerTextBox.Text)).ToList();
    }
    private void FilterBtn_Click(object sender, RoutedEventArgs e)
    {
        source = ApplyFilter(northWind);
        //page count resets based on filtered records.
    }
}

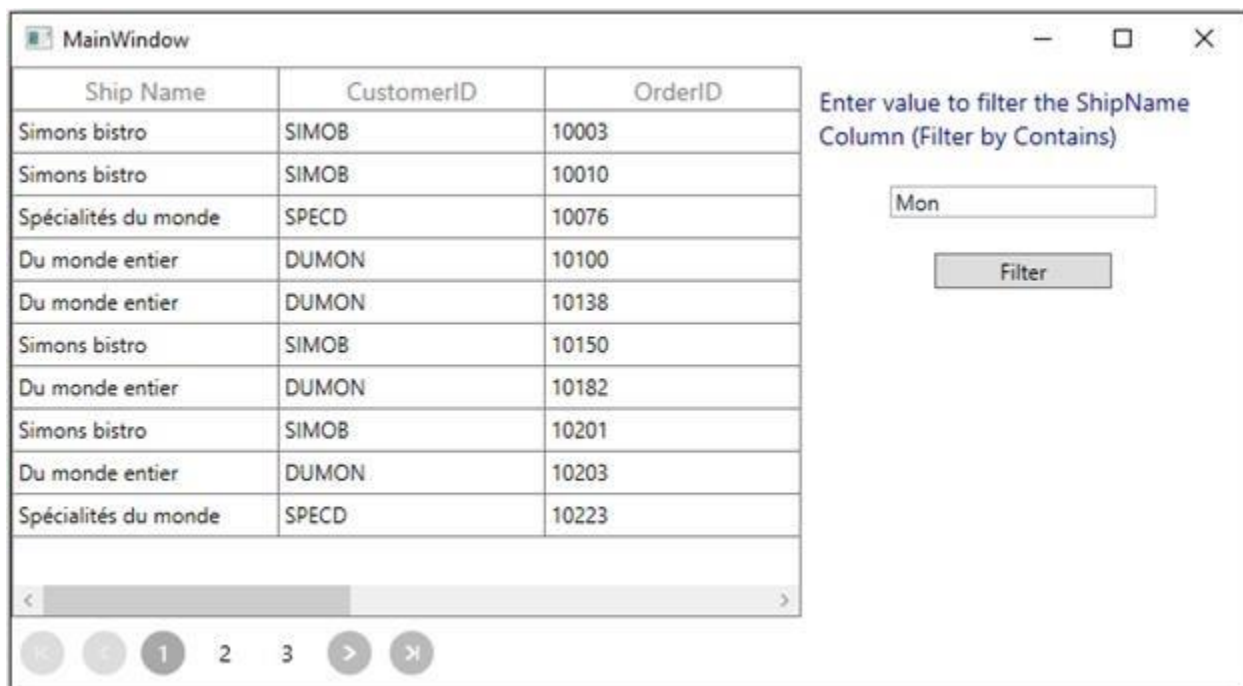
```

```

if (source.Count() < dataPager.PageSize)
this.dataPager.PageCount = 1;
else
{
var count = source.Count() / dataPager.PageSize;
if (source.Count() % dataPager.PageSize == 0)
this.dataPager.PageCount = count;
else
this.dataPager.PageCount = count + 1;
}
this.dataPager.MoveToPage(0);
}

```

Here, records are filtered based on the textbox text in clicking event of Filter button. Initially **PageCount** is 5 and it is changed as 3 once the records are filtered.



You can refer the [sample](#) from here.

#### Sorting complete collection

You can sort the complete collection with 'OnDemandPaging' by using [SfDataGrid.SortColumnChanging](#) event.

In this event, you can sort the complete underlying collection instead of sorting current page alone by resetting the caches.

#### XML

```

<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />

```



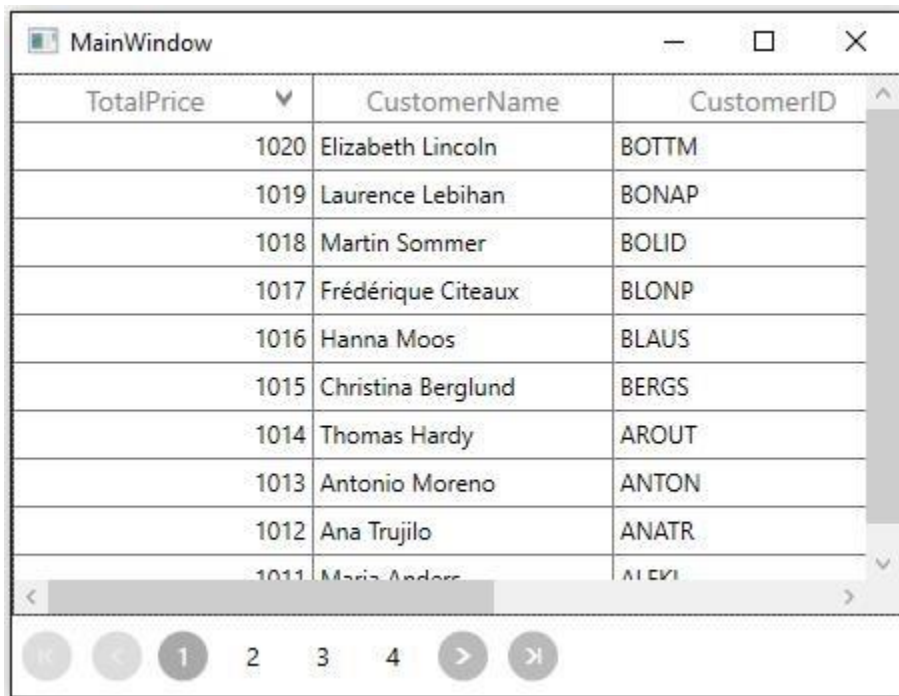
```
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<syncfusion:SfDataGrid x:Name="dataGrid"
Grid.Row="0"
ItemsSource="{Binding ElementName=dataPager,Path=PagedSource}"/>
<syncfusion:SfDataPager x:Name="dataPager"
PageCount="4"
PageSize="10"
UseOnDemandPaging="True"
Grid.Row="1"/>
</Grid>
```

## C#

```
using Syncfusion.Data.Extensions;
List<OrderInfo> source;
public MainWindow()
{
    InitializeComponent();
    source = new List<OrderInfo>();
    dataPager.OnDemandLoading += dataPager_OnDemandLoading;
    source = (this.DataContext as ViewModel).Orders;
    this.dataGrid.SortColumnsChanging += DataGrid_SortColumnsChanging;
}
private void DataGrid_SortColumnsChanging(object sender,
GridSortColumnsChangingEventArgs e)
{
    (dataPager.PagedSource as PagedCollectionView).ResetCache();
    (dataPager.PagedSource as
PagedCollectionView).ResetCacheForPage(dataPager.PageIndex);
    if (e.Action == NotifyCollectionChangedAction.Add || e.Action ==
NotifyCollectionChangedAction.Replace)
    {
        var sortDesc = e.AddedItems[0];
        if (sortDesc.SortDirection == ListSortDirection.Ascending)
        {
            //records are sorted in ascending order.
            source =
            source.OfQueryable().AsQueryable().OrderBy(sortDesc.ColumnName).Cast<OrderIn
fo>().ToList();
        }
        else
        {
            //records are sorted descending order.
            source =
            source.OfQueryable()
            .AsQueryable()
            .OrderByDescending(sortDesc.ColumnName)
            .Cast<OrderInfo>()
            .ToList();
        }
        this.dataPager.MoveToPage(dataPager.PageIndex);
    }
    private void dataPager_OnDemandLoading(object sender,
Syncfusion.UI.Xaml.Controls.DataPager.OnDemandLoadingEventArgs args)
{

```

```
dataPager.LoadDynamicItems (args.StartIndex,
source.Skip (args.StartIndex) .Take (args.PageSize) );
}
```



#### Loading ItemsSource for page using async and await

When you fetch the data from external server, it takes some time to load the data. In this case, you can delay the loading in `SfDataPager.OnDemandLoading` event using `async` and `await`.

Here `dataPagerOnDemandLoading` event is defined with `async` keyword to load the data by time delay. `GetEmployeesDetailsListAsync` method is invoked in `dataPagerOnDemandLoading` with `await` keyword which holds the execution until returning the data.

#### C#

```
private EmployeeInfoRepository repository;
public MainWindow()
{
    InitializeComponent();
    repository = new EmployeeInfoRepository();
}
//async method which return data with some delay
public async Task<List<Employees>> GetEmployeesDetailsListAsync(int
startIndex, int pageSize)
{
    var employees = new List<Employees>();
    //wait the method Execution to 2000 milliseconds
    System.Threading.Thread.Sleep(2000);
    for (int i = startIndex; i < (startIndex + pageSize); i++)
    {
        //Get the Data's to SfDataPager from ViewModel class
        employees.Add(repository.GetEmployees(i, pageSize));
    }
}
```

```

return employees;
}
//Delegate handler marked as async to use await inside
private async void dataPager_OnDemandLoading(object sender,
OnDemandLoadingEventArgs args)
{
var source = await GetEmployeesDetailsListAsync(args.StartIndex,
args.PageSize);
//Data's loaded to SfDataPager dynamically
dataPager.LoadDynamicItems(args.StartIndex, source.Take(args.PageSize));
//Resets the previously loaded page data's. It's optional
(dataPager.PagedSource as PagedCollectionView).ResetCache();
}

```

`GetEmployees` method in `EmployeeInfoRepository` returns the record to `SfDataPager`.

### C#

```

public class EmployeeInfoRepository
{
public EmployeeInfoRepository()
{
}
public List<Employees> GetEmployees(int startIndex, int pageSize)
{
int j = 0;
var employees = new List<Employees>();
for (int i = startIndex; i < (startIndex + pageSize); i++)
{
Employees employee = GetEmployee(employees);
employees.Add(employee);
}
return employees;
}
}

```

### Limitations

1. UI Filtering is not supported. You can code in application level to filter the data.
2. Data processing operations (Sorting, Grouping) are done only in the current page.
3. Deleting is not supported. You can code to delete row in application level.
4. Only the navigated pages are exported when 'OnDemandPaging' is enabled, if the navigated page cache is cleared then the corresponding page will not be exported.

### Data Virtualization in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) (SfDataGrid) provides support to handle the large amount of data through built-in virtualization features. With Data virtualization, [SfDataGrid.View](#) process the data in on-demand for better performance while loading large amount of data. Below are the different virtualization concepts available,

Concept	Usage
---------	-------

Data Virtualization	Use to load large amount of data in less time.
Incremental Loading	Use to load subset of data from the services or servers in less time while loading and scrolling. On-demand request also supported.
Paging	Use to load large amount of data in less time with the help of SfDataPager.
On-demand paging	Use to load data in on-demand. You can load data only for current page from server.

### Data Virtualization

You can load large amount of data in less time by setting [SfDataGrid.EnableDataVirtualization](#) property to true.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding EmployeeDetails}"
    EnableDataVirtualization="True">
```

#### C#

```
this.datagrid.EnableDataVirtualization = true;
```

### Working with GridVirtualizingCollectionView

You can load the large amount of data in less time in another way using [GridVirtualizingCollectionView](#) which is derived from [VirtualizingCollectionView](#) to [SfDataGrid.ItemsSource](#).

In the below code, **ViewModel** defined with **GridVirtualizingCollectionView** by passing complete records collection and bound to SfDataGrid.

#### C#

```
public class ViewModel
{
    private GridVirtualizingCollectionView _gridVirtualizingItemsSource;
    public GridVirtualizingCollectionView GridVirtualizingItemsSource
    {
        get { return _gridVirtualizingItemsSource; }
        set { _gridVirtualizingItemsSource = value; }
    }
    public ViewModel()
    {
        var _orders = this.GenerateOrders();
        GridVirtualizingItemsSource = new GridVirtualizingCollectionView(_orders);
    }
}
```

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    ColumnSizer="Star"
```

```
ItemsSource="{Binding GridVirtualizingItemsSource}" />
```

### Limitations

1. Data update using [LiveDataUpdateMode](#) is not supported.
2. Details view is not supported.
3. [AllowFrozenGroupHeaders](#) is not supported.
4. DataTable collection is not supported.

### Incremental Loading

DataGrid supports to load the data incrementally using [ISupportIncrementalLoading](#) interface.

[ISupportIncrementalLoading](#) interface has [LoadMoreItemsAsync](#) method which helps to load the data incrementally. [LoadMoreItemsAsync](#) called in on-demand while scrolling based on [HasMoreItems](#) property.

If [HasMoreItems](#) is false, SfDataGrid stops calling [LoadMoreItemsAsync](#). SfDataGrid have [IncrementalList](#) which is derived from [ISupportIncrementalLoading](#). You can use [IncrementalList](#) or create collection derived from [ISupportIncrementalLoading](#) and bind it [SfDataGrid.ItemsSource](#).

In the below code, [IncrementalList](#) is initialized by passing Action to its constructor for loading items incrementally.

### XML

```
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<syncfusion:SfDataGrid x:Name="dataGrid"
AllowFiltering="True"
AutoGenerateColumns="True"
ItemsSource="{Binding IncrementalItemsSource}" />
```

### C#

```
Public class ViewModel
{
    public ViewModel()
    {
        IncrementalItemsSource = new IncrementalList<OrderInfo>(LoadMoreItems) {
            MaxItemCount = 1000 };
    }
    private IncrementalList<OrderInfo> _incrementalItemsSource;
    public IncrementalList<OrderInfo> IncrementalItemsSource
    {
        get { return _incrementalItemsSource; }
        set { _incrementalItemsSource = value; }
    }
    /// <summary>
    /// Method to load items which assigned to the action of IncrementalList
    /// </summary>
    /// <param name="count"></param>
    /// <param name="baseIndex"></param>
```

```
void LoadMoreItems(uint count, int baseIndex)
{
    var _orders = GenerateOrders();
    var list = _orders.Skip(baseIndex).Take(50).ToList();
    IncrementalItemsSource.LoadItems(list);
}
}
```

You can download the sample from [here](#).

#### *Displaying animation when fetching data from services*

You can display animations when fetching data from service for [LoadMoreItemsAsync](#) method call, using [BackgroundWorker](#).

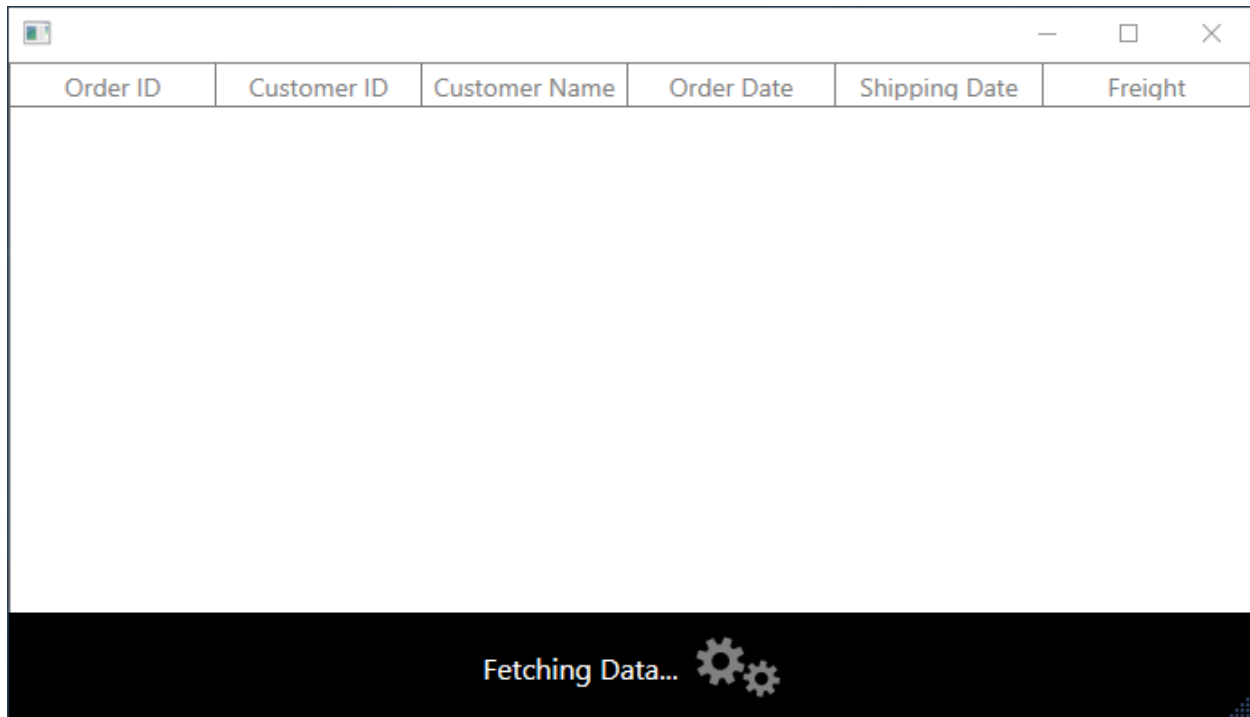
In the below code snippet data fetched from service using [BackgroundWorker](#) and [SfBusyIndicator](#) displayed over SfDataGrid based on [IsBusy](#) property in [ViewModel](#), until [BackgroundWorker](#) completes its action.

#### **XML**

```
<Window.DataContext>
<local:ViewModel />
</Window.DataContext>
<Window.Resources>
<local:BoolToVisibilityConverter x:Key="converter" />
</Window.Resources>
<Grid>
<syncfusion:SfDataGrid x:Name="dataGrid"
AllowFiltering="True"
AutoGenerateColumns="True"
ItemsSource="{Binding IncrementalItemsSource}"/>
<Border Height="60"
VerticalAlignment="Bottom"
Background="Black"
BorderBrush="Black"
BorderThickness="1"
Opacity="50"
Visibility="{Binding IsBusy,
Mode=TwoWay,
Converter={StaticResource converter}}">
<StackPanel HorizontalAlignment="Center"
VerticalAlignment="Center"
Orientation="Horizontal">
<TextBlock Margin="5"
VerticalAlignment="Center"
FontSize="16"
Foreground="White"
Text="Fetching Data..." />
<syncfusion:SfBusyIndicator Margin="5"
VerticalAlignment="Center"
Foreground="Gray"
AnimationType="Gear" />
</StackPanel>
</Border>
</Grid>
```

**C#**

```
public class ViewModel : INotifyPropertyChanged
{
    NorthwindEntities northwindEntity;
    public ViewModel()
    {
        string uri="http://services.odata.org/Northwind/Northwind.svc/";
        incrementalItemsSource = new IncrementalList<Order>(LoadMoreItems) {
            MaxItemCount = 1000;
            northwindEntity = new NorthwindEntities(new Uri(uri));
            IsBusy = false;
        }
        private bool isBusy;
        /// <summary>
        /// Gets or Sets whether to show the busy indicator.
        /// </summary>
        public bool IsBusy
        {
            get { return isBusy; }
            set { isBusy = value; RaisePropertyChanged("IsBusy"); }
        }
        /// <summary>
        /// Loads the item while SfDataGrid loading and scrolling.
        /// </summary>
        /// <param name="count">Specifies the fetch count to load data</param>
        /// <param name="baseIndex">Specifies the index to load data</param>
        void LoadMoreItems(uint count, int baseIndex)
        {
            BackgroundWorker worker = new BackgroundWorker();
            worker.DoWork += (o, ae) =>
            {
                DataServiceQuery<Order> query = northwindEntity.Orders.Expand("Customer");
                query = query.Skip<Order>(baseIndex).Take<Order>(50) as
                DataServiceQuery<Order>;
                IAsyncResult ar = query.BeginExecute(null, null);
                var items = query.EndExecute(ar);
                var list = items.ToList();
                Application.Current.Dispatcher.Invoke(new Action(() => {
                    IncrementalItemsSource.LoadItems(list); }));
            };
            worker.RunWorkerCompleted += (o, ae) =>
            {
                IsBusy = false;
            };
            IsBusy = true;
            worker.RunWorkerAsync();
        }
    }
}
```



You can download the sample from [here](#).

*LoadMore using ISupportIncrementalLoading*

You can fetch the data in some user action instead of scrolling using [IncrementalList.LoadItems](#) method.

In the below code, data fetched when you click the **Load Items** button.

#### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid x:Name="Root_Grid">
<Grid.RowDefinitions>
<RowDefinition Height="*/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="True"
ItemsSource="{Binding IncrementalItemsSource}"/>
<StackPanel Grid.Row="1" Orientation="Vertical">
<Button x:Name="loadByButton" Content="Load Items"
Command="{Binding DataContext.LoadItems , ElementName=dataGrid}"/>
</StackPanel>
</Grid>
</Window>
```

#### C#

```
public class ViewModel
{
    NorthwindEntities northwindEntity;
    private IncrementalList<Order> _incrementalItemsSource;
```



```

public IncrementalList<Order> IncrementalItemsSource
{
    get { return _incrementalItemsSource; }
    set { _incrementalItemsSource = value; }
}
private BaseCommand loadItems;
public BaseCommand LoadItems
{
    get
    {
        if (loadItems == null)
            loadItems = new BaseCommand(OnLoadItemsClicked, OnCanLoad);
        return loadItems;
    }
}
private static bool OnCanLoad(object obj)
{
    return true;
}
private void OnLoadItemsClicked(object obj)
{
    LoadMoreItems(5, this.IncrementalItemsSource.Count);
}
public ViewModel()
{
    string uri="http://services.odata.org/Northwind/Northwind.svc/";
    _incrementalItemsSource = new IncrementalList<Order>(LoadMoreItems) {
        MaxItemCount = 50 };
    northwindEntity = new NorthwindEntities(new Uri(uri));
}
/// <summary>
/// Method to load items which assigned to the action of IncrementalList
/// </summary>
/// <param name="count"></param>
/// <param name="baseIndex"></param>
void LoadMoreItems(uint count, int baseIndex)
{
    DataServiceQuery<Order> query = northwindEntity.Orders.Expand("Customer");
    query = query.Skip<Order>(baseIndex).Take<Order>(50) as
    DataServiceQuery<Order>;
    IAsyncResult ar = query.BeginExecute(null, null);
    var items = query.EndExecute(ar);
    var list = items.ToList();
    IncrementalItemsSource.LoadItems(list);
}
}

```

You can download the sample from [here](#).

#### Limitations

1. Deleting is not supported. You can code to delete row in application level.
2. Summary is not calculated based on [LiveDataUpdateMode](#).

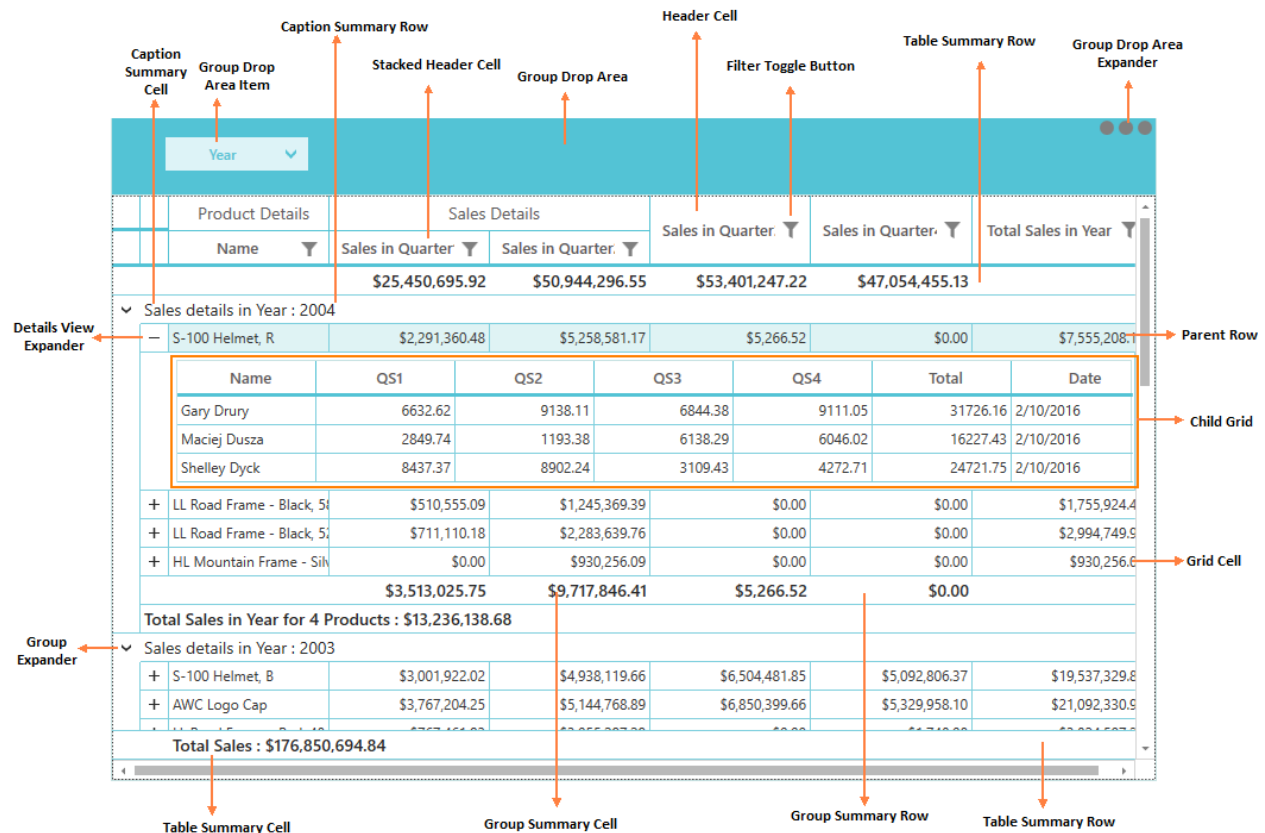
## Paging

SfDataGrid supports to load paged data source using [SfDataPager](#). You can use the paging in SfDataGrid by go through the [Paging](#) section.

## Styles and Templates in WPF DataGrid (SfDataGrid)

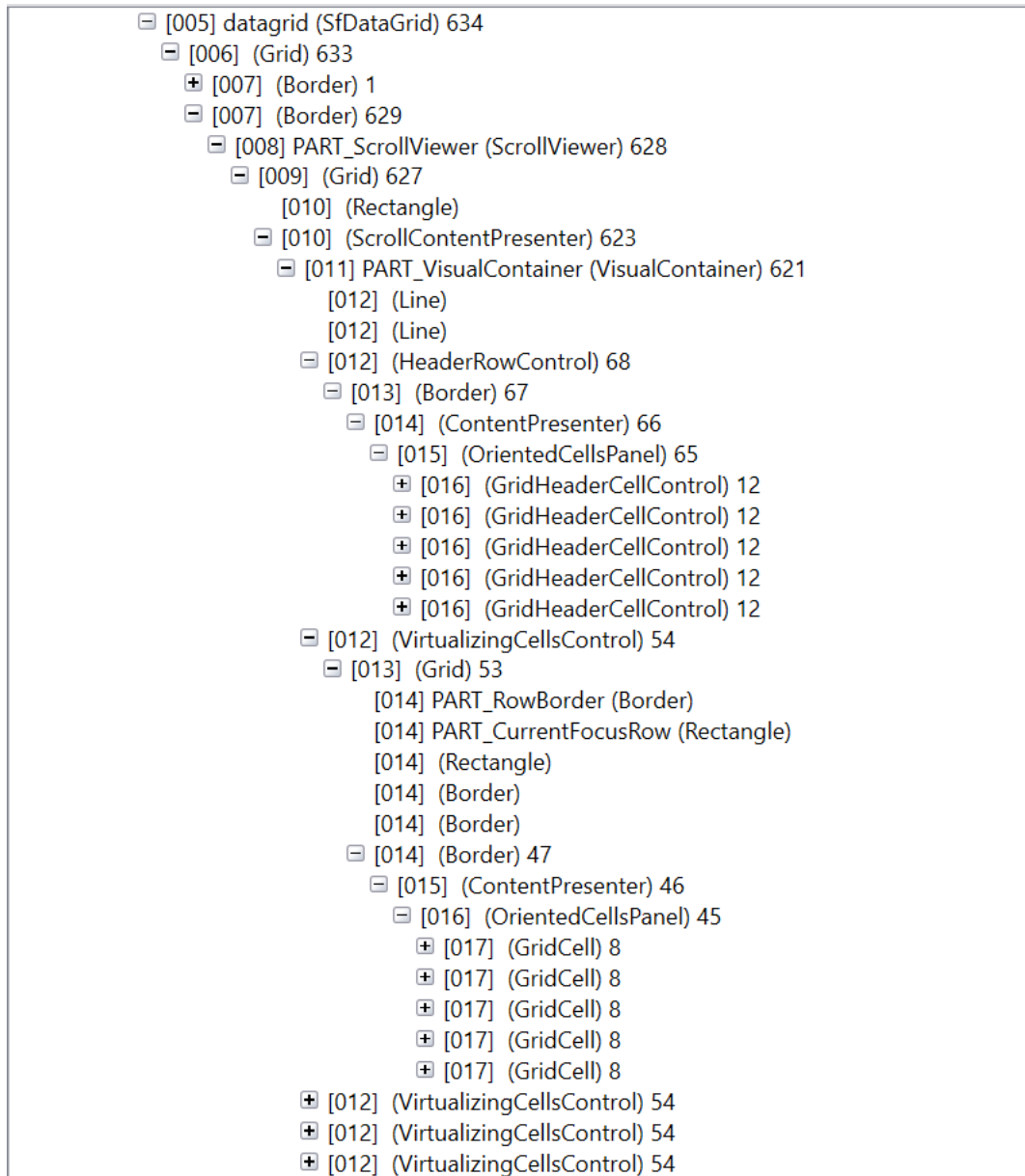
The appearance of [WPF DataGrid](#) (SfDataGrid) and its inner elements (example: Cell, Row, Header, Summary etc.) can be customized using various properties exposed and by editing the elements' Style.

## Control Structure of SfDataGrid



## Customizing Default Containers

WPF DataGrid (SfDataGrid) arranges the cell and row content using cell and row containers. Below screenshot shows the [VisualTree](#) of SfDataGrid where [HeaderCell](#) is loaded into the [HeaderCellControl](#) and data cells are loaded into the [VirtualizingCellsControl](#) container. [VirtualizingCellsControl](#) container uses [GridCell](#) to load the cell content.



[RowGenerator](#) class processes the creation and re-using of containers for SfDataGrid. You create your own containers by overriding [RowGenerator](#) class and setting it to [SfDataGrid.RowGenerator](#). Using this method to customize the row and cell containers allows for customizations that aren't possible through styling and conditional styling.

#### Row containers

Below table shows the different types of grid rows and its container.

RowType	Container
DataRow	<a href="#">VirtualizingCellsControl</a>
UnboundRow	<a href="#">UnBoundRowControl</a>
FilterRow	<a href="#">FilterRowControl</a>

DetailViewDataRow	<a href="#">DetailViewRowControl</a>
TableSummaryRow	<a href="#">TableSummaryRowControl</a>
HeaderRow	<a href="#">HeaderRowControl</a>
AddNewRow	<a href="#">AddNewRowControl</a>
CaptionSummaryRow	<a href="#">CaptionSummaryRowControl</a>
GroupSummaryRow	<a href="#">GroupSummaryRowControl</a>
StackedHeaderRow	<a href="#">GridStackedHeaderCellControl</a>

#### *Animating the data row when property changes*

You can customize the [DataRow](#) operations by overriding the `DataRow` class. You have to override the [GetDataRow](#) method in [RowGenerator](#) to load the customized `DataRow`.

Similarly, you can able to customize:

1. [GridUnboundRow](#)
2. [FilterRow](#)
3. [SpannedDataRow](#)

The below code example shows how to animate the `DataRow` when the row data is changed.

#### **C#**

```
this.dataGrid.RowGenerator = new CustomRowGenerator(this.dataGrid);
public class CustomDataRow : DataRow
{
    public CustomDataRow()
    : base()
    {
    }
    protected Storyboard sb = null;
    protected override void OnRowDataChanged()
    {
        base.OnRowDataChanged();
        if (this.WholeRowElement != null)
        {
            DoubleAnimation animation = new DoubleAnimation
            {
                From = 0,
                To = 1,
                Duration = new Duration(TimeSpan.FromMilliseconds(2000)),
                AutoReverse = true,
                FillBehavior = FillBehavior.Stop
            };
            Storyboard.SetTarget(animation, this.WholeRowElement);
            Storyboard.SetTargetProperty(animation, new
            PropertyPath(Path.OpacityProperty));
            sb = new Storyboard();
            sb.Children.Add(animation);
```

```

sb.Begin();
}
}
}
public class CustomRowGenerator : RowGenerator
{
    public CustomRowGenerator(SfDataGrid dataGrid)
    : base(dataGrid)
    { }
    protected override GridDataRow GetDataRow<T>(RowType type)
    {
        //Set the customized DataRow.
        if (typeof(T) == typeof(DataRow))
        return new CustomDataRow();
        return base.GetDataRow<T>(type);
    }
}

```

You can download a working demo for the above customization from [here](#).

OrderID	CustomerID	CustomerName	Country
1009	BGFDE	Sirert	Bulk
1010	BOTTM	Rakesh	hertion
1011	TGVFD	George	Oregon
1004	BERGS	Charles	Spain
1004	Bulk	John	Chicago
1005	BERGS	Charles	Spain
1006	TGVFD	Dintin	Britain
1007	YTREW	Friedo	James
1008	MNBGY	John	Oregon
1001	ALFKI	Bulk	Beverton
1002	ANATR	Oliver	Oregon
1003	ANTON	Brendon	Johanesberg
1004	YHGTR	John	Chicago
1005	BERGS	Charles	Spain
1006	TGVFD	Dintin	Britain
1007	YTREW	Friedo	Britain
1008	MNBGY	John	Oregon
1009	BGFDE	Sirert	Bulk

The below code example shows how to change the background color of the **VirtualizingCellsControl** when the value has been changed for a particular cell. This can be done by hooking the **DataContextChanged** and **PropertyChanged** event.

**C#**

```

this.dataGrid.RowGenerator = new CustomRowGenerator(this.dataGrid);
public class CustomVirtualizingCellsControl : VirtualizingCellsControl
{
    public CustomVirtualizingCellsControl()
    : base()
    {
        this.DataContextChanged +=
            CustomVirtualizingCellsControl_DataContextChanged;
    }
    private void CustomVirtualizingCellsControl_DataContextChanged(object
sender, DependencyPropertyChangedEventArgs e)
    {
        var newValue = e.NewValue as INotifyPropertyChanged;
        newValue.PropertyChanged += NewValue_PropertyChanged;
    }
    private void NewValue_PropertyChanged(object sender,
PropertyEventArgs e)
    {
        if (e.PropertyName == "CustomerID")
        this.Background = new SolidColorBrush(Colors.Pink);
    }
}
public class CustomRowGenerator : RowGenerator
{
    public CustomRowGenerator(SfDataGrid dataGrid)
    : base(dataGrid)
    { }
    protected override VirtualizingCellsControl GetVirtualizingCellsControl<T>()
    {
        //Set the customized VirtualizingCellsControl
        if (typeof(T) == typeof(VirtualizingCellsControl))
        return new CustomVirtualizingCellsControl();
        return base.GetVirtualizingCellsControl<T>();
    }
}

```

You can download a working demo for the above customization from [here](#).

*Cell containers*

Below table shows the different types of cells and its container.

CellType	Container
GridCell	<a href="#">OrientedCellsPanel</a>
GridUnBoundRowCell	{{' <a href="#">OrientedCellsPanel</a> '   markdownify }}
GridFilterRowCell	[OrientedCellsPanel]( <a href="https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.OrientedCellsPanel.html">https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.OrientedCellsPanel.html</a> )

*Animating the data cell when property changes*

You can customize the [GridCell](#) behavior by overriding the `GridCell` class. You have to override the [GetGridCell](#) method in [RowGenerator](#) to load the customized `GridCell`.

Similarly, you can able to customize:

1. [GridUnBoundRowCell](#)
2. [GridFilterRowCell](#)

The below code example shows how to animate the cell based on the changes occur in another cell using the `DataContextChanged` and `PropertyChanged` events.

**C#**

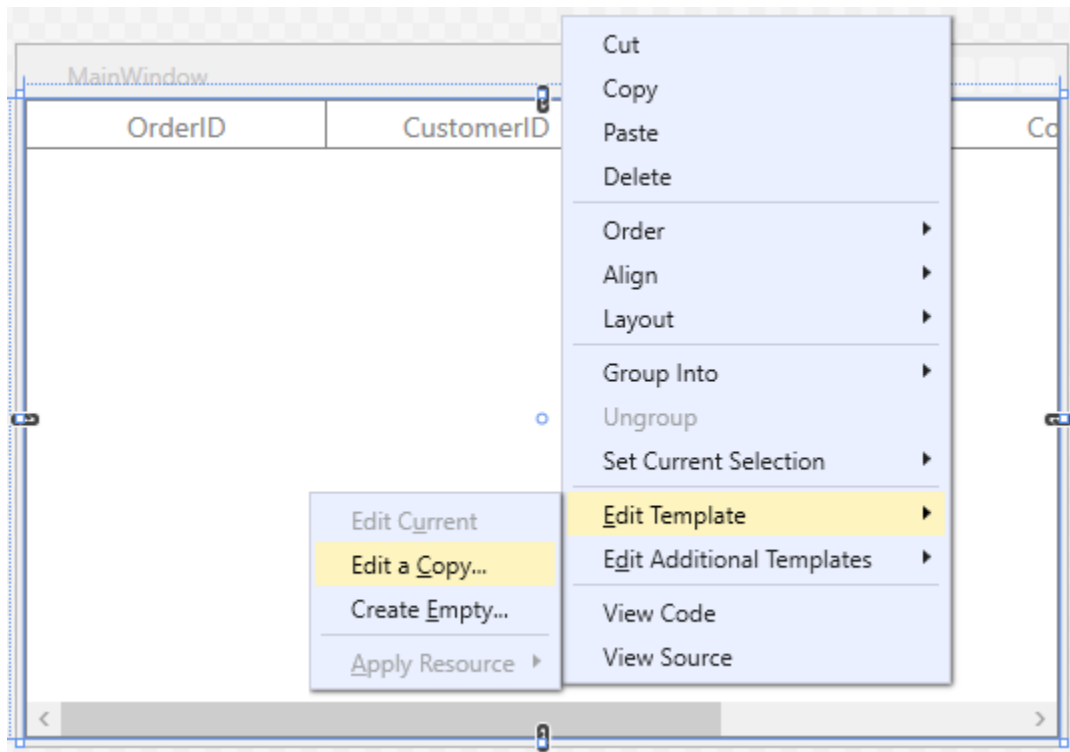
```
this.dataGrid.RowGenerator = new CustomRowGenerator(this.dataGrid);
public class CustomGridCell : GridCell
{
    public CustomGridCell() : base()
    {
        this.DataContextChanged += CustomGridCell_DataContextChanged;
    }
    private void CustomGridCell_DataContextChanged(object sender,
        DependencyPropertyChangedEventArgs e)
    {
        var newData = e.NewValue as INotifyPropertyChanged;
        newData.PropertyChanged += Data_PropertyChanged;
    }
    protected Storyboard sb = null;
    private void Data_PropertyChanged(object sender, PropertyChangedEventArgs e)
    {
        if (e.PropertyName == "CustomerID")
        {
            DoubleAnimation animation = new DoubleAnimation
            {
                From = 0,
                To = 1,
                Duration = new Duration(TimeSpan.FromMilliseconds(2000)),
                AutoReverse = false,
                FillBehavior = FillBehavior.HoldEnd
            };
            Storyboard.SetTarget(animation, this);
            Storyboard.SetTargetProperty(animation, new
                PropertyPath(Path.OpaCityProperty));
            sb = new Storyboard();
            sb.Children.Add(animation);
            sb.Begin();
        }
    }
    protected override void Dispose(bool isDisposing)
    {
        this.DataContextChanged -= CustomGridCell_DataContextChanged;
        base.Dispose(isDisposing);
    }
}
public class CustomRowGenerator : RowGenerator
```

```
{  
public CustomRowGenerator(SfDataGrid dataGrid)  
: base(dataGrid)  
{  
}  
protected override GridCell GetGridCell<T>()  
{  
return new CustomGridCell();  
}  
}
```

You can download a working demo for the above customization from [here](#).

#### Editing Style in Visual Studio Designer

You can edit the SfDataGrid style in Visual Studio Designer by right clicking it in design View and click **Edit Template**.

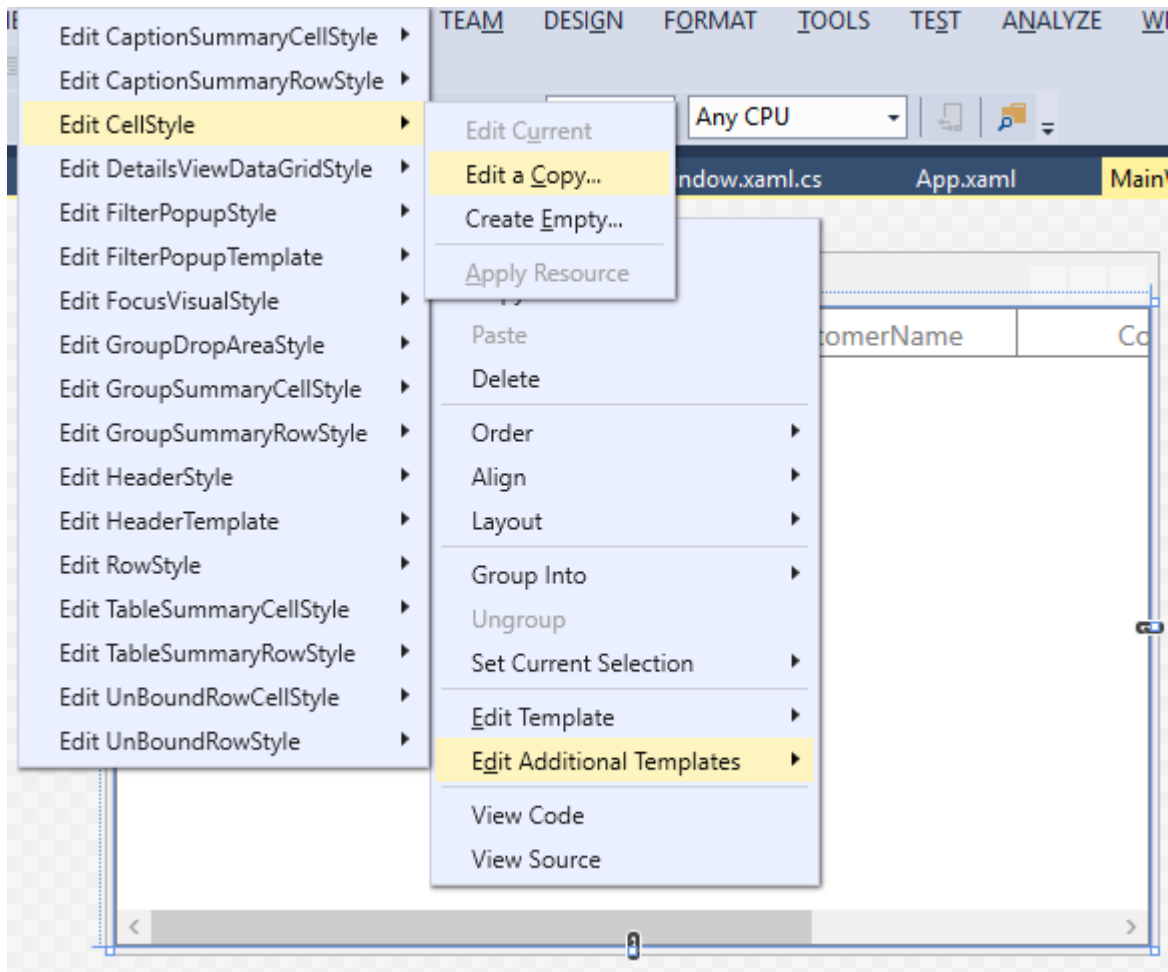


By clicking **Edit a Copy**, it will generate default template of SfDataGrid in **XAML view** and you can edit the default style.

#### Editing DataGrid Elements Style in Visual Studio Designer

You can edit the SfDataGrid elements style in Visual Studio Designer by right clicking it in designer view and click **Edit Additional Templates**.





You can edit or create new style for the following SfDataGrid elements through **Edit Additional Templates** option,

1. HeaderStyle
2. HeaderTemplate
3. CellStyle
4. RowStyle
5. GroupDropAreaStyle
6. CaptionSummaryCellStyle
7. CaptionSummaryRowStyle
8. GroupSummaryCellStyle
9. GroupSummaryRowStyle
10. TableSummaryCellStyle
11. TableSummaryRowStyle
12. UnBoundRowCellStyle
13. UnBoundRowStyle
14. FilterPopupStyle
15. FilterPopupTemplate
16. DetailsViewDataGridStyle

---

**Note:** Visual Studio Editing option is available from Visual Studio 2012 and higher versions only.

---

### Writing Style by TargetType

The appearance of WPF DataGrid (SfDataGrid) and its inner elements can be customized by writing style of TargetType to those control. If the key is not specified, then the style will be applied to all the SfDataGrid in its scope. You can apply specific to SfDataGrid or column or cell using various properties exposed.

### Styling Record cell

The record cells can be customized by writing style of TargetType [GridCell](#). You can set to particular SfDataGrid by setting [SfDataGrid.CellStyle](#) property and the particular column can be styled by setting [GridColumn.CellStyle](#) property. Underlying record will be the DataContext for [GridCell](#).

### XML

```
<Window.Resources>
<Style TargetType="syncfusion:GridCell" x:Key="customCellStyle">
<Setter Property="Background" Value="Bisque" />
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
CellStyle="{StaticResource customCellStyle}"
ItemsSource="{Binding Orders}"/>
```

You can also set the [CellStyle](#) to particular column in below way.

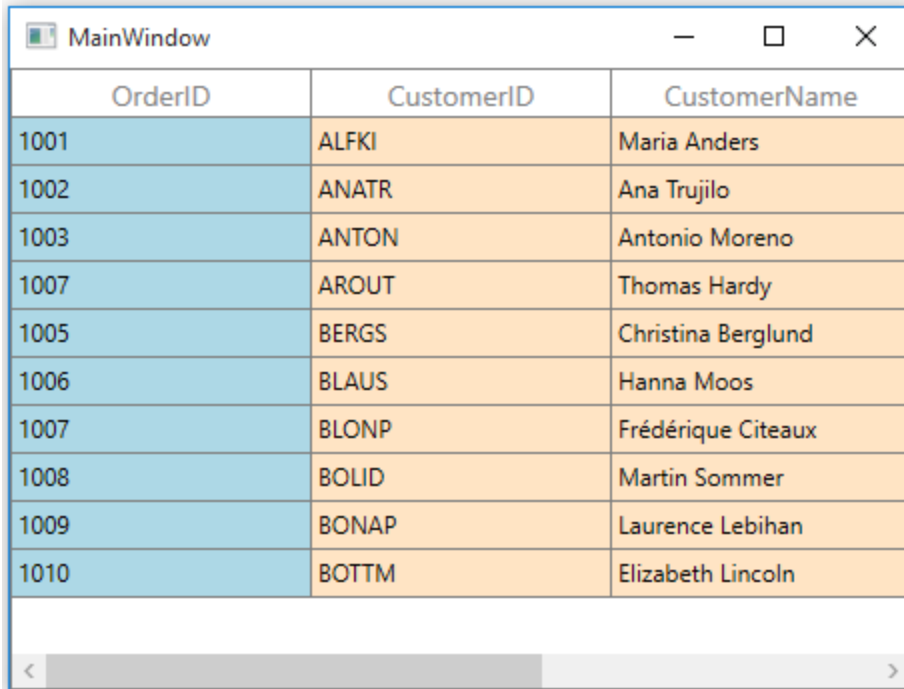
### XML

```
<syncfusion:GridTextColumn MappingName="OrderID">
<syncfusion:GridTextColumn.CellStyle>
<Style TargetType="syncfusion:GridCell">
<Setter Property="Background" Value="LightBlue" />
</Style>
</syncfusion:GridTextColumn.CellStyle>
</syncfusion:GridTextColumn>
```

---

**Note:** [GridColumn.CellStyle](#) takes higher priority than [SfDataGrid.CellStyle](#) property.

---



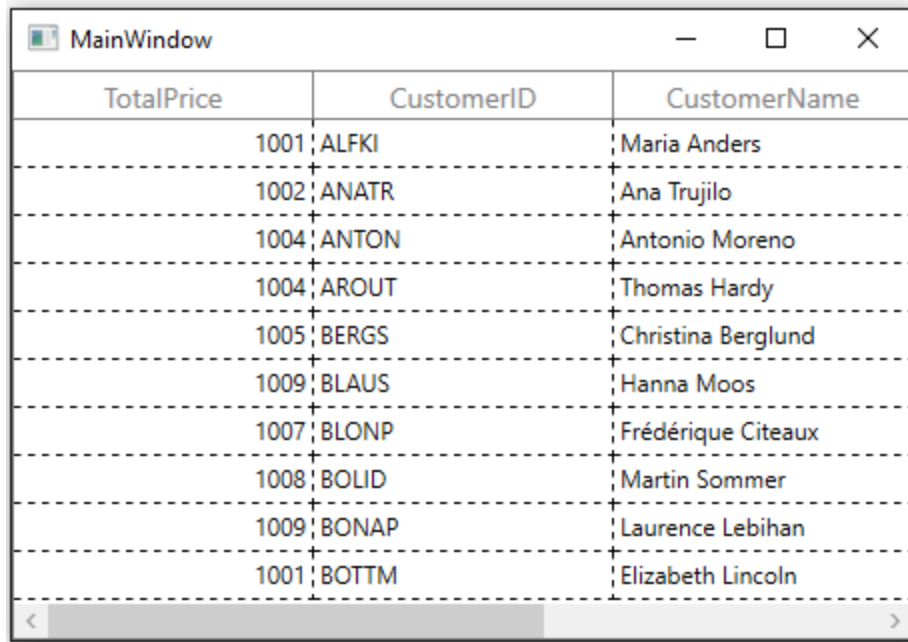
OrderID	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1007	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

*Changing Grid line border as dotted line*

You can change the gridline border as dotted line by customizing [GridCell.BorderBrush](#) property.

#### XML

```
<Window.Resources>
<Style TargetType="syncfusion:GridCell">
<Setter Property="BorderBrush">
<Setter.Value>
<DrawingBrush Viewport="0,0,7,7" ViewportUnits="Absolute" TileMode="Tile">
<DrawingBrush.Drawing>
<DrawingGroup>
<GeometryDrawing Brush="Black">
<GeometryDrawing.Geometry>
<GeometryGroup>
<RectangleGeometry Rect="0,0,50,50" />
<RectangleGeometry Rect="50,50,50,50" />
</GeometryGroup>
</GeometryDrawing.Geometry>
</GeometryDrawing>
</DrawingGroup>
</DrawingBrush.Drawing>
</DrawingBrush>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}">
```



TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1009	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1001	BOTTM	Elizabeth Lincoln

#### Changing Grid line color

You can also change the gridline color by setting [GridCell.BorderBrush](#) property.

#### XML

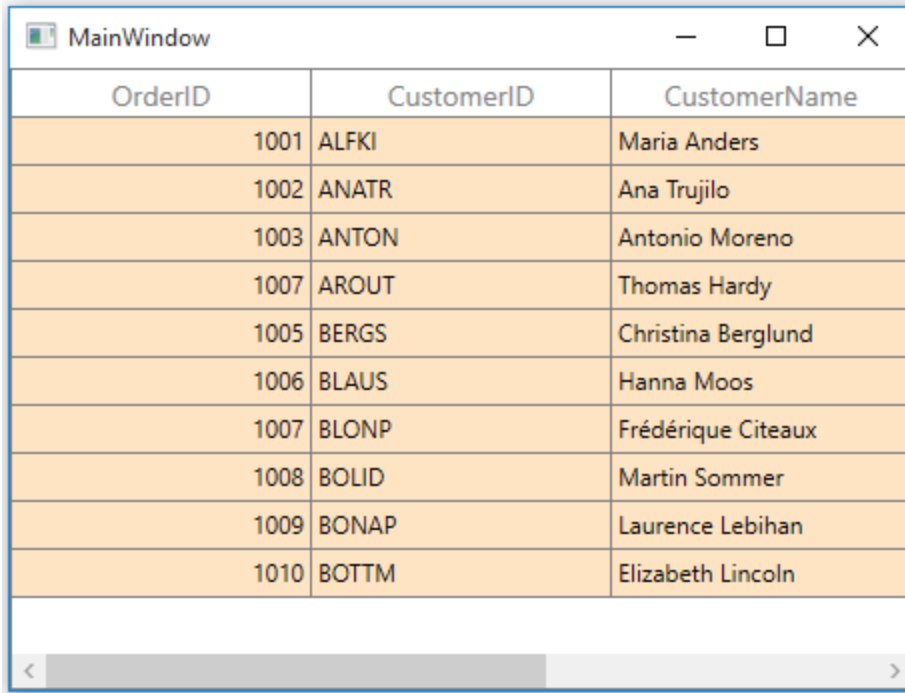
```
<Window.Resources>
<Style TargetType="syncfusion:GridCell">
<Setter Property="BorderBrush" Value="Green" />
</Style>
</Window.Resources>
```

#### Styling Record row

The record rows can be customized by writing style of TargetType [VirtualizingCellsControl](#). You can set to particular SfDataGrid by setting [SfDataGrid.RowStyle](#) property.

#### XML

```
<Window.Resources>
<Style TargetType="syncfusion:VirtualizingCellsControl"
x:Key="customRowStyle">
<Setter Property="Background" Value="Bisque"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
RowStyle="{StaticResource customRowStyle}"/>
```



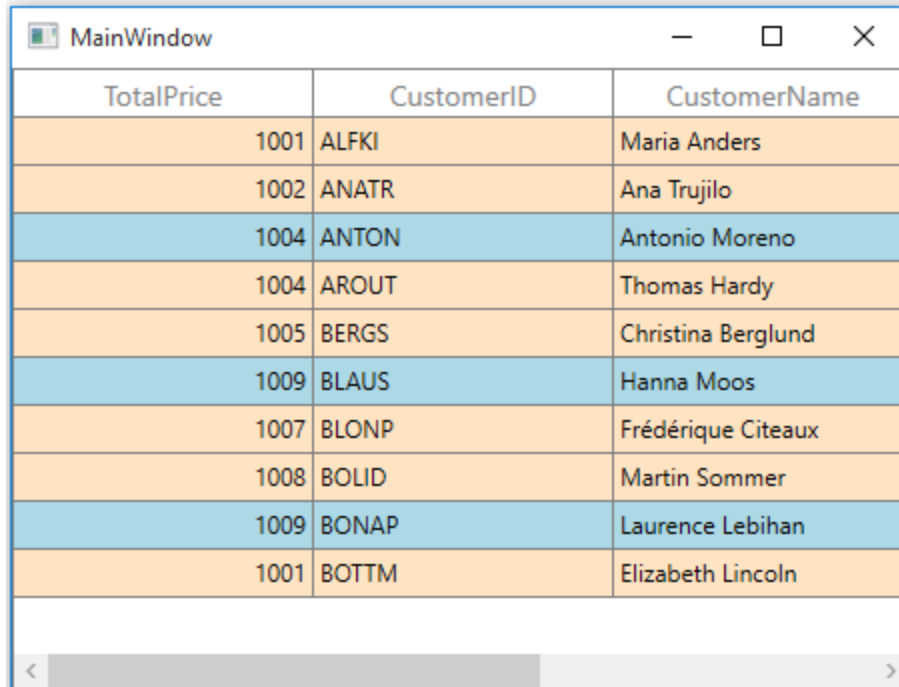
OrderID	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1007	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

### Alternating Row Style

You can style the alternate rows by setting [SfDataGrid.AlternatingRowStyle](#) and [SfDataGrid.RowStyle](#) property. [AlternateRowStyle](#) will be applied based on [SfDataGrid.AlternationCount](#) property.

### XML

```
<Window.Resources>
<Style TargetType="syncfusion:VirtualizingCellsControl"
x:Key="alternatingRowStyle">
<Setter Property="Background" Value="LightBlue"/>
</Style>
<Style TargetType="syncfusion:VirtualizingCellsControl" x:Key="RowStyle">
<Setter Property="Background" Value="Bisque"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AlternatingRowStyle="{StaticResource alternatingRowStyle}"
AlternationCount="3"
RowStyle="{StaticResource RowStyle}"
ItemsSource="{Binding Orders}"/>
```



TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1009	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1001	BOTTM	Elizabeth Lincoln

### Selection

The foreground and background for the selected row, cell can be customized by setting [SfDataGrid.RowSelectionBrush](#) and [SfDataGrid.SelectionForegroundBrush](#) property.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    RowSelectionBrush="DarkBlue"
    SelectionForegroundBrush="Bisque"
    ItemsSource="{Binding Orders}">
```

TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1009	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1001	BOTTM	Elizabeth Lincoln

Styling Column Header

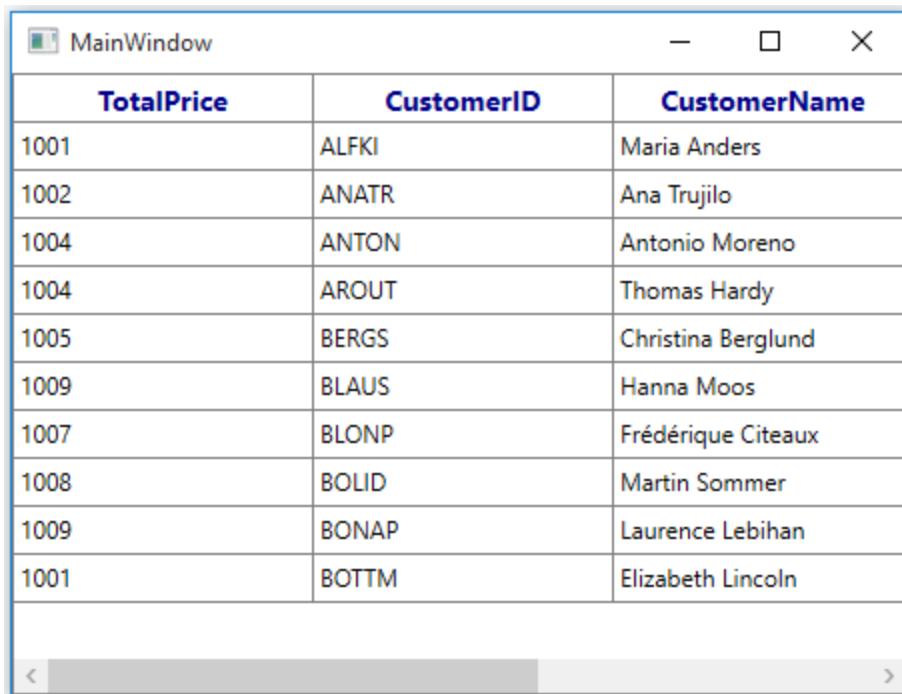
*Styling Header cell*

The header cell can be customized by writing style of TargetType [GridHeaderCellControl](#). You can set to particular SfDataGrid by setting [SfDataGrid.HeaderStyle](#) property and the particular column can be styled by setting [GridColumn.HeaderStyle](#) property.

**Note:** [GridColumn.HeaderStyle](#) takes higher priority than [SfDataGrid.HeaderStyle](#) property.

#### XML

```
<Window.Resources>
<Style TargetType="syncfusion:GridHeaderCellControl" x:Key="headerStyle">
<Setter Property="FontWeight" Value="Bold"/>
<Setter Property="FontSize" Value="14"/>
<Setter Property="Foreground" Value="DarkBlue"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
HeaderStyle="{StaticResource headerStyle}"
ItemsSource="{Binding Orders}"/>
```



TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1009	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1001	BOTTM	Elizabeth Lincoln

#### Styling DetailsViewDataGrid header

The header style can be applied to [DetailsViewDataGrid](#) alone by setting [HeaderStyle](#) property to [DetailsViewDataGrid](#) in both XAML and code behind.

#### XML

```
<Window.Resources>
<Style TargetType="syncfusion:GridHeaderCellControl" x:Key="header">
<Setter Property="Foreground" Value="DarkBlue"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="Details">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstDetailsViewGrid"
HeaderStyle="{StaticResource header}">
</syncfusion:SfDataGrid>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>
```

If [SfDataGrid.AutoGenerateRelations](#) is [true](#), you can set the header style to [DetailsViewDataGrid](#) in [SfDataGrid.AutoGenerateRelations](#) event.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateRelations="True"
ItemsSource="{Binding Orders}">
```



**C#**

```

this.dataGrid.AutoGeneratingRelations += dataGrid_AutoGeneratingRelations;
void dataGrid_AutoGeneratingRelations
(object sender, Syncfusion.UI.Xaml.Grid.AutoGeneratingRelationsArgs e)
{
    e.GridViewDefinition.DataGrid.HeaderStyle = (Style)this.Resources["header"];
}

```

*Styling Stacked Headers*

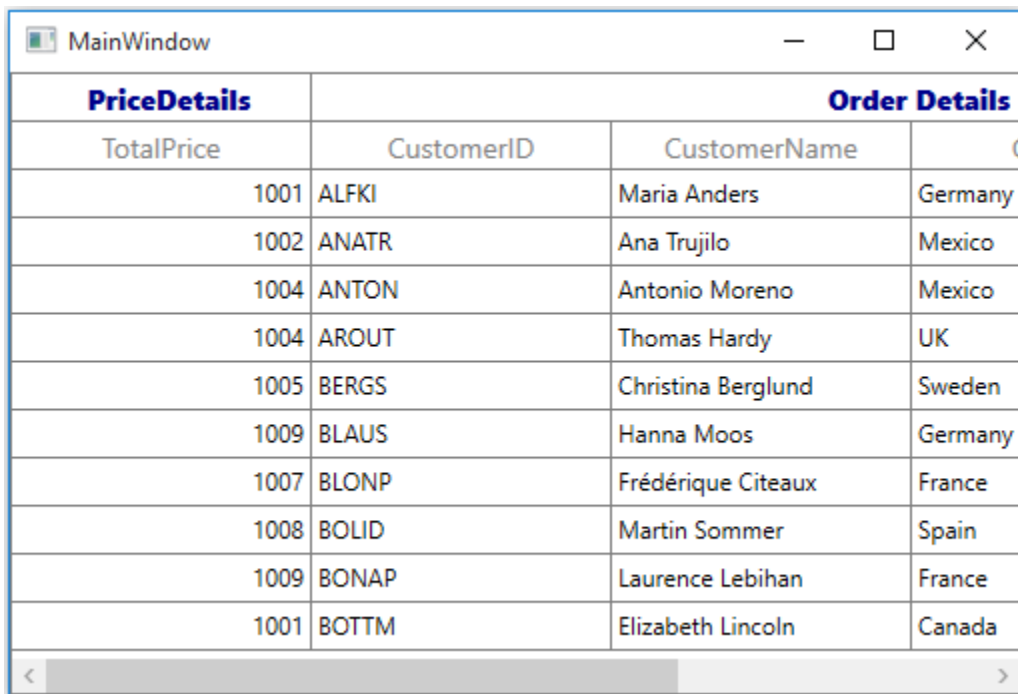
The appearance of stacked header can be customized by writing style of TargetType [GridStackedHeaderCellControl](#).

**XML**

```

<Window.Resources>
<Style TargetType="syncfusion:GridStackedHeaderCellControl">
<Setter Property="FontWeight" Value="ExtraBold"/>
<Setter Property="FontSize" Value="14"/>
<Setter Property="Foreground" Value="DarkBlue"/>
</Style>
</Window.Resources>

```



PriceDetails		Order Details	
TotalPrice	CustomerID	CustomerName	
1001	ALFKI	Maria Anders	Germany
1002	ANATR	Ana Trujilo	Mexico
1004	ANTON	Antonio Moreno	Mexico
1004	AROUT	Thomas Hardy	UK
1005	BERGS	Christina Berglund	Sweden
1009	BLAUS	Hanna Moos	Germany
1007	BLONP	Frédérique Citeaux	France
1008	BOLID	Martin Sommer	Spain
1009	BONAP	Laurence Lebihan	France
1001	BOTTM	Elizabeth Lincoln	Canada

*Setting different styles to StackedHeader*

You can apply the different style to stacked header by overriding the [default renderer](#) of StackedHeader.

**XML**

```

<Application.Resources>
<Style x:Key="style1" TargetType="syncfusion:GridStackedHeaderCellControl">
<Setter Property="Background" Value="LightBlue" />

```

```
<Setter Property="FontFamily" Value="Segoe UI" />
<Setter Property="FontStyle" Value="Italic" />
<Setter Property="FontWeight" Value="Bold"/>
</Style>
<Style x:Key="style2" TargetType="syncfusion:GridStackedHeaderCellControl">
<Setter Property="Background" Value="Bisque" />
<Setter Property="FontFamily" Value="Courier New" />
<Setter Property="FontStyle" Value="Oblique" />
<Setter Property="FontWeight" Value="Bold"/>
</Style>
</Application.Resources>
```

## C#

```
//Default GridStackedCellRenderer is removed.
this.dataGrid.CellRenderers.Remove("StackedHeader");
//Customized GridStackedCellRenderer is added.
this.dataGrid.CellRenderers.Add("StackedHeader", new
GridCustomStackedRenderer());
public class GridCustomStackedRenderer : GridStackedHeaderCellRenderer
{
public GridCustomStackedRenderer()
{
}
public override void OnInitializeEditElement(DataColumnBase dataColumn,
GridStackedHeaderCellControl uiElement, object dataContext)
{
if (dataColumn.ColumnIndex == 0)
uiElement.Style = App.Current.Resources["style1"] as Style;
else if (dataColumn.ColumnIndex == 2)
uiElement.Style = App.Current.Resources["style2"] as Style;
base.OnInitializeEditElement(dataColumn, uiElement, dataContext);
}
}
```

<i>ProductDetails</i>		<i>OrderDetails</i>	
Product ID	Product Name	Number of Orders	OrderDate
1001	Alice Mutton	10	5/2/2015
1002	NuNuCa Nuß-Nougat-Creme	10	5/20/2015
1003	Boston Crab Meat	100	5/3/2015
1004	Konbu	50	5/13/2015
1005	Boston Crab Meat	20	5/23/2015
1006	Raclette Courdavault	20	5/13/2015
1007	Wimmers gute Semmelknöde	20	5/13/2015
1008	Gorgonzola Telino	20	5/13/2015
1009	Fløtemysost	20	5/13/2015
1010	Chartreuse verte	20	5/13/2015
1011	Thüringer Rostbratwurst	20	5/13/2015
1012	Vegie-spread	20	5/13/2015
1013	Wimmers gute Semmelknöde	20	5/13/2015
1014	Gorgonzola Telino	20	5/13/2015
1015	Fløtemysost	20	5/13/2015

### Setting Default Style for one column

You can also skip the cell styling for particular column from other setting like [SfDataGrid.CellStyle](#) by setting [GridColumn.CellStyle](#) to null. Likewise, you can skip all the style properties in particular column (example: `HeaderStyle`).

### XML

```
<Window.Resources>
<Style TargetType="syncfusion:GridCell" x:Key="customCellStyle">
<Setter Property="Background" Value="Bisque" />
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
CellStyle="{StaticResource customCellStyle}"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn MappingName="OrderID"
CellStyle="{x:Null}" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#

```
this.dataGrid.Columns["OrderID"].CellStyle = null;
```

OrderID	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1007	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

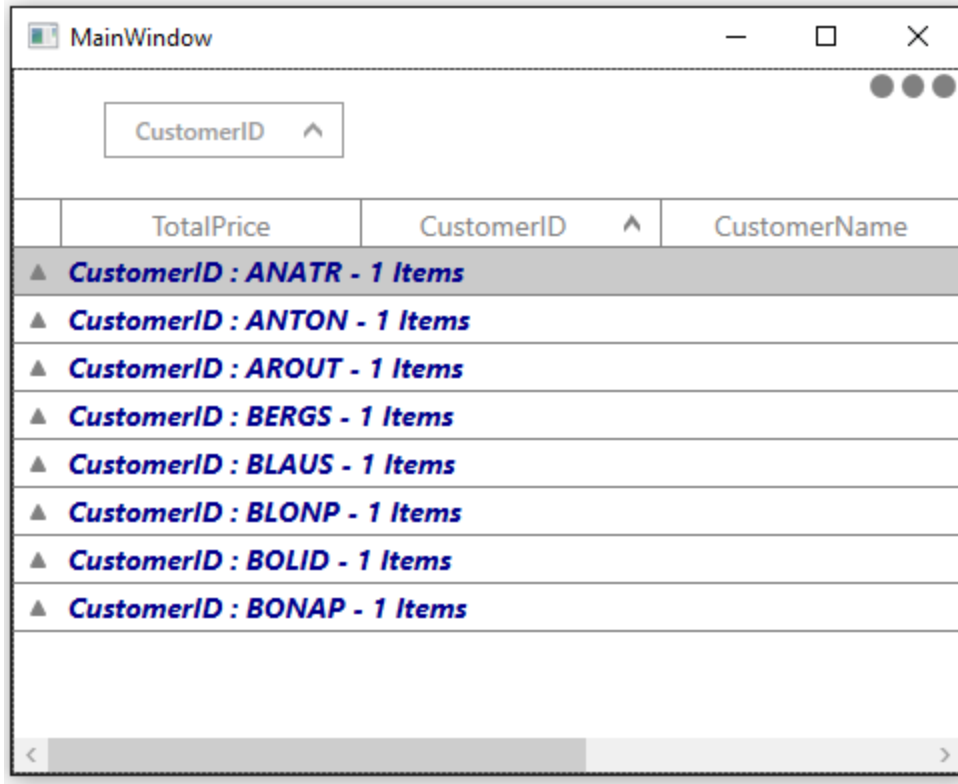
### Styling CaptionSummary

#### Styling CaptionSummary cells

The caption summary cells can be customized by writing style of TargetType [GridCaptionSummaryCell](#). You can set to particular SfDataGrid by setting [SfDataGrid.CaptionSummaryCellStyle](#) property.

### XML

```
<Window.Resources>
<Style TargetType="syncfusion:GridCaptionSummaryCell"
x:Key="customCaptionSummaryCell">
<Setter Property="FontWeight" Value="Bold"/>
<Setter Property="Foreground" Value="DarkBlue"/>
<Setter Property="FontStyle" Value="Italic"/>
<Setter Property="FontSize" Value="14"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
CaptionSummaryCellStyle="{StaticResource customCaptionSummaryCell}"
ItemsSource="{Binding Orders}"/>
```

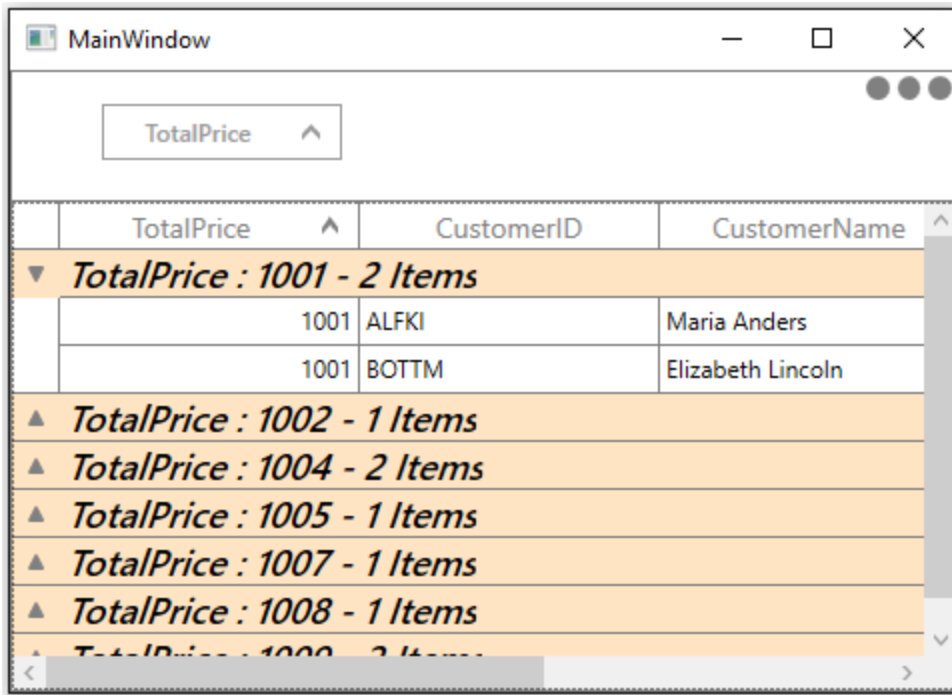


#### Styling CaptionSummary rows

The caption summary rows can be customized by writing style of TargetType [GridCaptionSummaryRowControl](#). You can set to particular SfDataGrid by setting [SfDataGrid.CaptionSummaryRowStyle](#) property.

#### XML

```
<Window.Resources>
<Style TargetType="syncfusion:CaptionSummaryRowControl"
x:Key="captionSummaryRowStyle">
<Setter Property="FontWeight" Value="SemiBold"/>
<Setter Property="Background" Value="Bisque"/>
<Setter Property="FontStyle" Value="Oblique"/>
<Setter Property="FontSize" Value="18"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
CaptionSummaryRowStyle="{StaticResource captionSummaryRowStyle}"
ItemsSource="{Binding Orders}"/>
```



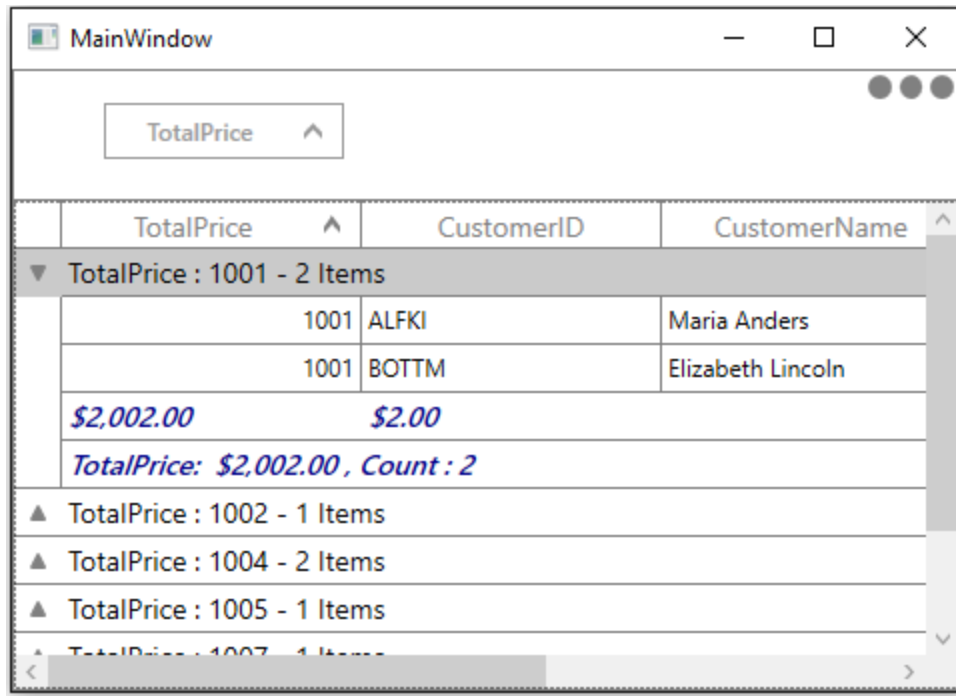
## Styling GroupSummary

### Styling GroupSummary cells

The group summary cells can be customized by writing style of TargetType [GridGroupSummaryCell](#). You can set to particular SfDataGrid by setting [SfDataGrid.GroupSummaryCellStyle](#) property.

### XML

```
<Window.Resources>
<Style TargetType="syncfusion:GridGroupSummaryCell"
x:Key="customGroupSummary">
<Setter Property="FontWeight" Value="SemiBold"/>
<Setter Property="Foreground" Value="DarkBlue"/>
<Setter Property="FontStyle" Value="Oblique"/>
<Setter Property="FontSize" Value="14"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
GroupSummaryCellStyle="{StaticResource customGroupSummary}"
ItemsSource="{Binding Orders}"/>
```

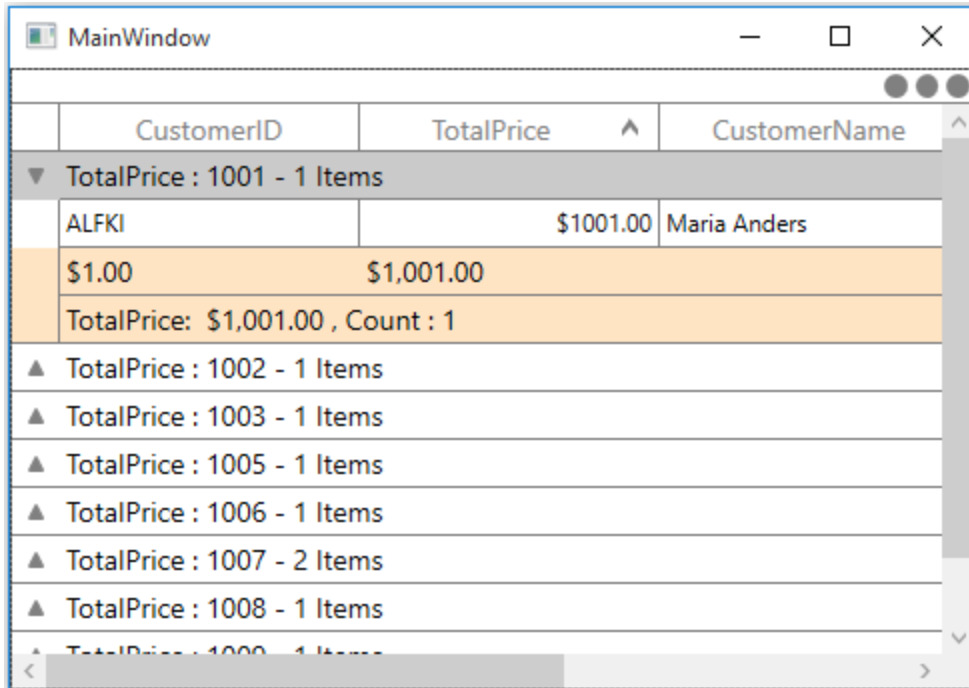


#### Styling GroupSummary rows

The group summary rows can be customized by writing style of TargetType [GridGroupSummaryRowControl](#). You can set to particular SfDataGrid by setting [SfDataGrid.GroupSummaryRowStyle](#) property.

#### XML

```
<Window.Resources>
<Style TargetType="syncfusion:GroupSummaryRowControl"
x:Key="customGroupSummaryRowControl">
<Setter Property="Background" Value="Bisque"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
GroupSummaryRowStyle="{StaticResource customGroupSummaryRowControl}"
ShowGroupDropArea="True"
ItemsSource="{Binding Orders}"/>
```



## Styling TableSummary

### Styling TableSummary cells

The table summary cells can be customized by writing style of TargetType [GridTableSummaryCell](#). You can set to particular SfDataGrid by setting [SfDataGrid.TableSummaryCellStyle](#) property.

### XML

```
<Window.Resources>
<Style x:Key="customTableSummary"
TargetType="syncfusion:GridTableSummaryCell">
<Setter Property="Foreground" Value="DarkBlue" />
<Setter Property="FontSize" Value="16" />
<Setter Property="FontWeight" Value="Bold" />
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
TableSummaryCellStyle="{StaticResource customTableSummary}"/>
```



TotalPrice	CustomerID	CustomerName
<b>\$8,048.00</b>	<b>\$8.00</b>	
1002	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1009	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
<b>Count : 8, Total Price : \$8,048.00</b>		

#### Styling TableSummary rows

The table summary rows can be customized by writing style of TargetType [GridTableSummaryRowControl](#). You can set to particular SfDataGrid by setting [SfDataGrid.TableSummaryRowStyle](#) property.

#### XML

```
<Window.Resources>
<Style TargetType="syncfusion:TableSummaryRowControl"
x:Key="tableSummaryRowStyle">
<Setter Property="Background" Value="Bisque"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
TableSummaryRowStyle="{StaticResource tableSummaryRowStyle}" />
```

CustomerID	TotalPrice	CustomerName	
\$10.00	\$10,058.00		
ALFKI	\$1001.00	Maria Anders	German
ANATR	\$1002.00	Ana Trujilo	Mexico
ANTON	\$1003.00	Antonio Moreno	Mexico
AROUT	\$1007.00	Thomas Hardy	UK
BERGS	\$1005.00	Christina Berglund	Sweden
BLAUS	\$1006.00	Hanna Moos	Germany
BLONP	\$1007.00	Frédérique Citeaux	France
BOLID	\$1008.00	Martin Sommer	Spain
BONAP	\$1009.00	Laurence Lebihan	France
Total Price : \$10,058.00, Count : 10			

## Styling UnboundRows

### Styling unbound row cells

The unbound row cells can be customized by writing style of TargetType [GridUnBoundRowCell](#). You can set to particular SfDataGrid by setting [SfDataGrid.UnBoundRowCellStyle](#) property.

### XML

```
<Style TargetType="syncfusion:GridUnBoundRowCell" x:Key="style">
  <Setter Property="FontWeight" Value="SemiBold"/>
</Style>
<syncfusion:SfDataGrid x:Name="sfDataGrid"
  ItemsSource="{Binding YearlySalesDetails}"
  UnBoundRowCellStyle="{StaticResource style}"/>
```

Name	Q1	Q2
<b>Σ Total Sales By Month</b>	<b>\$6,618.81</b>	<b>\$8,209.40</b>
S-100 Helmet, R	998.55	699.43
S-100 Helmet, B	655.14	588.32
AWC Logo Cap	835.98	932.53
Long-Sleeve Logo Jersey, M	150.66	739.64
L-Sleeve Jersey	281.99	669.35
Long-Sleeve Logo Jersey, XL	207.83	826.83
HL Road Frame - Red, 62	259.44	658.27
LL Road Frame - Black, 58	369.01	676.22
LL Road Frame - Black, 58	329.54	103.38
LL Road Frame - Red, 48	928.28	335.50
LL Road Frame - Red, 52	451.39	600.11
LL Road Frame - Red, 60	692.79	772.13

#### Styling unbound row

The unbound rows can be customized by writing style of TargetType [UnBoundRowControl](#). You can set to particular SfDataGrid by setting [SfDataGrid.UnBoundRowStyle](#) property.

#### XML

```
<Style TargetType="syncfusion:UnBoundRowControl" x:Key="rowStyle">
  <Setter Property="Foreground" Value="blue"/>
</Style>
<syncfusion:SfDataGrid x:Name="sfDataGrid"
  ItemsSource="{Binding YearlySalesDetails}"
  UnBoundRowStyle="{StaticResource rowStyle}"/>
```

Name	QS1	QS2
<b>Total Sales By Month</b>	<b>\$6,729.07</b>	<b>\$7,597.49</b>
S-100 Helmet, R	872.13	735.46
S-100 Helmet, B	168.44	388.68
AWC Logo Cap	444.54	784.70
Long-Sleeve Logo Jersey, M	460.96	962.01
L-Sleeve Jersey	213.25	328.94
Long-Sleeve Logo Jersey, XL	841.24	840.04
HL Road Frame - Red, 62	736.85	527.68
LL Road Frame - Black, 58	356.28	758.72
LL Road Frame - Black, 58	420.71	520.10
LL Road Frame - Red, 48	754.26	540.86
LL Road Frame - Red, 52	323.61	256.14
LL Road Frame - Red, 60	894.31	696.71
ML Road Frame - Red, 48	242.49	257.45

### Styling AddNewRow

The appearance of AddNewRow can be customized by writing style of TargetType [AddNewRowControl](#).

### XML

```
<Window.Resources>
<Style TargetType="Syncfusion:AddNewRowControl">
<Setter Property="AddNewRowText" Value="Enter value to add new row"/>
<Setter Property="FontWeight" Value="Bold"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AddNewRowPosition="Top"
ItemsSource="{Binding Orders}">
```

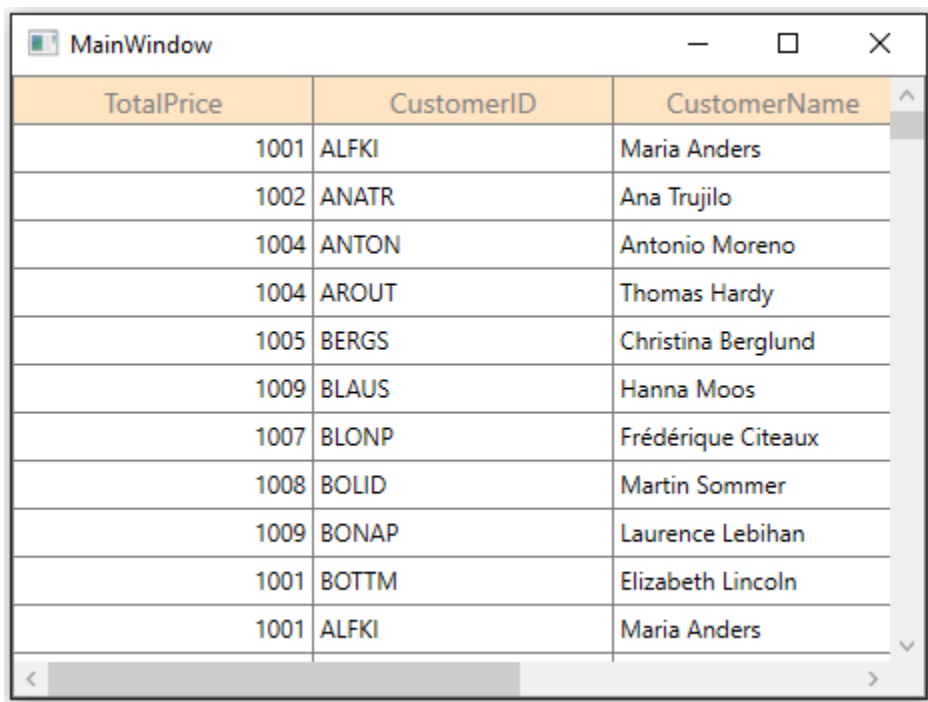
Amount	CustomerID	CustomerName
Enter value to add new row		
9.67	ALFKI	Bulk
2.89	ANATR	Oliver
1.67	ANTON	Brendon
0.00	YHGT	John
1.11	BERGS	Charles
2.77	TGVFD	Dintin
4.88	YTREW	Friedo
7.11	MNBGY	John
5.66	BGFDE	Sirert
8.11	BOTTM	Rakesh
5.99	TGVFD	George
0.87	BERGS	Charles

### Styling RowHeader

The appearance of header row can be customized by writing style of TargetType [HeaderRowControl](#).

#### XML

```
<Window.Resources>
<Style TargetType="syncfusion:HeaderRowControl">
<Setter Property="Background" Value="Bisque"/>
<Setter Property="BorderThickness" Value="1"/>
</Style>
</Window.Resources>
```



TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1009	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1001	BOTTM	Elizabeth Lincoln
1001	ALFKI	Maria Anders

### Displaying row index in row header cell

The appearance of row header can be customized by writing style of TargetType [RowHeaderCell](#).

You can also display the row index value in the row header cell by customizing its style.

#### XML

```
<Window.Resources>
<Style TargetType="syncfusion:GridRowHeaderCell">
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:GridRowHeaderCell">
<Border x:Name="PART_RowHeaderCellBorder"
Background="Bisque"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}">
<Grid>
<!--RowIndex is displayed here -->
<TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center"
Text="{Binding RowIndex,
```

```

RelativeSource={RelativeSource TemplatedParent}}"
TextAlignment="Center" />
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>

```

	TotalPrice	CustomerID	CustomerName
1	1001	ALFKI	Maria Anders
2	1002	ANATR	Ana Trujilo
3	1004	ANTON	Antonio Moreno
4	1004	AROUT	Thomas Hardy
5	1005	BERGS	Christina Berglund
6	1009	BLAUS	Hanna Moos
7	1007	BLONP	Frédérique Citeaux
8	1008	BOLID	Martin Sommer
9	1009	BONAP	Laurence Lebihan
10	1001	BOTTM	Elizabeth Lincoln

### Template Selectors

The [DataTemplateSelectors](#) can be used to set the custom templates to the cell or rows based on the data. You can set to particular SfDataGrid by setting [SfDataGrid.CellTemplateSelector](#) and the template can be set to particular column by setting [GridColumn.CellTemplateSelector](#).

Here, custom template applied to **TotalPrice** and **CustomerID** columns.

### XML

```

<Application.Resources>
<DataTemplate x:Key="CellTemplate1">
<TextBlock Foreground="DarkBlue" Text="{Binding Path=Value}" />
</DataTemplate>
<DataTemplate x:Key="CellTemplate2">
<TextBlock Foreground="DarkRed" Text="{Binding Path=Value}" />
</DataTemplate>
</Application.Resources>
<Window.Resources>
<local:GridColumnTemplateSelector x:Key="templateSelector"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"

```

```

CellTemplateSelector="{StaticResource templateSelector}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTemplateColumn MappingName="TotalPrice"
SetCellBoundValue="True" />
<syncfusion:GridTemplateColumn MappingName="CustomerName"
SetCellBoundValue="True"/>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```

**C#**

```

public class GridCellTemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
        {
            var data = (item as DataContextHelper).Record as OrderInfo;
            //custom logic is checked.
            if (data.TotalPrice < 1005)
            return Application.Current.Resources["CellTemplate1"] as DataTemplate;
            else
            return Application.Current.Resources["CellTemplate2"] as DataTemplate;
        }
    }
}

```

TotalPrice	CustomerName	CustomerID
1001	Maria Anders	ALFKI
1002	Ana Trujilo	ANATR
1003	Antonio Moreno	ANTON
1007	Thomas Hardy	AROUT
1005	Christina Berglund	BERGS
1006	Hanna Moos	BLAUS
1007	Frédérique Citeaux	BLONP
1008	Martin Sommer	BOLID
1009	Laurence Lebihan	BONAP
1010	Elizabeth Lincoln	BOTTM

*Changing HeaderTemplates*

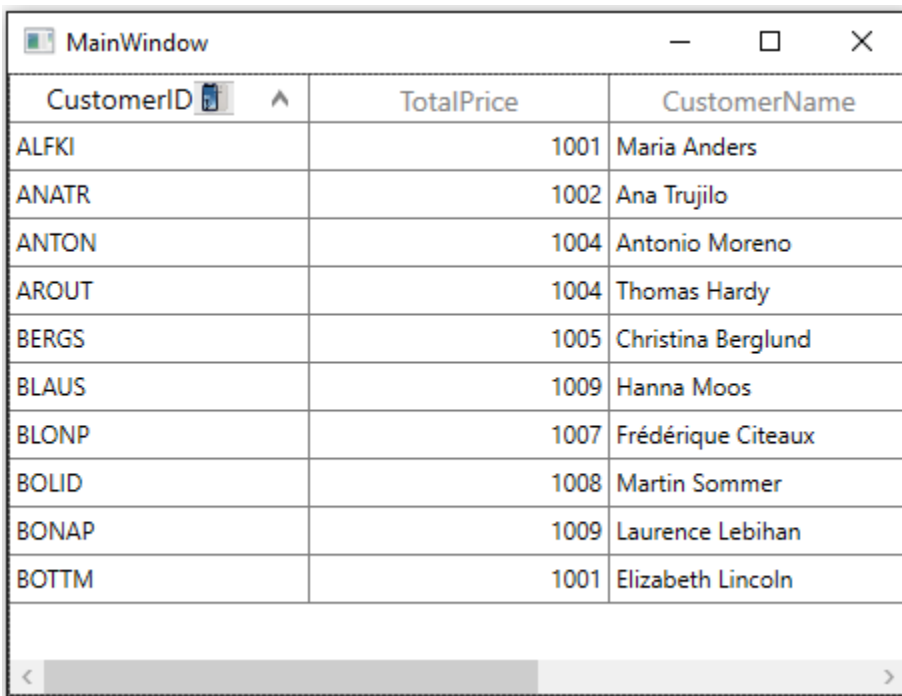
You can customize the appearance of particular SfDataGrid column header by setting [SfDataGrid.HeaderTemplate](#) and the particular column header can be customized by setting [GridColumn.HeaderTemplate](#).

**XML**

```

<DataTemplate x:Key="headerTemplate">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Grid.Column="0" VerticalAlignment="Center"
Foreground="Black" Text="{Binding}" />
<Image Source="/Assets/S3.png" Grid.Column="1" />
</Grid>
</DataTemplate>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
HeaderTemplate="{StaticResource headerTemplate}"/>

```



CustomerID	TotalPrice	CustomerName
ALFKI	1001	Maria Anders
ANATR	1002	Ana Trujilo
ANTON	1004	Antonio Moreno
AROUT	1004	Thomas Hardy
BERGS	1005	Christina Berglund
BLAUS	1009	Hanna Moos
BLONP	1007	Frédérique Citeaux
BOLID	1008	Martin Sommer
BONAP	1009	Laurence Lebihan
BOTTM	1001	Elizabeth Lincoln

#### Loading different editor elements in a same column

The different editor elements can be loaded in a same template column conditionally based on data by setting [GridTemplateColumn.EditTemplateSelector](#).

#### XML

```

<Application.Resources>
<DataTemplate x:Key="DatePicker">
<DatePicker/>
</DataTemplate>
<DataTemplate x:Key="textbox">
<TextBox/>
</DataTemplate>
</Application.Resources>
<Window.Resources>
<local:GridCellEditTemplateSelector x:Key="editSelector"/>
</Window.Resources>

```



```
<syncfusion:SfDataGrid x:Name="dataGrid"
AllowEditing="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTemplateColumn HeaderText="Employee Name"
MappingName="CustomerName"
EditTemplateSelector="{StaticResource editSelector}" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

For example, in the below code example `TextBox` or `DatePicker` will be loaded based on `TotalPrice` property of Underlying data.

### C#

```
public class GridCellEditTemplateSelector : DataTemplateSelector
{
    public override DataTemplate SelectTemplate(object item, DependencyObject container)
    {
        if((item as OrderInfo).TotalPrice < 1005)
            return Application.Current.Resources["textbox"] as DataTemplate;
        else
            return Application.Current.Resources["DatePicker"] as DataTemplate;
    }
}
```

### Styling DetailsViewDataGrid

The appearance of [DetailsViewDataGrid](#) can be customized by writing style of TargetType `DetailsViewDataGrid`. You can set to particular SfDataGrid by setting [SfDataGrid.DetailsViewDataGridStyle](#) property.

### XML

```
<Window.Resources>
<Style TargetType="{x:Type syncfusion:DetailsViewDataGrid}"
x:Key="detailsViewStyle">
<Setter Property="Background" Value="Bisque" />
<Setter Property="BorderBrush" Value="Blue" />
<Setter Property="FontWeight" Value="Bold"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateRelations="True"
DetailsViewDataGridStyle="{StaticResource detailsViewStyle}"
ItemsSource="{Binding Orders}"/>
```

TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1009	RIADIS	Hanna Moos

OrderID	NoOfOrder
1005	1
1006	2
1007	3
1008	4

Styling Filter popup

[Refer here for filter popup styling](#)

Styling Sort icon

The appearance of sort indicator can be customized by editing the style of [GridHeaderCellControl](#). Once the `GridHeaderCellControl` style is edited, go to `PART_SortButtonPresenter`.

*Default GridHeaderCellControl style*

#### **XML**

```
<syncfusion:SortDirectionToVisibilityConverter
x:Key="sortDirectionToVisibilityConverter" />
<syncfusion:SortDirectionToWidthConverter
x:Key="sortDirectionToWidthConverter" />
<Style TargetType="syncfusion:GridHeaderCellControl">
<Setter Property="Background" Value="Red" />
<Setter Property="BorderBrush" Value="Gray" />
<Setter Property="BorderThickness" Value="0,0,1,1" />
<Setter Property="HorizontalContentAlignment" Value="Left" />
<Setter Property="Padding" Value="5,3,5,3" />
<Setter Property="Foreground" Value="Gray" />
<Setter Property="FontSize" Value="14" />
<Setter Property="FontWeight" Value="Normal" />
<Setter Property="IsTabStop" Value="False" />
<Setter Property="syncfusion:VisualContainer.WantsMouseInput" Value="True" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:GridHeaderCellControl">
<Grid>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="HiddenColumnsResizingStates">
<VisualState x:Name="PreviousColumnHidden">
```

```

<Storyboard>
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_HeaderCellBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0" Value="3, 0, 1, 1" />
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="HiddenState">
<Storyboard>
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_HeaderCellBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0" Value="3, 0, 3, 1" />
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="NormalState" />
<VisualState x:Name="LastColumnHidden">
<Storyboard>
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_HeaderCellBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0" Value="0, 0, 3, 1" />
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="CommonStates">
<VisualState x:Name="MouseOver" />
<VisualState x:Name="Normal" />
</VisualStateGroup>
<VisualStateGroup x:Name="BorderStates">
<VisualState x:Name="NormalCell" />
<VisualState x:Name="FrozenColumnCell">
<Storyboard BeginTime="0">
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_HeaderCellBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0" Value="0,0,1,1" />
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="FooterColumnCell">
<Storyboard BeginTime="0">
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_FooterCellBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0" Value="1,0,1,1" />
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>

```

```

<VisualState x:Name="BeforeFooterColumnCell">
<Storyboard BeginTime="0">
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_FooterCellBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0" Value="0,0,0,1" />
</ThicknessAnimationUsingKeyFrames>
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_HeaderCellBorder"
Storyboard.TargetProperty="BorderThickness">
<EasingThicknessKeyFrame KeyTime="0" Value="0,0,0,1" />
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border x:Name="PART_FooterCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}" />
<Border x:Name="PART_HeaderCellBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
SnapsToDevicePixels="True">
<Grid Margin="{TemplateBinding Padding}" SnapsToDevicePixels="True">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<ContentPresenter HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"
VerticalAlignment="Center"
Focusable="False" />
<Border x:Name="PART_FilterPopUpPresenter" />
<Grid x:Name="PART_SortButtonPresenter"
Grid.Column="1"
SnapsToDevicePixels="True">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="0" MinWidth="{Binding Path=SortDirection,
Mode=OneWay, RelativeSource={RelativeSource TemplatedParent}},
Converter={StaticResource sortDirectionToWidthConverter}}" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Path Width="8.938"
Height="8.138"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="F1M753.644,-13.0589L753.736,-12.9639 753.557,-12.7816 732.137,8.63641
732.137,29.7119 756.445,5.40851 764.094,-2.24384 764.275,-2.42352
771.834,5.1286 796.137,29.4372 796.137,8.36163 774.722,-13.0589 764.181,-
23.5967 753.644,-13.0589z"
Fill="Gray"
SnapsToDevicePixels="True"
Stretch="Fill"

```

```

Visibility="{Binding Path=SortDirection,
RelativeSource={RelativeSource TemplatedParent},
ConverterParameter=Ascending,
Converter={StaticResource sortDirectionToVisibilityConverter}}">
<Path.RenderTransform>
<TransformGroup>
<TransformGroup.Children>
<RotateTransform Angle="0" />
<ScaleTransform ScaleX="1" ScaleY="1" />
</TransformGroup.Children>
</TransformGroup>
</Path.RenderTransform>
</Path>
<Path Width="8.938"
Height="8.138"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="F1M181.297,177.841L181.205,177.746 181.385,177.563 202.804,156.146
202.804,135.07 178.497,159.373 170.847,167.026 170.666,167.205
163.107,159.653 138.804,135.345 138.804,156.42 160.219,177.841
170.76,188.379 181.297,177.841z"
Fill="Gray"
SnapsToDevicePixels="True"
Stretch="Fill"
Visibility="{Binding Path=SortDirection,
RelativeSource={RelativeSource TemplatedParent},
ConverterParameter=Decending,
Converter={StaticResource sortDirectionToVisibilityConverter}}">
<Path.RenderTransform>
<TransformGroup>
<TransformGroup.Children>
<RotateTransform Angle="0" />
<ScaleTransform ScaleX="1" ScaleY="1" />
</TransformGroup.Children>
</TransformGroup>
</Path.RenderTransform>
</Path>
<TextBlock Grid.Column="1"
Margin="0,-4,0,0"
VerticalAlignment="Center"
FontSize="10"
Foreground="{TemplateBinding Foreground}"
SnapsToDevicePixels="True"
Text="{TemplateBinding SortNumber}"
Visibility="{TemplateBinding SortNumberVisibility}" />
</Grid>
<syncfusion:FilterToggleButton x:Name="PART_FilterToggleButton"
Grid.Column="2"
HorizontalAlignment="Stretch"
VerticalAlignment="Stretch"
SnapsToDevicePixels="True"
Visibility="{TemplateBinding FilterIconVisiblity}" />
</Grid>
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>

```

```
</Setter>
</Style>
```

Totally two paths will be present under the PART\_SortButtonPresenter. You can change the appearance of Ascending sort indicator by customizing first path present in this.

Here, height and color of the indicator is customized in the below code example.

#### *Customizing Ascending Sort Indicator*

##### **XML**

```
<Path Data="F1M753.644,-13.0589L753.736,-12.9639 753.557,-12.7816
732.137,8.63641 732.137,29.7119 756.445,5.40851 764.094,-2.24384 764.275,-
2.42352 771.834,5.1286 796.137,29.4372 796.137,8.36163 774.722,-13.0589
764.181,-23.5967 753.644,-13.0589z"
Fill="DarkBlue"
HorizontalAlignment="Center"
Height="15"
Stretch="Fill"
SnapsToDevicePixels="True"
VerticalAlignment="Center"
Width="12">
  <Path.RenderTransform>
    <TransformGroup>
      <RotateTransform Angle="0"/>
      <ScaleTransform ScaleY="1" ScaleX="1"/>
    </TransformGroup>
  </Path.RenderTransform>
  <Path.Visibility>
    <Binding ConverterParameter="Ascending" Path="SortDirection"
RelativeSource="{RelativeSource TemplatedParent}">
  <Binding.Converter>
    <syncfusion:SortDirectionToVisibilityConverter/>
  </Binding.Converter>
</Binding>
</Path.Visibility>
</Path>
```

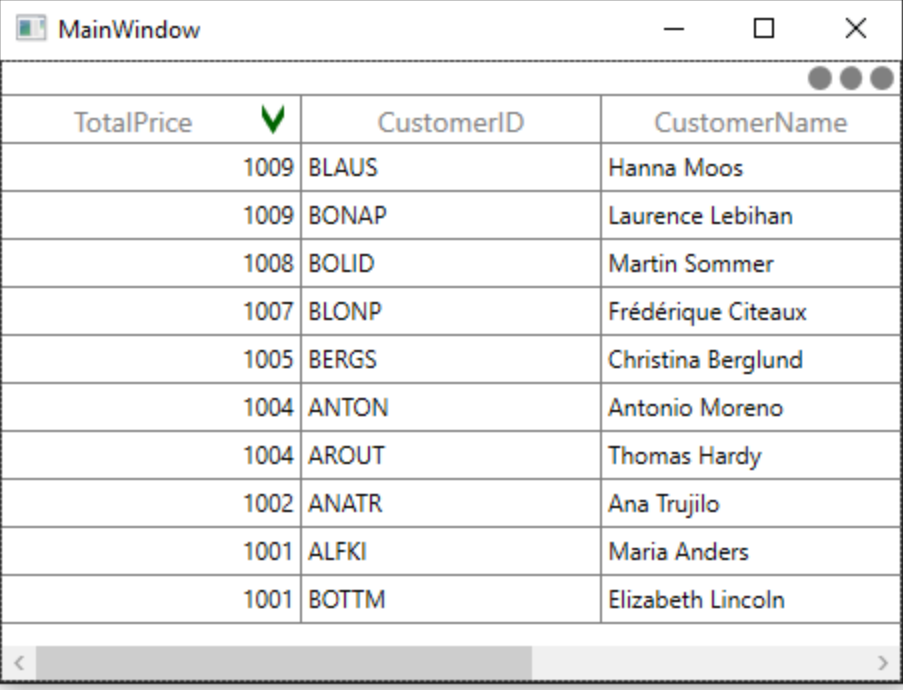
And also, you can change the appearance of Descending sort indicator by customizing second path present in PART\_SortButtonPresenter. For example, in the below code example height and color of the indicator is changed.

TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1001	BOTTM	Elizabeth Lincoln
1002	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BLAUS	Hanna Moos
1009	BONAP	Laurence Lebihan

*Customizing Descending Sort Indicator*

#### **XML**

```
<Path Data="F1M181.297,177.841L181.205,177.746 181.385,177.563
202.804,156.146 202.804,135.07 178.497,159.373 170.847,167.026
170.666,167.205 163.107,159.653 138.804,135.345 138.804,156.42
160.219,177.841 170.76,188.379 181.297,177.841z"
Fill="DarkGreen"
HorizontalAlignment="Center"
Height="15"
Stretch="Fill"
SnapsToDevicePixels="True"
VerticalAlignment="Center"
Width="11">
  <Path.RenderTransform>
    <TransformGroup>
      <RotateTransform Angle="0"/>
      <ScaleTransform ScaleY="1" ScaleX="1"/>
    </TransformGroup>
  </Path.RenderTransform>
  <Path.Visibility>
    <Binding ConverterParameter="Decending" Path="SortDirection"
      RelativeSource="{RelativeSource TemplatedParent}">
      <Binding.Converter>
        <syncfusion:SortDirectionToVisibilityConverter/>
      </Binding.Converter>
    </Binding>
  </Path.Visibility>
</Path>
```



TotalPrice	CustomerID	CustomerName
1009	BLAUS	Hanna Moos
1009	BONAP	Laurence Lebihan
1008	BOLID	Martin Sommer
1007	BLONP	Frédérique Citeaux
1005	BERGS	Christina Berglund
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1002	ANATR	Ana Trujilo
1001	ALFKI	Maria Anders
1001	BOTTM	Elizabeth Lincoln

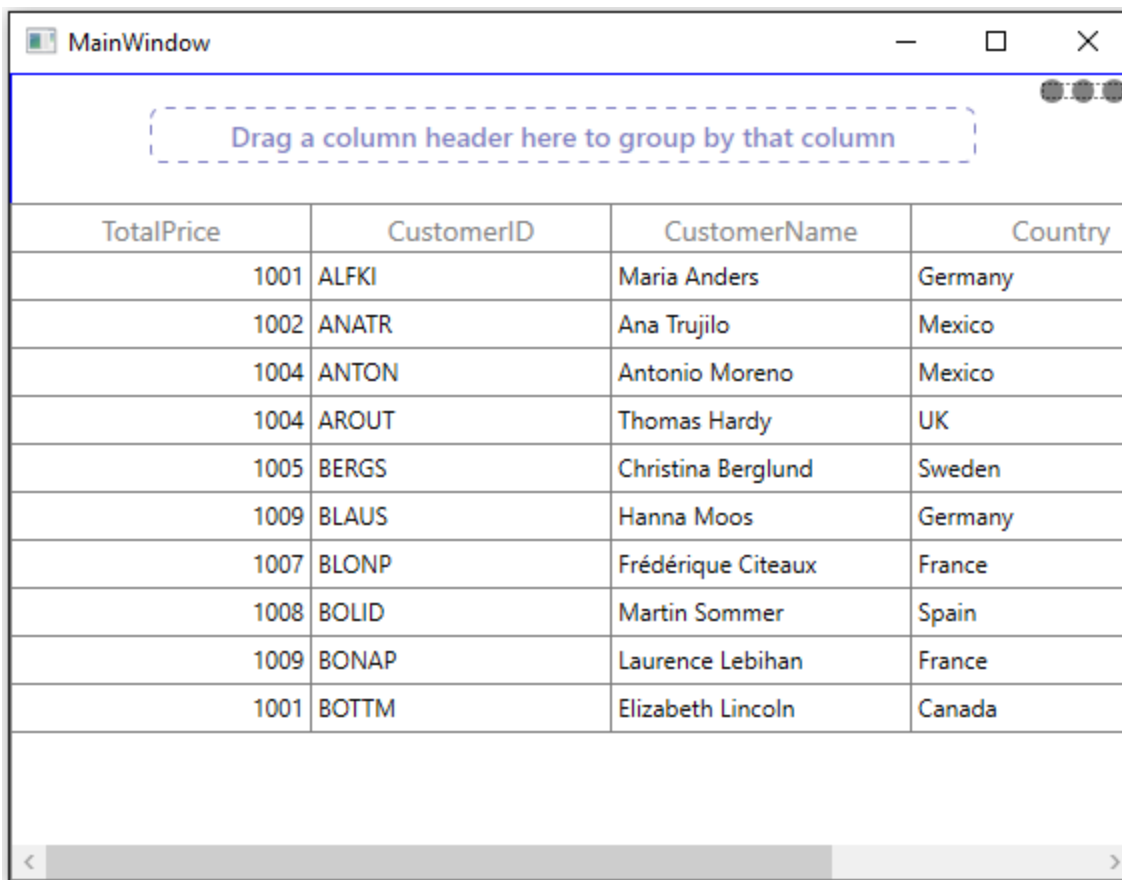
### Styling GroupDropArea

The appearance of [GroupDropArea](#) can be customized by writing style of TargetType `GroupDropArea`. You can disable the watermark displayed in `GroupDropArea` by setting [WaterMarkTextVisibility](#) as Collapsed.

### XML

```
<Window.Resources>
<Style TargetType="syncfusion:GroupDropArea">
<Setter Property="BorderBrush" Value="Blue"/>
<Setter Property="Foreground" Value="DarkBlue"/>
<Setter Property="FontWeight" Value="Medium"/>
<Setter Property="WatermarkTextVisibility" Value="Visible"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
ItemsSource="{Binding Orders}"/>
```





### Showing busy indicator before loading records

You can show the indication of data loading with the help of [BusyIndicator](#) by setting [BusyIndicator.IsBusy](#) as `True` and you can stop it by setting [BusyIndicator.IsBusy](#) as `false` in the `ItemsSourceChanged` event.

### XML

```
<Syncfusion:SfBusyIndicator Name="sfBusyIndicator"
    IsBusy="True"
    Margin="5"
    VerticalAlignment="Center"
    AnimationType="Gear"/>
```

### C#

```
sfDataGrid.Loaded += sfDataGrid_Loaded;
sfDataGrid.ItemsSourceChanged += sfDataGrid_ItemsSourceChanged;
async void sfDataGrid_Loaded(object sender, RoutedEventArgs e)
{
    this.sfDataGrid.ItemsSource = await (this.DataContext as
    ViewModel).GetRecords();
}
void sfDataGrid_ItemsSourceChanged(object sender,
GridItemsSourceChangedEventArgs e)
{
    sfBusyIndicator.IsBusy = false;
```

```
}

```

Customer ID	Ship City	Total Price
		

Conditional Styling in WPF DataGrid (SfDataGrid)

You can style the DataGrid and its inner elements (cells, rows and columns) conditionally based on data in three ways,

- 1. Using Converter
- 2. Using Data Triggers
- 3. Using StyleSelector

Approach	Performance
Using Converter	Provide good performance when compared other two ways.
Using Data Triggers	When compared to converter, performance is slow while styling more number of columns or rows.
Using StyleSelector	It affects scrolling performance while styling more number of columns based on number of columns visible.

## Cell style

### *Conditional styling of cells using converter*

The record cells ([GridCell](#)) can be customized conditionally by changing its property value based on cell value or data object using converter.

Here, GridCell background is changed using converter, where converter returns the value based on OrderID property of underlying record.

### XML

```
<Window.Resources>
<local:ColorConverter x:Key="converter"/>
</Window.Resources>
<syncfusion:GridTextColumn MappingName="TotalPrice">
<syncfusion:GridTextColumn.CellStyle>
<Style TargetType="syncfusion:GridCell">
<Setter Property="Background" Value="{Binding
Path=OrderID,Converter={StaticResource converter}}"/>
</Style>
</syncfusion:GridTextColumn.CellStyle>
</syncfusion:GridTextColumn>
```

### C#

```
public class ColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        int input = (int)value;
        //custom condition is checked based on data.
        if (input < 1003)
            return new SolidColorBrush(Colors.LightBlue);
        else if (input < 1007)
            return new SolidColorBrush(Colors.Bisque);
        else
            return DependencyProperty.UnsetValue;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

#### Condition styling of cells based on record using converter

You can also style the cells based on record instead of passing single property to converter, where converter returns the value based on underlying record. This can be assigned to `GridColumn.CellStyle` to style the column based on other column properties.

#### XML

```
<Window.Resources>
<local:ColorConverter x:Key="converter"/>
<Style TargetType="syncfusion:GridCell">
<Setter Property="Background" Value="{Binding Converter={StaticResource
converter}}"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}" />
```

#### C#

```
public class ColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var data = value as OrderInfo;
        //custom condition is checked based on data.
        if (data.OrderID < 1003)
            return new SolidColorBrush(Colors.LightBlue);
        else if (data.OrderID < 1007)
            return new SolidColorBrush(Colors.Bisque);
        else
            return DependencyProperty.UnsetValue;
    }
}
```

```

}
public object ConvertBack(object value, Type targetType, object parameter,
    CultureInfo culture)
{
    throw new NotImplementedException();
}
}

```

ID	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

### Conditional styling of cells using triggers

The record cells ([GridCell](#)) can be customized by setting [Style.Triggers](#) that apply property values based on specified conditions. Multiple conditions can be specified by setting [MultiDataTrigger](#).

### XML

```

<syncfusion:GridTextColumn MappingName="OrderID" >
<syncfusion:GridTextColumn.CellStyle>
<Style TargetType="syncfusion:GridCell">
<Style.Triggers>
<!--Background property set based on cell content-->
<DataTrigger Binding="{Binding Path=OrderID}" Value="1001">
<Setter Property="Background" Value="Bisque" />
</DataTrigger>
<!--Background property set based on multiple conditions-->
<MultiDataTrigger>
<MultiDataTrigger.Conditions>
<Condition Binding="{Binding Path=OrderID}" Value="1008" />
<Condition Binding="{Binding Path=CustomerID}" Value="BOLID" />
</MultiDataTrigger.Conditions>
<Setter Property="Background" Value="LightBlue" />
</MultiDataTrigger>
</Style.Triggers>
</Style>

```

```
</syncfusion:GridTextColumn.CellStyle>
</syncfusion:GridTextColumn>
```

Here, GridCell's are conditionally customized based on `OrderID` value.

TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1008	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

#### Conditional styling of cells using style selector

The record cells ([GridCell](#)) can be customized conditionally based on data by setting [SfDataGrid.CellStyleSelector](#) property and the particular column record cells can be customized by setting [GridColumn.CellStyleSelector](#) property and you can get the container as `GridCell` in the `CellStyleSelector`.

**Note:** `GridColumn.CellStyleSelector` takes higher priority than `SfDataGrid.CellStyleSelector` property.

#### XML

```
<Application.Resources>
<local:SelectorClass x:Key="styleSelector"/>
<Style x:Key="redCellStyle" TargetType="syncfusion:GridCell">
<Setter Property="Foreground" Value="Red" />
</Style>
<Style x:Key="blueCellStyle" TargetType="syncfusion:GridCell">
<Setter Property="Foreground" Value="DarkBlue" />
</Style>
</Application.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}" CellStyleSelector="{StaticResource
styleSelector}"/>
```

#### C#

```

public class SelectorClass : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var data = item as OrderInfo;
        if (data != null && ((container as
        GridCell).ColumnBase.GridColumn.MappingName == "TotalPrice"))
        {
            //custom condition is checked based on data.
            if (data.TotalPrice < 1005)
            {
                return App.Current.Resources["redCellStyle"] as Style;
            }
            return App.Current.Resources["blueCellStyle"] as Style;
        }
        return base.SelectStyle(item, container);
    }
}

```

Here, GridCell's are customized based on **TotalPrice** property of underlying record.

TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

Row style

*Conditional styling of rows using converter*

The record rows ([VirtualizingCellsControl](#)) can be customized conditionally by changing its property value based on 'cell value' or 'data object' by using **converter**, where converter returns the value based on Underlying record.

#### XML

```

<Window.Resources>
<local:ColorConverter x:Key="converter"/>
<Style TargetType="syncfusion:VirtualizingCellsControl">
<Setter Property="Background" Value="{Binding Converter={StaticResource
converter}}"/>

```

```

</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"/>

```

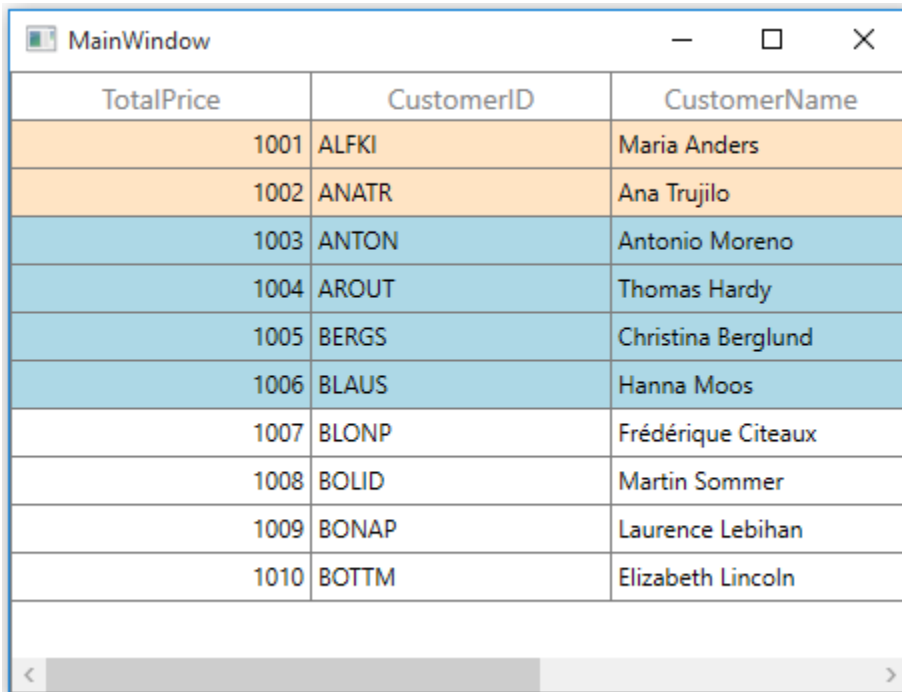
**C#**

```

public class ColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var input = (value as OrderInfo).TotalPrice;
        //custom condition is checked based on data.
        if (input < 1003)
            return new SolidColorBrush(Colors.Bisque);
        else if (input < 1007)
            return new SolidColorBrush(Colors.LightBlue);
        else
            return DependencyProperty.UnsetValue;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

Here, rows are customized based on **TotalPrice** property of underlying record.



TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln



*Conditional styling of rows using style selector*

The record rows ([VirtualizingCellsControl](#)) can be customized conditionally based on data by setting [SfDataGrid.RowStyleSelector](#) property and you can get the container as [VirtualizingCellsControl](#) in [StyleSelector](#).

**XML**

```
<Application.Resources>
<Style x:Key="rowStyle1" TargetType="syncfusion:VirtualizingCellsControl">
<Setter Property="Background" Value="Bisque" />
</Style>
<Style x:Key="rowStyle2" TargetType="syncfusion:VirtualizingCellsControl">
<Setter Property="Background" Value="Aqua" />
</Style>
</Application.Resources>
<Window.Resources>
<local:CustomRowStyleSelector x:Key="rowStyleSelector" />
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
RowStyleSelector="{StaticResource rowStyleSelector}"/>
```

**C#**

```
public class CustomRowStyleSelector : StyleSelector
{
public override Style SelectStyle(object item, DependencyObject container)
{
var row = (item as DataRowBase).RowData;
var data = row as OrderInfo;
if (data.TotalPrice < 1004)
return App.Current.Resources["rowStyle1"] as Style;
return App.Current.Resources["rowStyle2"] as Style;
}
}
```

Here, rows are customized based on [TotalPrice](#) property of underlying record.

TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1009	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1001	BOTTM	Elizabeth Lincoln

### Alternate row style

The appearance of alternating rows can be customized conditionally based on data by setting [SfDataGrid.AlternatingRowStyleSelector](#) property.

### XML

```
<Application.Resources>
<Style x:Key="rowStyle1" TargetType="syncfusion:VirtualizingCellsControl">
<Setter Property="Background" Value="Bisque" />
</Style>
<Style x:Key="rowStyle2" TargetType="syncfusion:VirtualizingCellsControl">
<Setter Property="Background" Value="Aqua" />
</Style>
</Application.Resources>
<Window.Resources>
<local:CustomRowStyleSelector x:Key="alternatingRowStyleSelector" />
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
AlternatingRowStyleSelector="{StaticResource alternatingRowStyleSelector}"/>
```

### C#

```
public class CustomRowStyleSelector : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var row = (item as DataRowBase).RowData;
        var data = row as OrderInfo;
        // Applying alternating background for rows.
        if (data.OrderID < 1006)
            return App.Current.Resources["rowStyle1"] as Style;
    }
}
```

```
return App.Current.Resources["rowStyle2"] as Style;
}
}
```

Here, alternating rows are customized based on **OrderID** property of underlying record.

Order ID	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1008	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

Caption summary cell style

*Conditional styling of caption summary cells using converter*

The appearance of caption summary cell can be customized conditionally based on summary value by using **converter**, where converter returns the value based on summary value.

### XML

```
<Window.Resources>
<local:ColorConverter x:Key="converter"/>
<Style TargetType="syncfusion:GridCaptionSummaryCell">
<Setter Property="Foreground" Value="{Binding Converter={StaticResource
converter}}"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}" />
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>
```

```

MappingName="CustomerID"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>

```

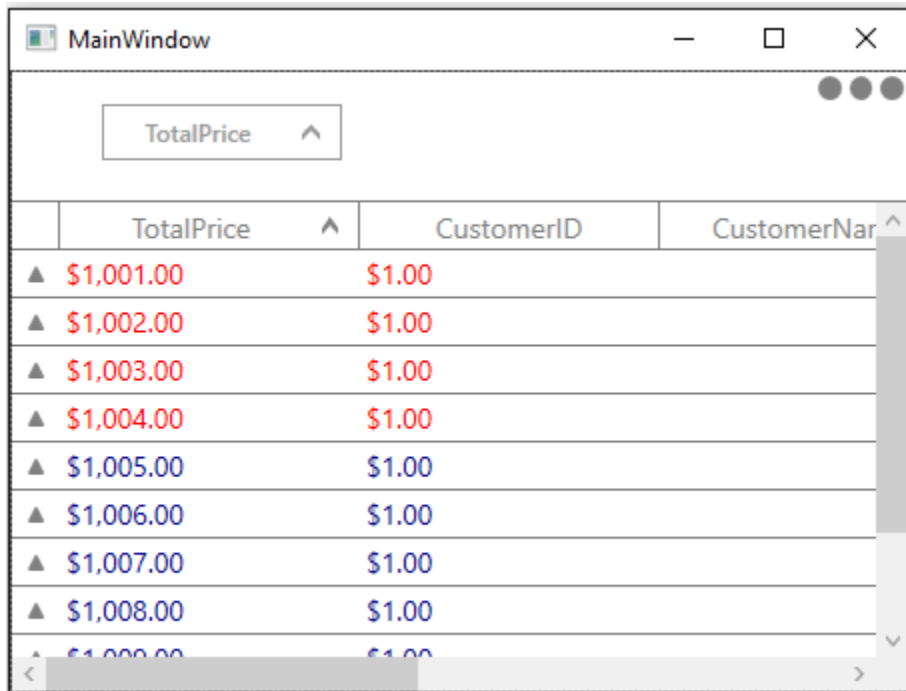
**C#**

```

public class ColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var summaryValue = (value as Group).SummaryDetails.SummaryValues[0];
        var aggregateValue = summaryValue.AggregateValues.ElementAt(0);
        var calculatedValue = aggregateValue.Value;
        //custom condition is checked.
        if ((double)calculatedValue < 1005)
            return new SolidColorBrush(Colors.Red);
        return new SolidColorBrush(Colors.DarkBlue);
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

Here, caption summary cells are customized based on **TotalPrice** summary value.



	TotalPrice ^	CustomerID	CustomerName ^
▲	\$1,001.00	\$1.00	
▲	\$1,002.00	\$1.00	
▲	\$1,003.00	\$1.00	
▲	\$1,004.00	\$1.00	
▲	\$1,005.00	\$1.00	
▲	\$1,006.00	\$1.00	
▲	\$1,007.00	\$1.00	
▲	\$1,008.00	\$1.00	
▲	\$1,009.00	\$1.00	

*Conditional styling of caption summary cells using style selector*

The appearance of caption summary cell can be customized conditionally based on summary value by setting [SfDataGrid.CaptionSummaryCellStyleSelector](#) and you can get the container as [GridCaptionSummaryCell](#) using StyleSelector.

**XML**

```
<Application.Resources>
<Style TargetType="syncfusion:GridCaptionSummaryCell"
x:Key="captionSummaryStyle ">
<Setter Property="Foreground" Value="Red"/>
</Style>
</Application.Resources>
<Window.Resources>
<local:SelectorClass x:Key="selector"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
CaptionSummaryCellStyleSelector="{StaticResource selector}"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
MappingName="CustomerID"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>
```

**C#**

```
public class SelectorClass : StyleSelector
{
public override Style SelectStyle(object item, DependencyObject container)
{
var summaryValue = (item as Group).SummaryDetails.SummaryValues[0];
var aggregateValue = summaryValue.AggregateValues.ElementAt(0);
var calculatedValue = aggregateValue.Value;
//custom condition is checked.
if ((double)calculatedValue < 0)
return App.Current.Resources["captionSummaryStyle"] as Style;
return base.SelectStyle(item, container);
}
}
```

Here, caption summary cells are customized based on [TotalPrice](#) summary value.

TotalPrice	CustomerID
(\$1,010.00)	\$1.00
(\$1,008.00)	\$1.00
(\$1,006.00)	\$1.00
(\$1,004.00)	\$1.00
(\$1,002.00)	\$1.00
\$1,001.00	\$1.00
\$1,003.00	\$1.00
\$1,005.00	\$1.00
\$1,007.00	\$1.00

*Conditional styling of caption summary cell based on column*

The caption summary cells can be conditionally customized summary column.

Here, caption summary cells are customized based on **TotalPrice** summary column.

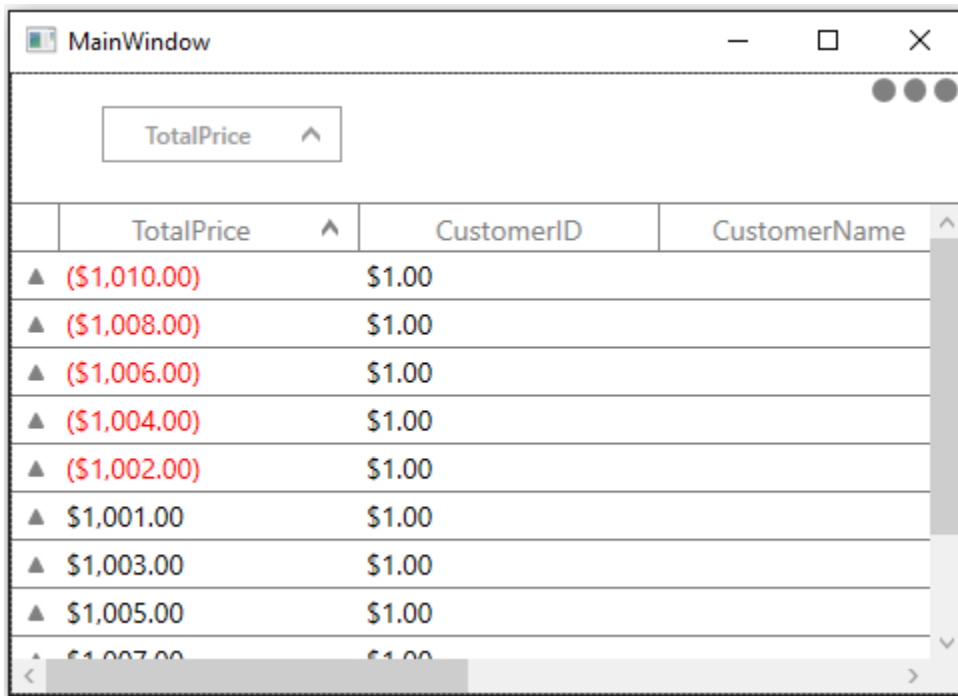
#### XML

```
<Application.Resources>
<Style TargetType="syncfusion:GridCaptionSummaryCell"
x:Key="captionSummaryStyle">
<Setter Property="Foreground" Value="Red"/>
</Style>
</Application.Resources>
<Window.Resources>
<local:SelectorClass x:Key="selector"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
CaptionSummaryCellStyleSelector="{StaticResource selector}"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate"/>
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
MappingName="CustomerID"
SummaryType="CountAggregate"/>
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>
```

```
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>
```

**C#**

```
public class SelectorClass : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var cell = container as GridCaptionSummaryCell;
        if (cell.ColumnBase.GridColumn.MappingName == "TotalPrice")
        {
            var groupKey = (int)(item as Group).Key;
            //custom condition is checked.
            if (groupKey < 0)
            {
                return App.Current.Resources["captionSummaryStyle"] as Style;
            }
            return null;
        }
    }
}
```



Caption summary row style

*Conditional styling of caption summary row using converter*

The appearance of caption summary row can be customized conditionally based on summary value by using **converter**, where converter returns the value based on summary value.

**XML**

```
<Window.Resources>
<local:ColorConverter x:Key="converter"/>
<Style TargetType="syncfusion:CaptionSummaryRowControl">
```

```

<Setter Property="Background" Value="{Binding Converter={StaticResource
converter}}"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow Title="Total Price : {price}"
ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
MappingName="CustomerID"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>

```

**C#**

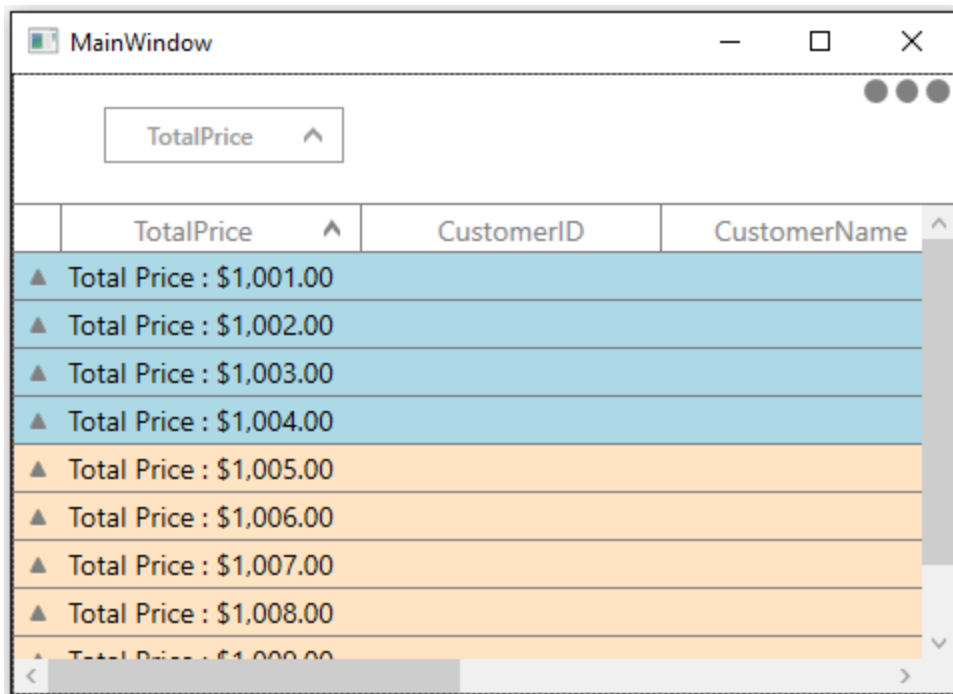
```

public class ColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var summaryValue = (value as Group).SummaryDetails.SummaryValues[0];
        var aggregateValue = summaryValue.AggregateValues.ElementAt(0);
        var calculatedValue = aggregateValue.Value;
        //custom condition is checked.
        if ((double)calculatedValue < 1005)
            return new SolidColorBrush(Colors.LightBlue);
        return new SolidColorBrush(Colors.Bisque);
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

Here, caption summary rows are customized based on **TotalPrice** summary value.





#### Conditional styling of caption summary row using style selector

In another way, appearance of caption summary row can be customized conditionally based on summary value by setting [SfDataGrid.CaptionSummaryRowStyleSelector](#) and you can get the container as [CaptionSummaryRowControl](#) in StyleSelector.

Here, caption summary rows are customized where [group key](#) value is negative.

#### XML

```
<Application.Resources>
<Style TargetType="syncfusion:CaptionSummaryRowControl"
x:Key="captionSummaryStyle">
<Setter Property="Background" Value="Bisque"/>
</Style>
</Application.Resources>
<Window.Resources>
<local:SelectorClass x:Key="styleSelector"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
CaptionSummaryRowStyleSelector="{StaticResource styleSelector}"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.CaptionSummaryRow>
<syncfusion:GridSummaryRow Title="Total Price : {price}"
ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
MappingName="CustomerID"
```

```

SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.CaptionSummaryRow>
</syncfusion:SfDataGrid>

```

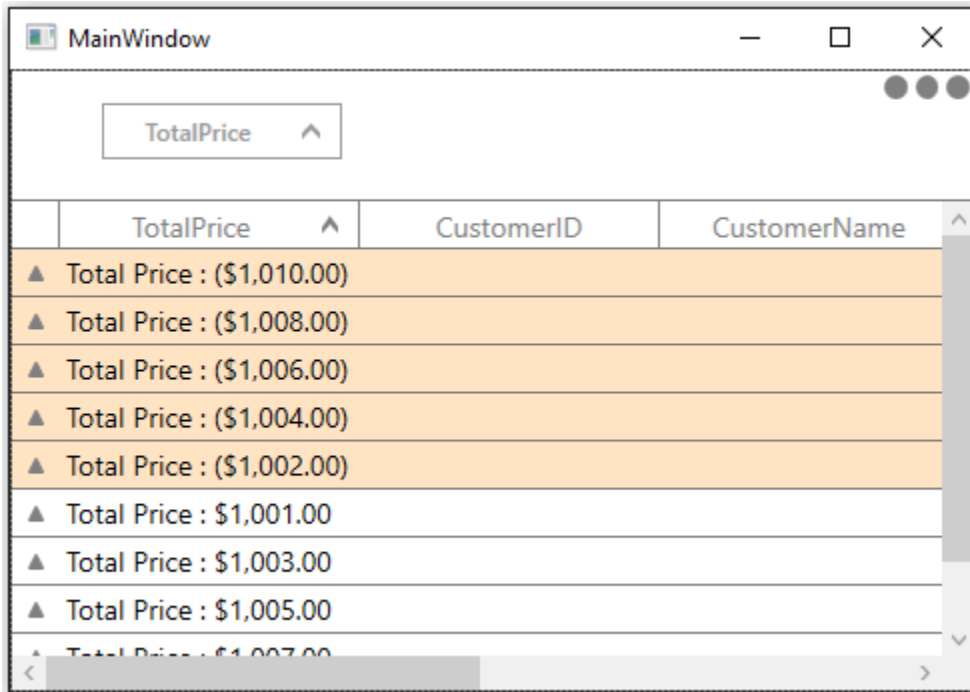
**C#**

```

public class SelectorClass : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var row = (item as SpannedDataRow).RowData;
        var groupKey = (int)(row as Group).Key;
        //custom condition is checked.
        if (groupKey < 0)
            return App.Current.Resources["captionSummaryStyle"] as Style;
        return null;
    }
}

```

Here, caption summary rows are customized based on **TotalPrice** summary value.



*Conditional styling of caption summary row based on group level*

The appearance of caption summary row can be conditionally customized based on [grouping level](#) using StyleSelector.

**XML**

```

<Application.Resources>
<Style x:Key="rowStyle1" TargetType="syncfusion:CaptionSummaryRowControl">
<Setter Property="Background" Value="LightPink" />

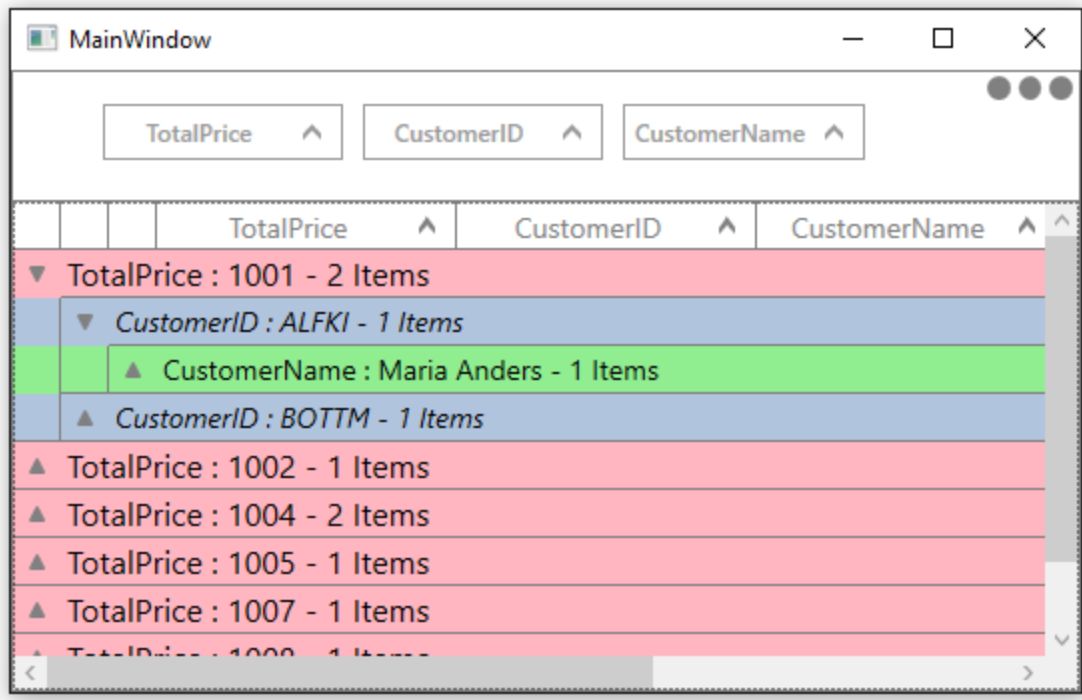
```

```
<Setter Property="FontSize" Value="16" />
</Style>
<Style x:Key="rowStyle2" TargetType="syncfusion:CaptionSummaryRowControl">
<Setter Property="Background" Value="LightSteelBlue" />
<Setter Property="FontStyle" Value="Italic" />
</Style>
<Style x:Key="rowStyle3" TargetType="syncfusion:CaptionSummaryRowControl">
<Setter Property="Background" Value="LightGreen" />
</Style>
</Application.Resources>
<Window.Resources>
<local:CustomCaptionSummaryRowStyleSelector x:Key="styleSelector" />
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
CaptionSummaryRowStyleSelector="{StaticResource styleSelector}"
ItemsSource="{Binding Orders}"
ShowGroupDropArea="True">
```

## C#

```
public class CustomCaptionSummaryRowStyleSelector : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var dataRow = item as DataRowBase;
        var level = dataRow.Level;
        //based on group levels, style applied to CaptionSummaryRow
        if (level == 1)
            return App.Current.Resources["rowStyle1"] as Style;
        else if (level == 2)
            return App.Current.Resources["rowStyle2"] as Style;
        else if (level == 3)
            return App.Current.Resources["rowStyle3"] as Style;
        return base.SelectStyle(item, container);
    }
}
```

Here, caption summary rows are customized based on **grouping level** (example: level1, level2, level3, etc.).



### Group summary cell style

Group summary cells can be customized conditionally by getting particular summary value from [SummaryValues](#) through converter or style selector. Likewise, you can also customize the group summary cell based on various properties exposed in [GridSummaryRow](#) (example: [ShowSummaryInRow](#) property).

### Conditional styling of group summary cell using converter

The appearance of group summary cell can be customized conditionally based on summary value by using 'converter', where converter returns the value based on summary value.

### XML

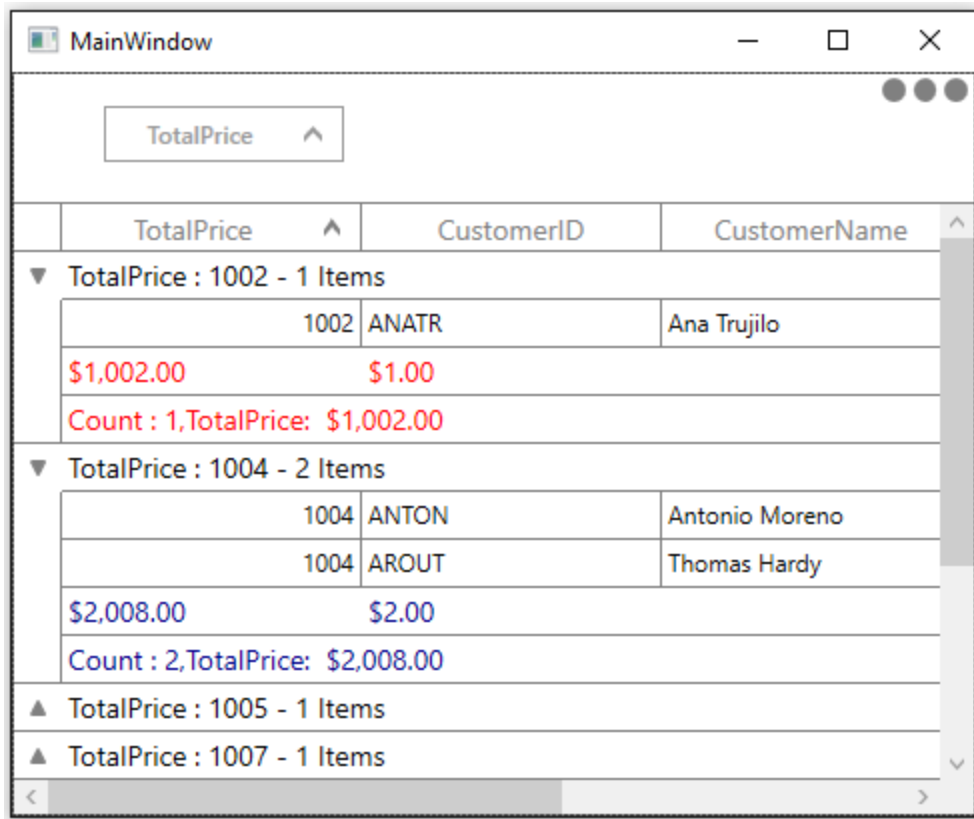
```
<Window.Resources>
<local:ColorConverter x:Key="converter"/>
<Style TargetType="syncfusion:GridGroupSummaryCell">
<Setter Property="Foreground" Value="{Binding Converter={StaticResource
converter}}"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.GroupSummaryRows>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
MappingName="CustomerID"
```

```
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
<syncfusion:GridSummaryRow Title="TotalPrice: {totalPrice}"
ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="totalPrice"
Format="{Sum:c}" />
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>
```

## C#

```
public class ColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var summaryValue = (value as SummaryRecordEntry).SummaryValues[0];
        var aggregateValue = summaryValue.AggregateValues.ElementAt(0);
        var calculatedValue = aggregateValue.Value;
        //custom condition is checked.
        if ((double)calculatedValue < 1500)
            return new SolidColorBrush(Colors.Red);
        return new SolidColorBrush(Colors.DarkBlue);
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Here, group summary cells are customized based on **TotalPrice** summary value.



#### Conditional styling of group summary cell using style selector

The appearance of group summary cell can be customized conditionally based on summary value by setting [SfDataGrid.GroupSummaryCellStyleSelector](#) and you can get the container as [GridGroupSummaryCell](#) in StyleSelector.

Here, group summary cells are customized based on summary values whether it's positive or negative.

#### XML

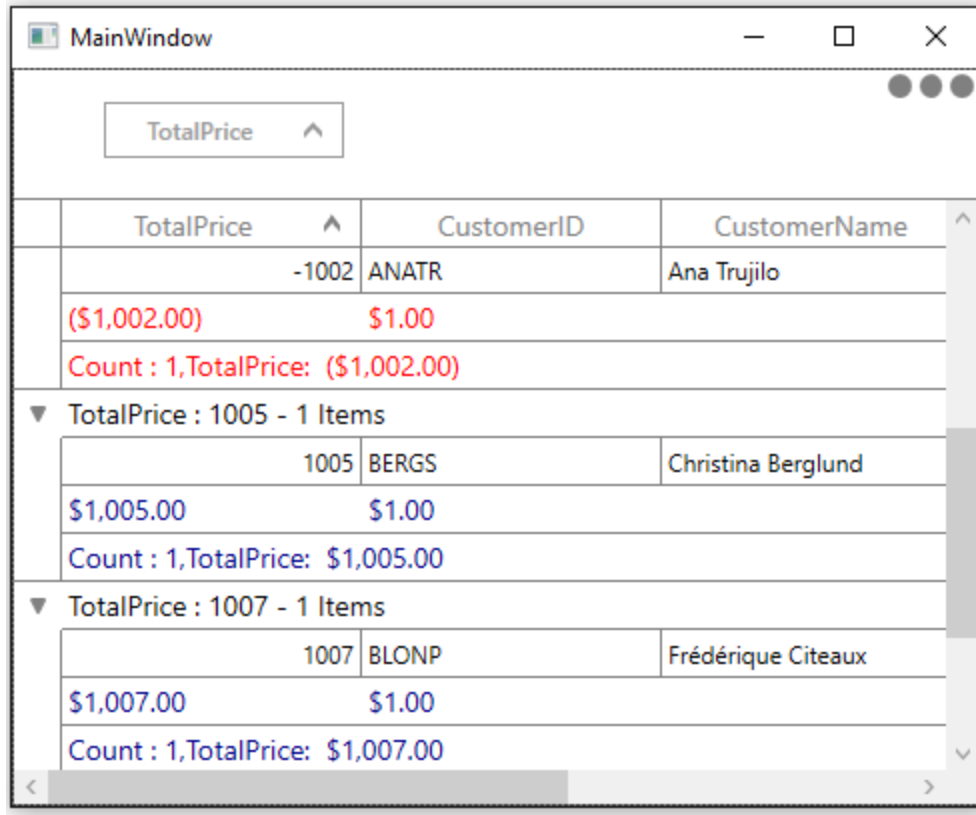
```
<Application.Resources>
<Style TargetType="syncfusion:GridGroupSummaryCell"
x:Key="customGroupSummary">
<Setter Property="Foreground" Value="DarkBlue"/>
</Style>
<Style TargetType="syncfusion:GridGroupSummaryCell"
x:Key="customGroupSummary1">
<Setter Property="Background" Value="Red"/>
</Style>
</Application.Resources>
<Window.Resources>
<local:SelectorClass x:Key="styleSelector"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
GroupSummaryCellStyleSelector="{StaticResource styleSelector}"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.GroupSummaryRows>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
```

```
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
MappingName="CustomerID"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
<syncfusion:GridSummaryRow Title="TotalPrice: {totalPrice}"
ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="totalPrice"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>
```

## C#

```
public class SelectorClass : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var summaryValue = (item as SummaryRecordEntry).SummaryValues[0];
        var aggregateValue = summaryValue.AggregateValues.ElementAt(0);
        var calculatedValue = aggregateValue.Value;
        //custom condition is checked.
        if ((double)calculatedValue < 0)
            return App.Current.Resources["customGroupSummary1"] as Style;
        return App.Current.Resources["customGroupSummary"] as Style;
    }
}
```

Here, group summary cells are customized based on **TotalPrice** summary value.



*Conditional styling of group summary cell based on column*

The group summary cells can be conditionally customized based on summary column.

Here, group summary cells are customized based on **TotalPrice** summary column.

#### XML

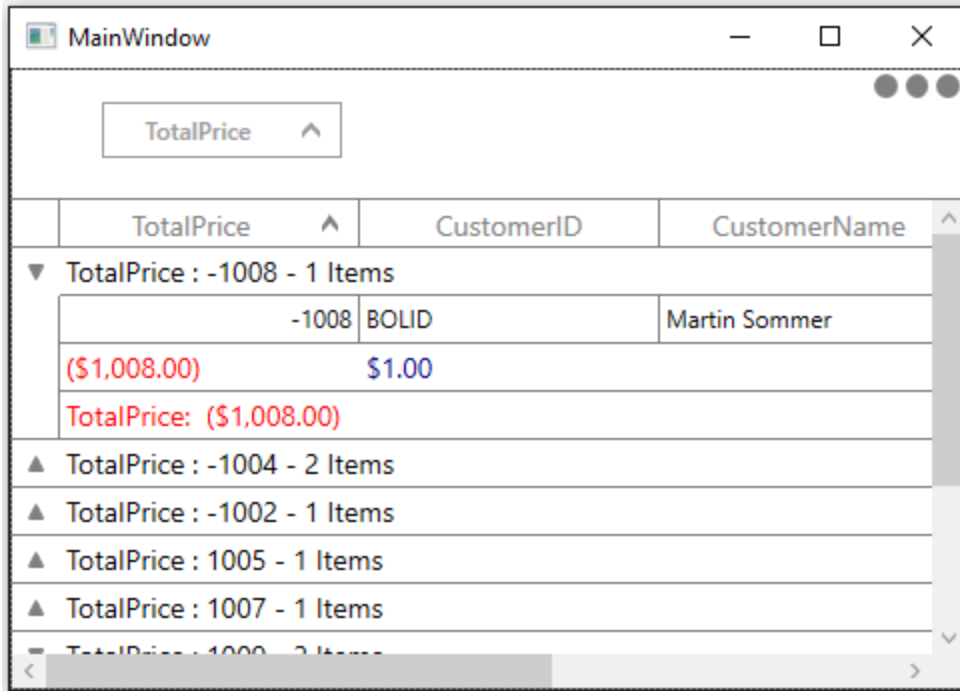
```
<Application.Resources>
<Style TargetType="syncfusion:GridGroupSummaryCell"
x:Key="customGroupSummary">
<Setter Property="Foreground" Value="DarkBlue"/>
</Style>
<Style TargetType="syncfusion:GridGroupSummaryCell"
x:Key="customGroupSummary1">
<Setter Property="Background" Value="Red"/>
</Style>
</Application.Resources>
<Window.Resources>
<local:SelectorClass x:Key="styleSelector"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
GroupSummaryCellStyleSelector="{StaticResource styleSelector}"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.GroupSummaryRows>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}" />
</syncfusion:GridSummaryColumn>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>
```



```
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
MappingName="CustomerID"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
<syncfusion:GridSummaryRow Title="TotalPrice: {totalPrice}"
ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="totalPrice"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>
```

## C#

```
public class SelectorClass : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var cell = container as GridGroupSummaryCell;
        if (cell.ColumnBase.GridColumn.MappingName == "TotalPrice")
        {
            var summaryValue = (item as SummaryRecordEntry).SummaryValues[0];
            var aggregateValue = summaryValue.AggregateValues.ElementAt(0);
            var calculatedValue = aggregateValue.Value;
            //custom condition is checked.
            if (aggregateValue.Key != "Count" && (double)calculatedValue < 0)
            return App.Current.Resources["customGroupSummary1"] as Style;
        }
        return App.Current.Resources["customGroupSummary"] as Style;
    }
}
```



### Group summary row style

Group summary row can be customized conditionally by getting particular summary value from [SummaryValues](#) through converter or style selector. Likewise, you can also customize the group summary row based on various properties exposed in [GridSummaryRow](#) (example: [ShowSummaryInRow](#) property).

### Conditional styling of group summary row using converter

The appearance of group summary row can be customized conditionally based on summary value by using 'converter', where converter returns the value based on summary value.

### XML

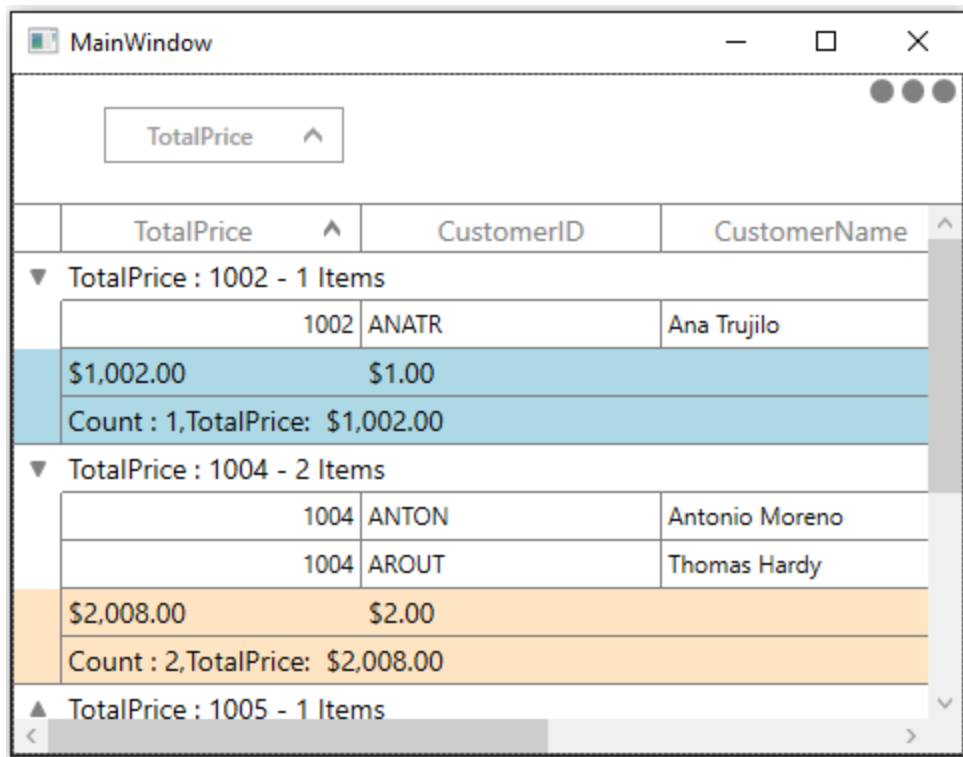
```
<Window.Resources>
<local:ColorConverter x:Key="converter"/>
<Style TargetType="syncfusion:GroupSummaryRowControl">
<Setter Property="Background" Value="{Binding Converter={StaticResource
converter}}"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowGroupDropArea="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.GroupSummaryRows>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
MappingName="CustomerID"
```

```
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
<syncfusion:GridSummaryRow Title="TotalPrice: {totalPrice}"
ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="totalPrice"
Format="{Sum:c}" />
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>
```

## C#

```
public class ColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var summaryValue = (value as SummaryRecordEntry).SummaryValues[0];
        var aggregateValue = summaryValue.AggregateValues.ElementAt(0);
        var calculatedValue = aggregateValue.Value;
        //custom condition is checked.
        if (aggregateValue.Key != "Count" && (double)calculatedValue < 1500)
            return new SolidColorBrush(Colors.LightBlue);
        return new SolidColorBrush(Colors.Bisque);
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Here, group summary rows are customized based on **TotalPrice** summary value.



#### Conditional styling of group summary row using style selector

The appearance of group summary row can be customized conditionally based on summary value by setting [SfDataGrid.GroupSummaryRowStyleSelector](#) and you can get the container as `GridGroupSummaryRowControl` in `StyleSelector`.

#### XML

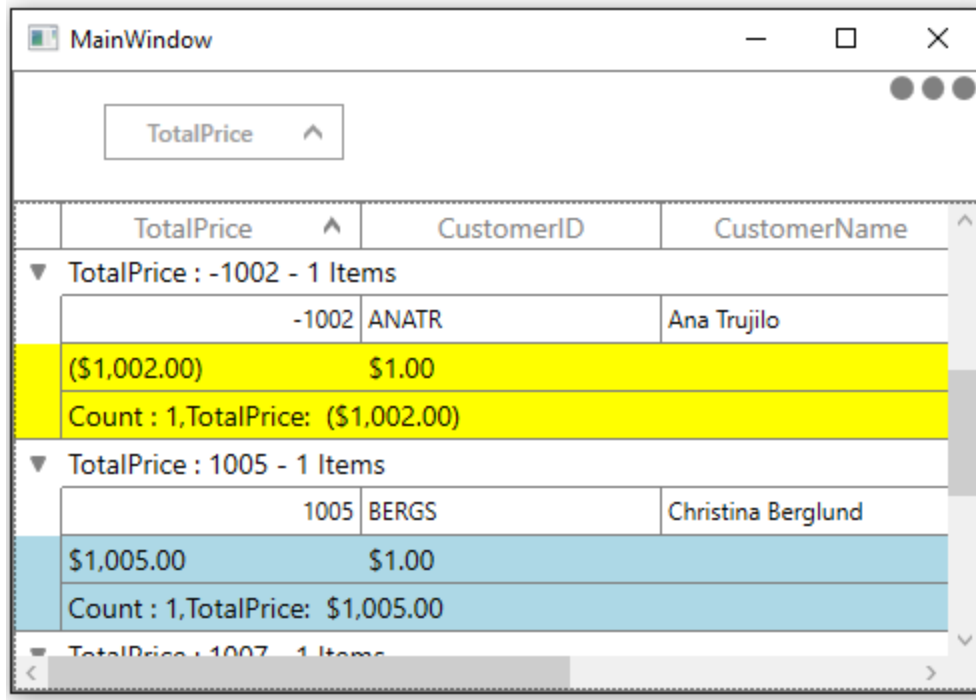
```
<Application.Resources>
<Style TargetType="syncfusion:GroupSummaryRowControl"
x:Key="customGroupSummary ">
<Setter Property="Background" Value="LightBlue"/>
</Style>
<Style TargetType="syncfusion:GroupSummaryRowControl"
x:Key="customGroupSummary1">
<Setter Property="Background" Value="Yellow"/>
</Style>
</Application.Resources>
<Window.Resources>
<local:SelectorClass x:Key="styleSelector"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
GroupSummaryRowStyleSelector="{StaticResource styleSelector}"
ShowGroupDropArea="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.GroupSummaryRows>
<syncfusion:GridSummaryRow ShowSummaryInRow="False">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow>
</syncfusion:GridSummaryRows>
</syncfusion:SfDataGrid>
```

```
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
MappingName="CustomerID"
SummaryType="CountAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
<syncfusion:GridSummaryRow Title="TotalPrice: {totalPrice}"
ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="totalPrice"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.GroupSummaryRows>
</syncfusion:SfDataGrid>
```

## C#

```
public class SelectorClass : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var summaryRecordEntry = (item as SpannedDataRow).RowData;
        var summaryValue = (summaryRecordEntry as
        SummaryRecordEntry).SummaryValues[0];
        var aggregateValue = summaryValue.AggregateValues.ElementAt(0);
        var calculatedValue = aggregateValue.Value;
        //custom condition is checked.
        if ( (double)calculatedValue < 0)
        return App.Current.Resources["customGroupSummary1"] as Style;
        return App.Current.Resources["customGroupSummary"] as Style;
    }
}
```

Here, group summary rows are customized based on **TotalPrice** summary value whether it's positive or negative.



### Table summary cell

Table summary cells can be customized conditionally by getting particular summary value from [SummaryValues](#) through [converter](#) or [style selector](#). Likewise, you can also customize the table summary cell based on various properties exposed in [GridSummaryRow](#) (example: [ShowSummaryInRow](#) property).

### Conditional styling of table summary cells using converter

The appearance of table summary cell can be customized conditionally based on summary value using [converter](#), where converter returns the value based on summary value.

### XML

```
<Window.Resources>
<local:ColorConverter x:Key="converter"/>
<Style TargetType="syncfusion:GridTableSummaryCell">
<Setter Property="Foreground" Value="{Binding Converter={StaticResource
converter}}"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowRowHeader="True"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow Position="Top" ShowSummaryInRow="False">
<syncfusion:GridTableSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
MappingName="CustomerID"
```

```
SummaryType="CountAggregate" />
</syncfusion:GridTableSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
<syncfusion:GridSummaryRow Title="Count : {count}, Total Price :
{totalPrice}" ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="count"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="totalPrice"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>
```

## C#

```
public class ColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var summaryValue = (value as SummaryRecordEntry).SummaryValues[0];
        var aggregateValue = summaryValue.AggregateValues.ElementAt(0);
        var calculatedValue = aggregateValue.Value;
        //custom condition is checked.
        if (aggregateValue.Key != "Count" && (double)calculatedValue < 1500)
            return new SolidColorBrush(Colors.Red);
        return new SolidColorBrush(Colors.LightBlue);
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Here, table summary cells are customized based on **TotalPrice** summary value.

TotalPrice	CustomerID	CustomerName	
\$12.00	\$8.00		
-1002	ANATR	Ana Trujilo	Mexico
-1004	ANTON	Antonio Moreno	Mexico
-1004	AROUT	Thomas Hardy	UK
1005	BERGS	Christina Berglund	Sweden
1009	BLAUS	Hanna Moos	Germany
1007	BLONP	Frédérique Citeaux	France
-1008	BOLID	Martin Sommer	Spain
1009	BONAP	Laurence Lebihan	France
Count : 8, Total Price : \$12.00			

#### Conditional styling of table summary cell using style selector

The appearance of table summary cell can be customized conditionally based on summary value by setting [SfDataGrid.TableSummaryCellStyleSelector](#) and you can get the container as [GridTableSummaryCell](#) in StyleSelector.

#### XML

```
<Application.Resources>
<Style TargetType="syncfusion:GridTableSummaryCell"
x:Key="customTableSummary">
<Setter Property="Foreground" Value="Red"/>
</Style>
<Style TargetType="syncfusion:GridTableSummaryCell"
x:Key="customTableSummary1">
<Setter Property="Foreground" Value="DarkBlue"/>
</Style>
</Application.Resources>
<Window.Resources>
<local:SelectorClass x:Key="styleSelector"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowRowHeader="True"
ItemsSource="{Binding Orders}"
TableSummaryCellStyleSelector="{StaticResource styleSelector}">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow Position="Top" ShowSummaryInRow="False">
<syncfusion:GridTableSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
```

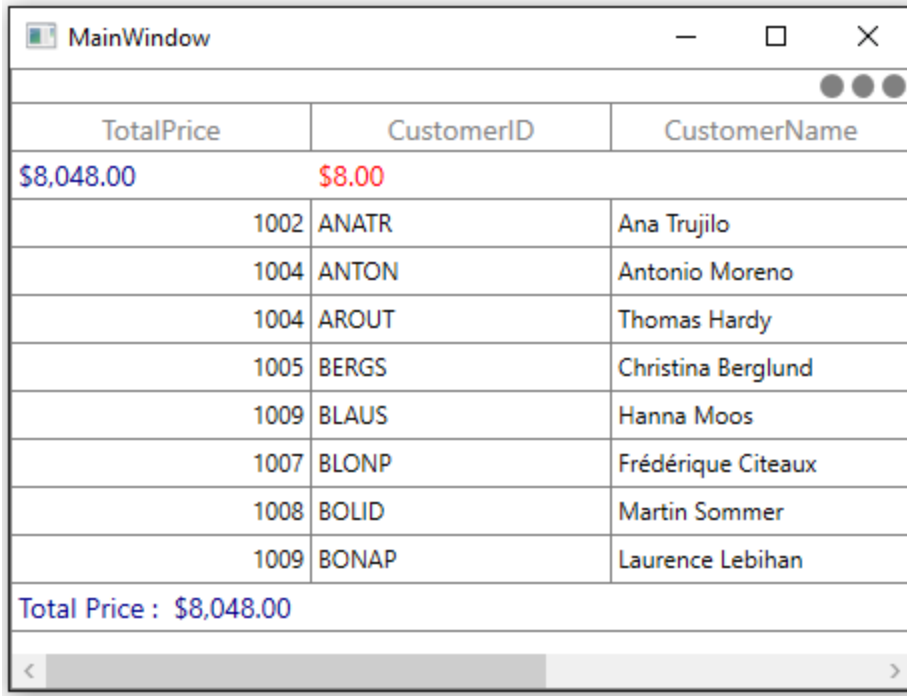


```
MappingName="CustomerID"
SummaryType="CountAggregate" />
</syncfusion:GridTableSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
<syncfusion:GridSummaryRow Title="Total Price : {totalPrice}"
ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="totalPrice"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>
```

## C#

```
public class SelectorClass : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var summaryValue = (item as SummaryRecordEntry).SummaryValues[0];
        var aggregateValue = summaryValue.AggregateValues.ElementAt(0);
        var calculatedValue = aggregateValue.Value;
        var cell = container as GridTableSummaryCell;
        //custom condition is checked.
        if ((double)calculatedValue < 8500 && cell.ColumnBase.GridColumn.MappingName
        == "TotalPrice")
        return App.Current.Resources["customTableSummary"] as Style;
        return App.Current.Resources["customTableSummary1"] as Style;
    }
}
```

Here, table summary cells are customized based on **TotalPrice** summary value.



TotalPrice	CustomerID	CustomerName
\$8,048.00	\$8.00	
1002	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1009	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
Total Price : \$8,048.00		

#### Conditional styling of table summary cell based on column

The table summary cells can be conditionally customized based on summary column.

Here, table summary cells are customized based on **TotalPrice** summary column.

#### XML

```
<Application.Resources>
<Style TargetType="syncfusion:GridTableSummaryCell"
x:Key="customTableSummary">
<Setter Property="Foreground" Value="Red"/>
</Style>
<Style TargetType="syncfusion:GridTableSummaryCell"
x:Key="customTableSummary1">
<Setter Property="Foreground" Value="DarkBlue"/>
</Style>
</Application.Resources>
<Window.Resources>
<local:SelectorClass x:Key="styleSelector"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowRowHeader="True"
ItemsSource="{Binding Orders}"
TableSummaryCellStyleSelector="{StaticResource styleSelector}">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow Position="Top" ShowSummaryInRow="False">
<syncfusion:GridTableSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
```

```

MappingName="CustomerID"
SummaryType="CountAggregate" />
</syncfusion:GridTableSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
<syncfusion:GridSummaryRow Title="Total Price : {totalPrice}"
ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="totalPrice"
Format="{Sum:c}" />
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>

```

**C#**

```

public class SelectorClass : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var cell = container as GridTableSummaryCell;
        // column name is checked.
        if (cell.ColumnBase.GridColumn.MappingName == "TotalPrice")
            return App.Current.Resources["TableSummaryStyle1"] as Style;
        return App.Current.Resources["TableSummaryStyle2"] as Style;
    }
}

```

TotalPrice	CustomerID	CustomerName
\$12.00	\$8.00	
-1002	ANATR	Ana Trujilo
-1004	ANTON	Antonio Moreno
-1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1009	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
-1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
Total Price : \$12.00		

### Table summary row style

Table summary rows can be customized conditionally by getting particular summary value from [SummaryValues](#) through converter or style selector. Likewise, you can also customize the table summary row based on various properties exposed in [GridSummaryRow](#) (example: [ShowSummaryInRow](#) property).

#### *Conditional styling of table summary row using converter*

The appearance of table summary row can be customized conditionally based on summary value using `converter`, where converter returns the value based on summary value.

#### XML

```
<Window.Resources>
<local:ColorConverter x:Key="converter"/>
<Style TargetType="syncfusion:TableSummaryRowControl">
<Setter Property="Background" Value="{Binding Converter={StaticResource
converter}}"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowRowHeader="True"
ItemsSource="{Binding Orders}">
```

#### C#

```
public class ColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var summaryValue = (value as SummaryRecordEntry).SummaryValues[0];
        var aggregateValue = summaryValue.AggregateValues.ElementAt(0);
        var calculatedValue = aggregateValue.Value;
        //custom condition is checked.
        if (aggregateValue.Key != "Count" && (double)calculatedValue < 1500)
            return new SolidColorBrush(Colors.Bisque);
        return new SolidColorBrush(Colors.LightBlue);
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Here, table summary rows are customized based on `TotalPrice` summary value.

TotalPrice	CustomerID	CustomerName
\$8,048.00	\$8.00	
	1002	ANATR
	1004	ANTON
	1004	AROUT
	1005	BERGS
	1009	BLAUS
	1007	BLONP
	1008	BOLID
	1009	BONAP
Count : 8, Total Price : \$8,048.00		

#### Conditional styling of table summary row using style selector

The appearance of table summary row can be customized conditionally based on summary value by setting [SfDataGrid.TableSummaryRowStyleSelector](#) and you can get the container as `GridTableSummaryRowControl` in `StyleSelector`.

#### XML

```
<Application.Resources>
<Style TargetType="syncfusion:TableSummaryRowControl"
x:Key="tableSummaryRowStyle">
<Setter Property="Background" Value="Bisque"/>
</Style>
<Style TargetType="syncfusion:TableSummaryRowControl"
x:Key="tableSummaryRowStyle1">
<Setter Property="Background" Value="LightBlue"/>
</Style>
</Application.Resources>
<Window.Resources>
<local:SelectorClass x:Key="styleSelector"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ShowRowHeader="True"
ItemsSource="{Binding Orders}"
TableSummaryRowStyleSelector="{StaticResource styleSelector}" >
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow Position="Top" ShowSummaryInRow="False">
<syncfusion:GridTableSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}"
```

```

MappingName="CustomerID"
SummaryType="CountAggregate" />
</syncfusion:GridTableSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
<syncfusion:GridSummaryRow Title="Count : {count}, Total Price :
{totalPrice}" ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="count"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="totalPrice"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>

```

**C#**

```

public class SelectorClass : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var summaryRecordEntry = (item as SpannedDataRow).RowData;
        var summaryValue = (summaryRecordEntry as
        SummaryRecordEntry).SummaryValues[0];
        var aggregateValue = summaryValue.AggregateValues.ElementAt(0);
        var calculatedValue = aggregateValue.Value;
        //custom condition is checked.
        if (aggregateValue.Key != "Count" && (double)calculatedValue < 0)
        return App.Current.Resources["customTableSummary"] as Style;
        return App.Current.Resources["customTableSummary1"] as Style;
    }
}

```

Here, table summary rows are customized based on **TotalPrice** summary value.

	TotalPrice	CustomerID	CustomerName
	\$8,048.00	\$8.00	
	1002	ANATR	Ana Trujilo
	1004	ANTON	Antonio Moreno
	1004	AROUT	Thomas Hardy
	1005	BERGS	Christina Berglund
	1009	BLAUS	Hanna Moos
	1007	BLONP	Frédérique Citeaux
	1008	BOLID	Martin Sommer
	1009	BONAP	Laurence Lebihan
Count : 8, Total Price : \$8,048.00			

Table summary cell alignment based on column

The alignment of summary cells can be customized conditionally based on summary column.

Here, horizontal alignment of table summary cells are changed based on column name. Likewise, you can change the horizontal alignment of group, caption summary cells.

#### XML

```
<Application.Resources>
<Style x:Key="TableSummaryStyle1"
TargetType="syncfusion:GridTableSummaryCell">
<Setter Property="HorizontalContentAlignment" Value="Center" />
</Style>
<Style x:Key="TableSummaryStyle2"
TargetType="syncfusion:GridTableSummaryCell">
<Setter Property="HorizontalContentAlignment" Value="Right" />
</Style>
</Application.Resources>
<Window.Resources>
<local:TableSummaryStyleSelector x:Key="tableSummaryStyleSelector" />
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
TableSummaryCellStyleSelector="{StaticResource tableSummaryStyleSelector}">
<syncfusion:SfDataGrid.TableSummaryRows>
<syncfusion:GridTableSummaryRow Position="Top" ShowSummaryInRow="False">
<syncfusion:GridTableSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="price"
Format="{Sum:c}"
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="customerID"
Format="{Count:c}" />
</syncfusion:GridTableSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>
```

```
MappingName="CustomerID"
SummaryType="CountAggregate" />
</syncfusion:GridTableSummaryRow.SummaryColumns>
</syncfusion:GridTableSummaryRow>
<syncfusion:GridSummaryRow Title="Count : {count}, Total Price :
{totalPrice}" ShowSummaryInRow="True">
<syncfusion:GridSummaryRow.SummaryColumns>
<syncfusion:GridSummaryColumn Name="count"
Format="{Sum:c}" />
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
<syncfusion:GridSummaryColumn Name="totalPrice"
Format="{Sum:c}" />
MappingName="TotalPrice"
SummaryType="DoubleAggregate" />
</syncfusion:GridSummaryRow.SummaryColumns>
</syncfusion:GridSummaryRow>
</syncfusion:SfDataGrid.TableSummaryRows>
</syncfusion:SfDataGrid>
```

## C#

```
public class TableSummaryStyleSelector : StyleSelector
{
    public override Style SelectStyle(object item, DependencyObject container)
    {
        var cell = container as GridTableSummaryCell;
        // Horizontal Alignments changed based on MappingName
        if (cell.ColumnBase.GridColumn.MappingName == "TotalPrice")
            return App.Current.Resources["TableSummaryStyle1"] as Style;
        return App.Current.Resources["TableSummaryStyle2"] as Style;
    }
}
```

Here, horizontal alignment of **TotalPrice** column alone left, other column horizontal alignment are changed into right.



TotalPrice	CustomerID	CustomerName
\$8,048.00	\$8.00	
1002	ANATR	Ana Trujilo
1004	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1009	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
Count : 8, Total Price : \$8,048.00		

### Row header style

The appearance of row header ([GridRowHeaderCell](#)) can be customized conditionally by changing its property value based on 'cell value' or 'data object' by using [converter](#), where converter returns the value based on Underlying record.

Here, row headers are customized based on [AmountPaid](#) property of underlying record.

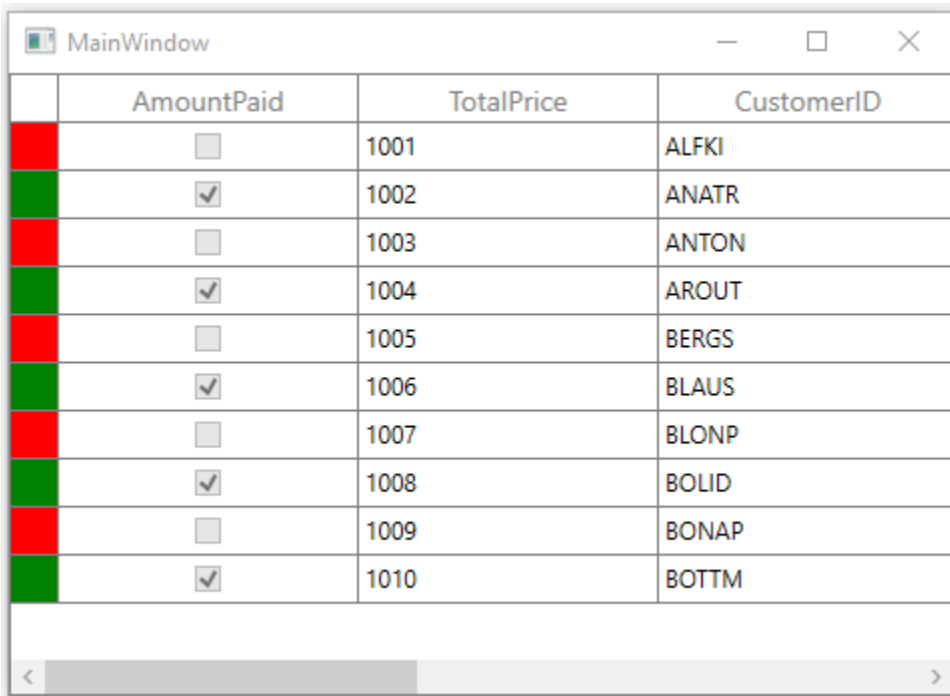
### XML

```
<Window.Resources>
<local:ColorConverter x:Key="converter" />
<Style TargetType="syncfusion:GridRowHeaderCell">
<Setter Property="Background" Value="{Binding Converter={StaticResource
converter}}" />
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
ShowRowHeader="True">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridCheckBoxColumn MappingName="AmountPaid" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#

```
public class ColorConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
{
var data = value as OrderInfo;
```

```
//custom condition is checked.
if (data.AmountPaid)
return Brushes.Green;
else
return Brushes.Red;
}
public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
{
throw new NotImplementedException();
}
}
```



	AmountPaid	TotalPrice	CustomerID
	<input type="checkbox"/>	1001	ALFKI
	<input checked="" type="checkbox"/>	1002	ANATR
	<input type="checkbox"/>	1003	ANTON
	<input checked="" type="checkbox"/>	1004	AROUT
	<input type="checkbox"/>	1005	BERGS
	<input checked="" type="checkbox"/>	1006	BLAUS
	<input type="checkbox"/>	1007	BLONP
	<input checked="" type="checkbox"/>	1008	BOLID
	<input type="checkbox"/>	1009	BONAP
	<input checked="" type="checkbox"/>	1010	BOTTM

### Grid Lines customization in WPF DataGrid (SfDataGrid)

SfDataGrid allows you to customize the grid lines visibility to vertical, horizontal, both or none. To achieve this, use the following properties.

[SfDataGrid.GridLinesVisibility](#): To set the border lines for the cells other than header and stacked header cells.

[SfDataGrid.HeaderLinesVisibility](#): To set the border lines only for header and stacked header cells.

The following are the list of options available to customize grid lines visibility,

- Both
- Vertical
- Horizontal
- None

Record rows

*Both*

The [GridLinesVisibility.Both](#) displays the DataGrid with both horizontal and vertical grid lines. By default GridLinesVisibility value set as Both.

#### XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
<syncfusion:SfDataGrid x:Name="sfDataGrid"
AutoGenerateColumns="True"
GridLinesVisibility="Both"
ItemsSource="{Binding Orders}"/>
```

#### C#

```
this.sfDataGrid.GridLinesVisibility = GridLinesVisibility.Both;
```

Order ID	Customer ID	Customer Name	Ship City	Country
1001	ALFKI	Maria Anders	Berlin	Germany
1002	ANATR	Ana Trujilo	Mexico D.F.	Mexico
1003	ANTON	Antonio Moreno	Mexico D.F.	Mexico
1004	AROUT	Thomas Hardy	London	UK
1005	BERGS	Christina Berglund	Lula	Sweden
1006	BLAUS	Hanna Moos	Mannheim	Germany
1007	BLONP	Frederique Citeaux	Strasbourg	France
1008	BOLID	Martin Sommer	Madrid	Spain
1009	BONAP	Laurence Lebihan	Marseille	France
1010	BOTTM	Elizabeth Lincoln	Tsawassen	Canada

*Horizontal*

The [GridLinesVisibility.Horizontal](#) displays the DataGrid with horizontal grid lines only.

#### XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
<syncfusion:SfDataGrid x:Name="sfDataGrid"
AutoGenerateColumns="True"
GridLinesVisibility="Horizontal"
ItemsSource="{Binding Orders}"/>
```

#### C#

```
this.sfDataGrid.GridLinesVisibility = GridLinesVisibility.Horizontal;
```

Order ID	Customer ID	Customer Name	Ship City	Country
1001	ALFKI	Maria Anders	Berlin	Germany
1002	ANATR	Ana Trujilo	Mexico D.F.	Mexico
1003	ANTON	Antonio Moreno	Mexico D.F.	Mexico
1004	AROUT	Thomas Hardy	London	UK
1005	BERGS	Christina Berglund	Lula	Sweden
1006	BLAUS	Hanna Moos	Mannheim	Germany
1007	BLONP	Frederique Citeaux	Strasbourg	France
1008	BOLID	Martin Sommer	Madrid	Spain
1009	BONAP	Laurence Lebihan	Marseille	France
1010	BOTTM	Elizabeth Lincoln	Tsawassen	Canada

### Vertical

The [GridLinesVisibility.Vertical](#) displays the DataGrid with vertical grid lines only.

### XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
<syncfusion:SfDataGrid x:Name="sfDataGrid"
AutoGenerateColumns="True"
GridLinesVisibility="Vertical"
ItemsSource="{Binding Orders}"/>
```

### C#

```
this.sfDataGrid.GridLinesVisibility = GridLinesVisibility.Vertical;
```

Order ID	Customer ID	Customer Name	Ship City	Country
1001	ALFKI	Maria Anders	Berlin	Germany
1002	ANATR	Ana Trujilo	Mexico D.F.	Mexico
1003	ANTON	Antonio Moreno	Mexico D.F.	Mexico
1004	AROUT	Thomas Hardy	London	UK
1005	BERGS	Christina Berglund	Lula	Sweden
1006	BLAUS	Hanna Moos	Mannheim	Germany
1007	BLONP	Frederique Citeaux	Strasbourg	France
1008	BOLID	Martin Sommer	Madrid	Spain
1009	BONAP	Laurence Lebihan	Marseille	France
1010	BOTTM	Elizabeth Lincoln	Tsawassen	Canada

### None

[GridLinesVisibility.None](#) displays the DataGrid without grid lines.

### XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
<dataGrid:SfDataGrid x:Name="sfDataGrid"
AutoGenerateColumns="True"
GridLinesVisibility="None"
ItemsSource="{Binding OrdersDetails}"/>
</dataGrid:SfDataGrid>
```

**C#**

```
this.sfDataGrid.GridLinesVisibility = GridLinesVisibility.None;
```

Order ID	Customer ID	Customer Name	Ship City	Country
1001	ALFKI	Maria Anders	Berlin	Germany
1002	ANATR	Ana Trujilo	Mexico D.F.	Mexico
1003	ANTON	Antonio Moreno	Mexico D.F.	Mexico
1004	AROUT	Thomas Hardy	London	UK
1005	BERGS	Christina Berglund	Lula	Sweden
1006	BLAUS	Hanna Moos	Mannheim	Germany
1007	BLONP	Frederique Citeaux	Strasbourg	France
1008	BOLID	Martin Sommer	Madrid	Spain
1009	BONAP	Laurence Lebihan	Marseille	France
1010	BOTTM	Elizabeth Lincoln	Tsawassen	Canada

**Header rows**

You can customize the DataGrid header lines visibility by using the [SfDataGrid.HeaderLinesVisibility](#) property. You can also customize the header lines visibility to horizontal, vertical, none or both. By default HeaderLinesVisibility value set as Both.

**XML**

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
<syncfusion:SfDataGrid x:Name="sfDataGrid"
AutoGenerateColumns="True"
HeaderLinesVisibility="Horizontal"
ItemsSource="{Binding Orders}"/>
```

**C#**

```
this.sfDataGrid.HeaderLinesVisibility = GridLinesVisibility.Horizontal;
```

Order ID	Customer ID	Customer Name	Ship City	Country
1001	ALFKI	Maria Anders	Berlin	Germany
1002	ANATR	Ana Trujilo	Mexico D.F.	Mexico
1003	ANTON	Antonio Moreno	Mexico D.F.	Mexico
1004	AROUT	Thomas Hardy	London	UK
1005	BERGS	Christina Berglund	Lula	Sweden
1006	BLAUS	Hanna Moos	Mannheim	Germany
1007	BLONP	Frederique Citeaux	Strasbourg	France
1008	BOLID	Martin Sommer	Madrid	Spain
1009	BONAP	Laurence Lebihan	Marseille	France
1010	BOTTM	Elizabeth Lincoln	Tsawassen	Canada

### Grid lines for Master-Details view

SfDataGrid allows you to customize the grid lines for Master-Details view also like parent DataGrid by changing the grid lines properties in GridViewDefinition.DataGrid.

#### XML

```
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
<syncfusion:SfDataGrid x:Name="sfDataGrid"
AutoGenerateColumns="True"
AutoGenerateRelations="False"
HideEmptyGridViewDefinition="True"
GridLinesVisibility="Horizontal"
HeaderLinesVisibility="Horizontal"
ItemsSource="{Binding Employees}">
<!-- FirstLevelNestedGrid is created here -->
<syncfusion:GridViewDefinition RelationalColumn="Sales">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AutoGenerateColumns="True"
GridLinesVisibility="Horizontal"
HeaderLinesVisibility="Horizontal">
</syncfusion:SfDataGrid>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>
```

#### C#

```
this.sfDataGrid.GridLinesVisibility = GridLinesVisibility.Horizontal;
this.sfDataGrid.HeaderLinesVisibility = GridLinesVisibility.Horizontal;
this.FirstLevelNestedGrid.GridLinesVisibility =
GridLinesVisibility.Horizontal;
this.FirstLevelNestedGrid.HeaderLinesVisibility =
GridLinesVisibility.Horizontal;
```

Employee ID	Order ID	City
1	1001	Berlin
	Order ID	Sales ID
	1001	A00001
	1001	A00002
2	1002	London
	Order ID	Sales ID
	1002	A00003
	1002	A00003
3	1003	Mexico
	Order ID	Sales ID
	1003	A00004
4	1004	France
5	1005	Canada

### Limitations

- Grid lines customization are not supported for RowHeader.

### Themes in WPF DataGrid (SfDataGrid)

SfDataGrid supports various built-in themes. Refer to the below links to apply themes for the SfDataGrid,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

Order Details			Shipment Details		Delivery Status
Order ID	Customer ID	Customer Name	Ship Country	Shipping Date	
Ship Country: Argentina - 20 Items					
10004	11004	Franklin	Argentina	2/13/2014	<input checked="" type="checkbox"/>
10007	11007	Martin	Argentina	11/23/2012	<input type="checkbox"/>
10014	11014	George	Argentina	10/11/2017	<input checked="" type="checkbox"/>
10032	11032	Martin	Argentina	4/16/2016	<input checked="" type="checkbox"/>
10036	11036	Zachary	Argentina	2/9/2011	<input checked="" type="checkbox"/>
10037	11037	James	Argentina	6/22/2002	<input type="checkbox"/>
10038	11038	Thomas	Argentina	3/4/2006	<input checked="" type="checkbox"/>
10042	11042	Rutherford	Argentina	2/13/2008	<input checked="" type="checkbox"/>
10047	11047	James	Argentina	6/12/2006	<input type="checkbox"/>
10048	11048	Abraham	Argentina	4/21/2004	<input checked="" type="checkbox"/>
10050	11050	Grover	Argentina	7/15/2018	<input checked="" type="checkbox"/>
10051	11051	Zachary	Argentina	3/23/2017	<input type="checkbox"/>
10060	11060	Rutherford	Argentina	4/3/2007	<input checked="" type="checkbox"/>
Total Customers: 300					

### Rows in WPF DataGrid (SfDataGrid)

This section explains about various row types in [WPF DataGrid](#) (SfDataGrid).

[StackedHeaderRow](#)[AddNewRow](#)[SummaryRow](#)[UnboundRow](#)

Row Header

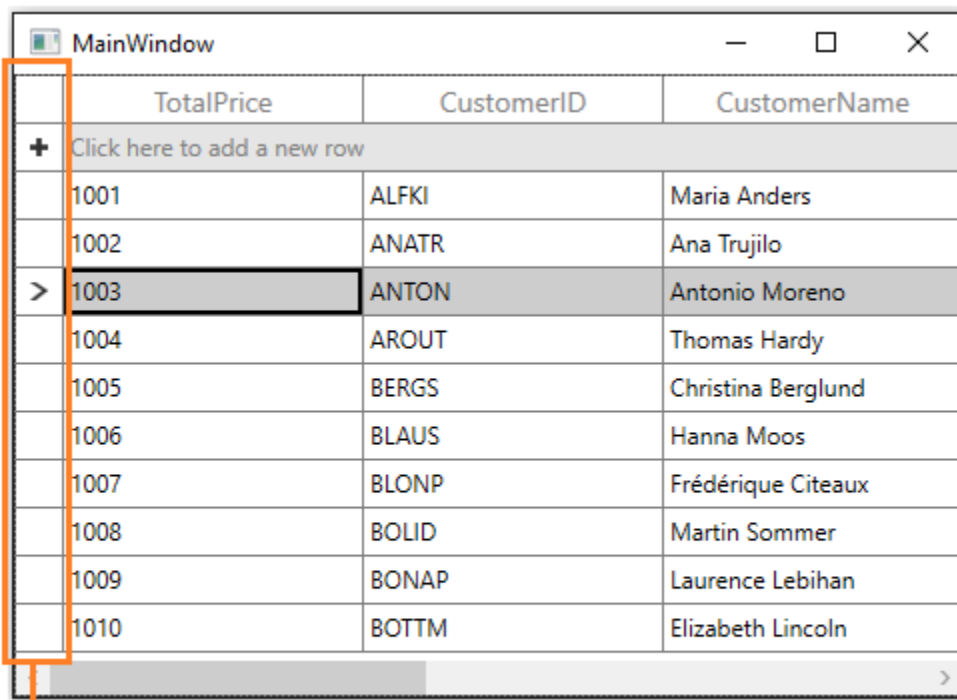
RowHeader is a special column used to indicate the status of row (current row, editing status, errors in row, etc.) which is placed as first cell of each row. You can show or hide the row header by setting [SfDataGrid.ShowRowHeader](#) property.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AddNewRowPosition="Top"
ItemsSource="{Binding Orders}"
ShowRowHeader="True">
```

**C#**

```
dataGrid.ShowRowHeader = true;
```



	TotalPrice	CustomerID	CustomerName
+ Click here to add a new row			
1001		ALFKI	Maria Anders
1002		ANATR	Ana Trujilo
> 1003		ANTON	Antonio Moreno
1004		AROUT	Thomas Hardy
1005		BERGS	Christina Berglund
1006		BLAUS	Hanna Moos
1007		BLONP	Frédérique Citeaux
1008		BOLID	Martin Sommer
1009		BONAP	Laurence Lebihan
1010		BOTTM	Elizabeth Lincoln

Row Header






See also.

[ShowRowIndex in RowHeader](#)



## Customizing RowHeader based on record

### Row indicators and its description

Row Indicator	Description
	Denotes the row which has current cell or selected item.
	Denotes row is being edited.
	Denotes row is AddNewRow.
	Denotes the row has errors.
	Denotes that the current row which has errors.

### Row header width

You can change the width of the row header by setting [SfDataGrid.RowHeaderWidth](#) property.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    ShowRowHeader="True"
    RowHeaderWidth="50"
    ItemsSource="{Binding Orders}" />
```

#### C#

```
dataGrid.RowHeaderWidth = 50;
```

### Display the row index in row header

You can display the corresponding row index in each row header, by customizing the `ControlTemplate` of `GridRowHeaderCell`. You have to bind the `RowIndex` property to `TextBlock.Text` like the below code example.

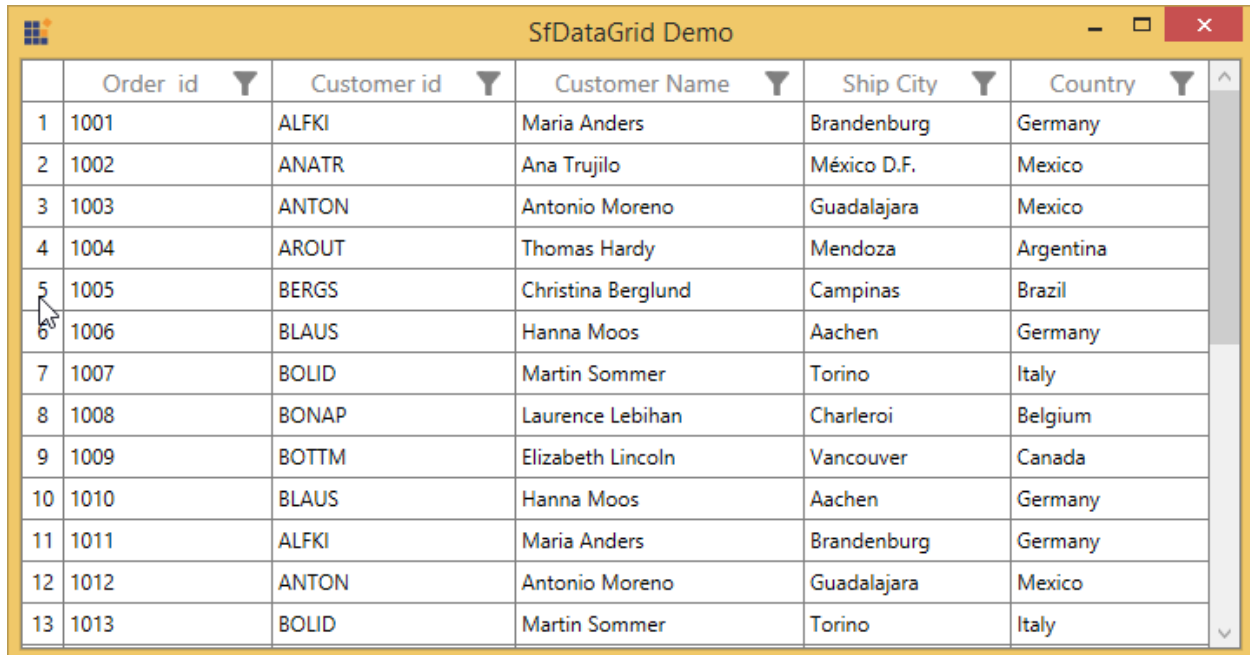
#### XML

```
<Style TargetType="syncfusion:GridRowHeaderCell">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="syncfusion:GridRowHeaderCell">
                <Border x:Name="PART_RowHeaderCellBorder"
                    Background="{TemplateBinding Background}"
                    BorderBrush="{TemplateBinding BorderBrush}"
                    BorderThickness="{TemplateBinding BorderThickness}">
                    <Grid>
                        <TextBlock HorizontalAlignment="Center"
                            VerticalAlignment="Center"
                            Text="{Binding RowIndex,RelativeSource={RelativeSource TemplatedParent}}"
                            TextAlignment="Center" />
                    </Grid>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

```

</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```



	Order id	Customer id	Customer Name	Ship City	Country
1	1001	ALFKI	Maria Anders	Brandenburg	Germany
2	1002	ANATR	Ana Trujilo	México D.F.	Mexico
3	1003	ANTON	Antonio Moreno	Guadalajara	Mexico
4	1004	AROUT	Thomas Hardy	Mendoza	Argentina
5	1005	BERGS	Christina Berglund	Campinas	Brazil
6	1006	BLAUS	Hanna Moos	Aachen	Germany
7	1007	BOLID	Martin Sommer	Torino	Italy
8	1008	BONAP	Laurence Lebihan	Charleroi	Belgium
9	1009	BOTTM	Elizabeth Lincoln	Vancouver	Canada
10	1010	BLAUS	Hanna Moos	Aachen	Germany
11	1011	ALFKI	Maria Anders	Brandenburg	Germany
12	1012	ANTON	Antonio Moreno	Guadalajara	Mexico
13	1013	BOLID	Martin Sommer	Torino	Italy

You can get the sample from [here](#).

*Change the current row indicator*

You can change the CurrentRowIndicator in the row header by customizing the control template of GridRowHeaderCell.

#### XML

```

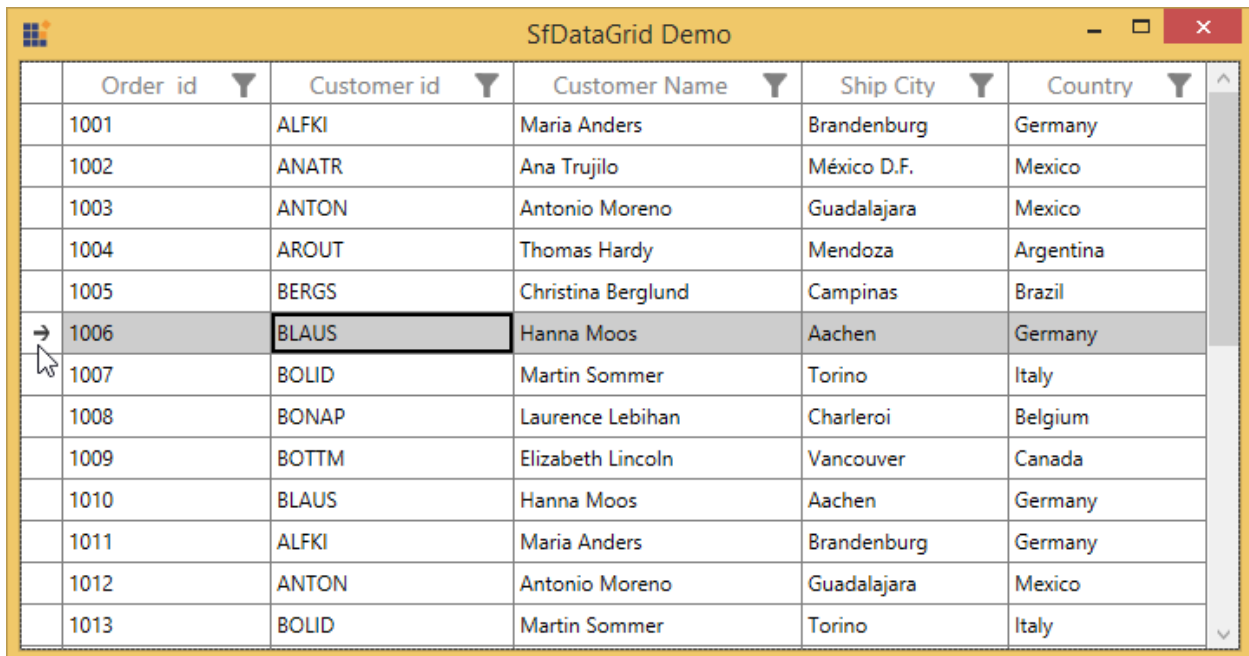
<Style TargetType="syncfusion:GridRowHeaderCell">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="syncfusion:GridRowHeaderCell">
        <Border x:Name="PART_RowHeaderCellBorder"
          Background="{TemplateBinding Background}"
          BorderBrush="{TemplateBinding BorderBrush}"
          BorderThickness="{TemplateBinding BorderThickness}">
          <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="IndicationStates">
              <VisualState x:Name="Normal">
                </VisualState>
              <VisualState x:Name="CurrentRow">
                <Storyboard>
                  <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="PART_RowHeaderIndicator"
                    Storyboard.TargetProperty="Data">
                    <DiscreteObjectKeyFrame KeyTime="0">
                      <DiscreteObjectKeyFrame.Value>

```

```

<Geometry>F1M-218.342,2910.79L-234.066,2926.52 -233.954,2926.63 -
225.428,2926.63 -210.87,2912.07 -206.495,2907.7 -225.313,2888.88 -
234.066,2888.88 -218.342,2904.6 -259.829,2904.6 -259.829,2910.79 -
218.342,2910.79z</Geometry>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Path x:Name="PART_RowHeaderIndicator"
Width="8.146"
Height="8.146"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Fill="#FF303030"
Stretch="Fill">
</Path>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```



	Order id	Customer id	Customer Name	Ship City	Country
	1001	ALFKI	Maria Anders	Brandenburg	Germany
	1002	ANATR	Ana Trujilo	México D.F.	Mexico
	1003	ANTON	Antonio Moreno	Guadalajara	Mexico
	1004	AROUT	Thomas Hardy	Mendoza	Argentina
	1005	BERGS	Christina Berglund	Campinas	Brazil
→	1006	BLAUS	Hanna Moos	Aachen	Germany
	1007	BOLID	Martin Sommer	Torino	Italy
	1008	BONAP	Laurence Lebihan	Charleroi	Belgium
	1009	BOTTM	Elizabeth Lincoln	Vancouver	Canada
	1010	BLAUS	Hanna Moos	Aachen	Germany
	1011	ALFKI	Maria Anders	Brandenburg	Germany
	1012	ANTON	Antonio Moreno	Guadalajara	Mexico
	1013	BOLID	Martin Sommer	Torino	Italy

You can get the sample from [here](#).

#### *Change the background of row header*

You can change the background color of the row header by customizing the style of the `GridRowHeaderCell`. You can change the background color with the converter based on the underlying collection.

#### **XML**

```

<Application.Resources>
<local:CustomConverter x:Key="converter"/>
<!--Customizing the RowHeader style-->
<Style TargetType= "syncfusion:GridRowHeaderCell">
<!--By using converter the Background color is changed based on the business
logic-->
<Setter Property="Background" Value="{Binding Converter={StaticResource
converter }}" />
</Style>
</Application.Resources>

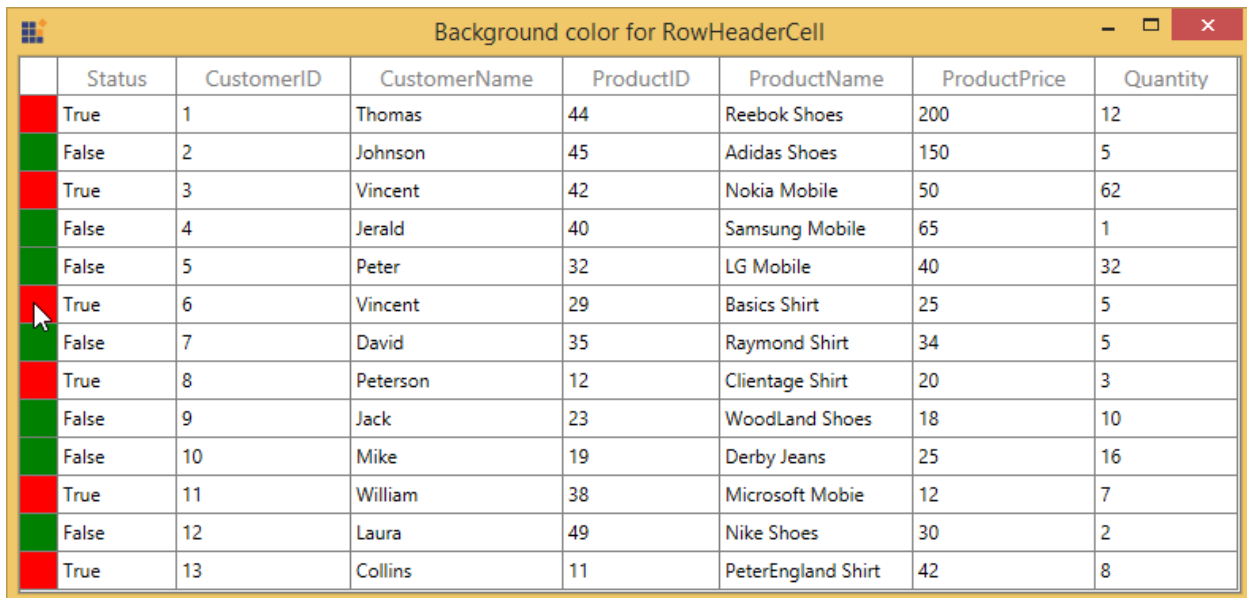
```

## C#

```

public class CustomConverter:IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        //Type casting the value as Data class(Business logic)
        var data = value as Data;
        //The Red color is applied to RowHeader if the status value is true
        otherwise the white color is applied
        if (data.Status == true)
            return Brushes.Red;
        else
            return Brushes.Green;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```



	Status	CustomerID	CustomerName	ProductID	ProductName	ProductPrice	Quantity
	True	1	Thomas	44	Reebok Shoes	200	12
	False	2	Johnson	45	Adidas Shoes	150	5
	True	3	Vincent	42	Nokia Mobile	50	62
	False	4	Jerald	40	Samsung Mobile	65	1
	False	5	Peter	32	LG Mobile	40	32
	True	6	Vincent	29	Basics Shirt	25	5
	False	7	David	35	Raymond Shirt	34	5
	True	8	Peterson	12	Clientage Shirt	20	3
	False	9	Jack	23	WoodLand Shoes	18	10
	False	10	Mike	19	Derby Jeans	25	16
	True	11	William	38	Microsoft Mobie	12	7
	False	12	Laura	49	Nike Shoes	30	2
	True	13	Collins	11	PeterEngland Shirt	42	8

You can get the sample from [here](#).

## Header Row

Header row is present in top of the WPF DataGrid which has column headers in it. Column header describes the caption to identify the column content.

TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1004	AROUT	Maria Anders Jscob Steve C
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

You can change the header row height by setting [SfDataGrid.HeaderRowHeight](#) property.

## Hiding Header row

You can hide the header row by setting [SfDataGrid.HeaderRowHeight](#) as 0 (zero).

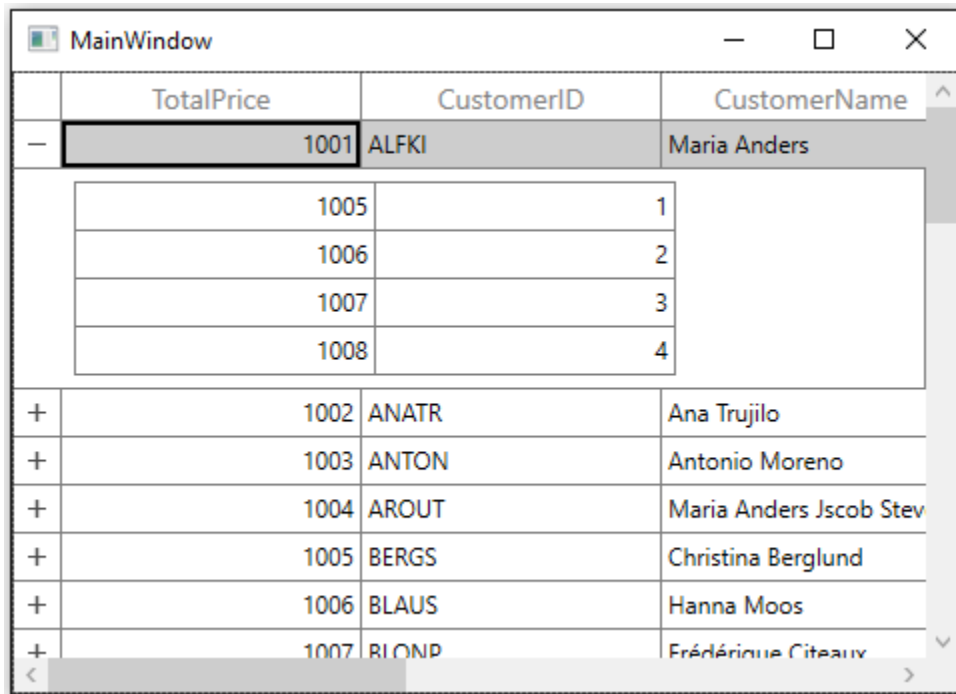
## XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    HeaderRowHeight="0"
    ItemsSource="{Binding Orders}">
```

You can also hide the header row of DetailsViewDataGrid by setting [HeaderRowHeight](#) as 0 (zero) to [ViewDefinition.DataGrid](#).

## XML

```
<syncfusion:SfDataGrid x:Name="dataGrid" ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.DetailsViewDefinition>
        <syncfusion:GridViewDefinition RelationalColumn="Details">
            <syncfusion:GridViewDefinition.DataGrid>
                <syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid" HeaderRowHeight="0" />
            </syncfusion:GridViewDefinition.DataGrid>
        </syncfusion:GridViewDefinition>
    </syncfusion:SfDataGrid.DetailsViewDefinition>
</ syncfusion:SfDataGrid>
```



### Freeze panes

WPF DataGrid provides support to freeze the rows and columns at top and bottom similar to excel. You can freeze the rows and columns by setting following properties,

Property Name	Description
<a href="#">FrozenRowCount</a>	Set the frozen rows count at top of the SfDataGrid.
<a href="#">FooterRowCount</a>	Set the footer rows count at bottom of the SfDataGrid.
<a href="#">FrozenColumnCount</a>	Set the frozen columns count in left side of the SfDataGrid.
<a href="#">FooterColumnCount</a>	Set the frozen columns in right side of the SfDataGrid.

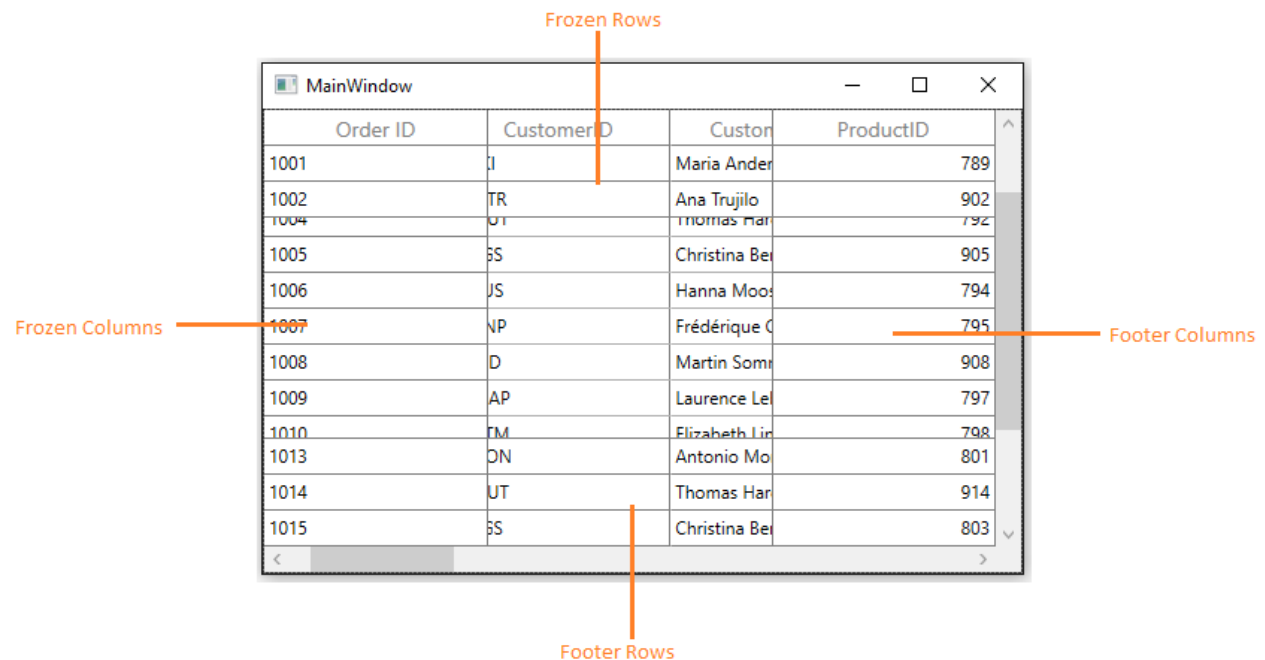
### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
FooterColumnCount="1"
FooterRowCount="3"
FrozenColumnCount="1"
FrozenRowCount="2"
ItemsSource="{Binding Orders}">
```

### C#

```
dataGrid.FooterColumnCount = 1;
```

```
dataGrid.FrozenColumnCount = 1;
dataGrid.FrozenRowCount = 2;
dataGrid.FrozenRowCount = 3;
```



### Differentiate frozen rows from normal rows

You can differentiate the frozen rows and footer rows from normal rows by writing style for [VirtualizingCellsControl](#) and by customizing the **FrozenRow** and **FooterRow** visual states.

### XML

```
<Style TargetType="{x:Type syncfusion:VirtualizingCellsControl}">
  <Setter Property="Background" Value="Transparent">
  <Setter Property="BorderBrush" Value="Gray">
  <Setter Property="BorderThickness" Value="0">
  <Setter Property="IsTabStop" Value="False">
  <Setter Property="FocusVisualStyle" Value="{x:Null}">
  <Setter Property="Template">
  <Setter.Value>
  <ControlTemplate TargetType="{x:Type syncfusion:VirtualizingCellsControl}">
    <Grid>
    <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="BorderStates">
    <VisualState x:Name="NormalRow">
    <VisualState x:Name="FrozenRow">
    <Storyboard BeginTime="0">
    <ThicknessAnimationUsingKeyFrames BeginTime="0"
    Duration="1"
    Storyboard.TargetName="PART_RowBorder"
    Storyboard.TargetProperty="BorderThickness">
    <!-- Border Thickness for Frozen rows -->
    <EasingThicknessKeyFrame KeyTime="0" Value="0, 0, 0, 3"/>
    </ThicknessAnimationUsingKeyFrames>
```

```

</Storyboard>
</VisualState>
<VisualState x:Name="FooterRow">
<Storyboard BeginTime="0">
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_RowBorder"
Storyboard.TargetProperty="BorderThickness">
<!-- Border Thickness for Footer rows -->
<EasingThicknessKeyFrame KeyTime="0" Value="0, 3, 0, 0" />
</ThicknessAnimationUsingKeyFrames>
<ThicknessAnimationUsingKeyFrames BeginTime="0"
Duration="1"
Storyboard.TargetName="PART_RowBorder"
Storyboard.TargetProperty="Margin">
<EasingThicknessKeyFrame KeyTime="0" Value="0, -1, 0, 0" />
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border x:Name="PART_RowBorder"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}" />
<Rectangle x:Name="PART_CurrentFocusRow"
Margin="{TemplateBinding CurrentFocusBorderMargin}"
Stroke="DarkGray"
StrokeDashArray="2,2"
StrokeThickness="1"
Visibility="{TemplateBinding CurrentFocusRowVisibility}" />
<Rectangle Clip="{TemplateBinding RowBackgroundClip}" Fill="{TemplateBinding
Background}" />
<Border Background="{TemplateBinding RowSelectionBrush}"
Clip="{TemplateBinding SelectionBorderClipRect}"
Visibility="{TemplateBinding SelectionBorderVisiblity}" />
<Border Background="{TemplateBinding RowHoverBackgroundBrush}"
BorderBrush="{TemplateBinding RowHoverBackgroundBrush}"
BorderThickness="{TemplateBinding RowHighlightBorderThickness}"
Clip="{TemplateBinding HighlightBorderClipRect}"
SnapsToDevicePixels="True"
Visibility="{TemplateBinding HighlightSelectionBorderVisiblity}" />
<Border BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}">
<ContentPresenter />
<Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```



TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln
1011	ALFKI	Maria Anders
1012	ANATR	Ana Trujilo
1013	ANTON	Antonia Mayra
1014	AROUT	Thomas Hardy
1015	BERGS	Christina Berglund

#### Disable drag and drop between frozen and non-frozen columns

You can disable the drag and drop between frozen and non-frozen columns by handling [QueryColumnDragging](#) event.

Using `Reason` property in [QueryColumnDraggingEventArgs](#), you can cancel the column dropping operation.

In the below code, if the Reason is `QueryColumnDraggingReason.Dropping` and the column is dragged from frozen region to non-frozen region or vice versa, you can cancel the dropping action by setting `e.Cancel` as `true` in the event.

#### C#

```
this.dataGrid.QueryColumnDragging += dataGrid_QueryColumnDragging;
void dataGrid_QueryColumnDragging(object sender,
QueryColumnDraggingEventArgs e)
{
    if (e.Reason == QueryColumnDraggingReason.Dropping)
    {
        //used to get frozen column index from the frozen column count
        var frozenColIndex = dataGrid.FrozenColumnCount +
        this.dataGrid.ResolveToStartColumnIndex();
        //cancels dragging from frozen column to non-frozen column
        if (e.From < frozenColIndex && e.To > frozenColIndex - 1)
            e.Cancel = true;
        // cancels dragging from non-frozen column to frozen column
        if (e.From > frozenColIndex && e.To < frozenColIndex ||
            (e.From == frozenColIndex && e.To < frozenColIndex))
            e.Cancel = true;
    }
}
```

### Limitations

1. When using `DetailsView` with freeze panes, exception will be raised like **DetailsView is not supported with Freeze panes support.**
2. When `AllowFrozenGroupHeaders` is true, frozen rows will not be considered.
3. SfDataGrid has support to freeze the number of rows from top or bottom. There is no support to freeze a specific row.

**Note:** 1. Header rows, table summary rows and row header are frozen regardless of `FrozenRowCount` and `FooterRowCount`.

2. `FrozenRowCount` and `FooterRowCount` values should be less than the number of rows and column visible.

### Row Height Customization in WPF DataGrid (SfDataGrid)

You can change the header row height by setting [SfDataGrid.HeaderRowHeight](#) and the other rows height can be changed by setting [SfDataGrid.RowHeight](#) property.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    RowHeight="30"
    HeaderRowHeight="50"
    ItemsSource="{Binding Orders}" />
```

#### C#

```
this.dataGrid.HeaderRowHeight = 50;
this.dataGrid.RowHeight = 30;
```

You can also change the particular row height using [VisualContainer.RowHeights](#) property.

#### C#

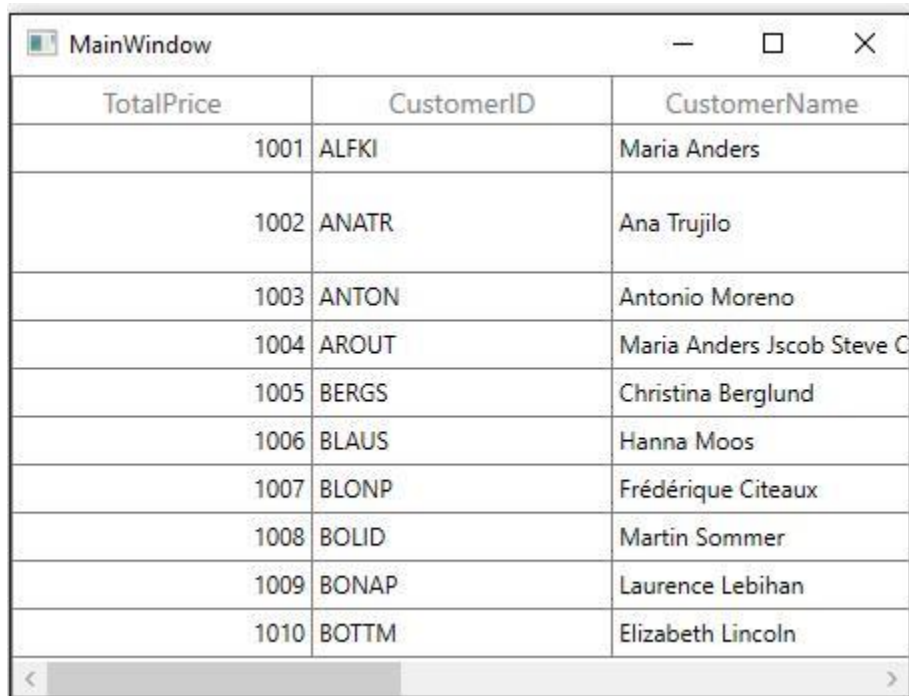
```
using Syncfusion.UI.Xaml.Grid.Helpers;
this.dataGrid.Loaded += dataGrid_Loaded;
void dataGrid_Loaded(object sender, RoutedEventArgs e)
{
    var VisualContainer = this.dataGrid.GetVisualContainer();
    //Set RowHeight to 2'nd row
    VisualContainer.RowHeights[2] = 50;
    VisualContainer.InvalidateMeasure();
}
```

You can also change the row height of particular row using [QueryRowHeight](#) event.

#### C#

```
this.dataGrid.QueryRowHeight += dataGrid_QueryRowHeight;
void dataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    if (e.RowIndex == 2) //Sets Height to the first row.
    {
        e.RowHeight = 50;
    }
}
```

```
{
e.Height = 50;
e.Handled = true;
}
```



TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1004	AROUT	Maria Anders Jscob Steve C
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

### QueryRowHeight event

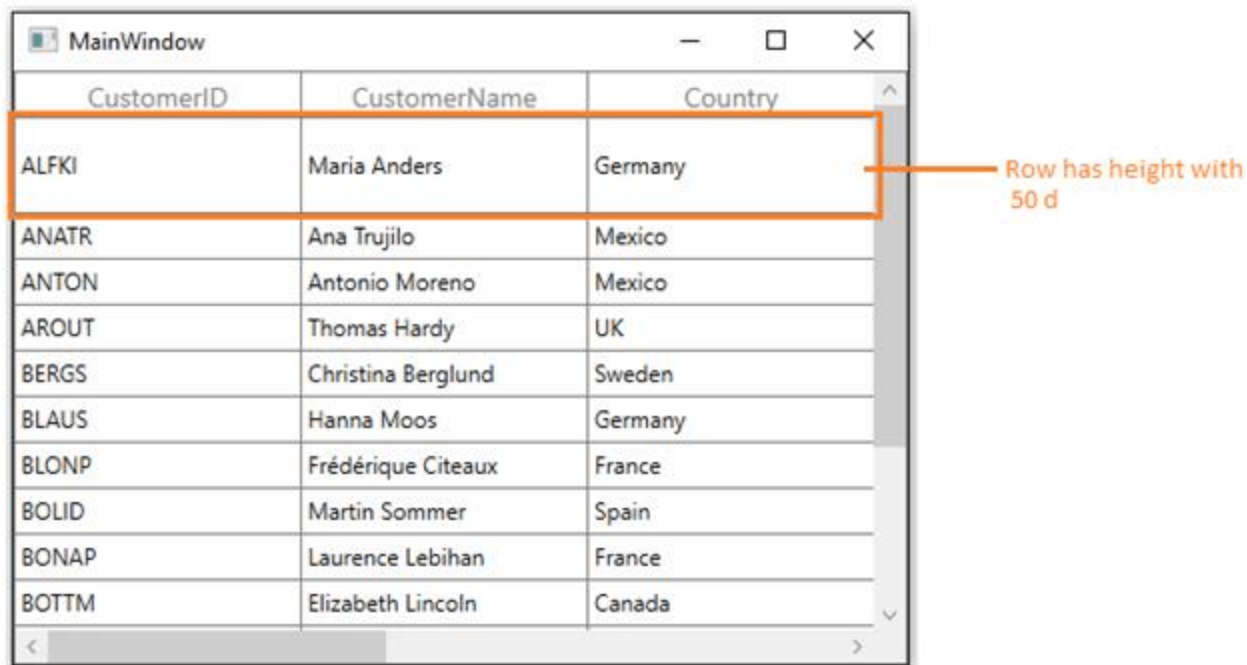
You can change the row height in on-demand based on the row index or row data using [SfDataGrid.QueryRowHeight](#) event.

QueryRowHeight event triggered for each row when it becomes visible. [QueryRowHeightEventArgs](#) provides information to [QueryRowHeight](#) event with following members,

- **RowIndex** – denotes index of the row in SfDataGrid.
- **Height** – Gets or sets the height of the row.
- **Handled** – Gets or sets a value indicating whether the [QueryRowHeight](#) event handled to change height of the row. Its default value is `false`.

### C#

```
this.dataGrid.QueryRowHeight += dataGrid_QueryRowHeight;
void dataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
if (e.RowIndex == 1) //Sets Height to the first row.
{
e.Height = 50;
e.Handled = true;
}
}
```



### Limitations

1. This event is not supported for **DetailsViewGrid**.

### Fit the Row Height based on its content

You can fit the row height based on its content in **QueryRowHeight** event handler using [GetAutoRowHeight](#) method. This improves the readability of the content and it does not affect the loading performance of the SfDataGrid as the **QueryRowHeight** event triggered for rows in on-demand.

**GetAutoRowHeight** method returns **true** when the row height is calculated for record & header rows and returns **false** for other rows. Calculated height based on content set to the **out** parameter and you can assign the calculated height to the **Height** property of [QueryRowHeightEventArgs](#).

Below are the parameter to **GetAutoRowHeight** method,

1. **RowIndex** – denotes the index of row in SfDataGrid.
2. **GridRowSizingOptions** – A class with properties to customize the row height calculation.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid" ItemsSource="{Binding Orders}">
  <syncfusion:SfDataGrid.Columns>
    <syncfusion:GridTextColumn MappingName="CustomerName" TextWrapping="Wrap" />
    <syncfusion:GridTextColumn MappingName="CustomerID" TextWrapping="Wrap" />
    <syncfusion:GridTextColumn MappingName="Country" TextWrapping="Wrap" />
  </syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

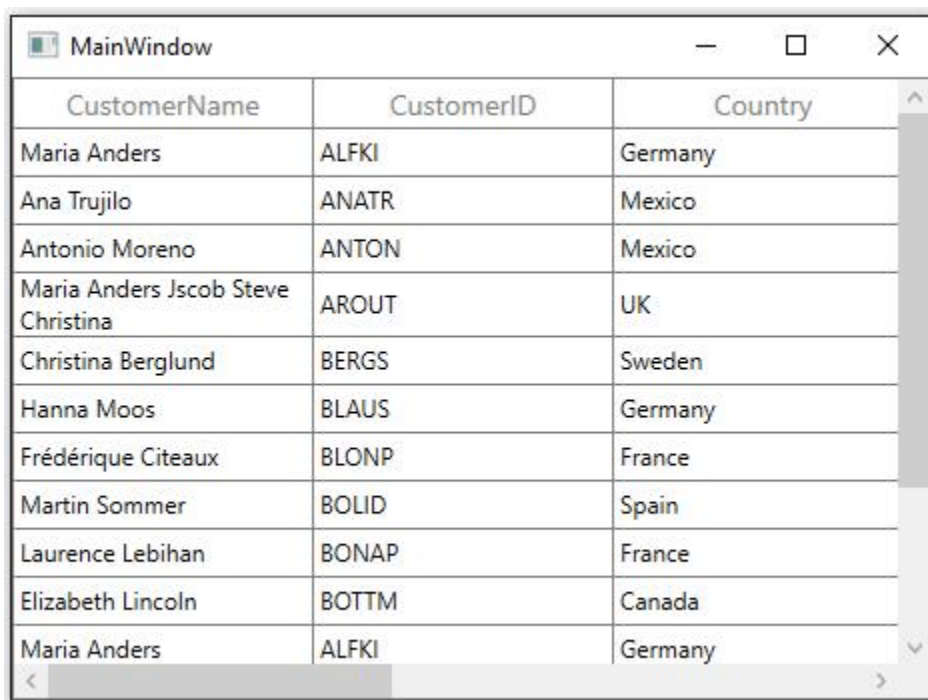
### C#

```

GridRowSizingOptions gridRowResizingOptions = new GridRowSizingOptions();
//To get the calculated height from GetAutoRowHeight method.
double autoHeight;
this.dataGrid.QueryRowHeight += dataGrid_QueryRowHeight;
void dataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    if (this.dataGrid.GridColumnSizer.GetAutoRowHeight(e.RowIndex,
        gridRowResizingOptions, out autoHeight))
    {
        if (autoHeight > 24)
        {
            e.Height = autoHeight;
            e.Handled = true;
        }
    }
}

```

Here, row heights are customized based on the large text content.



CustomerName	CustomerID	Country
Maria Anders	ALFKI	Germany
Ana Trujilo	ANATR	Mexico
Antonio Moreno	ANTON	Mexico
Maria Anders Jscob Steve Christina	AROUT	UK
Christina Berglund	BERGS	Sweden
Hanna Moos	BLAUS	Germany
Frédérique Citeaux	BLONP	France
Martin Sommer	BOLID	Spain
Laurence Lebihan	BONAP	France
Elizabeth Lincoln	BOTTM	Canada
Maria Anders	ALFKI	Germany

[GridRowSizingOptions](#)

[GridRowSizingOptions](#) have the following properties,

1. **ExcludeColumns** – If you want to skip specific column from row height calculation, you can add that column to [GridRowSizingOptions.ExcludeColumns](#). By default, [GetAutoRowHeight](#) method calculates the row height based on all columns.
2. **CanIncludeHiddenColumns** – If you want to consider the hidden columns while calculating row height, you can set [GridRowSizingOptions.CanIncludeHiddenColumns](#) as **true**.

### Calculate Height based on certain columns

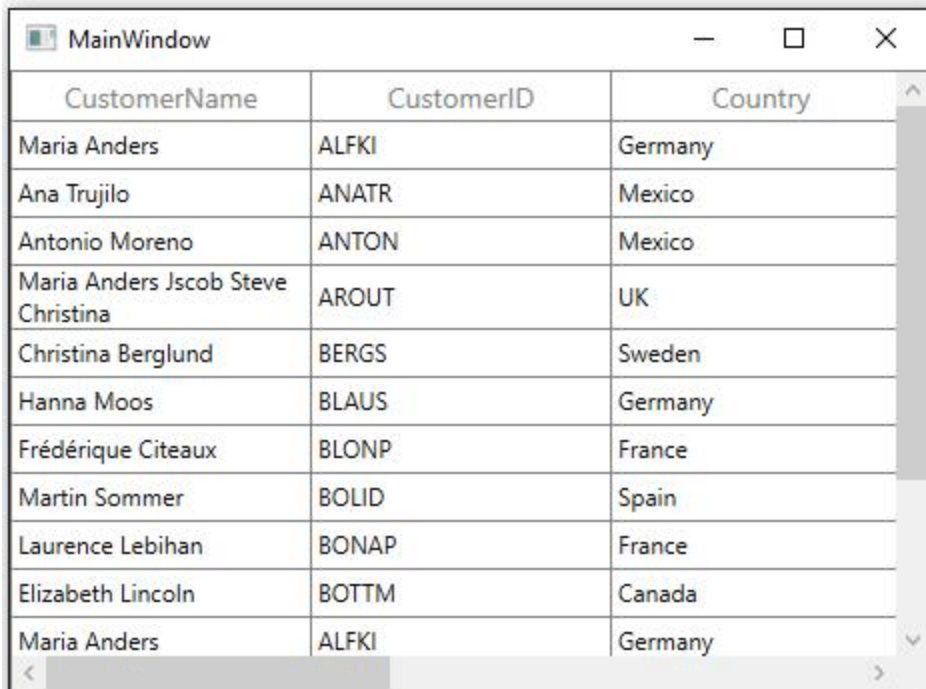
You can exclude columns from row height calculation using [GridRowSizingOptions.ExcludeColumns](#). This will help to reduce the count of loop run for height calculation for better performance.

You can add the columns which need to be excluded from height calculation using `GridRowSizingOptions.ExcludeColumns` collection.

### C#

```
GridRowSizingOptions gridRowResizingOptions = new GridRowSizingOptions();
//To get the calculated height from GetAutoRowHeight method.
double autoHeight = double.NaN;
// The list contains the column names that will be excluded from the height
// calculation in GetAutoRowHeight method.
List<string> excludeColumns = new List<string>() { "CustomerID", "Country"
};
this.dataGrid.QueryRowHeight += dataGrid_QueryRowHeight;
gridRowResizingOptions.ExcludeColumns = excludeColumns;
void dataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    if (this.dataGrid.GridColumnSizer.GetAutoRowHeight(e.RowIndex,
        gridRowResizingOptions, out autoHeight))
    {
        if (autoHeight > 24)
        {
            e.Height = autoHeight;
            e.Handled = true;
        }
    }
}
```

Here `CustomerID` and `Country` columns are excluded from height calculation and the row height is calculated based on `CustomerName` column only.



CustomerName	CustomerID	Country
Maria Anders	ALFKI	Germany
Ana Trujillo	ANATR	Mexico
Antonio Moreno	ANTON	Mexico
Maria Anders Jscob Steve Christina	AROUT	UK
Christina Berglund	BERGS	Sweden
Hanna Moos	BLAUS	Germany
Frédérique Citeaux	BLONP	France
Martin Sommer	BOLID	Spain
Laurence Lebihan	BONAP	France
Elizabeth Lincoln	BOTTM	Canada
Maria Anders	ALFKI	Germany

### Reset Row Height at runtime

You can reset height of the particular or all rows in View at runtime to get the updated height from `QueryRowHeight` event handler using below methods. You have to call [InvalidateMeasureInfo](#) method of `VisualContainer` to refresh the View.

- [InvalidateRowHeight](#) – Resets the height of particular row.

### C#

```
using Syncfusion.UI.Xaml.Grid.Helpers;
datagrid.InvalidateRowHeight(2);
datagrid.GetVisualContainer().InvalidateMeasureInfo();
```

- [RowHeightManager.Reset](#) – Resets the height for all rows in View.

### C#

```
using Syncfusion.UI.Xaml.Grid.Helpers;
this.datagrid.GetVisualContainer().RowHeightManager.Reset();
datagrid.GetVisualContainer().InvalidateMeasureInfo();
```

### Update Row Height while editing

You can set the height of the row based on the content after editing by refreshing the row height in [SfDataGrid.CurrentCellEndEdit](#) event.

You can call the `InvalidateRowHeight` method in [CurrentCellEndEdit](#) event to reset the particular row height. Then call the `InvalidateMeasureInfo` method of `VisualContainer` to refresh the view. Now the

QueryRowHeight event is called again for edited row alone and row height is calculated based on edited content.

### C#

```
using Syncfusion.UI.Xaml.Grid.Helpers;
GridRowSizingOptions gridRowResizingOptions = new GridRowSizingOptions();
//To get the calculated height from GetAutoRowHeight method.
double autoHeight = double.NaN;
this.dataGrid.QueryRowHeight += dataGrid_QueryRowHeight;
this.dataGrid.CurrentCellEndEdit += dataGrid_CurrentCellEndEdit;
void dataGrid_CurrentCellEndEdit(object sender, CurrentCellEndEditEventArgs args)
{
    dataGrid.InvalidateRowHeight(args.RowColumnIndex.RowIndex);
    dataGrid.GetVisualContainer().InvalidateMeasureInfo();
}
void dataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    if (this.dataGrid.GridColumnSizer.GetAutoRowHeight(e.RowIndex, gridRowResizingOptions, out autoHeight))
    {
        if (autoHeight > 24)
        {
            e.Height = autoHeight;
            e.Handled = true;
        }
    }
}
```

### Change HeaderRow Height based on its Content

By default, auto height is supported for the headers is QueryRowHeight event. If you want to set the auto height to header row alone, you can use the [GetHeaderIndex](#) method to decide whether the row index is header or not in QueryRowHeight event.

### XML

```
<Window.Resources>
<DataTemplate x:Key="headerTemplate">
<TextBlock Height="50"
FontWeight="Bold"
Foreground="DarkBlue"
Text="Total Amount of Price in this month"
TextWrapping="Wrap" />
</DataTemplate>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid" ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn HeaderTemplate="{StaticResource headerTemplate}"
MappingName="TotalPrice" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

### C#



```

GridRowSizingOptions gridRowResizingOptions = new GridRowSizingOptions();
//To get the calculated height from the GetAutoRowHeight method.
double autoHeight;
this.dataGrid.QueryRowHeight += dataGrid_QueryRowHeight;
void dataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    //checked whether the row index is header or not.
    if (this.dataGrid.GetHeaderIndex() == e.RowIndex)
    {
        if (this.dataGrid.GridColumnSizer.GetAutoRowHeight(e.RowIndex,
            gridRowResizingOptions, out autoHeight))
        {
            if (autoHeight > 24)
            {
                e.Height = autoHeight;
                e.Handled = true;
            }
        }
    }
}

```

Total Amount of Price in this month	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1004	AROUT	Maria Anders Jscob Steve
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

#### Change StackedHeaderRow Height based on its content

By default, auto height is supported for **StackedHeaderRows** in **QueryRowHeight** event. You can also set the auto height to the **StackedHeaderRows** alone using **QueryRowHeight** event by checking the row index with **StackedHeaderRows** count.

Also you can wrap stacked header text by writing style of TargetType [GridStackedHeaderCellControl](#) and set the **TextWrapping** as Wrap as below,

#### XML

```
<Style TargetType="syncfusion:GridStackedHeaderCellControl">
```

```
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="syncfusion:GridStackedHeaderCellControl">
<Border Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}">
<Grid Margin="{TemplateBinding Padding}">
<ContentPresenter HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}" VerticalAlignment="{TemplateBinding
VerticalContentAlignment}">
<ContentPresenter.Content>
<TextBlock Text="{Binding HeaderText}" TextWrapping="Wrap" />
</ContentPresenter.Content>
</ContentPresenter>
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
```

## C#

```
GridRowSizingOptions gridRowResizingOptions = new GridRowSizingOptions();
//To get the calculated height from the GetAutoRowHeight method.
double autoHeight;
this.dataGrid.QueryRowHeight += dataGrid_QueryRowHeight;
void dataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    if (e.RowIndex < this.dataGrid.StackedHeaderRows.Count)
    {
        if (this.dataGrid.GridColumnSizer.GetAutoRowHeight(e.RowIndex,
gridRowResizingOptions, out autoHeight))
        {
            if (autoHeight > 24)
            {
                e.Height = autoHeight;
                e.Handled = true;
            }
        }
    }
}
```

Customer Details with ID, Name		Customer Country and ShipCity Details	
CustomerID	CustomerName	Country	ShipCity
ALFKI	Maria Anders	Germany	Berlin
ANATR	Ana Trujilo	Mexico	México D.F.
ANTON	Antonio Moreno	Mexico	México D.F.
AROUT	Maria Anders Jscob Stev	UK	London
BERGS	Christina Berglund	Sweden	Luleå
BLAUS	Hanna Moos	Germany	Mannheim
BLONP	Frédérique Citeaux	France	Strasbourg
BOLID	Martin Sommer	Spain	Madrid
BONAP	Laurence Lebihan	France	Marseille
BOTTN	Elizabeth	Germany	Frankfurt

### Change TableSummaryRow Height

You can change the table summary row height by using [QueryRowHeight](#) event. You can use [IsTableSummaryIndex](#) extension method to identify whether the row is table summary or not by passing row index.

### C#

```
using Syncfusion.UI.Xaml.Grid.Helpers;
this.dataGrid.QueryRowHeight += dataGrid_QueryRowHeight;
void dataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    if (dataGrid.IsTableSummaryIndex(e.RowIndex))
    {
        e.Height = 40;
        e.Handled = true;
    }
}
```

TotalPrice	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1004	AROUT	Maria Anders Jscob Steve C
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln
Total Price : \$10,055.00		

### Row drag and drop in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) allows drag and drop the rows within and between controls by setting the [AllowDraggingRows](#) and [AllowDrop](#) property as true. It is also possible to drag and drop the rows between datagrid and other controls such as [ListView](#) and [SfTreeGrid](#). SfDataGrid allows dropping rows when [AllowDrop](#) is true and allows dragging when [AllowDraggingRows](#) is true.

#### XML

```
<Syncfusion:SfDataGrid x:Name="datagrid"
    AllowDraggingRows="True"
    AllowDrop="True"
    ItemsSource="{Binding Source}"/>
```

#### C#

```
this.datagrid.AllowDraggingRows = true;
this.datagrid.AllowDrop = true;
```

While dropping, the dragged records can be added above or below to the target record based on its drop position

For example, if you dropped record at the bottom of the targeted record, it will be added below the targeted record.

	Order ID	Customer ID	Ship Name	Ship City	Ship Address	Employee ID
	10054	RICAR	Ricardo Adocicados	Rio de Janeiro	Av. Copacabana, 267	52.39
	10055	BLONP	Blondel père et fils	Strasbourg	24, place Kléber	135.33
	10056	LILAS	LILA-Supermercado	Barquisimeto	Carrera 52 con Ave. Bolí	20.09
↓	10057	THECR	The Cracker Box	Butte	55 Grizzly Peak Rd.	4.03
↑	10058	HUNGO	Hungry Owl	Johnstown Road		101.66
	10059	LILAS	LILA-Supermercado	Barquisimeto	Carrera 52 con Ave. Bolí	22.45
	10060	SEVES	Seven Seas Imports	London	90 Wadhurst Rd.	101.91
	10061	EASTC	Eastern Connection	London	35 King George	4.34
	10062	ALFKI	Alfred's Futterkiste	Berlin	Obere Str. 57	47.22
	10063	HUNGO	Hungry Owl All-Night Gr	Cork	8 Johnstown Road	15.33
	10064	VAFFE	Vaffeljernet	Århus	Smagsløget 45	10.27

If you drop above the targeted record, it will be added above the targeted record

	Order ID	Customer ID	Ship Name	Ship City	Ship Address	Employee ID
	10223	SPECD	Spécialités du monde	Paris	25, rue Lauriston	16.62
	10224	RICAR	Ricardo Adocicados	Rio de Janeiro	Av. Copacabana, 267	24.58
	10225	SAVEA	Save-a-lot Markets	Boise	187 Suffolk Ln.	147.73
↓	10226	SAVEA	Save-a-lot Markets	Boise	187 Suffolk Ln.	358.75
↑	10227	TRADH	Tradição Hipermercado Sã	Boa Vista, 414		2.15
	10228	CENTC	Centro comercial Moch	Me	a 999	36.16
	10229	LAZYK	Lazy K Kountry Store	Walla Walla	12 Orchestra Terrace	8.81
	10230	BOTTM	Bottom-Dollar Markets	Tsawassen	23 Tsawassen Blvd.	151.14
>	10231	BOTTM	Bottom-Dollar Markets	Tsawassen	23 Tsawassen Blvd.	21.3
	10232	DRACD	Drachenblut Delikatess	Aachen	Walserweg 21	12.48
	10233	BERGS	Berglunds snabbköp	Luleå	Berguvsvägen 8	39.47
	10234	GROSR	GROSELLA-Restaurante	Caracas	5ª Ave. Los Palos Grand	30.34
	10235	VICTE	Victuailles en stock	Lyon	2, rue du Commerce	7.58

### Dragging multiple rows

WPF DataGrid (SfDataGrid) allows to drag multiple selected rows. To enable multiple selection, set the [SfDataGrid.SelectionMode](#) as **Multiple** or **Extended**.

**Note:** The drag selection cannot be performed while the [AllowDraggingRows](#) enabled as **true** in the SfDataGrid.

	Order ID	Customer ID	Ship Name	Ship City	Ship Address	Employee ID
	10000	FRANS	Franchi S.p.A.	Torino	Via Monte Bianco 34	4.45
	10001	MEREP	Mère Paillarde	Montréal	43 rue St. Laurent	79.45
	10002	FOLKO	Folk och få HB	Bräcke	Åkergatan 24	36.18
	10003	SIMOB	Simons bistro	København	Vinbæltet 34	18.59
	10004	VAFFE	Vaffeljernet	Århus	Smagsløget 45	20.12
	10005	WARTH	Wartian Herkku	Oulu	Torikatu 38	4.13
>	10006	FRANS	Franchi S.p.A.	Torino	Via Monte Bianco 34	3.62
	10007	MORGK	Morgenstern Gesundk	Leipzig	Heerstr. 22	36.19
↓	10008	FURIB	Furia Bacalhau e Fruto	Lisboa	Jardim das rosas n. 32	74.22
↑	10009		Seas Imports	London	90 Wadhurst Rd.	49.21
	10010		s bistro	København	Vinbæltet 34	3.01

### Drag and drop events

SfDataGrid triggers the following events when drag and drop:

#### Drag start event

[DragStart](#) event occurs when you start to drag the records in datagrid. The [GridRowDragStartEventArgs](#) has the following member, which provides information for the [DragStart](#) event.

- [DraggingRecords](#) : Gets the Records which contains the data associated while dragging the rows.

### C#

```

this.sfDataGrid.RowDragDropController.DragStart +=
RowDragDropController_DragStart;
private void RowDragDropController_DragStart(object sender,
GridRowDragStartEventArgs e)
{
}

```

#### Drag over event

[DragOver](#) event occurs continuously while record is dragged within the target SfDataGrid. The [GridRowDragOverEventArgs](#) has the following members, which provide information for the [DragOver](#) event.

- [Data](#) : Gets a data object that contains the data associated while dragging the rows.
- [DropPosition](#) : Gets a value indicating the drop position which is based on dropped location
- [IsFromOutsideSource](#) : Gets a value indicating whether the dragging item is from same DataGrid or not.

\*[ShowDragUI](#) : Gets or sets a value indicating the default Dragging UI.

- [TargetRecord](#) : Gets a value indicating the target record which is going to drop.

**C#**

```
this.sfDataGrid.RowDragDropController.DragOver +=  
RowDragDropController_DragOver;  
private void RowDragDropController_DragOver(object sender,  
GridRowDragOverEventArgs e)  
{  
}
```

*Drag leave event*

[DragLeave](#) event occurs when leave a drag-and-drop operation. The [GridRowDragLeaveEventArgs](#) has the following members, which provide information for the [DragLeave](#) event.

- [Data](#) : Gets a data object that contains the data associated while dragging the rows.
- [DropPosition](#) : Gets a value indicating the drop position which is based on dropped location
- [IsFromOutSideSource](#) : Gets a value indicating whether the dragging item is from same DataGrid or not.
- [TargetRecord](#) : Gets a value indicating the target record which is going to drop.

**C#**

```
this.sfDataGrid.RowDragDropController.DragLeave +=  
RowDragDropController_DragLeave;  
private void RowDragDropController_DragLeave(object sender,  
GridRowDragLeaveEventArgs e)  
{  
}
```

*Drop event*

[Drop](#) event occurs when a record is dropping within the target SfDataGrid. The [GridRowDropEventArgs](#) has the following members, which provide information for the [Drop](#) event.

- [Data](#) : Gets a data object that contains the data associated while dragging the rows.
- [DraggingRecords](#) : Gets the Records which contains the data associated while dragging the rows.
- [DropPosition](#) : Gets a value indicating the drop position which is based on dropped location
- [IsFromOutSideSource](#) : Gets a value indicating whether the dragging item is from same DataGrid or not.
- [TargetRecord](#) : Gets a value indicating the target record which is going to drop.

**C#**

```
this.sfDataGrid.RowDragDropController.Drop += RowDragDropController_Drop;  
private void RowDragDropController_Drop(object sender, GridRowDropEventArgs  
e)  
{  
}
```

### *Dropped event*

[Dropped](#) event occurs when a record is dropping within the target SfDataGrid. The [GridRowDroppedEventArgs](#) has the following members, which provide information for the **Drop** event.

- [Data](#) : Gets a data object that contains the data associated while dragging the rows.
- [DropPosition](#) : Gets a value indicating the drop position which is based on dropped location
- [IsFromOutsideSource](#) : Gets a value indicating whether the dragging item is from same DataGrid or not.
- [TargetRecord](#) : Gets a value indicating the target record which is going to drop.

### **C#**

```
this.sfDataGrid.RowDragDropController.Dropped +=  
RowDragDropController_Dropped;  
private void RowDragDropController_Dropped(object sender,  
GridRowDroppedEventArgs e)  
{  
}
```

### Changing the row drop indicator

By default, the drop position will be indicated with arrows. To change the drop indicator as line, then set the [sfDataGrid.RowDropIndicatorMode](#) as **Line**.

### **XML**

```
<syncfusion:SfDataGrid Name="dataGrid"  
ItemsSource="{Binding OrdersListDetails}"  
AllowDrop="True"  
AllowDraggingRows="True"  
RowDropIndicatorMode="Line" />
```

### **C#**

```
this.dataGrid.RowDropIndicatorMode =  
Syncfusion.UI.Xaml.Grid.DropIndicatorMode.Line;
```



Order ID	Customer ID	Product Name	Quantity	Unit Price	Contact Number
10000	FRANS	Alice Mutton	50	\$32.80	999121235
10001	FRANS	Boston Crab Meat	57	\$56.70	999121236
10002	FRANS	Raclette Courdavault	29	\$336.20	999121237
10003	FRANS	Wimmers gute	46	\$70.30	999121238
10004	MEREP	Raclette Courdavault	48	\$42.50	999121239
10005	MEREP	Alice Mutton	35	\$41.00	999121240
10011	FOLKO	Raclette Courdavault	38	\$70.30	999121246
10006	MEREP	Alice Mutton		79.90	999121241
10010	FOLKO	Wimmers gute		42.50	999121245
10007	MEREP	Wimmers gute	48	\$434.20	999121242
10008	FOLKO	Alice Mutton	52	\$26.80	999121243
10009	FOLKO	Alice Mutton	41	\$63.70	999121244
10012	SIMOB	NuNuCa Nuß-Nougat-Crem	26	\$63.70	999121247
10013	SIMOB	Raclette Courdavault	26	\$23.30	999121248
10014	SIMOB	Alice Mutton	39	\$26.80	999121249
10015	SIMOB	Alice Mutton	28	\$512.00	999121250
10016	WARTH	NuNuCa Nuß-Nougat-Crem	32	\$434.20	999121251

### Customizing row drag and drop operation

#### Disable dragging of certain rows in WPF DataGrid

You can restrict the dragging of certain rows in SfDataGrid by using the [GridRowDragDropController.DragStart](#) event.

#### C#

```
this.sfDataGrid.RowDragDropController.DragStart +=
RowDragDropController_DragStart;
private void RowDragDropController_DragStart(object sender,
Syncfusion.UI.Xaml.Grid.GridRowDragStartEventArgs e)
{
    var records = e.DraggingRecords;
    var orders = records[0] as Orders;
    // You can restrict the dragging for certain rows based on the record value
    also.
    var rowIndex = this.sfDataGrid.ResolveToRowIndex(orders);
    var recordIndex = this.sfDataGrid.ResolveToRecordIndex(rowIndex);
    if (recordIndex > 5)
        e.Handled = true;
}
```

#### Disable dropping over certain rows in WPF DataGrid

You can restrict the dropping the records in certain rows in SfDataGrid by using the [GridRowDragDropController.Drop](#) event.

#### C#

```
this.sfDataGrid.RowDragDropController.Drop += RowDragDropController_Drop;
private void RowDragDropController_Drop(object sender, GridRowDropEventArgs
e)
{
    var record = e.TargetRecord;
    if (record == null)
        return;
}
```

```

var orders = (record as RecordEntry).Data as Orders;
// You can restrict the dropping for certain rows based on the target record
value also.
var rowIndex = this.sfDataGrid.ResolveToRowIndex(orders);
var recordIndex = this.sfDataGrid.ResolveToRecordIndex(rowIndex);
if (recordIndex > 5)
e.Handled = true;
}

```

### Disable the default drag UI

You can disable the draggable popup by setting the [ShowDragUI](#) as `false` in the [DragOver](#) event of [GridRowDragDropController.DragOver](#) event .

### C#

```

this.sfDataGrid.RowDragDropController.DragOver +=
RowDragDropController_DragOver;
private void RowDragDropController_DragOver(object sender,
GridRowDragOverEventArgs e)
{
e.ShowDragUI = false;
}

```

### Customizing draggable Popup

To customize draggable popup, use the [RowDragDropTemplate](#) property in the SfDataGrid.

### XML

```

<DataTemplate x:Key="template">
<Border x:Name="border" Width="250"
Background="#ecec"
BorderBrush="#c8c8c8" Height="60"
BorderThickness="1.2">
<Grid VerticalAlignment="Center"
HorizontalAlignment="Left">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<TextBlock Padding="12,0,0,0" Text="Dragging rows count :" FontSize="14"
FontFamily="Segoe UI"
Foreground="#333333" FontWeight="Regular" Background="SkyBlue" />
<TextBlock Text="{Binding DraggingRecords.Count}" FontSize="14"
FontFamily="Segoe UI"
FontWeight="Regular"
Foreground="#333333"
Grid.Column="1" Margin="-100,0,0,0"/>
<Separator Grid.Row="1" Height="2" BorderBrush="#c8c8c8"
HorizontalAlignment="Stretch" BorderThickness="1"
VerticalAlignment="Stretch" Width="250"/>

```

```

<TextBlock Text="Drop status:"
Foreground="#333333"
Padding="12,0,0,0" Background="SkyBlue"
FontFamily="Segoe UI"
FontWeight="Regular"
FontSize="14"
Grid.Row="2"/>
<TextBlock Text="{Binding DragStatus}"
FontSize="14"
FontFamily="Segoe UI"
FontWeight="Regular"
Foreground="#333333"
Margin="-163,0,0,0"
Grid.Row="2"
Grid.Column="1"/>
</Grid>
</Border>
</DataTemplate>
<Syncfusion:SfDataGrid x:Name="datagrid"
AllowDraggingRows="True"
AllowDrop="True"
RowDragDropTemplate="{StaticResource template}"
ItemsSource="{Binding Source}"/>

```

	Order ID	Customer ID	Ship Name	Ship City	Ship Address	Employee ID
>	10002	FOLKO	Folk och få HB	Bräcke	Åkergatan 24	36.18
	10003	SIMOB	Simons bistro	København	Vinbæltet 34	18.59
	10004	VAFFE	Vaffeljernet	Århus	Smagsløget 45	20.12
↓	10000	FRANS	Franchi S.p.A.	Torino	Via Monte Bianco 34	4.45
↑	10001	MEREP	Dragging rows count : 1		43 rue St. Laurent	79.45
	10005	WARTH	Drop status:DropBelow		Torikatu 38	4.13
	10006	FRANS	Franchi S.p.A.	Torino	Via Monte Bianco 34	3.62
	10007	MORGK	Morgenstern Gesunc	Leipzig	Heerstr. 22	36.19
	10008	FURIB	Furia Bacalhau e Frut	Lisboa	Jardim das rosas n. 3.	74.22
	10009	SEVES	Seven Seas Imports	London	90 Wadhurst Rd.	49.21
	10010	SIMOB	Simons bistro	København	Vinbæltet 34	3.01
	10011	WELLI	Wellington Importad	Resende	Rua do Mercado, 12	31.54

#### Customizing draggable popup to show corresponding dragging row data

You can customize the dragging popup to show the corresponding drag row data by customizing the `RowDragDropTemplate` with binding the data from the `DraggingRecords` property.

#### XML

```

<DataTemplate x:Key="dragdroptemplate">
<Border x:Name="border" Width="250"
Background="#ecec"
BorderBrush="#c8c8c8" Height="60"
BorderThickness="1.2">
<Grid VerticalAlignment="Center" HorizontalAlignment="Left">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<TextBlock Text="OrderID : " Grid.Row="0" Grid.Column="0" />
<TextBlock Text="{Binding DraggingRecords[0].OrderID}" Grid.Row="0"
Grid.Column="1"/>
<TextBlock Text="CustomerID : " Grid.Row="1" Grid.Column="0"/>
<TextBlock Text="{Binding DraggingRecords[0].CustomerID}" Grid.Row="1"
Grid.Column="1"/>
</Grid>
</Border>
</DataTemplate>
<Syncfusion:SfDataGrid x:Name="datagrid"
AllowDraggingRows="True"
AllowDrop="True"
RowDragDropTemplate="{StaticResource dragdroptemplate}"
ItemsSource="{Binding OrderDetails}"/>

```

	Order ID	Customer ID	Ship Name	Ship City	Ship Address	Employee ID
>	10001	MEREP	Mère Paillarde	Montréal	43 rue St. Laurent	79.45
	10002	FOLKO	Folk och få HB	Bräcke	Åkergatan 24	36.18
	10003	SIMOB	Simons bistro	København	Vinbæltet 34	18.59
↓	10000	FRANS	Franchi S.p.A.	Torino	Via Monte Bianco 34	4.45
↑	10004	VAFPE	OrderID : 10001 CustomerID : MEREP		Smagsløget 45	20.12
	10005	WARTH			Torikatu 38	4.13
	10006	FRANS	Franchi S.p.A.	Torino	Via Monte Bianco 34	3.62
	10007	MORGK	Morgenstern Gesund	Leipzig	Heerstr. 22	36.19
	10008	FURIB	Furia Bacalhau e Frut	Lisboa	Jardim das rosas n. 32	74.22
	10009	SEVES	Seven Seas Imports	London	90 Wadhurst Rd.	49.21
	10010	SIMOB	Simons bistro	København	Vinbæltet 34	3.01
	10011	WELLI	Wellington Importada	Resende	Rua do Mercado, 12	31.54

*Reorder the source collection while drag and drop the rows*

You can reorder the source collection after drag and drop the row by handling [RowDragDropController.Dropped](#) event.

**C#**

```
sfGrid.RowDragDropController.Dropped += OnRowDropped;
private void OnRowDropped(object sender, GridRowDroppedEventArgs e)
{
    if (e.DropPosition != DropPosition.None)
    {
        // Get Dragging records
        ObservableCollection<object> draggingRecords = e.Data.GetData("Records") as
        ObservableCollection<object>;
        // Gets the TargetRecord from the underlying collection using record index
        // of the TargetRecord (e.TargetRecord)
        ViewModel model = sfGrid.DataContext as ViewModel;
        OrderInfo targetRecord = model.OrdersFirstGrid[(int)e.TargetRecord];
        // Use Batch update to avoid data operations in SfDataGrid during records
        // removing and inserting
        sfGrid.BeginInit();
        // Removes the dragging records from the underlying collection
        foreach (OrderInfo item in draggingRecords)
        {
            model.OrdersFirstGrid.Remove(item);
        }
        // Find the target record index after removing the records
        int targetIndex = model.OrdersFirstGrid.IndexOf(targetRecord);
        int insertionIndex = e.DropPosition == DropPosition.DropAbove ? targetIndex
        : targetIndex + 1;
        insertionIndex = insertionIndex < 0 ? 0 : insertionIndex;
        // Insert dragging records to the target position
        for (int i = draggingRecords.Count - 1; i >= 0; i--)
        {
            model.OrdersFirstGrid.Insert(insertionIndex, draggingRecords[i] as
            OrderInfo);
        }
        sfGrid.EndInit();
    }
}
```

*Row drag and drop between DataGrid and ListView*

To perform dragging between the [ListView](#) and [SfDataGrid](#), by using the [GridRowDragDropController.DragStart](#) and [GridRowDragDropController.Drop](#) events. And you must set the [AllowDrop](#) property as [true](#) in the [ListView](#) while doing the drag and drop operation from [SfDataGrid](#) with [ListView](#) control.

**C#**

```
this.dataGrid.RowDragDropController.DragStart += sfGrid_DragStart;
this.dataGrid.RowDragDropController.Drop += sfGrid_Drop;
this.listView.PreviewMouseMove += ListView_PreviewMouseMove;
this.listView.Drop += ListView_Drop;
/// <summary>
/// customize the DragStart event.Restrict the certain record from dragging.
```

```

/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void sfGrid_DragStart(object sender, GridRowDragStartEventArgs e)
{
    var draggingRecords = e.DraggingRecords[0] as OrderInfo;
    if (draggingRecords.CustomerName == "Martin")
    {
        e.Handled = true;
    }
}

/// <summary>
/// Customize the Drop event.restrict the certain record and Drop position
from drop.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void sfGrid_Drop(object sender, GridRowDropEventArgs e)
{
    if (e.IsFromOutsideSource)
    {
        ObservableCollection<object> DraggingRecords = new
        ObservableCollection<object>();
        if (e.Data.GetDataPresent("ListViewRecords"))
            DraggingRecords = e.Data.GetData("ListViewRecords") as
            ObservableCollection<object>;
        else
            DraggingRecords = e.Data.GetData("Records") as ObservableCollection<object>;
        var draggingRecords = DraggingRecords[0] as OrderInfo;
        int dropIndex = (int)e.TargetRecord;
        var dropPosition = e.DropPosition.ToString();
        IList collection = AssociatedObject.sfGrid.View.SourceCollection as IList;
        if (dropPosition == "DropAbove")
        {
            dropIndex--;
            collection.Insert(dropIndex, draggingRecords);
        }
        else
        {
            dropIndex++;
            collection.Insert(dropIndex, draggingRecords);
        }
        (AssociatedObject.listView.ItemsSource as
        ObservableCollection<OrderInfo>).Remove(draggingRecords as OrderInfo);
        e.Handled = true;
    }
}

/// <summary>
/// List view initiates the DragDrop operation
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ListView_PreviewMouseMove(object sender, MouseEventArgs e)
{
    if (e.LeftButton == MouseButtonState.Pressed)
    {
        ListBox dragSource = null;
    }
}

```

```

var records = new ObservableCollection<object>();
ListBox parent = (ListBox)sender;
dragSource = parent;
object data = GetDataFromListBox(dragSource, e.GetPosition(parent));
records.Add(data);
var dataObject = new DataObject();
dataObject.SetData("ListViewRecords", records);
dataObject.SetData("ListView", AssociatedObject.listView);
if (data != null)
{
    DragDrop.DoDragDrop(parent, dataObject, DragDropEffects.Move);
}
}
e.Handled = true;
}
/// <summary>
/// Get the data from list box control
/// </summary>
/// <param name="source"></param>
/// <param name="point"></param>
/// <returns></returns>
private static object GetDataFromListBox(ListBox source, Point point)
{
    UIElement element = source.InputHitTest(point) as UIElement;
    if (element != null)
    {
        object data = DependencyProperty.UnsetValue;
        while (data == DependencyProperty.UnsetValue)
        {
            data = source.ItemContainerGenerator.ItemFromContainer(element);
            if (data == DependencyProperty.UnsetValue)
            {
                element = VisualTreeHelper.GetParent(element) as UIElement;
            }
            if (element == source)
            {
                return null;
            }
        }
        if (data != DependencyProperty.UnsetValue)
        {
            return data;
        }
    }
    return null;
}
/// <summary>
/// ListView Drop event.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ListView_Drop(object sender, DragEventArgs e)
{
    ObservableCollection<object> DraggingRecords = new
    ObservableCollection<object>();
    if (e.Data.GetDataPresent("ListViewRecords"))
    {

```

```

DraggingRecords = e.Data.GetData("ListViewRecords") as
ObservableCollection<object>;
var listViewRecord = DraggingRecords[0] as OrderInfo;
(AssociatedObject.listView.ItemsSource as
ObservableCollection<OrderInfo>).Remove(listViewRecord);
(this.AssociatedObject.DataContext as
ViewModel).OrdersListView.Add(listViewRecord );
}
else
{
DraggingRecords = e.Data.GetData("Records") as ObservableCollection<object>;
var record = DraggingRecords[0] as OrderInfo;
this.AssociatedObject.sfGrid.View.Remove(record);
(this.AssociatedObject.DataContext as ViewModel).OrdersListView.Add(record);
}
}

```

OrderID	CustomerID	CustomerName	Country	ShipCity		
1001	ALFKI	Maria Anders	Germany	Berlin	Berlin	
1002	ANATR	Ana Trujillo	Mexico	Mexico D.F.	Mexico D.F.	
1003	ANTON	Antonio Moreno	Mexico	Mexico D.F.	London	
1004	AROUT	Thomas Hardy	UK	London	Lula	
1005	BERGS	Christina Berglund	Sweden	Lula	Mannheim	
⬇	1006	BLAUS	Hanna Moos	Germany	Mannheim	
⬆	1007	BLONP	Frederique Cite aux	Dragging rows count : 9 Drop status : Drop below		Strasbourg
1008	BOLID	Martin				Madrid
1009	BONAP	Laurence	France	Marseille	Marseille	
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen	Tsawassen	

You can download the [sample](#).

Row drag and drop between two DataGrid's

To perform the dragging operation between two datagrid by using the [GridRowDragDropController.DragStart](#) , [GridRowDragDropController.Drop](#) , [GridRowDragDropController.DragOver](#) and [GridRowDragDropController.Dropped](#) events.

### C#

```

this.firstDataGrid.RowDragDropController.DragStart += sfGrid_DragStart;
this.firstDataGrid.RowDragDropController.Drop += sfGrid_Drop;
this.firstDataGrid.RowDragDropController.Dropped += sfGrid_Dropped;
this.secondDataGrid.RowDragDropController.DragOver += grid_DragOver;
/// <summary>
/// customize the DragStart event.Restrict the certain record from dragging.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void sfGrid_DragStart(object sender, GridRowDragStartEventArgs e)
{
var record = e.DraggingRecords[0] as OrderInfo;
if (record.CustomerName == "Martin")
{
e.Handled = true;
}
}

```



```
/// <summary>
/// Customize the DragOver event.Disable the DragUI
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void grid_DragOver(object sender, GridRowDragOverEventArgs e)
{
    e.ShowDragUI = false;
    e.Handled = true;
}
/// <summary>
/// Customize the Drop event
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void sfGrid_Drop(object sender, GridRowDropEventArgs e)
{
    var record = e.DraggingRecords[0] as OrderInfo;
    var dropPosition = e.DropPosition.ToString();
    if (dropPosition == "DropAbove")
    {
        e.Handled = true;
    }
    if (record.ShipCity == "Mexico D.F.")
    {
        e.Handled = true;
    }
}
/// <summary>
/// Customize the Dropped event.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void sfGrid_Dropped(object sender, GridRowDroppedEventArgs e)
{
    ObservableCollection<object> draggingRecords = new
    ObservableCollection<object>();
    draggingRecords = e.Data.GetData("Records") as ObservableCollection<object>;
    var items = draggingRecords[0] as OrderInfo;
    var records = AssociatedObject.firstDataGrid.View.Records.ToList();
    IList collection = AssociatedObject.firstDataGrid.ItemsSource as IList;
    for (int i = 0; i < records.Count; i++)
    {
        var orderData = records[i].Data as OrderInfo;
        if (orderData.OrderID == items.OrderID)
        {
            collection.Remove(items);
            collection.Insert(i, orderData);
        }
    }
    AssociatedObject.firstDataGrid.ItemsSource = collection;
}
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	Mexico D.F.
1003	ANTON	Antonio Moreno	Mexico	Mexico D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Lula
↓	1007	BLONP	Frederique Cite aux	France
↑	1008	BOLID	Martin	Spain
	1009	BONAP	Lar	Marseille
	1010	BOTTM	Eli	Tsawassen

Dragging rows count : 1

Drop status : Drop above

OrderID	CustomerID	CustomerName	Country	ShipCity
101	ALFKI	Maria Anders	Germany	Berlin
102	ANATR	Ana Trujillo	Mexico	Mexico D.F.
1006	BLAUS	Hanna Moos	Germany	Mannheim
103	ANTON	Antonio Moreno	Mexico	Mexico D.F.
104	AROUT	Thomas Hardy	UK	London
105	BERGS	Christina Berglund	Sweden	Lula
106	BLAUS	Hanna Moos	Germany	Mannheim

You can download the [sample](#);

Row drag and drop between DataGrid and TreeGrid

To perform the dragging operation between **SfDataGrid** and **SfTreeGrid** by using the [GridRowDragDropController.Drop](#) and [TreeGridRowDragDropController.Drop](#) events.

### C#

```

this.sfDataGrid.RowDragDropController.Drop += sfDataGrid_Drop;
this.sfTreeGrid.RowDragDropController.Drop += sfTreeGrid_Drop;
/// <summary>
/// Customized TreeGrid Drop event.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void sfTreeGrid_Drop(object sender, TreeGridRowDropEventArgs e)
{
    if (e.IsFromOutsideSource)
    {
        var draggingRecord = e.Data.GetData("Records") as
ObservableCollection<object>;
        var record = draggingRecord[0] as EmployeeInfo;
        var dropPosition = e.DropPosition.ToString();
        var newItem = new EmployeeInfo();
        var rowIndex
=AssociatedObject.sfTreeGrid.ResolveToRowIndex(e.TargetNode.Item);
        if (dropPosition != "None" && rowIndex != -1)
        {
            if (AssociatedObject.sfTreeGrid.View is TreeGridSelfRelationalView)
            {
                var treeNode = e.TargetNode;

```

```
if (treeNode == null)
return;
var data = treeNode.Item;
AssociatedObject.sfTreeGrid.SelectionController.SuspendUpdates();
var dropIndex = -1;
TreeNode parentNode = null;
if (dropPosition == "DropBelow" || dropPosition == "DropAbove")
{
parentNode = treeNode.ParentNode;
if (parentNode == null)
{
var treeNodeItem = treeNode.Item as EmployeeInfo;
newItem = new EmployeeInfo() { FirstName = record.FirstName, LastName =
record.LastName, ID = record.ID, Salary = record.Salary, Title =
record.Title, ReportsTo = treeNodeItem.ReportsTo };
}
else
{
var parentNodeItems = parentNode.Item as EmployeeInfo;
newItem = new EmployeeInfo() { FirstName = record.FirstName, LastName =
record.LastName, ID = record.ID, Salary = record.Salary, Title =
record.Title, ReportsTo = parentNodeItems.ID };
}
}
else if (dropPosition == "DropAsChild")
{
if (!treeNode.IsExpanded)
AssociatedObject.sfTreeGrid.ExpandNode(treeNode);
parentNode = treeNode;
var parentNodeItems = parentNode.Item as EmployeeInfo;
newItem = new EmployeeInfo() { FirstName = record.FirstName, LastName =
record.LastName, ID = record.ID, Salary = record.Salary, Title =
record.Title, ReportsTo = parentNodeItems.ID };
}
IList sourceCollection = null;
if (dropPosition == "DropBelow" || dropPosition == "DropAbove")
{
if (treeNode.ParentNode != null)
{
var collection =
AssociatedObject.sfTreeGrid.View.GetPropertyAccessProvider().GetValue(treeNo
de.ParentNode.Item, AssociatedObject.sfTreeGrid.ChildPropertyName) as
IEnumerable;
sourceCollection = GetSourceListCollection(collection);
}
else
{
sourceCollection =
GetSourceListCollection(AssociatedObject.sfTreeGrid.View.SourceCollection);
}
dropIndex = sourceCollection.IndexOf(data);
if (dropPosition == "DropBelow")
{
dropIndex += 1;
}
}
else if (dropPosition == "DropAsChild")
```

```

{
    var collection =
        AssociatedObject.sfTreeGrid.View.GetPropertyAccessProvider().GetValue(data,
            AssociatedObject.sfTreeGrid.ChildPropertyName) as IEnumerable;
    sourceCollection = GetSourceListCollection(collection);
    if (sourceCollection == null)
    {
        var list =
            data.GetType().GetProperty(AssociatedObject.sfTreeGrid.ChildPropertyName).PropertyType.CreateNew() as IList;
        if (list != null)
        {
            AssociatedObject.sfTreeGrid.View.GetPropertyAccessProvider().SetValue(treeNode.Item, AssociatedObject.sfTreeGrid.ChildPropertyName, list);
            sourceCollection = list;
        }
    }
    dropIndex = sourceCollection.Count;
}
sourceCollection.Insert(dropIndex, newItem);
AssociatedObject.sfTreeGrid.SelectionController.ResumeUpdates();
(AssociatedObject.sfTreeGrid.SelectionController as
    TreeGridRowSelectionController).RefreshSelection();
e.Handled = true;
}
}
AssociatedObject.sfDataGrid.View.Remove(record);
}
}

/// <summary>
/// Gets the source collection of TreeGrid
/// </summary>
/// <param name="collection"></param>
/// <returns></returns>
private IList GetSourceListCollection(IEnumerable collection)
{
    IList list = null;
    if (collection == null)
        collection = AssociatedObject.sfTreeGrid.View.SourceCollection;
    if ((collection as IList) != null)
    {
        list = collection as IList;
    }
    return list;
}

/// <summary>
/// Customize the Drop event.restrict the certain record and Drop position
/// from drop.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void sfDataGrid_Drop(object sender, GridRowDropEventArgs e)
{
    if (e.IsFromOutsideSource)
    {
        var draggingRecord = e.Data.GetData("Nodes") as
            ObservableCollection<TreeNode>;
    }
}

```

```

var record = draggingRecord[0].Item as EmployeeInfo;
int dropIndex = (int)e.TargetRecord;
var dropPosition = e.DropPosition.ToString();
if (record.Title == "Manager")
{
    e.Handled = true;
    return;
}
IList collection = null;
collection = AssociatedObject.sfDataGrid.View.SourceCollection as IList;
if (dropPosition == "DropAbove")
{
    dropIndex--;
    collection.Insert(dropIndex, record);
}
else
{
    dropIndex++;
    collection.Insert(dropIndex, record);
}
AssociatedObject.sfTreeGrid.View.Remove(record);
e.Handled = true;
}
}

```

	FirstName	LastName	ID	Title	Salary	ReportsTo
<input type="checkbox"/>	Ferando	Joseph	2	Management	2000000.00	-1
	Andrew	Fuller	9	Vice President	1200000.00	2
	Janet	Leverling	10	GM	1000000.00	2
	Steven	Buchanan	11	Manager	900000.00	2
<input type="checkbox"/>	John	Adams	3	Accounts	2000000.00	-1
>	Nancy	Davolio	12	Accounts Manager	850000.00	3
	Margaret	Peacock	13	Accountant	700000.00	3
	Michael	Suyama	14	Accountant	700000.00	3
	Robert	King	15	Accountant	650000.00	3

	FirstName	LastName	ID	Title	Salary	ReportsTo
	Susmi	Joseph	800	Management	2000000.00	1
	Ramya	Fuller	100	Vice President	1200000.00	2
	Priya	Leverling			1000000.00	2
	Vicky	Buchanan			900000.00	8
	Nancy	Davolio			850000.00	4

You can download the sample [here](#).

## Merge Cells in WPF DataGrid (SfDataGrid)

DataGrid allows you to merge the range of adjacent cells using [QueryCoveredRange](#) event. Merged cells can be exported and printed.

[QueryCoveredRange](#) event occurs when each cell gets arranged and the custom range will be stored for visible rows and columns in [SfDataGrid.CoveredCells](#). This event is not fired for the cells that are not visible and also for the cells that are already in [SfDataGrid.CoveredCells](#). When scrolling the merged

range will be added for newly added rows & columns through this event and also removed for the rows & columns which are out of view.

[GridQueryCoveredRangeEventArgs](#) of the [QueryCoveredRange](#) event provides information about the cell triggered this event. [GridQueryCoveredRangeEventArgs.OriginalSender](#) returns the DataGrid fired this event for DetailsView. By [GridQueryCoveredRangeEventArgs.Range](#) property, the adjacent cells can be merged.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
SelectionUnit="Cell"
NavigationMode="Cell"
QueryCoveredRange="dataGrid_QueryCoveredRange"/>
```

#### C#

```
dataGrid.QueryCoveredRange += dataGrid_QueryCoveredRange;
dataGrid.SelectionUnit = GridSelectionUnit.Cell;
dataGrid.NavigationMode = Syncfusion.UI.Xaml.Grid.NavigationMode.Cell;
void dataGrid_QueryCoveredRange(object sender,
GridQueryCoveredRangeEventArgs e)
{
}
```

#### Merging cells

You can span a cell in a row and column by merging the range of cells by setting [CoveredCellInfo](#) (by defining Left, Right, Top & Bottom) to [GridQueryCoveredRangeEventArgs.Range](#) and handling the event.

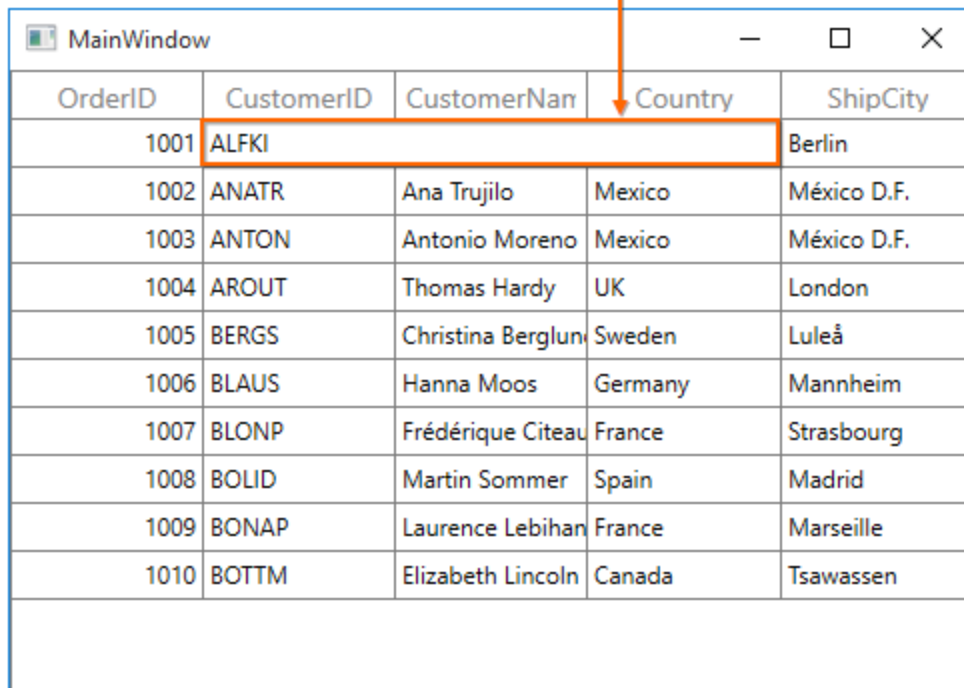
##### *Merging cells horizontally by fixed range*

You can merge the columns in a row by setting the column range using Left and Right properties of [CoveredCellInfo](#).

#### C#

```
void dataGrid_QueryCoveredRange(object sender,
GridQueryCoveredRangeEventArgs e)
{
    if (e.RowColumnIndex.RowIndex == 1)
    {
        if (e.RowColumnIndex.ColumnIndex >= 1 && e.RowColumnIndex.ColumnIndex <= 3)
        {
            e.Range = new CoveredCellInfo(1, 3, 1, 1);
            e.Handled = true;
        }
    }
}
```

Cells merged horizontally



OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI			Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeau	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Leblond	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

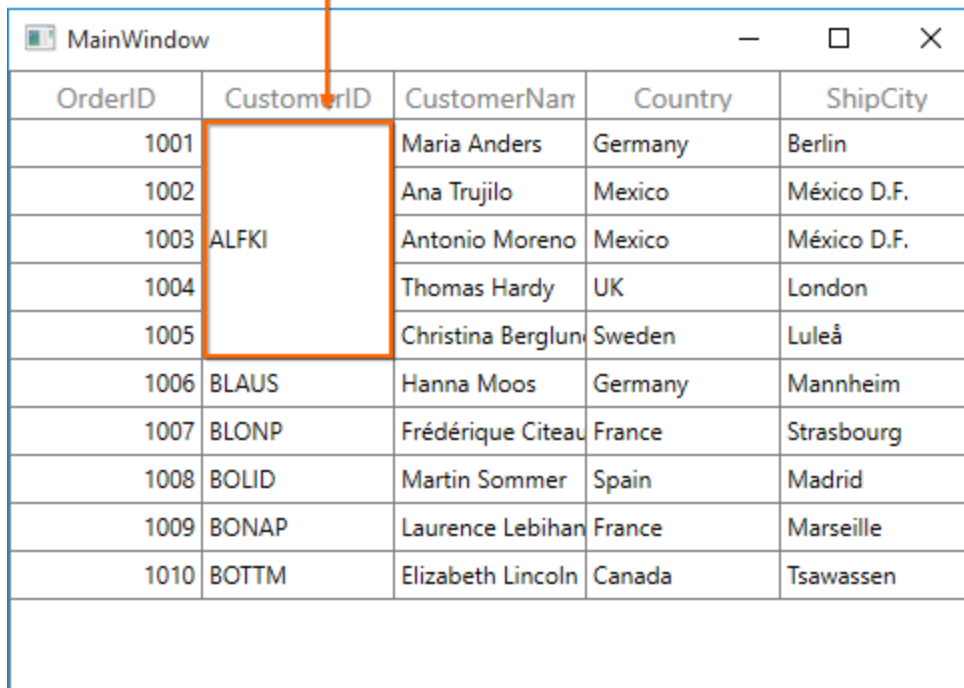
*Merging cells vertically by fixed range*

You can merge the range of rows for a particular column by setting the row range using Top and Bottom properties of [CoveredCellInfo](#).

**C#**

```
void dataGrid_QueryCoveredRange(object sender,
GridQueryCoveredRangeEventArgs e)
{
    if (e.RowColumnIndex.ColumnIndex == 1)
    {
        if (e.RowColumnIndex.RowIndex >= 1 && e.RowColumnIndex.RowIndex <= 5)
        {
            e.Range = new CoveredCellInfo(1, 1, 1, 5);
            e.Handled = true;
        }
    }
}
```

Cells merged vertically



OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002		Ana Trujilo	Mexico	México D.F.
1003		Antonio Moreno	Mexico	México D.F.
1004		Thomas Hardy	UK	London
1005		Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeau	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Leblanc	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

*Merging range of cells*

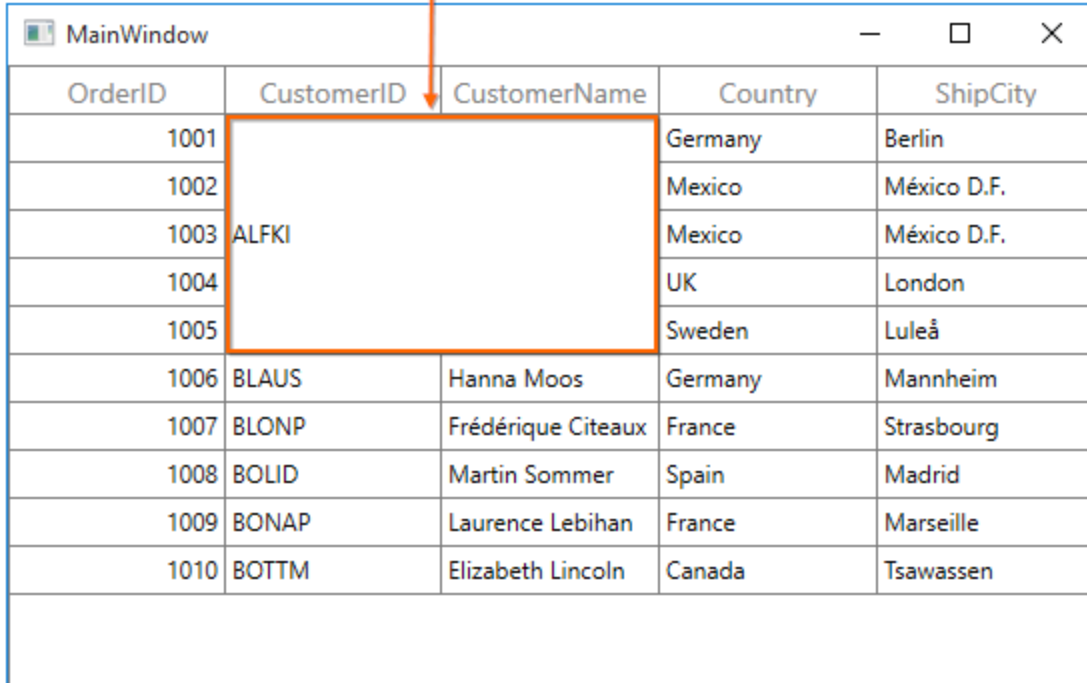
You can merge the range of rows and columns by setting the range using Left, Right, Top and Bottom properties of [CoveredCellInfo](#).

**C#**

```
private void dataGrid_QueryCoveredRange(object sender,
GridQueryCoveredRangeEventArgs e)
{
    //Merge range
    if (e.RowColumnIndex.ColumnIndex == 1 && e.RowColumnIndex.RowIndex == 1)
    {
        e.Range = new CoveredCellInfo(1, 2, 1, 5);
        e.Handled = true;
    }
}
```



**Merged range of cells**



OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI		Germany	Berlin
1002			Mexico	México D.F.
1003			Mexico	México D.F.
1004			UK	London
1005			Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

### Merging cells based on the content

You can merge the redundant data in adjacent cells in a row or columns using [QueryCoveredRange](#) event.

In the below code, `GetRange` method returns range for a cell based on adjacent cells content. From range from `GetRange` method `QueryCoveredRange` handler sets the range if the calculated range is already not exist in the [SfDataGrid.CoveredCells](#) using [CoveredCells.IsInRange](#) method.

### C#

```
dataGrid.ItemsSourceChanged += dataGrid_ItemsSourceChanged;
dataGrid.QueryCoveredRange += dataGrid_QueryCoveredRange;
/// <summary>
/// Reflector for SfDataGrid's data.
/// </summary>
IPropertyAccessProvider reflector = null;
/// <summary>
/// ItemsSourceChanged event handler.
/// </summary>
void dataGrid_ItemsSourceChanged(object sender,
GridItemsSourceChangedEventArgs e)
{
    if (dataGrid.View != null)
        reflector = dataGrid.View.GetPropertyAccessProvider();
    else
        reflector = null;
}
/// <summary>
```

```

/// QueryCoveredRange event handler
/// </summary>
void dataGrid_QueryCoveredRange(object sender,
GridQueryCoveredRangeEventArgs e)
{
    var range = GetRange(e.GridColumn, e.RowColumnIndex.RowIndex,
e.RowColumnIndex.ColumnIndex, e.Record);
    if (range == null)
        return;
    // You can know that the range is already exist in Covered Cells by
IsInRange method.
    if (!dataGrid.CoveredCells.IsInRange(range))
    {
        e.Range = range;
        e.Handled = true;
    }
    //If the calculated range is already exist in CoveredCells, you can get the
range using SfDataGrid.GetConflictRange (CoveredCellInfo coveredCellInfo)
extension method.
}
/// <summary>
/// Method to get the covered range based on cell value.
/// </summary>
/// <param name="column"></param>
/// <param name="rowIndex"></param>
/// <param name="columnIndex"></param>
/// <param name="rowData"></param>
/// <returns> Compares the adjacent cell value and returns the range
</returns>
/// <remark> If the method find that the adjacent values are equal by
horizontal then it will merge vertically. And vice versa</remarks>
private CoveredCellInfo GetRange.GridColumn column, int rowIndex, int
columnIndex, object rowData)
{
    var range = new CoveredCellInfo(columnIndex, columnIndex, rowIndex,
rowIndex);
    object data = reflector.GetFormattedValue(rowData, column.MappingName);
    GridColumn leftColumn = null;
    GridColumn rightColumn = null;
    // total rows count.
    int recordsCount = this.dataGrid.GroupColumnDescriptions.Count != 0 ?
(this.dataGrid.View.TopLevelGroup.DisplayElements.Count +
this.dataGrid.TableSummaryRows.Count + this.dataGrid.UnBoundRows.Count +
(this.dataGrid.AddNewRowPosition == AddNewRowPosition.Top ? +1 : 0)) :
(this.dataGrid.View.Records.Count + this.dataGrid.TableSummaryRows.Count +
this.dataGrid.UnBoundRows.Count + (this.dataGrid.AddNewRowPosition ==
AddNewRowPosition.Top ? +1 : 0));
    // Merge Horizontally
    // compare right column
    for (int i = dataGrid.Columns.IndexOf(column); i <
this.dataGrid.Columns.Count - 1; i++)
    {
        var compareData = reflector.GetFormattedValue(rowData, dataGrid.Columns[i +
1].MappingName);
        if (compareData == null)
            break;
        if (!compareData.Equals(data))

```

```

break;
rightColumn = dataGrid.Columns[i + 1];
}
// compare left column.
for (int i = dataGrid.Columns.IndexOf(column); i > 0; i--)
{
var compareData = reflector.GetFormattedValue(rowData, dataGrid.Columns[i - 1].MappingName);
if (compareData == null)
break;
if (!compareData.Equals(data))
break;
leftColumn = dataGrid.Columns[i - 1];
}
if (leftColumn != null || rightColumn != null)
{
// set left index
if (leftColumn != null)
{
var leftColumnIndex =
this.dataGrid.ResolveToScrollColumnIndex(this.dataGrid.Columns.IndexOf(leftC
olumn));
range = new CoveredCellInfo(leftColumnIndex, range.Right, range.Top,
range.Bottom);
}
// set right index
if (rightColumn != null)
{
var rightColumnIndex =
this.dataGrid.ResolveToScrollColumnIndex(this.dataGrid.Columns.IndexOf(right
Column));
range = new CoveredCellInfo(range.Left, rightColumnIndex, range.Top,
range.Bottom);
}
return range;
}
// Merge Vertically from the row index.
int previousRowIndex = -1;
int nextRowIndex = -1;
// Get previous row data.
var startIndex = dataGrid.ResolveStartIndexBasedOnPosition();
for (int i = rowIndex - 1; i >= startIndex; i--)
{
var previousData = this.dataGrid.GetRecordEntryAtRowIndex(i);
if (previousData == null || !previousData.IsRecords)
break;
var compareData = reflector.GetFormattedValue((previousData as
RecordEntry).Data, column.MappingName);
if (compareData == null)
break;
if (!compareData.Equals(data))
break;
previousRowIndex = i;
}
// get next row data.
for (int i = rowIndex + 1; i < recordsCount + 1; i++)
{

```

```

var nextData = this.dataGrid.GetRecordEntryAtRowIndex(i);
if (nextData == null || !nextData.IsRecords)
break;
var compareData = reflector.GetFormattedValue((nextData as
RecordEntry).Data, column.MappingName);
if (compareData == null)
break;
if (!compareData.Equals(data))
break;
nextRowIndex = i;
}
if (previousRowIndex != -1 || nextRowIndex != -1)
{
if (previousRowIndex != -1)
range = new CoveredCellInfo(range.Left, range.Right, previousRowIndex,
range.Bottom);
if (nextRowIndex != -1)
range = new CoveredCellInfo(range.Left, range.Right, range.Top,
nextRowIndex);
return range;
}
return null;
}

```

### Merge cells in Master-Details View

Master- details view allows you to merge the range of cells using the [QueryCoveredRange](#) event of [ViewDefinition.DataGrid](#). You can get the **DetailsViewDataGrid** which triggered the event from [GridQueryCoveredRangeEventArgs.OriginalSender](#) of the [QueryCoveredRange](#) event.

### XML

```

<syncfusion:SfDataGrid x:Name="dataGrid"
SelectionUnit="Cell"
NavigationMode="Cell"
AutoGenerateColumns="True" ColumnSizer="Star"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.DetailsViewDefinition>
<syncfusion:GridViewDefinition RelationalColumn="ProductDetails">
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AutoGenerateColumns="True"
QueryCoveredRange="FirstLevelNestedGrid_QueryCoveredRange"
/>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>

```

### C#

```

dataGrid.SelectionUnit = GridSelectionUnit.Cell;
dataGrid.NavigationMode = Syncfusion.UI.Xaml.Grid.NavigationMode.Cell;
FirstLevelNestedGrid.QueryCoveredRange
+=FirstLevelNestedGrid_QueryCoveredRange;

```

```
private void FirstLevelNestedGrid_QueryCoveredRange(object sender,
GridQueryCoveredRangeEventArgs e)
{
    if (e.RowColumnIndex.ColumnIndex == 1)
    {
        if (e.RowColumnIndex.RowIndex >= 2 && e.RowColumnIndex.RowIndex <= 3)
        {
            e.Range = new CoveredCellInfo(1, 1, 2, 3);
            e.Handled = true;
        }
    }
}
```

Cells merged in details view

	OrderID	CustomerID	CustomerName	Country	ShipCity
+	1001	ALFKI	Maria Anders	Germany	Berlin
	OrderID	ProductName	DateOfMonth		
	1001	Bike1	5/27/2015		
	1001	Bike2	5/20/2015		
	1001		5/27/2015		
+	1002	ANATR	Ana Trujilo	Mexico	México D.F.
+	1003	ANTON	Antonio Moreno	Mexico	México D.F.
+	1004	AROUT	Thomas Hardy	UK	London
+	1005	BERGS	Christina Berglund	Sweden	Luleå
+	1006	BLAUS	Hanna Moos	Germany	Mannheim
+	1007	BLONP	Frédérique Citeaux	France	Strasbourg
+	1008	BOLID	Martin Sommer	Spain	Madrid

### Merging range of parent cells

You can't vertically merge the cells of row when it has details view. You can check whether the row have details view or not using [CanMergeNextRows](#) extension method of [MergedCellHelper](#).

### C#

```
void dataGrid_QueryCoveredRange(object sender,
GridQueryCoveredRangeEventArgs e)
{
    if (!dataGrid.CanMergeNextRows(rowData))
    {
        return null;
    }
}
```

Refreshing the merged cells at runtime

#### *Add covered range*

You can add the range to [SfDataGrid.CoveredCells](#) at run time using [AddRange](#) extension method in [MergedCellHelper](#). You have to invalidate the [VisualContainer](#) by calling [InvalidateMeasureInfo](#) method which reflects the changes in UI.

#### **C#**

```
sfDataGrid.AddRange(new Syncfusion.UI.Xaml.Grid.CoveredCellInfo(2, 3, 3, 3));  
sfDataGrid.GetVisualContainer().InvalidateMeasureInfo();
```

#### *Remove covered range*

You can remove the range from [SfDataGrid.CoveredCells](#) at run time by [RemoveRange](#) extension method of [MergedCellHelper](#). You have to invalidate the [VisualContainer](#) by calling [InvalidateMeasureInfo](#) method which reflects the changes in UI.

#### **C#**

```
dataGrid.RemoveRange(new CoveredCellInfo(2, 4, 1, 1));  
dataGrid.GetVisualContainer().InvalidateMeasureInfo();
```

#### *Trigger QueryCoveredRange event programmatically*

You can trigger [SfDataGrid.QueryCoveredRange](#) event for particular cell by removing its range from [SfDataGrid.CoveredCells](#) and invalidating the [VisualContainer](#) by calling [InvalidateMeasureInfo](#) method. You can get range for particular cell in CoveredCells by passing row and column index to [SfDataGrid.CoveredCells.GetCoveredCellInfo](#) method.

For example, if you want to merge the adjacent cells with same content while editing then you can remove the range and refresh the container in [SfDataGrid.CurrentCellEndEdit](#) event.

#### **C#**

```
void dataGrid_CurrentCellEndEdit(object sender, CurrentCellEndEditEventArgs args)  
{  
    var rowIndex = args.RowColumnIndex.RowIndex;  
    var columnIndex = args.RowColumnIndex.ColumnIndex;  
    var range = dataGrid.CoveredCells.GetCoveredCellInfo(rowIndex, columnIndex);  
    dataGrid.RemoveRange(range);  
    dataGrid.GetVisualContainer().InvalidateMeasure();  
}
```

Exporting merged cells

#### *Export merged cells to Excel*

You can export the merged cells to excel by setting the [ExcelExportingOptions.ExportMergedCells](#) property.

#### **C#**

```
ExcelExportingOptions excelExportingOption = new ExcelExportingOptions();  
excelExportingOption.ExportMergedCells = true;
```

*Export merged cells to PDF*

You can export the merged cells to PDF by setting the [PdfExportingOptions.ExportMergedCells](#) property.

**C#**

```
PdfExportingOptions pdfExportingOption = new PdfExportingOptions();
pdfExportingOption.ExportMergedCells = true;
```

*Limitations*

Below are the limitation when using Cell Merging in SfDataGrid.

1. Row selection is not supported.
2. Heterogeneous rows can't be merged.
3. Cell loaded with Template Selector can't be merged.
4. [AllowFrozenGroupHeaders](#) is not supported.
5. With DetailsViewDefinition, Cell merging is not supported if [HideEmptyGridViewDefinition](#) is false or record has DetailsViewDataGrid.

How to

*Allow cell merging with Row Selection or NavigationMode as Row or AllowFrozenGroupHeaders is true*

SfDataGrid does not allow cell merging when [SelectionUnit](#) is Row or [NavigationMode](#) is Row or [AllowFrozenGroupHeaders](#) is true. You can overcome this behavior by setting [ExternalExceptionThrownEventArgs.Handled](#) to true using [ExternalExceptionThrown](#) event.

**C#**

```
dataGrid.ExternalExceptionThrown += dataGrid_ExternalExceptionThrown;
private void
dataGrid_ExternalExceptionThrown(ExternalExceptionThrownEventArgs args)
{
    if (args.Exception is NotSupportedException && args.Exception.Message ==
        "Merged Cell will not support when SfDataGrid.SelectionUnit is Row or Any,
        or SfDataGrid.NavigationMode is Row or SfDataGrid.AllowFrozenGroupedHeaders
        is true")
    {
        args.Handled = true;
    }
}
```

## Unbound Rows in WPF DataGrid (SfDataGrid)

SfDataGrid allows you to add **additional rows** at top and also bottom of the SfDataGrid which are **not bound with data object** from underlying data source. You can add unbound rows using [SfDataGrid.UnBoundRows](#) collection property. You can add any no of unbound rows to SfDataGrid. Unbound rows can be exported and printed.

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid" ItemsSource="{Binding Orders}">
  <syncfusion:SfDataGrid.UnBoundRows>
    <syncfusion:GridUnBoundRow Position="Top"/>
  </syncfusion:SfDataGrid.UnBoundRows>
</syncfusion:SfDataGrid>
```

**C#**

```
this.dataGrid.UnBoundRows.Add(new GridUnBoundRow() {Position =
UnBoundRowsPosition.Top});
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Leblond	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

**Note:** [AllowFrozenGroupHeaders](#) is not supported with unbound rows.

## Positioning unbound rows

Unbound row can be placed in top or bottom of the SfDataGrid. Unbound row positioned based on [GridUnBoundRow.Position](#) and [GridUnBoundRow.ShowBelowSummary](#) properties.

Below table shows the unbound row positioning based on property settings of **Position** and **ShowBelowSummary**.

UnBoundRowPosition	ShowBelowSummary	Position in DataGrid
Top	True	Unbound row placed at top, right above the record rows. In this position, unbound row is selectable and editable.
Top	False	Unbound row placed at top, right next to Header row. In this position, unbound row is not selectable, not editable and frozen when scrolling.
Bottom	True	Unbound row placed at bottom of SfDataGrid. In this position, unbound row is not selectable, not editable and frozen when scrolling.



Bottom	False	Unbound row placed at bottom, right below record rows. In this position, unbound row is selectable and editable.
--------	-------	--

Below screen shot shows different unbound rows placed in all possible positions.

Position="Top"  
ShowBelowSummary="False"

Position="Top"  
ShowBelowSummary="True"

Position="Bottom"  
ShowBelowSummary="False"

Position="Bottom"  
ShowBelowSummary="True"

Populating data for unbound rows

You can populate data for the unbound row by handling [QueryUnBoundRow](#) event of SfDataGrid. This event occurs for each cell in unbound row whenever the row gets refreshed.

[GridUnBoundRowEventArgs](#) of the [QueryUnBoundRow](#) event provides information about the cell triggered this event. [GridUnBoundRowEventArgs.OriginalSender](#) returns the DataGrid fired this event for DetailsView.

You can get or set the [GridUnBoundRowEventArgs.Value](#) property based on the [UnBoundAction](#). If [UnBoundAction](#) is [QueryData](#) then you can set the value for display. If the [UnBoundAction](#) is [CommitData](#) then you can get the edited value.

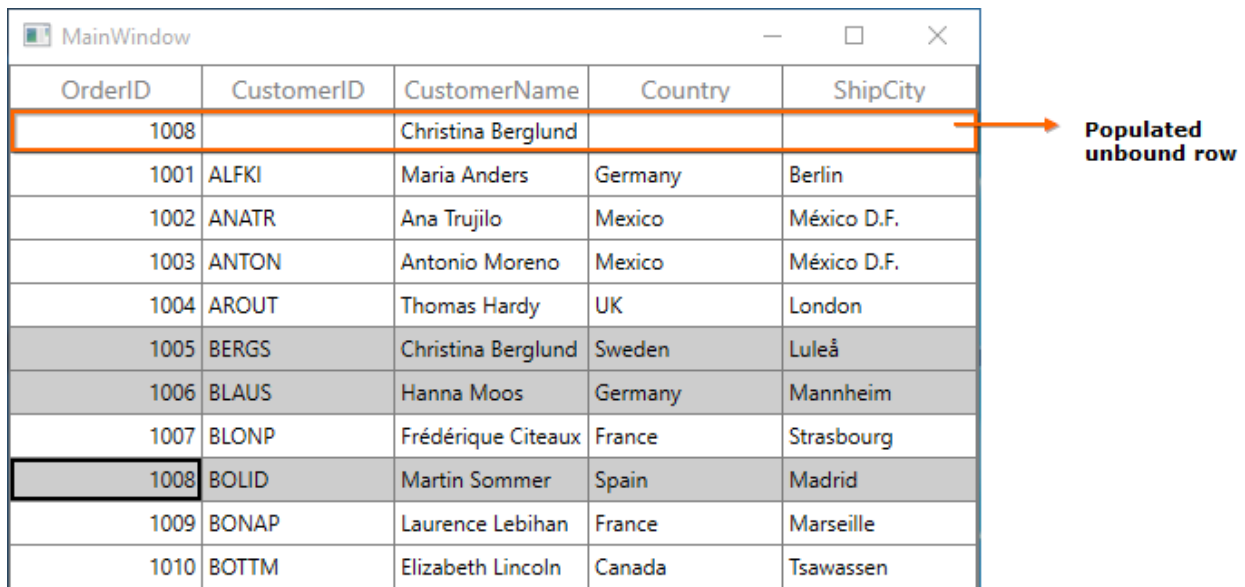
### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
SelectionMode="Multiple" >
<syncfusion:SfDataGrid.UnBoundRows>
<syncfusion:GridUnBoundRow Position="Top"/>
</syncfusion:SfDataGrid.UnBoundRows>
</syncfusion:SfDataGrid>
```

For example, now unbound row populated based on selected items in SfDataGrid.

### C#

```
this.dataGrid.SelectedItems.Add(collection[4]);
this.dataGrid.SelectedItems.Add(collection[5]);
this.dataGrid.SelectedItems.Add(collection[7]);
dataGrid.QueryUnBoundRow += dataGrid_QueryUnBoundRow;
void dataGrid_QueryUnBoundRow(object sender, GridUnBoundRowEventArgs e)
{
    if (e.UnBoundAction == UnBoundActions.QueryData)
    {
        if (e.RowColumnIndex.ColumnIndex == 0)
        {
            e.Value = (dataGrid.SelectedItems.OrderBy(item => (item as
            OrderInfo).OrderID).Last() as OrderInfo).OrderID;
            e.Handled = true;
        }
        else if (e.RowColumnIndex.ColumnIndex == 2)
        {
            e.Value = (dataGrid.SelectedItems.First(item => (item as
            OrderInfo).CustomerName.Contains("g")) as OrderInfo).CustomerName;
            e.Handled = true;
        }
    }
}
```



OrderID	CustomerID	CustomerName	Country	ShipCity
1008		Christina Berglund		
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

Refreshing the Unbound Rows at runtime

*Add/Remove unbound rows*

You can add or remove unbound rows using [SfDataGrid.UnBoundRows](#) property which reflects in UI immediately.

*Trigger QueryUnBoundRow event programmatically*

You can trigger the [QueryUnBoundRow](#) event for the unbound row cells at runtime by invalidating the unbound row by calling [SfDataGrid.InValidateUnBoundRow](#) method and invalidating the [VisualContainer](#) by calling [InvalidateMeasureInfo](#) method.

**C#**

```
using Syncfusion.UI.Xaml.Grid.Helpers;
dataGrid.InValidateUnBoundRow(dataGrid.UnBoundRows[0]);
dataGrid.GetVisualContainer().InvalidateMeasureInfo();
```

## Editing in unbound rows

*Cancel the editing for unbound row cell*

You can cancel the editing of unbound row cell by handling the [SfDataGrid.CurrentCellBeginEdit](#) event with the help of [SfDataGrid.GetUnBoundRow](#) method and row index.

**C#**

```
using Syncfusion.UI.Xaml.Grid;
dataGrid.CurrentCellBeginEdit += dataGrid_CurrentCellBeginEdit;
void dataGrid_CurrentCellBeginEdit(object sender,
CurrentCellBeginEditEventArgs args)
{
    var unboundRow = dataGrid.GetUnBoundRow(args.RowColumnIndex.RowIndex);
    if (unboundRow == null)
        return;
    args.Cancel = true;
}
```

*Saving edited unbound row cell value to external source*

You can get the edited value of unbound row cell from [GridUnBoundRowEventArgs.Value](#) property of [QueryUnBoundRow](#) event when [UnBoundAction](#) is [CommitData](#).

**C#**

```
void dataGrid_QueryUnBoundRow(object sender, GridUnBoundRowEventArgs e)
{
    if (e.UnBoundAction == UnBoundActions.CommitData)
    {
        var editedValue = e.Value;
    }
}
```

## Styling in Unbound rows

*Unbound row style*

You can customize the style of unbound row by writing style of TargetType [UnBoundRowControl](#) or setting [SfDataGrid.UnBoundRowStyle](#) property.

**XML**

```
<Window.Resources>
<Style TargetType="syncfusion:UnBoundRowControl">
<Setter Property="FontWeight" Value="Bold"/>
</Style>
```

```

</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
SelectionMode="Multiple">
<syncfusion:SfDataGrid.UnBoundRows>
<syncfusion:GridUnBoundRow Position="Top"/>
</syncfusion:SfDataGrid.UnBoundRows>
</syncfusion:SfDataGrid>

```

OrderID	CustomerID	CustomerName	Country	ShipCity
1008		Christina Berglund		
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

#### Unbound row cell's style

You can customize the style of unbound row cell by writing style of TargetType [GridUnBoundRowCell](#) or setting [SfDataGrid.UnBoundRowCellStyle](#) property.

#### XML

```

<Window.Resources>
<local:UnboundCellStyleConverter x:Key="unboundRowCellStyleConverter"/>
<Style TargetType="syncfusion:GridUnBoundRowCell">
<Setter Property="FontWeight" Value="{Binding
RelativeSource={RelativeSource Self}, Converter={StaticResource
unboundRowCellStyleConverter}}"/>
</Style>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
ItemsSource="{Binding Orders}"
SelectionMode="Multiple">
<syncfusion:SfDataGrid.UnBoundRows>
<syncfusion:GridUnBoundRow Position="Top"/>
</syncfusion:SfDataGrid.UnBoundRows>
</syncfusion:SfDataGrid>

```

#### C#

```

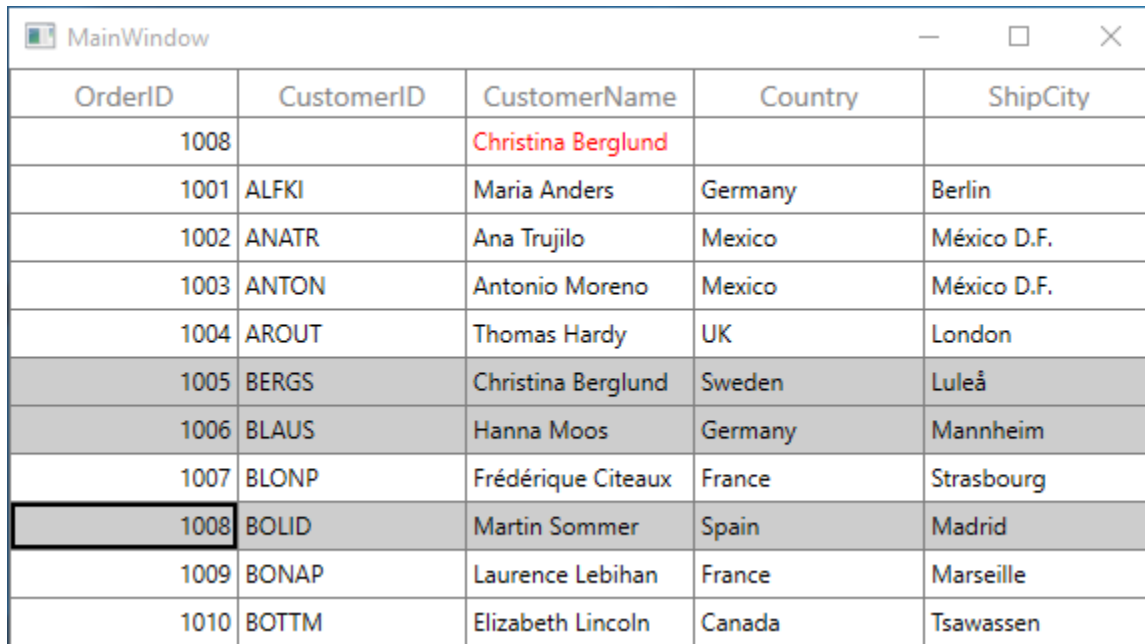
public class UnboundCellStyleConverter : IValueConverter

```

```

{
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        var unboundRowCell = value as GridUnBoundRowCell;
        if (unboundRowCell == null ||
            unboundRowCell.ColumnBase.GridUnBoundRowEventArgs.Value == null) return
            null;
        if (unboundRowCell.ColumnBase.GridUnBoundRowEventArgs.Value.ToString().Conta
            ins("g"))
            return new SolidColorBrush(Colors.Red);
        return new SolidColorBrush(Colors.Black);
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        return value;
    }
}

```



OrderID	CustomerID	CustomerName	Country	ShipCity
1008		Christina Berglund		
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

### Customize the Unbound Row's behavior

SfDataGrid allows you to customize the operations like key navigation and UI related interactions by overriding the corresponding renderer associated with the unbound row cell. Each renderer have set of virtual methods for handling the behaviors. Creating new renderers also supported.

Below table lists the available cell types for unbound row and its renderers.

Cell Type	Renderer
UnBoundTemplateColumn	<a href="#">GridUnBoundRowCellTemplateRenderer</a>
UnBoundTextColumn	<a href="#">GridUnBoundRowCellTextBoxRenderer</a>

The renderer of unbound row cell defined by [GridUnBoundRowEventArgs.CellType](#) property in the [QueryUnBoundRow](#) event. If the [GridUnBoundRowEventArgs.CellType](#) not defined then the [UnBoundTextColumn](#) set as default cell type of [GridUnBoundRowCell](#).

If [GridUnBoundRowEventArgs.CellTemplate](#) and [GridUnBoundRowEventArgs.EditTemplate](#) properties defined then [UnBoundTemplateColumn](#) set as cell type of [GridUnBoundRowCell](#).

#### *Overriding Existing CellType*

You can customize the unbound row cell behavior by overriding existing renderer and replace the default one in [SfDataGrid.UnBoundRowCellRenderers](#).

Below [GridUnBoundRowCellTextBoxRenderer](#) is customized to change the foreground.

#### **C#**

```
public class GridUnBoundRowCellTextBoxRendererExt :
    GridUnBoundRowCellTextBoxRenderer
{
    public override void OnInitializeDisplayElement(DataColumnBase dataColumn,
        TextBlock uiElement, object dataContext)
    {
        base.OnInitializeDisplayElement(dataColumn, uiElement, dataContext);
        var cellValue = dataColumn.GridUnBoundRowEventArgs != null &&
            dataColumn.GridUnBoundRowEventArgs.Value != null ?
            dataColumn.GridUnBoundRowEventArgs.Value.ToString() :
            string.Empty;
        uiElement.Text = cellValue;
        uiElement.Foreground = new SolidColorBrush(Colors.Orange);
    }
    public override void OnInitializeEditElement(DataColumnBase dataColumn,
        TextBox uiElement, object dataContext)
    {
        base.OnInitializeEditElement(dataColumn, uiElement, dataContext);
        var cellValue = (dataColumn.GridUnBoundRowEventArgs != null &&
            dataColumn.GridUnBoundRowEventArgs.Value != null) ?
            dataColumn.GridUnBoundRowEventArgs.Value.ToString() :
            string.Empty;
        uiElement.Text = cellValue.ToString();
    }
}
```

In the below code default renderer replaced using the above custom renderer in [SfDataGrid.UnBoundRowCellRenderers](#).

#### **C#**

```
dataGrid.UnBoundRowCellRenderers.Remove("UnBoundTextColumn");
dataGrid.UnBoundRowCellRenderers.Add("UnBoundTextColumn", new
    GridUnBoundRowCellTextBoxRendererExt());
```

MainWindow				
OrderID	CustomerID	CustomerName	Country	ShipCity
1008		Christina Berglund		
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

### Custom Renderer

You can customize the unbound row cell by creating new renderer, deriving from [GridUnBoundRowCellRenderer](#) and setting the [GridUnBoundRowEventArgs.CellType](#) property.

Below code creates `DatePickerController` to load the [DatePicker](#) as editor element in the first cell of unbound row.

### C#

```
public class DatePickerController : GridUnBoundRowCellRenderer<TextBlock,
DatePicker>
{
    /// <summary>
    /// Constructor of the renderer.
    /// </summary>
    public DatePickerController()
    {
    }

    /// <summary>
    /// Display element creation.
    /// </summary>
    /// <returns></returns>
    protected override TextBlock OnCreateDisplayUIElement()
    {
        return new TextBlock();
    }

    /// <summary>
    /// Edit Element creation.
    /// </summary>
    /// <returns></returns>
    protected override DatePicker OnCreateEditUIElement()
    {
        return new DatePicker();
    }

    /// <summary>
```

```

/// Initialize the value for display element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnInitializeDisplayElement(DataColumnBase dataColumn,
TextBlock uiElement, object dataContext)
{
uiElement.Text = dataColumn.GridUnBoundRowEventArgs.Value.ToString();
}
/// <summary>
/// Updates the value for display element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnUpdateDisplayBinding(DataColumnBase dataColumn,
TextBlock uiElement, object dataContext)
{
uiElement.Text = dataColumn.GridUnBoundRowEventArgs.Value.ToString();
}
#region Edit Element
/// <summary>
/// Initialize the value for edit element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="uiElement"></param>
/// <param name="dataContext"></param>
public override void OnInitializeEditElement(DataColumnBase dataColumn,
DatePicker uiElement, object dataContext)
{
uiElement.SelectedDate =
(DateTime?)dataColumn.GridUnBoundRowEventArgs.Value;
uiElement.Tag = dataColumn;
}
/// <summary>
/// Updates the value for edit element.
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="element"></param>
/// <param name="dataContext"></param>
public override void OnUpdateEditBinding(DataColumnBase dataColumn,
DatePicker element, object dataContext)
{
element.SelectedDate = (DateTime?)dataColumn.GridUnBoundRowEventArgs.Value;
element.Tag = dataColumn;
}
/// <summary>
/// Method to wire the Selection Changed event
/// </summary>
/// <param name="uiElement"></param>
protected override void OnWireEditUIElement(DatePicker uiElement)
{
base.OnWireEditUIElement(uiElement);
uiElement.SelectedDateChanged += uiElement_SelectedDateChanged;
}
/// <summary>

```



```

/// Method to un wire the Selection Changed event.
/// </summary>
/// <param name="uiElement"></param>
protected override void OnUnwireEditUIElement(DatePicker uiElement)
{
    base.OnUnwireEditUIElement(uiElement);
    uiElement.SelectedDateChanged -= uiElement_SelectedDateChanged;
}
/// <summary>
/// Method to raise the CurrentCellValueChangedEvent.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
public void uiElement_SelectedDateChanged(object sender,
System.Windows.Controls.SelectionChangedEventArgs e)
{
    var datePicker = sender as DatePicker;
    if (datePicker.Tag is DataColumn)
    {
        var dataColumn = datePicker.Tag as DataColumn;
        dataColumn.GridUnBoundRowEventArgs.Value = (sender as
DatePicker).SelectedDate;
        DataGrid.RaiseQueryUnBoundRow(dataColumn.GridUnBoundRowEventArgs.GridUnbound
dRow, UnBoundActions.CommitData, dataColumn.GridUnBoundRowEventArgs.Value,
dataColumn.GridColumn, dataColumn.GridUnBoundRowEventArgs.CellType, new
Syncfusion.UI.Xaml.ScrollAxis.RowColumnIndex(dataColumn.RowIndex,
dataColumn.ColumnIndex));
    }
}
#endregion
#region Update
/// <summary>
/// Update display value and raise event
/// </summary>
/// <param name="dataColumn"></param>
/// <param name="currentRendererElement"></param>
protected override void OnEditingComplete(DataColumnBase dataColumn,
FrameworkElement currentRendererElement)
{
    dataColumn.GridUnBoundRowEventArgs.Value = (currentRendererElement as
DatePicker).Text;
    DataGrid.RaiseQueryUnBoundRow(dataColumn.GridUnBoundRowEventArgs.GridUnbound
dRow, UnBoundActions.CommitData, dataColumn.GridUnBoundRowEventArgs.Value,
dataColumn.GridColumn, dataColumn.GridUnBoundRowEventArgs.CellType, new
Syncfusion.UI.Xaml.ScrollAxis.RowColumnIndex(dataColumn.RowIndex,
dataColumn.ColumnIndex));
}
#endregion
}

```

In the below code newly created renderer added in [SfDataGrid.UnBoundRowCellRenderers](#).

#### C#

```

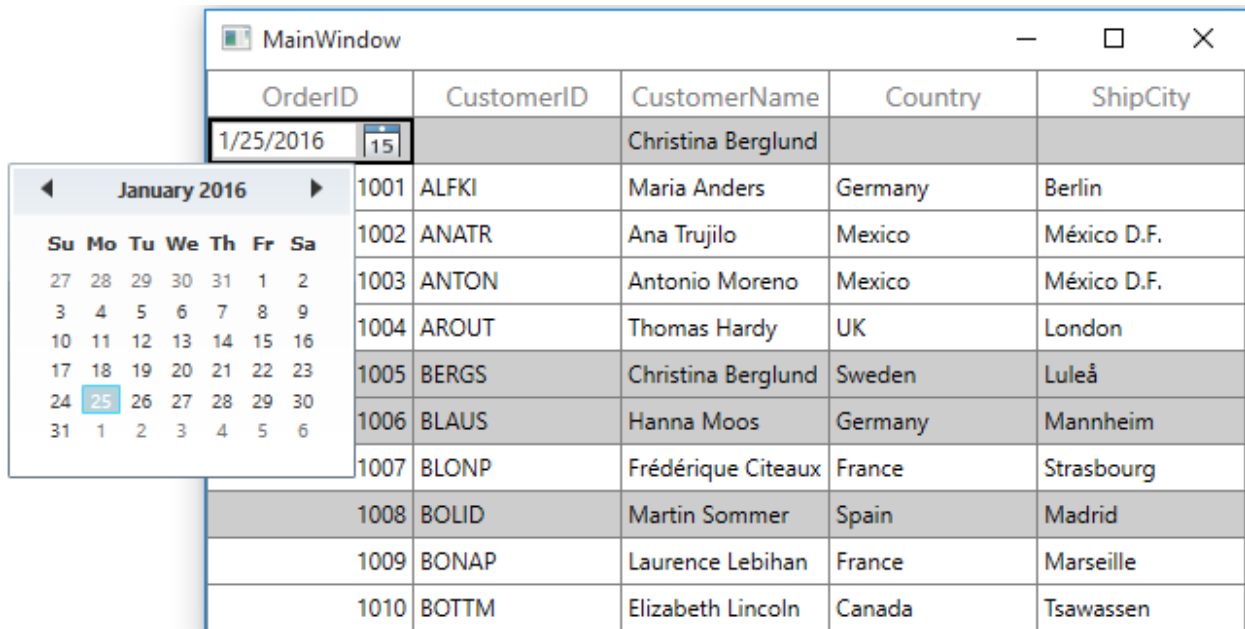
dataGrid.UnBoundRowCellRenderers.Add("DatePickerRenderer", new
DatePickerRenderer());

```

Below code sets the [CellType](#) as `DatePickerRenderer`.

### C#

```
void dataGrid_QueryUnBoundRow(object sender, GridUnBoundRowEventArgs e)
{
    if (e.UnBoundAction == UnBoundActions.QueryData)
    {
        if (e.RowColumnIndex.ColumnIndex == 0)
        {
            e.CellType = "DatePickerRenderer";
            e.Value = DateTime.Now;
            e.Handled = true;
        }
        else if (e.RowColumnIndex.ColumnIndex == 2)
        {
            e.Value = (dataGrid.SelectedItems.First(item => (item as OrderInfo).CustomerName.Contains("g")) as OrderInfo).CustomerName;
            e.Handled = true;
        }
    }
    if (e.UnBoundAction == UnBoundActions.CommitData)
    {
        date = (DateTime)e.Value;
    }
}
```



### Templating unbound row cells

You can customize the unbound row cells using [GridUnBoundRowEventArgs.CellTemplate](#) property.

### XML

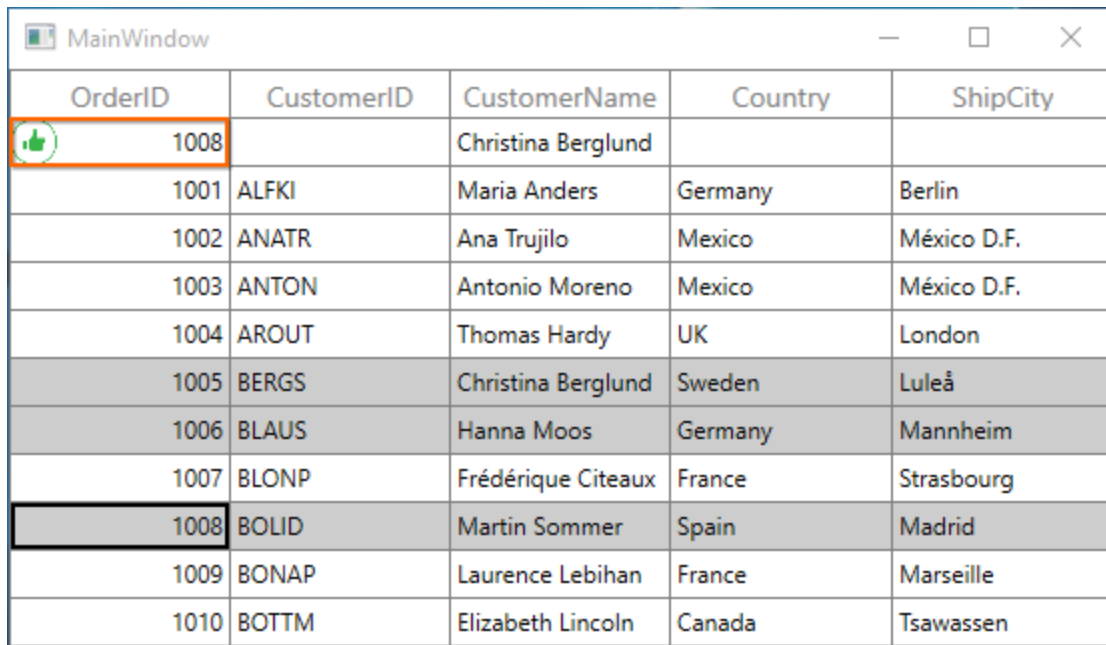
```
<DataTemplate x:Key="UnBoundRowCellTemplate">
    <Grid>
    <Grid.ColumnDefinitions>
```


```
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="{Binding Path=}"
Grid.Column="1" Margin="0,0,3,0"
TextWrapping="Wrap"
VerticalAlignment="Center"
HorizontalAlignment="Right" />
<Image Source="\Images\thumb_yes.png" HorizontalAlignment="Left" />
</Grid>
</DataTemplate>
```

<br/>

### C#

```
void dataGrid_QueryUnBoundRow(object sender, GridUnBoundRowEventArgs e)
{
    if (e.UnBoundAction == UnBoundActions.QueryData)
    {
        if (e.RowColumnIndex.ColumnIndex == 0)
        {
            e.CellType = "UnBoundTemplateColumn";
            e.CellTemplate = App.Current.Resources["UnBoundRowCellTemplate"] as
            DataTemplate;
            e.Value = (dataGrid.SelectedItems.OrderBy(item => (item as
            OrderInfo).OrderID).Last() as OrderInfo).OrderID;
            e.Handled = true;
        }
        else if (e.RowColumnIndex.ColumnIndex == 2)
        {
            e.Value = (dataGrid.SelectedItems.First(item => (item as
            OrderInfo).CustomerName.Contains("g")) as OrderInfo).CustomerName;
            e.Handled = true;
        }
    }
}
```



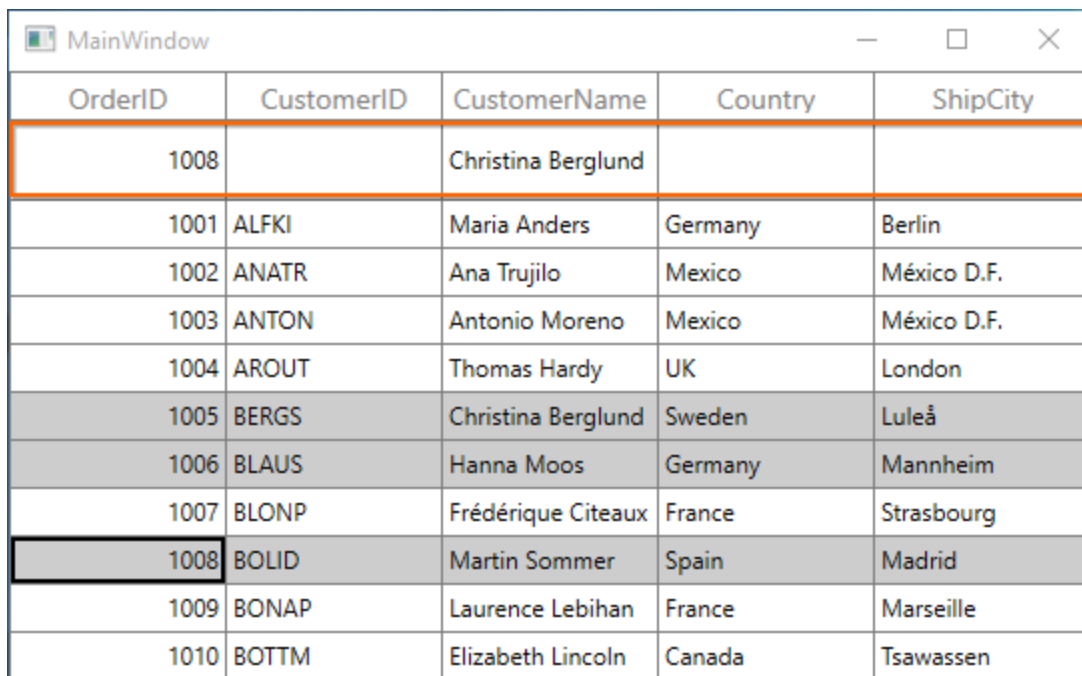
OrderID	CustomerID	CustomerName	Country	ShipCity
 1008		Christina Berglund		
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

### Changing unbound row height

You can change the height of unbound row using [SfDataGrid.QueryRowHeight](#) event.

### C#

```
using Syncfusion.UI.Xaml.Grid;
dataGrid.QueryRowHeight += dataGrid_QueryRowHeight;
void dataGrid_QueryRowHeight(object sender, QueryRowHeightEventArgs e)
{
    if (dataGrid.IsUnBoundRow(e.RowIndex))
    {
        e.Height = 40;
        e.Handled = true;
    }
}
```



OrderID	CustomerID	CustomerName	Country	ShipCity
1008		Christina Berglund		
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

Exporting Unbound rows

*Export unbound rows to Excel*

You can export the unbound rows to excel by setting the [ExcelExportingOptions.ExportUnBoundRows](#) property.

**C#**

```
ExcelExportingOptions excelExportingOption = new ExcelExportingOptions();
excelExportingOption.ExportUnBoundRows = true;
```

*Export unbound rows to PDF*

You can export the unbound rows to PDF by setting the [PdfExportingOptions.ExportUnBoundRows](#) property.

**C#**

```
PdfExportingOptions pdfExportingOption = new PdfExportingOptions();
pdfExportingOption.ExportUnBoundRows = true;
```

Get unbound rows

You can get the unbound row of specified row index using [GetUnBoundRow](#) method.

**C#**

```
using Syncfusion.UI.Xaml.Grid;
var unboundRow = dataGrid.GetUnBoundRow(1);
```

Merging with Unbound rows

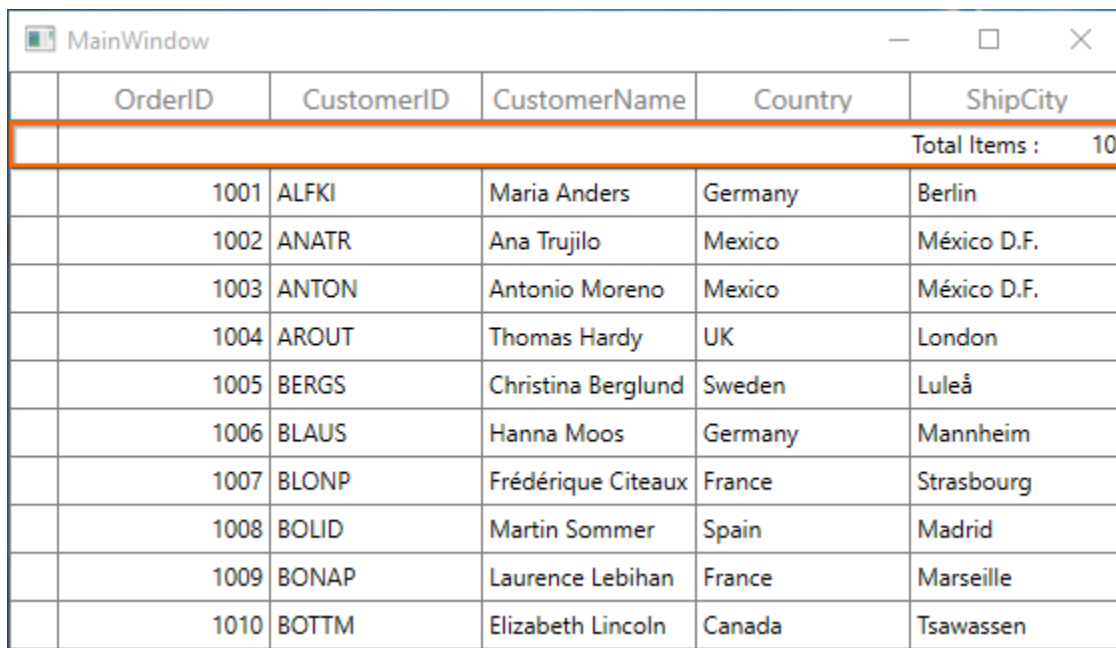
You can merge the unbound row cell by setting the Left, Right, Top and Bottom properties of [CoveredCellInfo](#) with the help of [GetUnBoundRow](#) method and RowIndex.

**C#**

```

using Syncfusion.UI.Xaml.Grid;
dataGrid.QueryCoveredRange += dataGrid_QueryCoveredRange;
void dataGrid_QueryCoveredRange(object sender,
GridQueryCoveredRangeEventArgs e)
{
    var unboundRow = this.dataGrid.GetUnBoundRow(e.RowColumnIndex.RowIndex);
    if(unboundRow == null)
        return;
    if(e.RowColumnIndex.ColumnIndex == 1)
    {
        e.Range = new CoveredCellInfo(e.RowColumnIndex.ColumnIndex,
        (e.OriginalSender as SfDataGrid).Columns.Count, e.RowColumnIndex.RowIndex,
        e.RowColumnIndex.RowIndex);
        e.Handled = true;
    }
}

```



	OrderID	CustomerID	CustomerName	Country	ShipCity
	Total Items :				10
	1001	ALFKI	Maria Anders	Germany	Berlin
	1002	ANATR	Ana Trujilo	Mexico	México D.F.
	1003	ANTON	Antonio Moreno	Mexico	México D.F.
	1004	AROUT	Thomas Hardy	UK	London
	1005	BERGS	Christina Berglund	Sweden	Luleå
	1006	BLAUS	Hanna Moos	Germany	Mannheim
	1007	BLONP	Frédérique Citeaux	France	Strasbourg
	1008	BOLID	Martin Sommer	Spain	Madrid
	1009	BONAP	Laurence Lebihan	France	Marseille
	1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

**Unbound row for Master-details view**

Master-details view also allows you to add [additional rows](#) to [ViewDefinition.DataGrid](#) which are **not bound with data object** from underlying data source.

You can get the DetailsViewDataGrid using [GridUnBoundRowEventArgs.OriginalSender](#) of the [QueryUnBoundRow](#) event, which fired the event and rendered in UI.

**XML**

```

<syncfusion:SfDataGrid x:Name="dataGrid"
    NavigationMode="Cell"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}">
    <syncfusion:SfDataGrid.DetailsViewDefinition>
    <syncfusion:GridViewDefinition RelationalColumn="ProductDetails">

```

```
<syncfusion:GridViewDefinition.DataGrid>
<syncfusion:SfDataGrid x:Name="FirstLevelNestedGrid"
AutoGenerateColumns="True">
<syncfusion:SfDataGrid.UnBoundRows>
<syncfusion:GridUnBoundRow Position="Top"
ShowBelowSummary="True"/>
</syncfusion:SfDataGrid.UnBoundRows>
</syncfusion:SfDataGrid>
</syncfusion:GridViewDefinition.DataGrid>
</syncfusion:GridViewDefinition>
</syncfusion:SfDataGrid.DetailsViewDefinition>
</syncfusion:SfDataGrid>
```

## C#

```
this.FirstLevelNestedGrid.QueryUnBoundRow +=
FirstLevelNestedGrid_QueryUnBoundRow;
void FirstLevelNestedGrid_QueryUnBoundRow(object sender,
GridUnBoundRowEventArgs e)
{
if (e.UnBoundAction == UnBoundActions.QueryData)
{
if (e.RowColumnIndex.ColumnIndex == 0)
{
e.Value = "Total Items";
e.Handled = true;
}
else if (e.RowColumnIndex.ColumnIndex == 1)
{
e.Value = (e.OriginalSender as SfDataGrid).View.Records.Count();
e.Handled = true;
}
}
}
```

MainWindow

	OrderID	CustomerID	CustomerName	Country	ShipCity															
+	1001	ALFKI	Maria Anders	Germany	Berlin															
-	1002	ANATR	Ana Trujilo	Mexico	México D.F.															
<table><tr><th>OrderID</th><th>ProductName</th><th>DateOfMonth</th></tr><tr><td>Total Items</td><td>3</td><td></td></tr><tr><td>1002</td><td>Bike5</td><td>5/7/2014</td></tr><tr><td>1002</td><td>Bike1</td><td>5/20/2012</td></tr><tr><td>1002</td><td>Bike3</td><td>5/27/2015</td></tr></table>						OrderID	ProductName	DateOfMonth	Total Items	3		1002	Bike5	5/7/2014	1002	Bike1	5/20/2012	1002	Bike3	5/27/2015
OrderID	ProductName	DateOfMonth																		
Total Items	3																			
1002	Bike5	5/7/2014																		
1002	Bike1	5/20/2012																		
1002	Bike3	5/27/2015																		
+	1003	ANTON	Antonio Moreno	Mexico	México D.F.															
+	1004	AROUT	Thomas Hardy	UK	London															
+	1005	BERGS	Christina Berglund	Sweden	Luleå															
+	1006	BLAUS	Hanna Moos	Germany	Mannheim															
+	1007	BLONP	Frédérique Citeaux	France	Strasbourg															

### Unbound Column in WPF DataGrid (SfDataGrid)

SfDataGrid allows you to add **additional columns** which are **not bound with data object** from underlying data source. You can add unbound column using [GridUnBoundColumn](#) class. Unbound columns supports for sorting, filtering, grouping, exporting and printing as normal columns.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridUnBoundColumn Expression="UnitPrice*Discount"
HeaderText="Discount Price"
MappingName="DiscountPrice" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### C#

```
this.dataGrid.Columns.Add(new GridUnBoundColumn() { HeaderText = "Discount
Price", MappingName = "DiscountPrice", Expression = "UnitPrice*Discount" });
```





OrderID	CustomerID	CustomerName	Discount Price	ShipCity
1001	ALFKI	Maria Anders	25	Berlin
1002	ANATR	Ana Trujillo	0	México D.F.
1003	ANTON	Antonio Moreno	75	México D.F.
1004	AROUT	Thomas Hardy	0	London
1005	BERGS	Christina Berglund	33	Luleå
1006	BLAUS	Hanna Moos	100	Mannheim
1007	BLONP	Frédérique Citeaux	75	Strasbourg
1008	BOLID	Martin Sommer	0	Madrid
1009	BONAP	Laurence Lebihan	25	Marseille
1010	BOTTM	Elizabeth Lincoln	100	Tsawassen

**Note:** It is mandatory to specify the `GridColumn.MappingName` for `GridUnBoundColumn` with some name to identify the column. It is not necessary to define name of field in the data object.

#### Populating data for unbound column

You can populate the data for unbound column by setting [Expression](#) or [Format](#) property or through [QueryUnBoundColumnValue](#) event.

#### Using Expression

You can specify the arithmetic or logic expression using `Expression` property to compute the display value. By default `GridUnBoundColumn` evaluates the expression with casing. You can disable the casing while evaluate the expression by setting [CaseSensitive](#) property to `false`.

Below are the list of Arithmetic and logical operations supported.

Arithmetic operations	Operator
Add	+
Subtract	-
Multiply	*
Divide	/
Power	^
Mod	%
Greater Than	>
Less Than	<
Equal	=
GreaterThanOrEqual	>=
LessThanOrEqual	<=

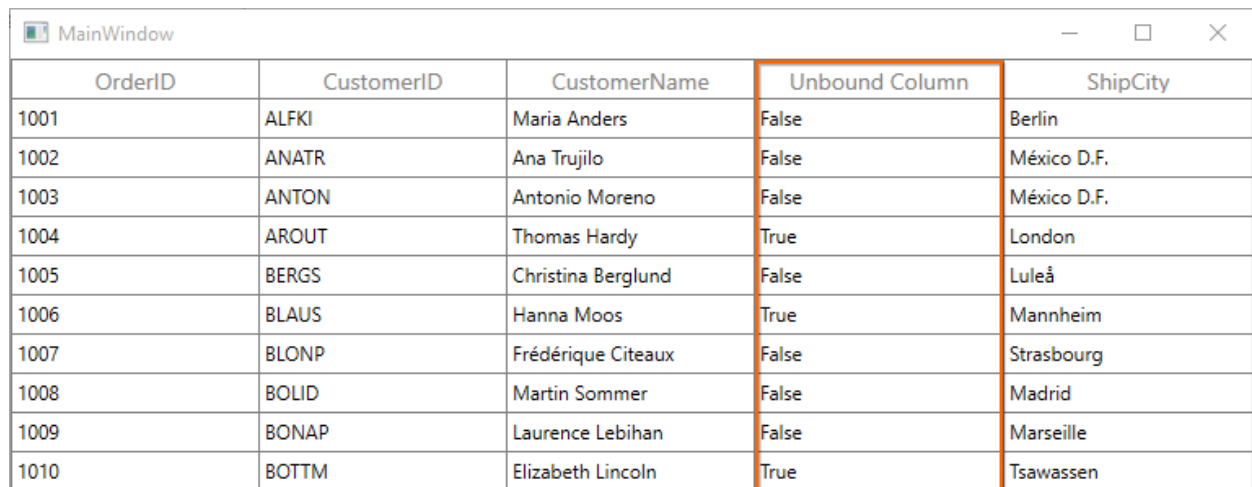
Logical operations	Operators
AND	(char)135
OR	(char)136
NOT	(char)137

**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridUnBoundColumn HeaderText="Unbound Column"
MappingName="UnboundColumn" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

**C#**

```
(this.dataGrid.Columns[3] as GridUnBoundColumn).Expression = "Discount *
UnitPrice > 0" + (char)135 + "UnitPrice * Quantity > 100";
```



OrderID	CustomerID	CustomerName	Unbound Column	ShipCity
1001	ALFKI	Maria Anders	False	Berlin
1002	ANATR	Ana Trujillo	False	México D.F.
1003	ANTON	Antonio Moreno	False	México D.F.
1004	AROUT	Thomas Hardy	True	London
1005	BERGS	Christina Berglund	False	Luleå
1006	BLAUS	Hanna Moos	True	Mannheim
1007	BLONP	Frédérique Citeaux	False	Strasbourg
1008	BOLID	Martin Sommer	False	Madrid
1009	BONAP	Laurence Lebihan	False	Marseille
1010	BOTTM	Elizabeth Lincoln	True	Tsawassen

*Using Format*

You can format the values of other columns and display the formatted value in unbound column using [Format](#) property.

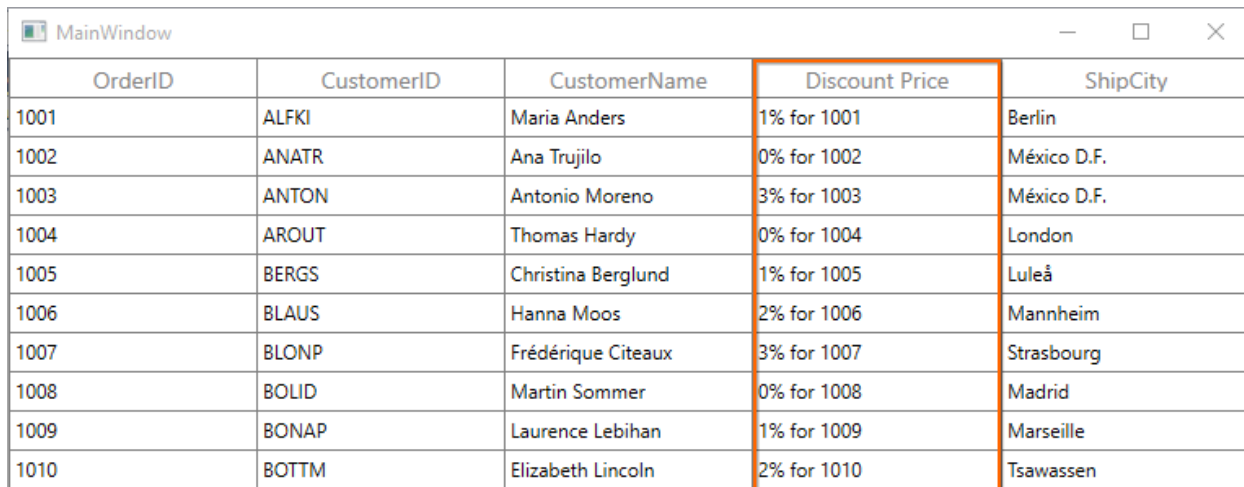
**XML**

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridUnBoundColumn Format="'{Discount}% for {OrderID}'"
HeaderText="Discount Price"
MappingName="DiscountPrice" />
</syncfusion:SfDataGrid.Columns>
```

```
</syncfusion:SfDataGrid>
```

**C#**

```
this.dataGrid.Columns.Add(new GridUnBoundColumn() { HeaderText = "Discount Price", MappingName = "DiscountPrice", Format = "'{Discount}% for {OrderID}'" });
```



OrderID	CustomerID	CustomerName	Discount Price	ShipCity
1001	ALFKI	Maria Anders	1% for 1001	Berlin
1002	ANATR	Ana Trujilo	0% for 1002	México D.F.
1003	ANTON	Antonio Moreno	3% for 1003	México D.F.
1004	AROUT	Thomas Hardy	0% for 1004	London
1005	BERGS	Christina Berglund	1% for 1005	Luleå
1006	BLAUS	Hanna Moos	2% for 1006	Mannheim
1007	BLONP	Frédérique Citeaux	3% for 1007	Strasbourg
1008	BOLID	Martin Sommer	0% for 1008	Madrid
1009	BONAP	Laurence Lebihan	1% for 1009	Marseille
1010	BOTTM	Elizabeth Lincoln	2% for 1010	Tsawassen

*Using QueryUnBoundColumnValue event*

You can populate the data for unbound column by handling the [QueryUnBoundColumnValue](#) event.

[GridUnBoundColumnEventArgs](#) of the [QueryUnBoundColumnValue](#) event provides the information about the cell triggered this event. [GridUnBoundColumnValueEventArgs.OriginalSender](#) returns the DataGrid fired this event for DetailsView.

You can get or set the [GridUnBoundColumnEventArgs.Value](#) property based on the [UnBoundAction](#).

- [UnBoundAction - QueryData](#) denotes the event triggered to query value and cell information.
- [UnBoundAction - CommitData](#) denotes the event triggered to save the edited value.

**C#**

```
this.dataGrid.QueryUnboundColumnValue += dataGrid_QueryUnboundColumnValue;
void dataGrid_QueryUnboundColumnValue(object sender,
GridUnboundColumnEventArgs e)
{
    if (e.UnBoundAction == UnBoundActions.QueryData)
    {
        var unitPrice =
Convert.ToDouble(e.Record.GetType().GetProperty("UnitPrice").GetValue(e.Record));
        var disCount =
Convert.ToDouble(e.Record.GetType().GetProperty("Discount").GetValue(e.Record));
        var save = unitPrice * disCount;
        e.Value = save.ToString() + "$";
    }
}
```

```
}

```

OrderID	CustomerID	CustomerName	Discount Price	ShipCity
1001	ALFKI	Maria Anders	25\$	Berlin
1002	ANATR	Ana Trujilo	0\$	México D.F.
1003	ANTON	Antonio Moreno	75\$	México D.F.
1004	AROUT	Thomas Hardy	0\$	London
1005	BERGS	Christina Berglund	33\$	Luleå
1006	BLAUS	Hanna Moos	100\$	Mannheim
1007	BLONP	Frédérique Citeaux	75\$	Strasbourg
1008	BOLID	Martin Sommer	0\$	Madrid
1009	BONAP	Laurence Lebihan	25\$	Marseille
1010	BOTTM	Elizabeth Lincoln	100\$	Tsawassen

Refreshing the unbound column at runtime

You can trigger the [QueryUnBoundColumnValue](#) event or recalculate the value for the cells of unbound column at runtime by calling [UpdateUnboundColumn](#) method.

**C#**

```
this.dataGrid.CurrentCellEndEdit += dataGrid_CurrentCellEndEdit;
void dataGrid_CurrentCellEndEdit(object sender, CurrentCellEndEditEventArgs args)
{
    this.dataGrid.UpdateUnboundColumn(this.dataGrid.CurrentItem, "Discount");
}
```

Editing unbound column

*Cancel the editing for unbound column cell*

You can cancel the editing of unbound column cell by handling the [SfDataGrid.CurrentCellBeginEdit](#) event.

**C#**

```
dataGrid.CurrentCellBeginEdit += dataGrid_CurrentCellBeginEdit;
void dataGrid_CurrentCellBeginEdit(object sender,
CurrentCellBeginEditEventArgs args)
{
    args.Cancel = args.Column is GridUnBoundColumn;
}
```

*Saving edited value of unbound column using QueryUnBoundColumnValue.*

You can get the edited value of unbound column from [GridUnboundColumnEventArgs.Value](#) property of [QueryUnBoundColumnValue](#) event when UnBoundAction is [CommitData](#).

**C#**

```
this.dataGrid.QueryUnboundColumnValue += dataGrid_QueryUnboundColumnValue;
void dataGrid_QueryUnboundColumnValue(object sender,
GridUnboundColumnEventArgs e)
```

```
{
    if (e.UnBoundAction == UnBoundActions.CommitData)
    {
        var editedValue = e.Value;
    }
}
```

#### Read unbound column values

You can get the value of `GridUnBoundColumn` using [GetUnBoundCellValue](#) method.

#### C#

```
using Syncfusion.UI.Xaml.Grid.Helpers;
this.dataGrid.CurrentCellValueChanged += dataGrid_CurrentCellValueChanged;
void dataGrid_CurrentCellValueChanged(object sender,
CurrentCellValueChangedEventArgs args)
{
    var updateValue = this.dataGrid.GetUnBoundCellValue(dataGrid.Columns[5],
this.dataGrid.CurrentItem);
}
```

#### Styling unbound column

You can customize the style of unbound column by writing style of TargetType [GridCell](#) or setting [GridColumn.CellStyle](#) property.

In the below code snippet, Foreground of the cells in `GridUnBoundColumn` changed based on its content.

#### XML

```
<Window.Resources>
<local:UnboundCellStyleConverter x:Key="unboundCellStyleConverter"/>
</Window.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridUnBoundColumn HeaderText="Discount Price"
MappingName="DiscountPrice">
<syncfusion:GridUnBoundColumn.CellStyle>
<Style TargetType="syncfusion:GridCell">
<Setter Property="Foreground" Value="{Binding
RelativeSource={RelativeSource Self}, Converter={StaticResource
unboundCellStyleConverter}}" />
</Style>
</syncfusion:GridUnBoundColumn.CellStyle>
</syncfusion:GridUnBoundColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

#### C#

```
public class UnboundCellStyleConverter : IValueConverter
{
```

```

public object Convert(object value, Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
{
    var unboundCell = value as GridCell;
    var unboundValue = (unboundCell.Content as TextBlock).Text;
    var data =
        System.Convert.ToInt16(unboundValue.Substring(0, unboundValue.Length - 1));
    if (data > 30)
        return new SolidColorBrush(Colors.DarkGreen);
    return new SolidColorBrush(Colors.Red);
}

public object ConvertBack(object value, Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
{
    return value;
}
}

```



OrderID	CustomerID	CustomerName	Discount Price	Country
1001	ALFKI	Maria Anders	25\$	Germany
1002	ANATR	Ana Trujilo	0\$	Mexico
1003	ANTON	Antonio Moreno	75\$	Mexico
1004	AROUT	Thomas Hardy	0\$	UK
1005	BERGS	Christina Berglund	33\$	Sweden
1006	BLAUS	Hanna Moos	100\$	Germany
1007	BLONP	Frédérique Citeaux	75\$	France
1008	BOLID	Martin Sommer	0\$	Spain
1009	BONAP	Laurence Lebihan	25\$	France
1010	BOTTM	Elizabeth Lincoln	100\$	Canada

You can refer the [Styling](#) section of [GridColumn](#) for more information.

#### Customize the Unbound column behavior

SfDataGrid allows you to customize the operations like key navigation and UI related interactions by overriding the corresponding renderer associated with the unbound column.

Below table lists the available cell types for unbound column.

Cell Type	Renderer
UnBoundTemplateColumn	<a href="#">GridUnBoundCellTemplateRenderer</a>
UnBoundTextColumn	<a href="#">GridUnBoundCellTextBoxRenderer</a>

If the [GridUnBoundColumn.EditTemplate](#) not defined then the [UnBoundTextColumn](#) set as default cell type of [GridUnBoundColumn](#).

If [GridUnBoundColumn.EditTemplate](#) property defined then [UnBoundTemplateColumn](#) set as cell type of [GridUnBoundColumn](#).

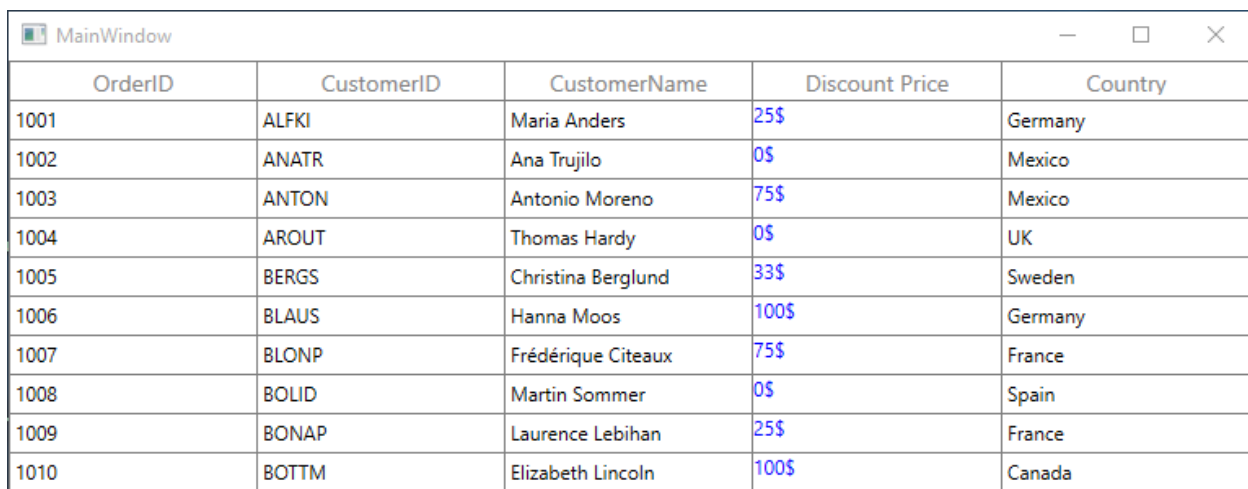
*Overriding Existing CellType*

You can customize the unbound row cell behavior by overriding existing renderer and replace the default one in [SfDataGrid.CellRenderers](#).

In the below code snippet, [GridUnBoundCellTextBoxRenderer](#) is customized to change the foreground and replaced the default renderer with customized renderer in [SfDataGrid.CellRenderer](#) collection.

**C#**

```
this.dataGrid.CellRenderers.Remove("UnBoundTextColumn");
this.dataGrid.CellRenderers.Add("UnBoundTextColumn", new
GridUnBoundCellTextBoxRendererExt());
public class GridUnBoundCellTextBoxRendererExt :
GridUnBoundCellTextBoxRenderer
{
    public override void OnInitializeDisplayElement(DataColumnBase dataColumn,
    TextBlock uiElement, object dataContext)
    {
        object cellValue = null;
        if (dataContext != null)
            cellValue = DataGrid.GetUnBoundCellValue(dataColumn.GridColumn,
            dataContext);
        uiElement.Text = cellValue == null ? string.Empty : cellValue.ToString(); ;
        uiElement.Foreground = new SolidColorBrush(Colors.Blue);
    }
    public override void OnInitializeEditElement(DataColumnBase dataColumn,
    TextBox uiElement, object dataContext)
    {
        object cellValue = null;
        if (dataContext != null)
            cellValue = DataGrid.GetUnBoundCellValue(dataColumn.GridColumn,
            dataContext);
        uiElement.Text = cellValue == null ? string.Empty : cellValue.ToString();
    }
}
```



OrderID	CustomerID	CustomerName	Discount Price	Country
1001	ALFKI	Maria Anders	25\$	Germany
1002	ANATR	Ana Trujillo	0\$	Mexico
1003	ANTON	Antonio Moreno	75\$	Mexico
1004	AROUT	Thomas Hardy	0\$	UK
1005	BERGS	Christina Berglund	33\$	Sweden
1006	BLAUS	Hanna Moos	100\$	Germany
1007	BLONP	Frédérique Citeaux	75\$	France
1008	BOLID	Martin Sommer	0\$	Spain
1009	BONAP	Laurence Lebihan	25\$	France
1010	BOTTM	Elizabeth Lincoln	100\$	Canada

*Custom Renderer*

You can change the renderer of unbound column by removing the predefined cell type value from [CellRenderers](#) collection and add the newly derived renderer from [GridVirtualizingCellRenderer](#). Refer

the [Create the renderer for existing column section](#) for more information to create the custom renderer in columns section.

#### Templating unbound column

You can load any WPF control in the display mode for `GridUnBoundColumn` by setting [GridColumn.CellTemplate](#) property. In edit mode, corresponding editor will be loaded based on column type. You can refer the [CellTemplate](#) section of `GridColumn` and [GridTemplateColumn](#) for more information.

#### Performance in WPF DataGrid (SfDataGrid)

SfDataGrid provides various built-in options to optimize the performance when handling large amount of data or high frequency updates.

#### Improving scrolling performance

You can improve the scrolling performance in SfDataGrid by setting [SfDataGrid.ScrollMode](#) property to **Async**.

This property enables the SfDataGrid to scroll the rows **asynchronously** with fade in animation.

#### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ItemsSource="{Binding EmployeeDetails}"
    ScrollMode="Async">
```

#### C#

```
this.sfDataGrid.ScrollMode = ScrollMode.Async;
```

#### Improving loading performance

##### *Data virtualization for loading*

You can load the large amount of data in less time using built-in [Data Virtualization](#).

##### *Improving loading and scrolling when using conditional styling*

You can style the cell and row conditionally in below three ways,

1. Using converters
2. Using Data triggers
3. Using Style selectors

Conditional styling using converter provides better performance compare to Data Trigger approach and Style selector approach. You can refer [Styles and Template](#) section for more information.

##### *Improving loading and scrolling performance using LightweightTemplate*

You can improve the loading and scrolling performance in SfDataGrid by setting [SfDataGrid.UseDrawing](#) property as **Default**. When using this property, the grid cell content and its borders are drawn instead of loading `UIElement` and hence it reduces the `VisualTree` of SfDataGrid to improve the loading and scrolling performance.

#### XML



```
<Syncfusion:SfDataGrid x:Name="dataGrid"
    UseDrawing="Default"
    ItemsSource="{Binding OrderInfoCollection }">
```

### C#

```
this.dataGrid.UseDrawing = UseDrawing.Default;
```

### Limitations

1. Searching is not supported.
2. Validation need to be achieved by using Template.
3. Cannot able to set **BorderThickness** in four ways (Left, Top, Right, Bottom). We can use one double value for setting the **BorderThickness**.
4. **UseDrawing** is not supported to get the property value while using dynamic type so need to use **UseBindingValue** as **true**
5. **UseDrawing** is completely supported for below grid elements.
  - GridCell
  - GridUnBoundRowCell
6. **UseDrawing** is not supported to UIElement content for the below grid elements (Support only for border).
  - GridIndentCell
  - GridCaptionSummaryCell
  - GridGroupSummaryCell
  - GridTableSummaryCell
7. **UseDrawing** is not supported for the below grid elements.
  - GridHeaderCellControl
  - GridRowHeaderCell
  - GridRowHeaderIndentCell
  - GridFilterCell
  - GridDetailsViewExpanderCell
  - GridHeaderIndentCell

### Improving performance when doing batch updates

SfDataGrid allows you to add, remove and update more number of records efficiently when you are having sorting, grouping and more summaries in place. By default, SfDataGrid responds to the collection changes and updates the UI instantly. If you are doing bulk or more updates to grid then you can follow below steps for better performance,

1. Invoke [SfDataGrid.View.BeginInit](#) before update the data.
2. After that update underlying collection.
3. Then call [SfDataGrid.View.EndInit](#) method, to refresh the View and UI. Now summaries, sort order and groups will be updated as expected.

### C#

```
//Batch Updates
//Suspends data manipulation operations in View
```

```
this.dataGrid.View.BeginInit();  
// Add, remove and update the underlying collection.  
//Resumes data manipulation operations and refresh the View.  
this.dataGrid.View.EndInit();
```

### Adding columns efficiently

SfDataGrid allows you to add more number of columns to [SfDataGrid.Columns](#) collection efficiently. Adding or removing more no of columns to collection, updates the UI for each time which negatively impact the performance.

You can improve the performance while adding, removing columns by suspending all the UI updates using [Suspend](#) and resume the updates after adding columns using [Resume](#) methods. You have to refresh the UI using [RefreshColumns](#) method.

### C#

```
using Syncfusion.UI.Xaml.Grid.Helpers;  
this.dataGrid.Columns.Suspend();  
// Add or Remove More columns  
this.dataGrid.Columns.Resume();  
this.dataGrid.RefreshColumns();
```

### Optimizing summary calculation performance

SfDataGrid optimizes the summary calculation when updating the underlying collection. It calculates the summaries optimistically by listening the data updates and using old calculated summary values without doing complete recalculation.

Below sections explains how SfDataGrid handles the updates efficiently for different data operations and what you have to do in application for the same.

#### Adding Record

SfDataGrid considers only the **added** item value and the current summary value instead of recalculating the summary based on all records. Based on these two values recalculates the summary efficiently.

#### Removing a Record

SfDataGrid considers only the **removed** item value and the current summary value instead of recalculating the summary based on all records. Based on these two values recalculates the summary efficiently.

#### Property Change in a record

SfDataGrid considers only the changed item value and the current aggregated value instead of recalculating the summary based on all records. For this you have to implement [INotifyPropertyChanging](#) and [INotifyPropertyChanged](#) interface to your Data Model.

Below code to enable summary calculation optimization by inheriting **INotifyPropertyChanging** and **INotifyPropertyChanged** interface to Data Model.

### C#

```
public class OrderInfo : INotifyPropertyChanging, INotifyPropertyChanged  
{  
    private int orderID;  
    public int OrderID  
{
```

```

get { return orderID; }
set
{
    this.RaisePropertyChanging("OrderID");
    orderID = value;
    this.RaisePropertyChanged("OrderID");
}
}
public void RaisePropertyChanged(string propName)
{
    if (this.PropertyChanged != null)
        this.PropertyChanged(this, new PropertyChangedEventArgs(propName));
}
public event PropertyChangedEventHandler PropertyChanged;
public void RaisePropertyChanging(string propName)
{
    if (this.PropertyChanging != null)
        this.PropertyChanging(this, new PropertyChangingEventArgs(propName));
}
public event PropertyChangingEventHandler PropertyChanging;
}

```

#### *Loading performance - On demand summary calculation for group and caption summary*

You can calculate the Caption and Group summary on-demand by setting [SfDataGrid.SummaryCalculationMode](#) as [CalculationMode.OnDemandCaptionSummary](#) or [CalculationMode.OnDemandGroupSummary](#). You can set this property when you are loading more number of summary columns on summary row or more number of group summaries to improve loading performance. On-demand summary calculation will calculate summaries for the summary rows which are visible and summaries for other rows will be calculated only when it comes into view.

#### **XML**

```

<Syncfusion:SfDataGrid x:Name="datagrid"
    SummaryCalculationMode="OnDemandCaptionSummary"
    ItemsSource="{Binding OrderInfoCollection }">

```

#### **C#**

```

this.datagrid.SummaryCalculationMode =
    CalculationMode.OnDemandCaptionSummary |
    CalculationMode.OnDemandGroupSummary;

```

#### *Improving UI Filter loading time*

SfDataGrid allows you to open filter popup in less time by setting [CanGenerateUniqueItems](#) property to false. By default [GridFilterControl](#) loads unique items in popup which takes more time to load.

[CanGenerateUniqueItems](#) property loading [TextBox](#) to filter instead of [ComboBox](#) in advanced filter UI View.

#### **XML**

```

<Window.Resources>
<Style TargetType="syncfusion:GridFilterControl">
<Setter Property="FilterMode" Value="AdvancedFilter" />

```

```
</Style>
<Style TargetType="syncfusion:AdvancedFilterControl">
<Setter Property="CanGenerateUniqueItems" Value="False" />
</Style>
</Window.Resources>
```

Improving performance while adding multiple FilterPredicates to the column in loop  
For more details, refer the [Filtering](#) section.

## Interactive Features in WPF DataGrid (SfDataGrid)

### Column Chooser

SfDataGrid allows you to show and hide the Columns from the view at runtime via drag and drop using ColumnChooser. You can enable a column chooser in an application by creating an instance for GridColumnChooserController and assign to SfDataGrid.GridColumnDragDropController.

While dropping columns in ColumnChooser window, the particular column will be hidden by setting GridColumn.IsHidden as true.

### C#

```
ColumnChooser chooserWindow;
void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
    chooserWindow = new ColumnChooser(this.dataGrid);
    chooserWindow.Resources.MergedDictionaries.Clear();
    chooserWindow.ClearValue(ColumnChooser.StyleProperty);
    //Resources has been added to the Merged Dictionaries
    chooserWindow.Resources.MergedDictionaries.Add(this.MainGrid.Resources.MergedDictionaries[0]);
    this.dataGrid.GridColumnDragDropController = new
    GridColumnChooserController(this.dataGrid, chooserWindow);
    //ColumnChooser Window will open
    chooserWindow.Show();
    chooserWindow.Owner = this;
}
```

Order ID	Ship Name	Shipped Date	Ship Country	Ship Via	Freight
10000	Franchi S.p.A.	5/15/1991	Italy	3	
10001	Mère Paillard	5/23/1991	Canada	3	
10002	Folk och få HB	5/17/1991	Sweden	3	
10003	Simons bistro	5/24/1991	D	1	
10004	Vaffeljernet	5/20/1991	D	2	
10005	Wartian Herkku	5/24/1991	F	3	
10006	Franchi S.p.A.	5/24/1991	I	1	
10007	Morgenstern Gesundkost	6/11/1991	G	3	
10008	Furia Bacalhau e Frutos do Mar	5/29/1991	P	1	
10009	Seven Seas Imports	5/31/1991	U	1	
10010	Simons bistro	5/30/1991	D	1	
10011	Wellington Importadora	6/3/1991	B	3	
10012	LINO-Delicateses	6/3/1991	V	1	\$
10013	Richter Supermarkt	6/7/1991	Switzerland	3	
10014	GROSELLA-Restaurante	6/12/1991	Venezuela	3	
10015	Piccolo und mehr	6/20/1991	Austria	1	
10016	Folies gourmandes	7/11/1991	France	2	\$
10017	Blondel père et fils	6/10/1991	France	1	\$
10018	Rattlesnake Canyon Grocery	7/5/1991	USA	1	
10019	Magazzini Alimentari Riuniti	6/20/1991	Italy	3	
10020	Vins et alcools Chevalier	6/26/1991	France	1	

### Custom Column Chooser

You can create custom UI for the column chooser and you can enable this view manually like below code example. You need to maintain separate collection in **ViewModel** to maintain the hidden columns which will bound to custom view.

### XML

```
<syncfusion:ChromelessWindow >
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="50" />
</Grid.RowDefinitions>
<ListBox x:Name="listBox"
Grid.Row="0"
Margin="0,5"
HorizontalAlignment="Stretch"
BorderThickness="0"
ItemContainerStyle="{StaticResource ListBoxItemStyle1}"
ItemsSource="{Binding ColumnCollection}">
<ListBox.ItemTemplate>
<StaticResource ResourceKey="MyDataTemplate" />
</ListBox.ItemTemplate>
</ListBox>
<StackPanel Grid.Row="1" Margin="20,0,0,0"
VerticalAlignment="Stretch"
Background="Transparent"
Orientation="Horizontal">
<Button Margin="5"
Click="OKButton_Click">
```

```

Content="OK"
Style="{StaticResource ButtonStyle}" />
<Button Margin="5"
Content="Cancel"
IsCancel="True"
Style="{StaticResource ButtonStyle}" />
</StackPanel>
</Grid>
<syncfusion:ChromelessWindow.Resources>
<Style x:Key="FocusVisual">
<Setter Property="Control.Template">
<Setter.Value>
<ControlTemplate>
<Rectangle Margin="2"
SnapsToDevicePixels="true"
Stroke="{DynamicResource {x:Static SystemColors.ControlTextBrushKey}}"
StrokeDashArray="1 2"
StrokeThickness="1" />
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
<SolidColorBrush x:Key="Item.MouseOver.Background" Color="#1F26A0DA" />
<SolidColorBrush x:Key="Item.MouseOver.Border" Color="#a826A0Da" />
<SolidColorBrush x:Key="Item.SelectedInactive.Background" Color="#3DDADADA" />
<SolidColorBrush x:Key="Item.SelectedInactive.Border" Color="#FFDADADA" />
<SolidColorBrush x:Key="Item.SelectedActive.Background" Color="#3D26A0DA" />
<SolidColorBrush x:Key="Item.SelectedActive.Border" Color="#FF26A0DA" />
//ListBoxItem is Customized through the DataTemplate
<Style x:Key="ListBoxItemStyle1" TargetType="ListBoxItem">
<Setter Property="SnapsToDevicePixels" Value="True" />
<Setter Property="Padding" Value="4,1" />
<Setter Property="HorizontalContentAlignment" Value="{Binding
HorizontalContentAlignment, RelativeSource={RelativeSource
AncestorType={x:Type ItemsControl}}}" />
<Setter Property="VerticalContentAlignment" Value="{Binding
VerticalContentAlignment, RelativeSource={RelativeSource
AncestorType={x:Type ItemsControl}}}" />
<Setter Property="Background" Value="Transparent" />
<Setter Property="BorderBrush" Value="Transparent" />
<Setter Property="BorderThickness" Value="1" />
<Setter Property="FocusVisualStyle" Value="{StaticResource FocusVisual}" />
<DataTemplate x:Key="MyDataTemplate">
<Grid Margin="0,3,0,3">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<CheckBox Margin="0,2,0,0"
HorizontalAlignment="Center"
VerticalAlignment="Center"
IsChecked="{Binding IsChecked,Mode=TwoWay}"
Padding="2,0,0,0">
</CheckBox>
<TextBlock Grid.Column="1"
HorizontalAlignment="Left"

```

```

VerticalAlignment="Center"
FontFamily="Segoe UI"
FontSize="14"
Padding="10,0,0,0"
Text="{Binding Name}" />
</Grid>
</DataTemplate>
//Customizing the Button by using the VisualState
<Style x:Key="ButtonStyle" TargetType="Button">
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="Button">
<Grid>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="CommonStates">
<VisualState x:Name="Normal" />
<VisualState x:Name="MouseOver" />
<VisualState x:Name="Pressed">
<Storyboard>
<ColorAnimationUsingKeyFrames Storyboard.TargetName="textBlock"
Storyboard.TargetProperty="(TextElement.Foreground).(SolidColorBrush.Color)"
>
<EasingColorKeyFrame KeyTime="0" Value="White" />
</ColorAnimationUsingKeyFrames>
<ColorAnimationUsingKeyFrames Storyboard.TargetName="border"
Storyboard.TargetProperty="(Panel.Background).(SolidColorBrush.Color)">
<EasingColorKeyFrame KeyTime="0" Value="#FF55C5D5" />
</ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Disabled" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border x:Name="border"
Width="80"
Height="24"
Background="Gray">
<TextBlock x:Name="textBlock"
HorizontalAlignment="Center"
VerticalAlignment="Center"
FontSize="14"
Foreground="White"
Text="{TemplateBinding Content}" />
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</syncfusion:ChromelessWindow.Resources>
</syncfusion:ChromelessWindow>

```

**C#**

```
var visibleColumns = this.AssociatedObject.dataGrid.Columns;
```

```
ObservableCollection<ColumnChooserItems> totalColumns =
GetColumnsDetails(visibleColumns);
CustomColumnChooserViewModel chooserViewModel = new
CustomColumnChooserViewModel(totalColumns);
CustomColumnChooser ColumnChooserView = new
CustomColumnChooser(chooserViewModel);
ColumnChooserView.Owner = Application.Current.MainWindow;
chooserWindow.Visibility = System.Windows.Visibility.Collapsed;
if ((bool)ColumnChooserView.ShowDialog())
ClickOKButton(chooserViewModel.ColumnCollection,
this.AssociatedObject.dataGrid);
viewModel.ShowColumnChooser = false;
//This will add or remove the Columns from the SfDataGrid
public void ClickOKButton(ObservableCollection<ColumnChooserItems>
ColumnCollection, SfDataGrid dataGrid)
{
foreach (var item in ColumnCollection)
{
var column = dataGrid.Columns.FirstOrDefault(v => v.MappingName ==
item.Name);
if (column != null)
{
if (item.IsChecked == false && !column.IsHidden)
column.IsHidden = true;
else if (item.IsChecked == true && column.IsHidden == true)
{
if (column.Width == 0)
column.Width = 150;
column.IsHidden = false;
}
}
}
viewModel.ShowColumnChooser = false;
}
```



Order ID	Order Date	Ship Name	Customer ID	Shipped Date	Ship Address
10000	5/10/1991	Franchi S.p.A.	FRANS	5/15/1991	Via Monte Bianco 34
10001	5/13/1991	Mère Paillarde	MEREP	5/23/1991	43 rue St. Laurent
10002	5/14/1991	Folk och få HB	FOLKO	5/17/1991	Åkergatan 24
10003	5/15/1991	Simons bistro		5/24/1991	Vinbæltet 34
10004	5/16/1991	Vaffeljernet		5/20/1991	Smagsløget 45
10005	5/20/1991	Wartian Herkku		5/24/1991	Torikatu 38
10006	5/21/1991	Franchi S.p.A.		5/24/1991	Via Monte Bianco 34
10007	5/22/1991	Morgenstern Ge		6/11/1991	Heerstr. 22
10008	5/23/1991	Furia Bacalhau e		5/29/1991	Jardim das rosas n. 3
10009	5/24/1991	Seven Seas Impo		5/31/1991	90 Wadhurst Rd.
10010	5/28/1991	Simons bistro		5/30/1991	Vinbæltet 34
10011	5/29/1991	Wellington Impo		6/3/1991	Rua do Mercado, 12
10012	5/30/1991	LINO-Delicatese		6/3/1991	Ave. 5 de Mayo Pork
10013	5/31/1991	Richter Supermarkt	RICSU	6/7/1991	Starenweg 5
10014	6/3/1991	GROSELLA-Restaurante	GROSR	6/12/1991	5ª Ave. Los Palos Gra
10015	6/5/1991	Piccolo und mehr	PICCO	6/20/1991	Geislweg 14
10016	6/6/1991	Folies gourmandes	FOLIG	7/11/1991	184, chaussée de To
10017	6/7/1991	Blondel père et fils	BLONP	6/10/1991	24, place Kléber
10018	6/10/1991	Rattlesnake Canyon Grocer	RATTC	7/5/1991	2817 Milton Dr.
10019	6/11/1991	Magazzini Alimentari Riuniti	MAGAA	6/20/1991	Via Ludovico il Moro
10020	6/12/1991	Vies et Associés Chevalier	VINET	6/26/1991	50 rue de l'Abbaye

### Appearance Customization

You can change the default appearance of the column chooser window by customizing the style of **ColumnChooser**. You can directly change the **Title** and **WaterMarkText** like the below code example.

### C#

```
ColumnChooser chooserWindow;
void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
    chooserWindow = new ColumnChooser(this.dataGrid);
    chooserWindow.Resources.MergedDictionaries.Clear();
    chooserWindow.ClearValue(ColumnChooser.StyleProperty);
    chooserWindow.Resources.MergedDictionaries.Add(this.MainGrid.Resources.MergedDictionaries[0]);
    this.dataGrid.GridColumnDragDropController = new
    GridColumnChooserController(this.dataGrid, chooserWindow);
    chooserWindow.Title = "Columns";
    chooserWindow.WaterMarkText = "Drag the columns";
    chooserWindow.Show();
    chooserWindow.Owner = this;
}
```

Order ID	Order Date	Ship Name	Customer ID	Shipped Date	Ship Address
10000	5/10/1991	Franchi S.p.A.	FRANS	5/15/1991	Via Monte Bianco 34
10001	5/13/1991	Mère Paillarde	MEREP	5/23/1991	43 rue St. Laurent
10002	5/14/1991	Folk och få HB	FOLKO	5/17/1991	Åkergatan 24
10003	5/15/1991	Simons bistro		5/24/1991	Vinbæltet 34
10004	5/16/1991	Vaffeljernet		5/20/1991	Smagsløget 45
10005	5/20/1991	Wartian Herkku		5/24/1991	Torikatu 38
10006	5/21/1991	Franchi S.p.A.		5/24/1991	Via Monte Bianco 34
10007	5/22/1991	Morgenstern Ge		6/11/1991	Heerstr. 22
10008	5/23/1991	Furia Bacalhau e		5/29/1991	Jardim das rosas n. 3
10009	5/24/1991	Seven Seas Impo		5/31/1991	90 Wadhurst Rd.
10010	5/28/1991	Simons bistro		5/30/1991	Vinbæltet 34
10011	5/29/1991	Wellington Impo		6/3/1991	Rua do Mercado, 12
10012	5/30/1991	LINO-Delicatese		6/3/1991	Ave. 5 de Mayo Pork
10013	5/31/1991	Richter Supermarkt	RICSU	6/7/1991	Starenweg 5
10014	6/3/1991	GROSELLA-Restaurante	GROSR	6/12/1991	5ª Ave. Los Palos Gra
10015	6/5/1991	Piccolo und mehr	PICCO	6/20/1991	Geislweg 14
10016	6/6/1991	Folies gourmandes	FOLIG	7/11/1991	184, chaussée de To
10017	6/7/1991	Blondel père et fils	BLONP	6/10/1991	24, place Kléber
10018	6/10/1991	Rattlesnake Canyon Grocer	RATTC	7/5/1991	2817 Milton Dr.
10019	6/11/1991	Magazzini Alimentari Riuniti	MAGAA	6/20/1991	Via Ludovico il Moro
10020	6/12/1991	Vier et deux Chevalier	VINET	6/26/1991	50 rue de l'Abbaye

## ToolTip in WPF DataGrid (SfDataGrid)

ToolTip provides the support to show the pop-up window that displays the information when the mouse hovers in cells of SfDataGrid.

### Record cell tooltip

You can enable the ToolTip for the [GridCell](#) by setting the [SfDataGrid.ShowToolTip](#) as `true`.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="True"
    ShowToolTip="True"
    ItemsSource="{Binding Orders}" >
</syncfusion:SfDataGrid>
```

### C#

```
this.dataGrid.ShowToolTip = true;
```

You can enable the ToolTip for the particular column by setting the [GridColumn.ShowToolTip](#) as `true`.

### XML

```
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn HeaderText="Order ID" ShowToolTip="True"
    MappingName="OrderID" />
<syncfusion:GridTextColumn HeaderText="CustomerID" ShowToolTip="True"
    MappingName="CustomerID" />
</syncfusion:SfDataGrid.Columns>
```

**C#**

```
this.dataGrid.Columns["OrderID"].ShowToolTip = true;
this.dataGrid.Columns["CustomerID"].ShowToolTip = true;
```

**Note:** GridColumn.ShowToolTip takes higher priority than [SfDataGrid.ShowToolTip](#).

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1003	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1011	ANATR	Ana Trujillo	Mexico	México D.F.
1012	BERGS	Christina Berglund	Sweden	Luleå
1013	BLONP	Frédérique Citeaux	France	Strasbourg

**Header tooltip**

You can enable the ToolTip for the header cell by setting the [GridColumn.ShowHeaderToolTip](#) as `true`.

**XML**

```
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn HeaderText="Order ID" ShowHeaderToolTip="True"
MappingName="OrderID" />
</syncfusion:SfDataGrid.Columns>
```

**C#**

```
this.dataGrid.Columns["OrderID"].ShowHeaderToolTip = true;
```



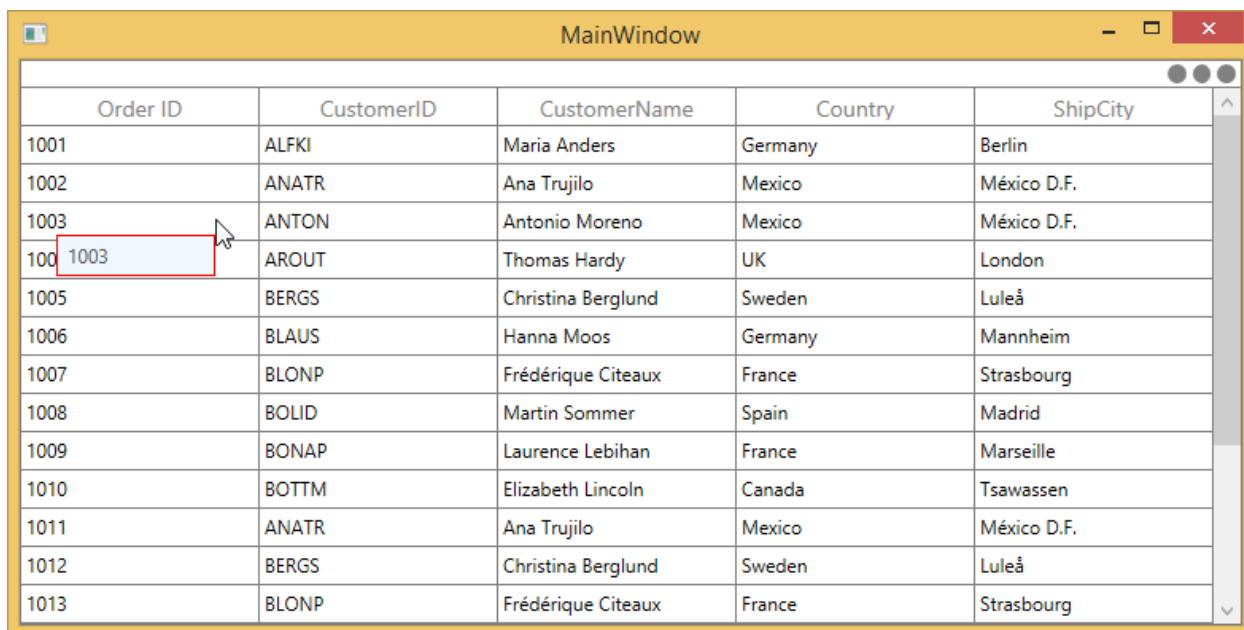
OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1011	ANATR	Ana Trujillo	Mexico	México D.F.
1012	BERGS	Christina Berglund	Sweden	Luleå
1013	BLONP	Frédérique Citeaux	France	Strasbourg

### ToolTip Customization

You can change the appearance of the ToolTip by customizing the style with TargetType as ToolTip.

### XML

```
<Window.Resources>
<Style x:Name="ToolTipStyle" TargetType="ToolTip">
<Setter Property="BorderThickness" Value="1,1,1,1" />
<Setter Property="BorderBrush" Value="Red" />
<Setter Property="Background" Value="AliceBlue" />
</Style>
</Window.Resources>
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn HeaderText="Order ID" ShowToolTip="True"
MappingName="OrderID" />
</syncfusion:SfDataGrid.Columns>
```



Order ID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1003	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen
1011	ANATR	Ana Trujillo	Mexico	México D.F.
1012	BERGS	Christina Berglund	Sweden	Luleå
1013	BLONP	Frédérique Citeaux	France	Strasbourg

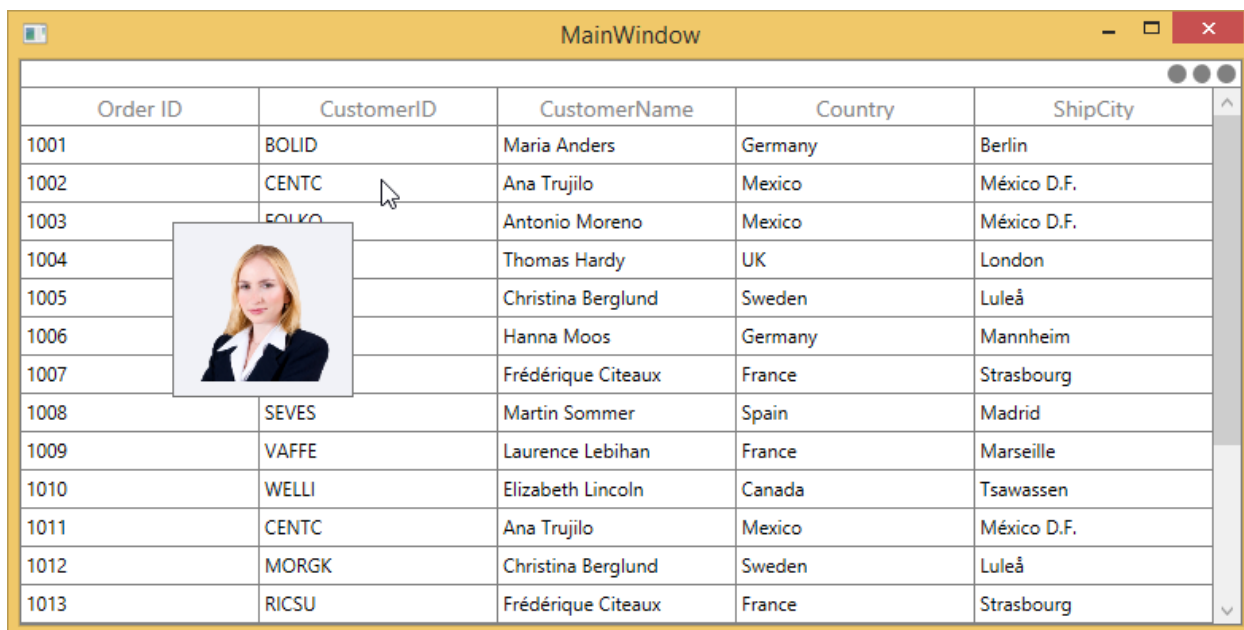
You can customize the template of ToolTip by using the [GridColumn.ToolTipTemplate](#) and [GridColumn.ToolTipTemplateSelector](#) properties.

#### *Customize the ToolTip using ToolTipTemplate*

You can customize the appearance of the ToolTip for particular column by setting `GridColumn.ToolTipTemplate`. And you can also customize the appearance of header ToolTip for particular column by [GridColumn.HeaderToolTipTemplate](#) property.

#### **XML**

```
<Window.Resources>
<local:StringToImageConverter x:Key="ImageConverter" />
<DataTemplate x:Key="TemplateToolTip">
<Image Height="100" Width="100" Source="{Binding
CustomerID,Converter={StaticResource ImageConverter}}"/>
</DataTemplate>
</Window.Resources>
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn HeaderText="Order ID" ShowToolTip="True"
MappingName="OrderID" />
<syncfusion:GridTextColumn HeaderText="CustomerID"
ToolTipTemplate="{StaticResource TemplateToolTip}" ShowToolTip="True"
MappingName="CustomerID" />
</syncfusion:SfDataGrid.Columns>
```



Order ID	CustomerID	CustomerName	Country	ShipCity
1001	BOLID	Maria Anders	Germany	Berlin
1002	CENTC	Ana Trujillo	Mexico	México D.F.
1003	FOLKO	Antonio Moreno	Mexico	México D.F.
1004		Thomas Hardy	UK	London
1005		Christina Berglund	Sweden	Luleå
1006		Hanna Moos	Germany	Mannheim
1007		Frédérique Citeaux	France	Strasbourg
1008	SEVES	Martin Sommer	Spain	Madrid
1009	VAFFE	Laurence Lebihan	France	Marseille
1010	WELLI	Elizabeth Lincoln	Canada	Tsawassen
1011	CENTC	Ana Trujillo	Mexico	México D.F.
1012	MORGK	Christina Berglund	Sweden	Luleå
1013	RICSU	Frédérique Citeaux	France	Strasbourg

You can get the sample from [here](#).

#### *Customize the ToolTip with ToolTipTemplateSelector*

The different ToolTip template can be loaded in a same column conditionally based on data by setting `GridColumn.ToolTipTemplateSelector`

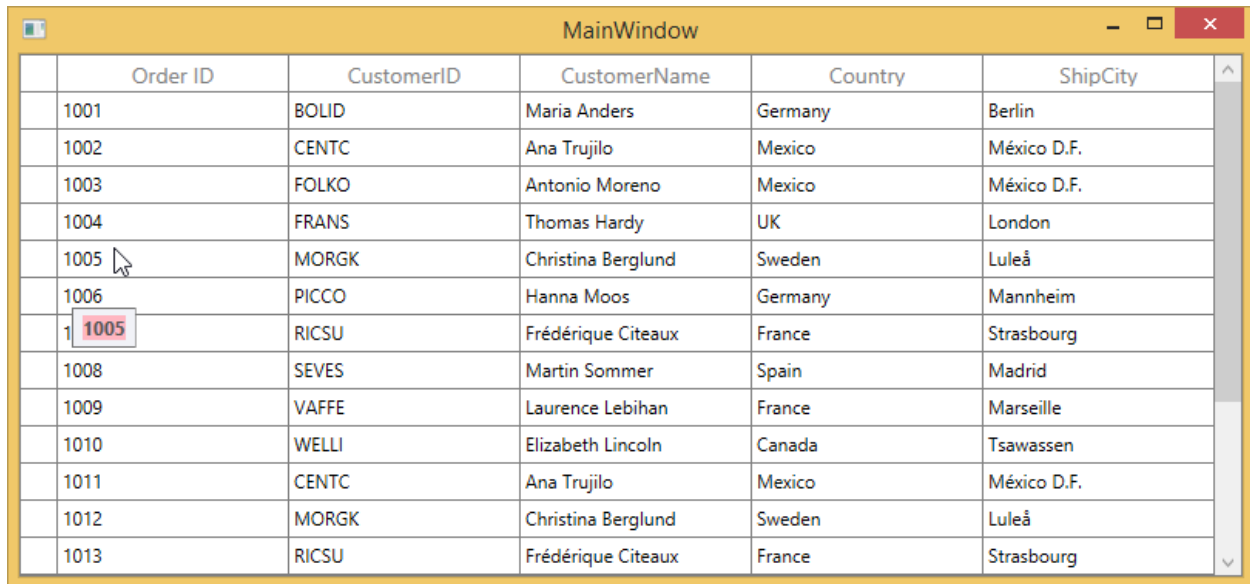
#### **XML**

```
<Window.Resources>
<DataTemplate x:Key="ToolTip1">
<Grid>
<Rectangle Fill="Transparent"/>
<TextBlock Text="{Binding OrderID}" FontWeight="Bold"
Background="LightPink"/>
</Grid>
</DataTemplate>
<DataTemplate x:Key="ToolTip2">
<Grid>
<Rectangle Fill="Transparent"/>
<TextBlock Text="{Binding OrderID}" FontStyle="Italic"
Background="LightGreen"/>
</Grid>
</DataTemplate>
</Window.Resources>
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTextColumn HeaderText="Order ID" ShowToolTip="True"
MappingName="OrderID" >
<syncfusion:GridTextColumn.ToolTipTemplateSelector>
<local:ToolTipTemplateSelector
AlternateTemplate="{StaticResource ToolTip2}"
DefaultTemplate="{StaticResource ToolTip1}" />
</syncfusion:GridTextColumn.ToolTipTemplateSelector>
</syncfusion:GridTextColumn>
</syncfusion:SfDataGrid.Columns>
```

**C#**

```
public class ToolTipTemplateSelector : DataTemplateSelector
{
    private DataTemplate _defaultTemplate;
    /// <summary>
    /// Gets or sets DefaultTemplate.
    /// </summary>
    public DataTemplate DefaultTemplate
    {
        get { return _defaultTemplate; }
        set { _defaultTemplate = value; }
    }
    private DataTemplate _alternateTemplate;
    /// <summary>
    /// Gets or Sets AlternateTemplate.
    /// </summary>
    public DataTemplate AlternateTemplate
    {
        get { return _alternateTemplate; }
        set { _alternateTemplate = value; }
    }
    public override System.Windows.DataTemplate SelectTemplate(object item,
        System.Windows.DependencyObject container)
    {
        //The item that comes from ToolTipTemplate is DataContextHelper. When set
        SetCellBoundValue to true, it sets DataContextHelper as DataContext to
        DataTemplate. Refer property section of CellTemplate.
        OrderInfo dataUnit = item as OrderInfo;
        if (dataUnit == null) return this.DefaultTemplate;
        //use reflection to retrieve property
        Type type = dataUnit.GetType();
        PropertyInfo property = type.GetProperty("OrderID");
        //To see what template needs to be select according to the specified
        property value.
        if (property.GetValue(dataUnit, null).ToString().Contains('9') ||
            property.GetValue(dataUnit, null).ToString().Contains('4'))
            return this.AlternateTemplate;
        else
            return this.DefaultTemplate;
    }
}
```

The below image refers the **DefaultTemplate** which is applied through **ToolTipTemplateSelector**.



	Order ID	CustomerID	CustomerName	Country	ShipCity
	1001	BOLID	Maria Anders	Germany	Berlin
	1002	CENTC	Ana Trujilo	Mexico	México D.F.
	1003	FOLKO	Antonio Moreno	Mexico	México D.F.
	1004	FRANS	Thomas Hardy	UK	London
	1005	MORGK	Christina Berglund	Sweden	Luleå
	1006	PICCO	Hanna Moos	Germany	Mannheim
1	1005	RICSU	Frédérique Citeaux	France	Strasbourg
	1008	SEVES	Martin Sommer	Spain	Madrid
	1009	VAFFE	Laurence Lebihan	France	Marseille
	1010	WELLI	Elizabeth Lincoln	Canada	Tsawassen
	1011	CENTC	Ana Trujilo	Mexico	México D.F.
	1012	MORGK	Christina Berglund	Sweden	Luleå
	1013	RICSU	Frédérique Citeaux	France	Strasbourg

The below image refers the **AlternateTemplate** which is applied through **ToolTipTemplateSelector**.



	Order ID	CustomerID	CustomerName	Country	ShipCity
	1001	BOLID	Maria Anders	Germany	Berlin
	1002	CENTC	Ana Trujilo	Mexico	México D.F.
	1003	FOLKO	Antonio Moreno	Mexico	México D.F.
	1004	FRANS	Thomas Hardy	UK	London
	1005	MORGK	Christina Berglund	Sweden	Luleå
	1006	PICCO	Hanna Moos	Germany	Mannheim
	1007	RICSU	Frédérique Citeaux	France	Strasbourg
	1008	SEVES	Martin Sommer	Spain	Madrid
	1009	VAFFE	Laurence Lebihan	France	Marseille
	1009	WELLI	Elizabeth Lincoln	Canada	Tsawassen
	1011	CENTC	Ana Trujilo	Mexico	México D.F.
	1012	MORGK	Christina Berglund	Sweden	Luleå
	1013	RICSU	Frédérique Citeaux	France	Strasbourg

You can get the sample from [here](#).

## Events

### CellToolTipOpening event

The **CellToolTipOpening** event occurs when any tooltip of the cell is opened. The **CellToolTipOpening** event receives the **GridCellToolTipOpeningEventArgs** as argument which has the following properties:

<ul>

<li> <a

href="https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.CellToolTipOpeningEventArgs.html#SyncfusionUIXamlGridCellToolTipOpeningEventArgs\_Column">Column :</a> Gets the hovered cell column in the SfTreeGrid.</li>



- <li> <a href="https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.CellToolTipOpeningEventArgs.html#SyncfusionUIXamlGridCellToolTipOpeningEventArgs\_Record">Record :</a> Gets the data context of hovered cell.</li>
- <li> <a href="https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.CellToolTipOpeningEventArgs.html#SyncfusionUIXamlGridCellToolTipOpeningEventArgs\_RowColumnIndex">RowColumnIndex :</a> Gets the row and column index of the hovered cell.</li>
- <li> <a href="https://help.syncfusion.com/cr/wpf/Syncfusion.UI.Xaml.Grid.CellToolTipOpeningEventArgs.html#SyncfusionUIXamlGridCellToolTipOpeningEventArgs\_ToolTip">ToolTip :</a> Gets the tooltip of the hovered cells.</li>

**XML**

```
<syncfusion:SfDataGrid Name="DataGrid"
    CellToolTipOpening="DataGrid_CellToolTipOpening"
    AutoGenerateColumns="True"
    ItemsSource="{Binding Orders}">
```

**C#**

```
this.DataGrid.CellToolTipOpening += DataGrid_CellToolTipOpening;
private void DataGrid_CellToolTipOpening(object sender,
    Syncfusion.UI.Xaml.Grid.GridCellToolTipOpeningEventArgs e)
{
}
```

## Context menu in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) (SfDataGrid) provides an entirely customizable context menu to expose the functionality on user interface. You can create context menu for different rows in an efficient manner.

The below code example shows the context menu with command bindings.

**XML**

```
<ContextMenu Style="{x:Null}">
    <MenuItem Command="{Binding CopyCommand, Source={StaticResource
        employeeInfoViewModel}}"
        CommandParameter="{Binding}" Header="Copy">
    </MenuItem>
</ContextMenu>
```

**C#**

```
public class BaseCommand : ICommand
{
    #region Fields
    readonly Action<object> _execute;
    readonly Predicate<object> _canExecute;
    #endregion
}
```

```

#region Constructors
    /// <summary>
    /// Creates a new command that can always execute.
    /// </summary>
    /// <param name="execute">The execution logic.</param>
    public BaseCommand(Action<object> execute)
    : this(execute, null)
    {
    }
    /// <summary>
    /// Creates a new command.
    /// </summary>
    /// <param name="execute">The execution logic.</param>
    /// <param name="canExecute">The execution status logic.</param>
    public BaseCommand(Action<object> execute, Predicate<object> canExecute)
    {
        if (execute == null)
            throw new ArgumentNullException("execute");
        _execute = execute;
        _canExecute = canExecute;
    }
#endregion
#region ICommand Members
    public bool CanExecute(object parameter)
    {
        return _canExecute == null ? true : _canExecute(parameter);
    }
    public event EventHandler CanExecuteChanged
    {
        add { CommandManager.RequerySuggested += value; }
        remove { CommandManager.RequerySuggested -= value; }
    }
    public void Execute(object parameter)
    {
        _execute(parameter);
    }
#endregion
}
public class EmployeeInfoViewModel : INotifyPropertyChanged
{
    public EmployeeInfoViewModel()
    {
        CopyCommand = new ContextMenuDemo.BaseCommand(ShowMessage);
    }
    private ICommand copyCommand
    public ICommand CopyCommand
    {
        get
        {
            return copyCommand;
        }
        set
        {
            copyCommand = value;
        }
    }
    public void ShowMessage(object obj)

```

```
{  
    if (obj is GridRecordContextMenuInfo)  
    {  
        var grid = (obj as GridRecordContextMenuInfo).DataGrid;  
        grid.GridCopyPaste.Copy();  
    }  
}
```

### Context menu for record rows

You can set the context menu for the data rows by using [SfDataGrid.RecordContextMenu](#) property.

#### XML

```
<syncfusion:SfDataGrid.RecordContextMenu>  
    <ContextMenu>  
        <MenuItem x:Name="Cut" Header="Cut" />  
        <MenuItem x:Name="Copy" Header="Copy" />  
        <MenuItem x:Name="Paste" Header="Paste" />  
        <MenuItem x:Name="Delete" Header="Delete" />  
    </ContextMenu>  
</syncfusion:SfDataGrid.RecordContextMenu>
```

#### C#

```
this.dataGrid.RecordContextMenu = new ContextMenu();  
this.dataGrid.RecordContextMenu.Items.Add(new MenuItem() { Header = "Cut" });  
this.dataGrid.RecordContextMenu.Items.Add(new MenuItem() { Header = "Copy" });  
this.dataGrid.RecordContextMenu.Items.Add(new MenuItem() { Header = "Paste" });  
this.dataGrid.RecordContextMenu.Items.Add(new MenuItem() { Header = "Delete" });
```

EmployeeId	EmployeeName	EmployeeAge	EmployeeArea	EmployeeStatus	EmployeeDesignation
EmployeeAge : 12 - 1 Items					
1002	Peter	12	UK	True	Junior Architect
Total Group Items : 1					
EmployeeAge : 20 - 19 Items					
1003	James	20	UAE	False	Team Lead
1004	Oliver	20	USA	True	General Manager
1010	Aravind	20	India	True	Senior Architect
1013	Praveen	20	UAE	False	Team Lead
1014	Stephen	20	USA	True	General Manager
1020	Amy Christam	20	India	True	Senior Architect
1023	Mahendra Gupta	20	UAE	False	Team Lead
1024	Ben Thompson	20	USA	True	General Manager
1025	Sindhya	20	India	True	Senior Architect
1026	Sindhya	20	India	True	Senior Architect
1027	Sindhya	20	India	True	Senior Architect
1028	Sindhya	20	India	True	Senior Architect
Total Employee : 36					

While binding the menu item using CommandBinding you can get the command parameter as `GridRecordContextMenuInfo` which contains the record of the corresponding row.

#### XML

```
<syncfusion:SfDataGrid.RecordContextMenu>
  <MenuItem Command="{Binding Source={x:Static
Member=local:ContextMenuCommands.Copy}}"
  CommandParameter="{Binding}"
  Header="Copy">
</MenuItem>
</syncfusion:SfDataGrid.RecordContextMenu>
```

#### C#

```
private static void OnCopyClicked(object obj)
{
    if (obj is GridRecordContextMenuInfo)
    {
        var grid = (obj as GridRecordContextMenuInfo).DataGrid;
        var record = (obj as GridRecordContextMenuInfo).Record;
        grid.GridCopyPaste.Copy();
    }
}
```

Context menu for column header

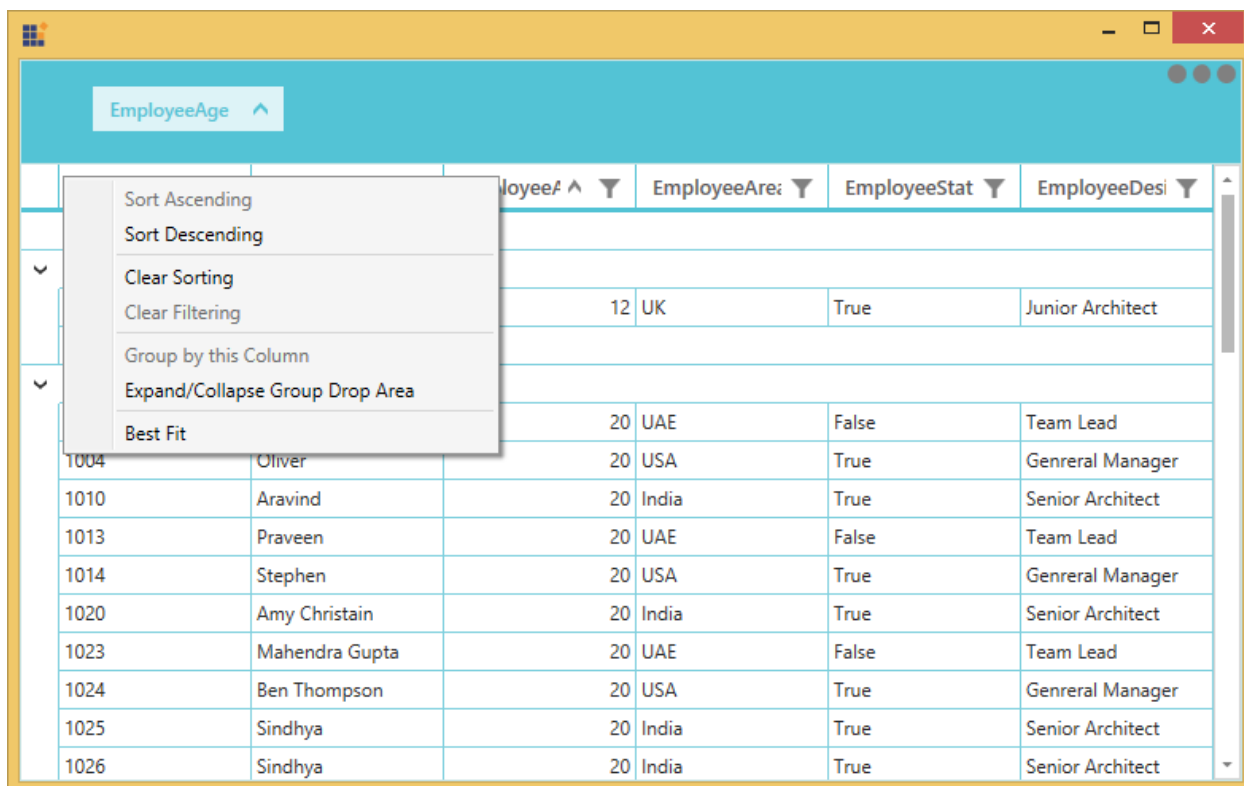
You can set the context menu for the header by using [SfDataGrid.HeaderContextMenu](#) property.

#### XML

```
<syncfusion:SfDataGrid.HeaderContextMenu>
<ContextMenu>
<MenuItem x:Name=" SortAscending " Header="SortAscending" />
<MenuItem x:Name=" SortDescending " Header="SortDescending" />
<MenuItem x:Name=" ClearSorting " Header="ClearSorting" />
<MenuItem x:Name=" ClearFiltering " Header="ClearFiltering" />
<MenuItem x:Name=" Group by this column " Header="Group by this column" />
<MenuItem x:Name=" Expand/Collapse Group Drop Area " Header="Expand/Collapse
Group Drop Area" />
<MenuItem x:Name=" BestFit " Header="BestFit" />
</ContextMenu>
</syncfusion:SfDataGrid.HeaderContextMenu>
```

## C#

```
this.dataGrid.HeaderContextMenu = new ContextMenu();
this.dataGrid.HeaderContextMenu.Items.Add(new MenuItem() { Header =
"SortAscending" });
this.dataGrid.HeaderContextMenu.Items.Add(new MenuItem() { Header =
"SortDescending" });
this.dataGrid.HeaderContextMenu.Items.Add(new MenuItem() { Header =
"ClearSorting" });
this.dataGrid.HeaderContextMenu.Items.Add(new MenuItem() { Header = "
ClearFiltering " });
this.dataGrid.HeaderContextMenu.Items.Add(new MenuItem() { Header = "Group
by this column" });
this.dataGrid.HeaderContextMenu.Items.Add(new MenuItem() { Header =
"Expand/Collapse Group Drop Area" });
this.dataGrid.HeaderContextMenu.Items.Add(new MenuItem() { Header =
"BestFit" });
```



While binding the menu item using CommandBinding you can get the parameter as `GridColumnContextMenuInfo` which contains the particular GridColumn.

#### XML

```
<syncfusion:SfDataGrid.HeaderContextMenu>
  <MenuItem Command="{Binding Source={x:Static
Member=local:ContextMenuCommands.SortAscending}}"
  CommandParameter="{Binding}"
  Header="Sort Ascending">
</MenuItem>
</syncfusion:SfDataGrid.HeaderContextMenu>
```

#### C#

```
private static void OnSortAscendingClicked(object obj)
{
    if (obj is GridColumnContextMenuInfo)
    {
        var grid = (obj as GridColumnContextMenuInfo).DataGrid;
        var column = (obj as GridColumnContextMenuInfo).Column;
        grid.SortColumnDescriptions.Clear();
        grid.SortColumnDescriptions.Add(new SortColumnDescription() { ColumnName =
column.MappingName, SortDirection = ListSortDirection.Ascending });
    }
}
```

Context menu for group drop area

You can set the context menu for the GroupDropArea by using [SfDataGrid.GroupDropAreaContextMenu](#) property.

#### XML

```
<syncfusion:SfDataGrid.GroupDropAreaContextMenu>
<ContextMenu>
<MenuItem x:Name=" ExpandAll " Header="ExpandAll" />
<MenuItem x:Name=" CollapseAll " Header="CollapseAll" />
<MenuItem x:Name=" ClearGroups " Header="ClearGroups" />
</ContextMenu>
</syncfusion:SfDataGrid.GroupDropAreaContextMenu>
```

#### C#

```
this.dataGrid.GroupDropAreaContextMenu = new ContextMenu();
this.dataGrid.GroupDropAreaContextMenu.Items.Add(new MenuItem() { Header =
"ExpandAll" });
this.dataGrid.GroupDropAreaContextMenu.Items.Add(new MenuItem() { Header =
"CollapseAll" });
this.dataGrid.GroupDropAreaContextMenu.Items.Add(new MenuItem() { Header =
"ClearGroups" });
```

EmployeeId	EmployeeName	EmployeeAge	EmployeeArea	EmployeeStatus	EmployeeDesignation
EmployeeAge : 12 - 1 Items					
1002	Peter	12	UK	True	Junior Architect
Total Employee : 1					
EmployeeAge : 20 - 19 Items					
1003	James	20	UAE	False	Team Lead
1004	Oliver	20	USA	True	General Manager
1010	Aravind	20	India	True	Senior Architect
1013	Praveen	20	UAE	False	Team Lead
1014	Stephen	20	USA	True	General Manager
1020	Amy Christain	20	India	True	Senior Architect
1023	Mahendra Gupta	20	UAE	False	Team Lead
1024	Ben Thompson	20	USA	True	General Manager
1025	Sindhya	20	India	True	Senior Architect
1026	Sindhya	20	India	True	Senior Architect
1027	Sindhya	20	India	True	Senior Architect
1028	Sindhya	20	India	True	Senior Architect
Total Employee : 36					

While binding the menu item using CommandBinding you can get the parameter as [GroupDropAreaContextMenuInfo](#).

#### XML

```
<syncfusion:SfDataGrid.GroupDropAreaContextMenu>
<MenuItem Command="{Binding Source={x:Static
Member=local:ContextMenuCommands.ExpandAll}}"
CommandParameter="{Binding}"
Header="Expand All">
</MenuItem>
</syncfusion:SfDataGrid.GroupDropAreaContextMenu>
```

**C#**

```
private static void OnFullExpandClicked(object obj)
{
    if (obj is Syncfusion.UI.Xaml.Grid.GridContextMenuInfo)
    {
        var grid = (obj as Syncfusion.UI.Xaml.Grid.GridContextMenuInfo).DataGrid;
        grid.ExpandAllGroup();
    }
}
```

Context menu for group item

You can set the context menu for the group drop item by using [SfDataGrid.GroupDropItemContextMenu](#) property.

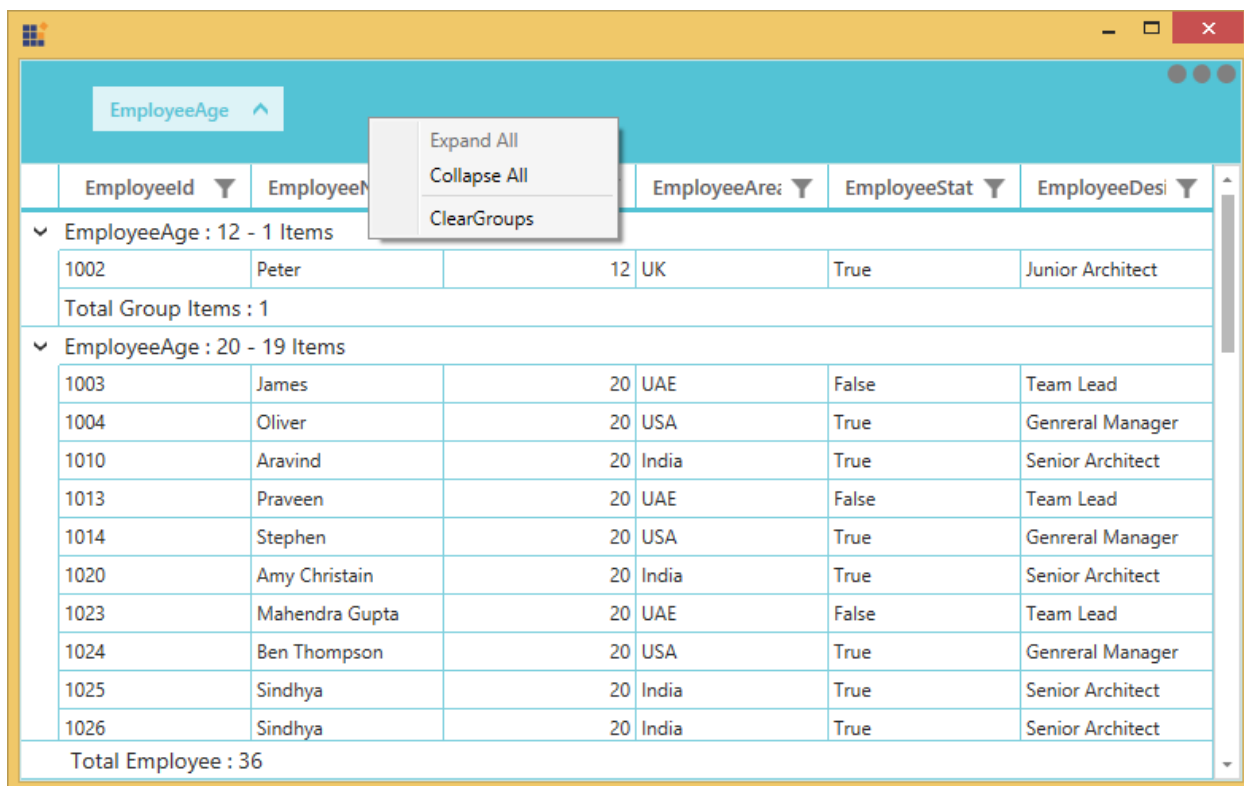
**XML**

```
<syncfusion:SfDataGrid.GroupDropItemContextMenu>
<ContextMenu>
<MenuItem x:Name=" ExpandAll " Header="ExpandAll" />
<MenuItem x:Name=" CollapseAll " Header="CollapseAll" />
<MenuItem x:Name=" SortAscending " Header="SortAscending" />
<MenuItem x:Name=" SortDescending " Header="SortDescending" />
<MenuItem x:Name=" ClearGroup " Header="ClearGroup" />
<MenuItem x:Name=" ClearSorting " Header="ClearSorting" />
</ContextMenu>
</syncfusion:SfDataGrid.GroupDropItemContextMenu>
```

**C#**

```
this.dataGrid.GroupDropItemContextMenu = new ContextMenu();
this.dataGrid.GroupDropItemContextMenu.Items.Add(new MenuItem() { Header =
"ExpandAll" });
this.dataGrid.GroupDropItemContextMenu.Items.Add(new MenuItem() { Header =
"CollapseAll" });
this.dataGrid.GroupDropItemContextMenu.Items.Add(new MenuItem() { Header =
"SortAscending" });
this.dataGrid.GroupDropItemContextMenu.Items.Add(new MenuItem() { Header =
"SortDescending" });
this.dataGrid.GroupDropItemContextMenu.Items.Add(new MenuItem() { Header =
"ClearGroup" });
this.dataGrid.GroupDropItemContextMenu.Items.Add(new MenuItem() { Header =
"ClearSorting" });
```





While binding the menu item using CommandBinding you can get the parameter as `GridColumnContextMenuInfo` which contains the particular GridColumn.

#### XML

```
<syncfusion:SfDataGrid.GroupDropItemContextMenu>
  <MenuItem Command="{Binding Source={x:Static
    Member=local:ContextMenuCommands.CollapseAll}}"
    CommandParameter="{Binding}"
    Header="Collapse All">
  </MenuItem>
</syncfusion:SfDataGrid.GroupDropItemContextMenu>
```

#### C#

```
private static void OnFullCollapseClicked(object obj)
{
    if (obj is GridContextMenuInfo)
    {
        var grid = (obj as GridContextMenuInfo).DataGrid;
        grid.CollapseAllGroup();
    }
}
```

Context menu for caption summary wow

You can set the context menu for the group caption by using [SfDataGrid.GroupCaptionContextMenu](#) property.

#### XML

```

<syncfusion:SfDataGrid.GroupCaptionContextMenu>
  <ContextMenu>
    <MenuItem x:Name=" Expand " Header="Expand" />
    <MenuItem x:Name=" Collapse " Header="Collapse" />
  </ContextMenu>
</syncfusion:SfDataGrid.GroupCaptionContextMenu>

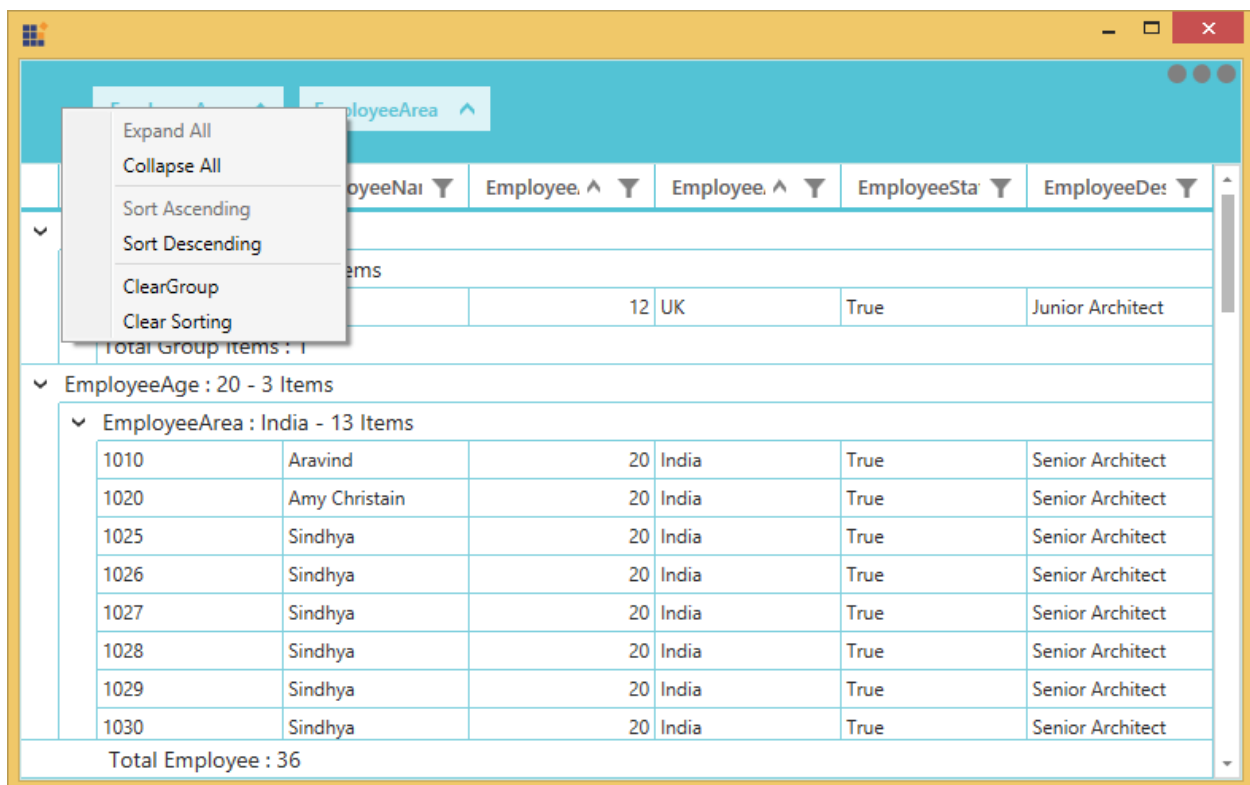
```

**C#**

```

this.dataGrid.GroupCaptionContextMenu = new ContextMenu();
this.dataGrid.GroupCaptionContextMenu.Items.Add(new MenuItem() { Header =
"Expand" });
this.dataGrid.GroupCaptionContextMenu.Items.Add(new MenuItem() { Header =
"Collapse" });

```



While binding the menu item using CommandBinding you can get the command parameter as `GridRecordContextMenuInfo` which contains the record of the corresponding row.

**XML**

```

<syncfusion:SfDataGrid.GroupCaptionContextMenu>
  <MenuItem Command="{Binding Source={x:Static
Member=local:ContextMenuCommands.Expand}} "
  CommandParameter="{Binding}"
  Header="Expand" />
</syncfusion:SfDataGrid.GroupCaptionContextMenu>

```

**C#**

```
private static void OnExpandClicked(object obj)
{
    if (obj is GridRecordContextMenuInfo)
    {
        var grid = (obj as GridRecordContextMenuInfo).DataGrid;
        var group = (obj as GridRecordContextMenuInfo).Record as Group;
        grid.ExpandGroup(group);
    }
}
```

Context menu for group summary row

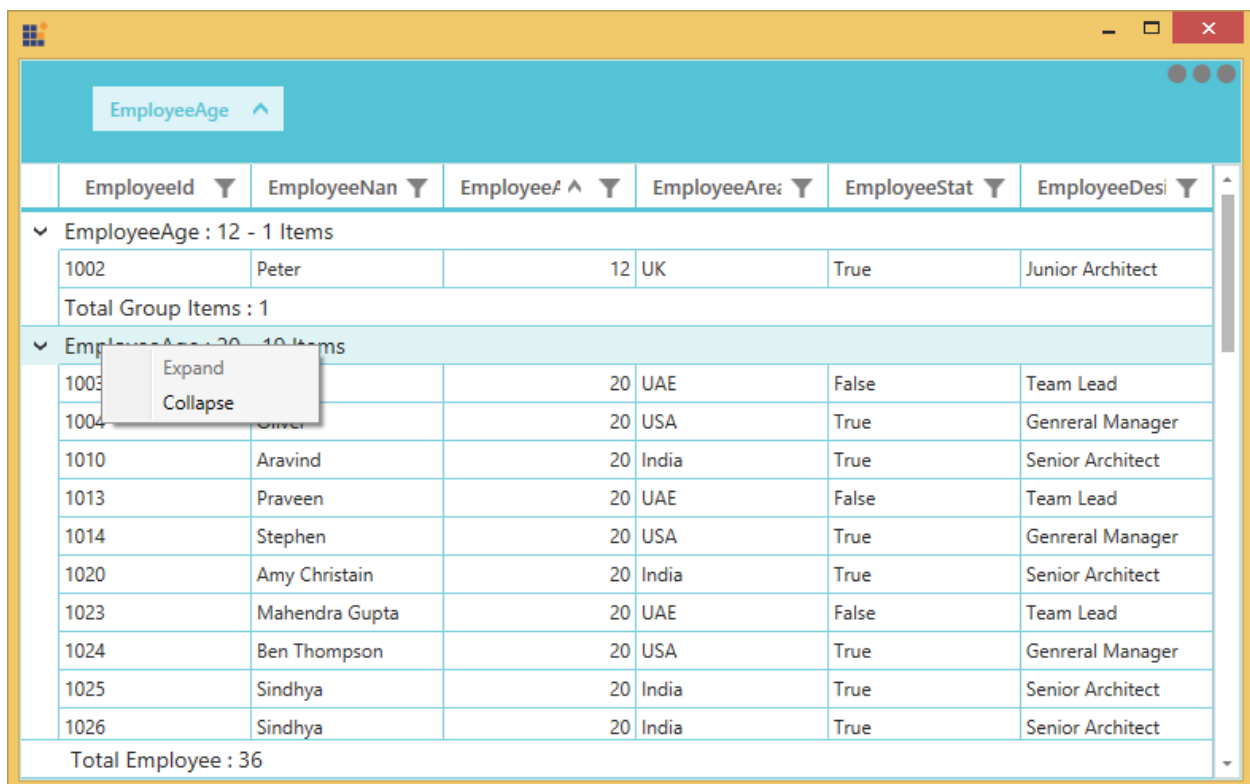
You can set the context menu for the group summary by using [SfDataGrid.GroupSummaryContextMenu](#) property.

#### XML

```
<syncfusion:SfDataGrid.GroupSummaryContextMenu>
<ContextMenu>
<MenuItem x:Name="ClearSummary" Header="ClearSummary" />
</ContextMenu>
</syncfusion:SfDataGrid.GroupSummaryContextMenu>
```

#### C#

```
this.dataGrid.GroupSummaryContextMenu = new ContextMenu();
this.dataGrid.GroupSummaryContextMenu.Items.Add(new MenuItem() { Header =
"ClearSummary" });
```



While binding the menu item using CommandBinding you can get the command parameter as `GridRecordContextMenuInfo` which contains the record of the corresponding row.

### XML

```
<syncfusion:SfDataGrid.GroupSummaryContextMenu>
  <MenuItem Command="{Binding Source={x:Static
    Member=local:ContextMenuCommands.ClearSummary}}"
    CommandParameter="{Binding}"
    Header="Clear Summary">
  </MenuItem>
</syncfusion:SfDataGrid.GroupSummaryContextMenu>
```

### C#

```
private static void OnClearSummaryClicked(object obj)
{
    if (obj is GridRecordContextMenuInfo)
    {
        var grid = (obj as GridRecordContextMenuInfo).DataGrid;
        if (grid.GroupSummaryRows.Any())
            grid.GroupSummaryRows.Clear();
    }
}
```

Context menu for table summary row

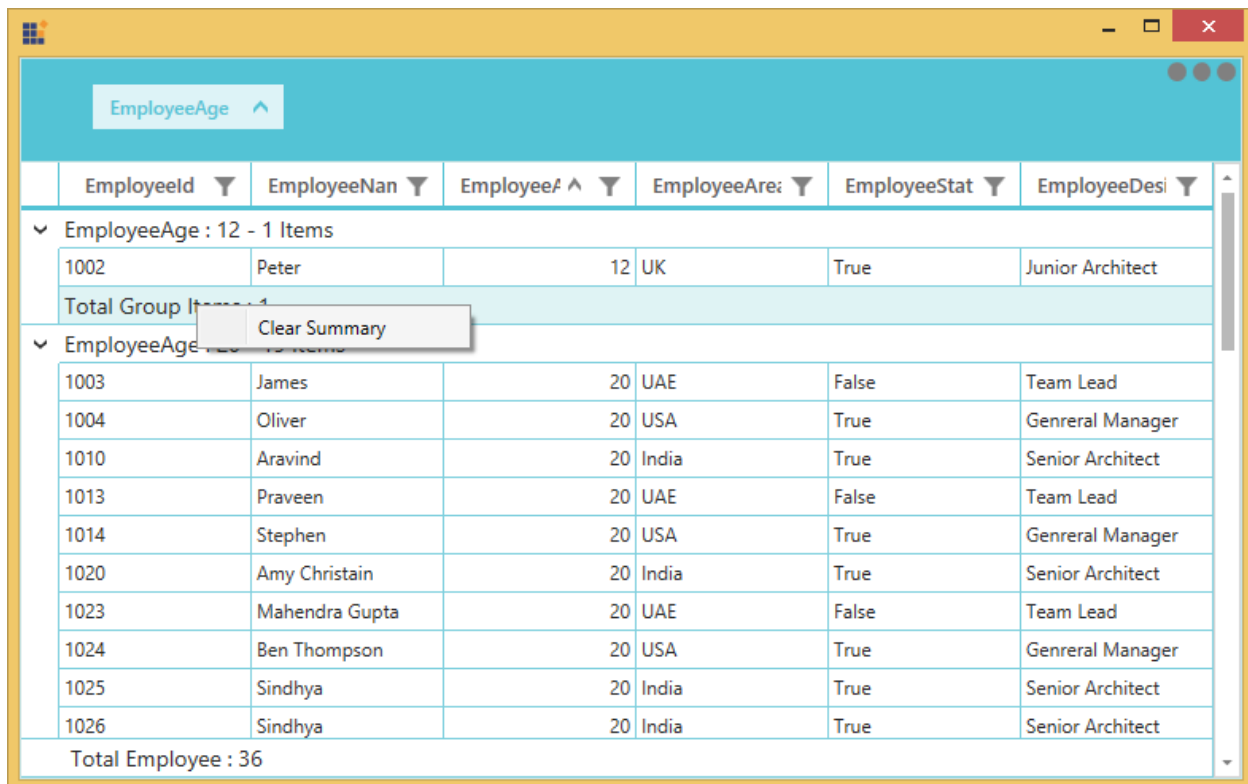
You can set the context menu for the table summary by using [SfDataGrid.TableSummaryContextMenu](#) property.

### XML

```
<syncfusion:SfDataGrid.TableSummaryContextMenu>
  <ContextMenu>
    <MenuItem x:Name="TableSummaryCount" Header="Count" />
    <MenuItem x:Name="TableSummaryMax" Header="Max" />
    <MenuItem x:Name="TableSummaryMin" Header="Min" />
    <MenuItem x:Name="TableSummaryMin" Header="Average" />
    <MenuItem x:Name="TableSummaryMin" Header="Sum" />
  </ContextMenu>
</syncfusion:SfDataGrid.TableSummaryContextMenu>
```

### C#

```
this.dataGrid.TableSummaryContextMenu = new ContextMenu();
this.dataGrid.TableSummaryContextMenu.Items.Add(new MenuItem() { Header =
    "Count" });
this.dataGrid.TableSummaryContextMenu.Items.Add(new MenuItem() { Header =
    "Max" });
this.dataGrid.TableSummaryContextMenu.Items.Add(new MenuItem() { Header =
    "Min" });
this.dataGrid.TableSummaryContextMenu.Items.Add(new MenuItem() { Header =
    "Average" });
this.dataGrid.TableSummaryContextMenu.Items.Add(new MenuItem() { Header =
    "Sum" });
```



While binding the menu item using CommandBinding you can get the command parameter as `GridRecordContextMenuInfo` which contains the record of the corresponding row.

### XML

```
<syncfusion:SfDataGrid.TableSummaryContextMenu>
  <MenuItem Command="{Binding Source={x:Static
Member=local:ContextMenuCommands.TotalSummaryCount}}}"
  CommandParameter="{Binding}"
  Header="Count">
  </MenuItem>
</syncfusion:SfDataGrid.TableSummaryContextMenu>
```

### C#

```
private static void OnTotalSummaryCountClicked(object obj)
{
    if (obj is GridRecordContextMenuInfo)
    {
        var grid = (obj as GridRecordContextMenuInfo).DataGrid;
        var record = (obj as GridRecordContextMenuInfo).Record as
        SummaryRecordEntry;
        if (record != null)
        {
            var summaryRow = new GridSummaryRow() { Name = "totalGroupSummaryRow", Title =
            "{totalSummary}", ShowSummaryInRow = true };
            summaryRow.SummaryColumns.Add(new GridSummaryColumn() { Name =
            "totalSummary", MappingName = "EmployeeId", SummaryType =
            SummaryType.CountAggregate, Format = "Total Employee Count : {Count}" });
            grid.TableSummaryRows.Clear();
        }
    }
}
```

```
grid.TableSummaryRows.Add(summaryRow);
}
}
}
```

## Events

### [GridContextMenuOpening](#)

[GridContextMenuOpening](#) event occurs while opening the context menu in SfDataGrid.

[GridContextMenuEventArgs](#) has the following members which provides the information about GridContextMenuOpening event.

- [ContextMenu](#) – Gets the corresponding context menu.
- [ContextMenuInfo](#) – Returns the context menu info based on the row which opens the context menu.
- [ContextMenuType](#) – Returns the type of context menu.
- [RowColumnIndex](#) – [RowColumnIndex](#) of the context menu which is currently going to open. [RowColumnIndex](#) is updated only for the [RecordContextMenu](#) and remains left empty.
- [Handled](#) – Indicates whether the [GridContextMenuOpening](#) event is handled or not.

## Customization of context menu

*Change the menu item when the context menu opening.*

You can use the [GridContextMenuOpening](#) event to change the menu item when the context menu opening.

### XML

```
<syncfusion:SfDataGrid.RecordContextMenu>
  <ContextMenu>
    <MenuItem Command="{Binding Source={x:Static
Member=local:ContextMenuCommands.Cut}}"
CommandParameter="{Binding}"
Header="Cut">
  </MenuItem>
    <MenuItem Command="{Binding Source={x:Static
Member=local:ContextMenuCommands.Copy}}"
CommandParameter="{Binding}"
Header="Copy">
  </MenuItem>
    <MenuItem Command="{Binding Source={x:Static
Member=local:ContextMenuCommands.Paste}}"
CommandParameter="{Binding}"
Header="Paste">
  </MenuItem>
  </ContextMenu>
</syncfusion:SfDataGrid.RecordContextMenu>
```

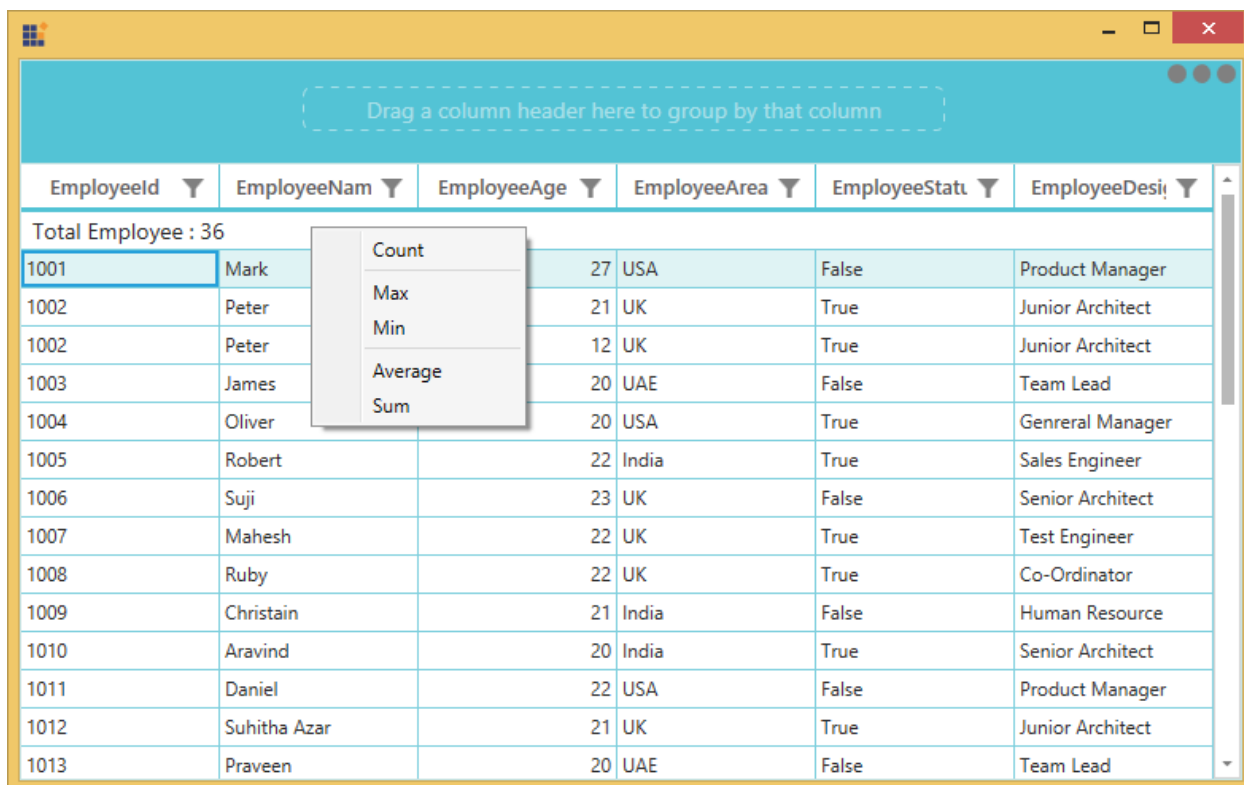
### C#

```
this.dataGrid.GridContextMenuOpening += dataGrid_GridContextMenuOpening;
void dataGrid_GridContextMenuOpening(object sender, GridContextMenuEventArgs e)
```

```

{
e.ContextMenu.Items.Clear();
if(e.ContextMenuType == ContextMenuType.RecordCell)
{
e.ContextMenu.Items.Add(new MenuItem() { Header="Record"});
e.ContextMenu.Items.Add(new MenuItem() { Header = "Data" });
}
}

```



### Changing background of Context menu

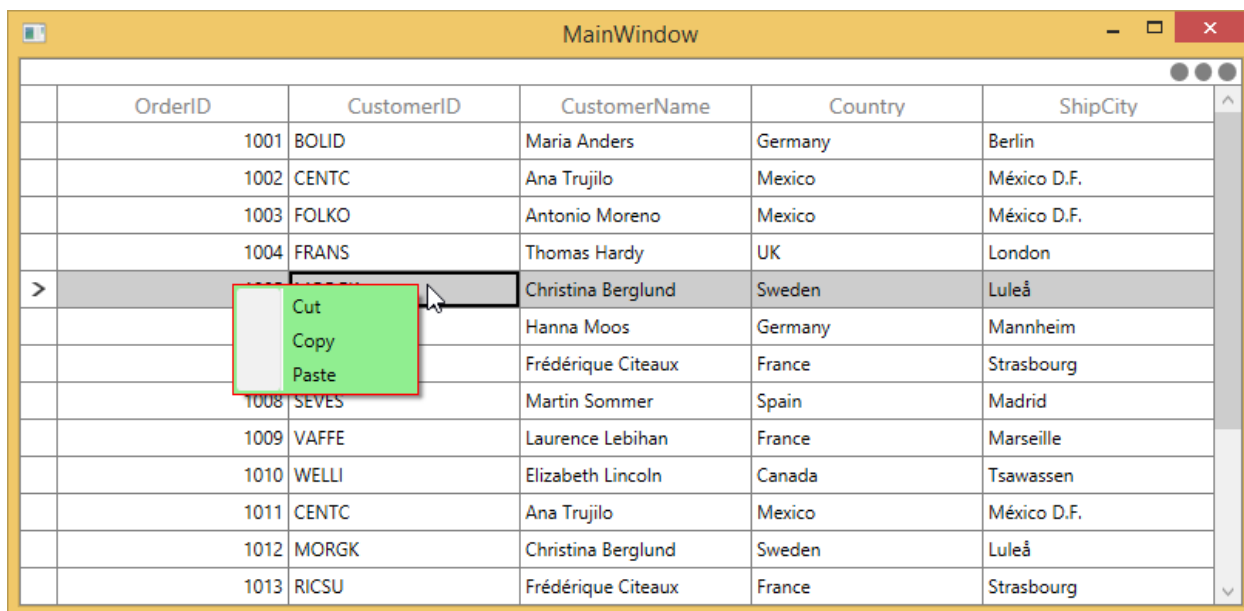
You can change the appearance of the context menu by customizing the style with TargetType as ContextMenu.

### XML

```

<Style x:Name="ToolTipStyle" TargetType="ContextMenu">
<Setter Property="BorderThickness" Value="1,1,1,1" />
<Setter Property="BorderBrush" Value="Red" />
<Setter Property="Background" Value="LightGreen" />
</Style>
<ContextMenu>
<MenuItem x:Name="Cut" Header="Cut" />
<MenuItem x:Name="Copy" Header="Copy" />
<MenuItem x:Name="Paste" Header="Paste" />
</ContextMenu>

```



OrderID	CustomerID	CustomerName	Country	ShipCity
1001	BOLID	Maria Anders	Germany	Berlin
1002	CENTC	Ana Trujilo	Mexico	México D.F.
1003	FOLKO	Antonio Moreno	Mexico	México D.F.
1004	FRANS	Thomas Hardy	UK	London
>		Christina Berglund	Sweden	Luleå
		Hanna Moos	Germany	Mannheim
		Frédérique Citeaux	France	Strasbourg
1008	SEVES	Martin Sommer	Spain	Madrid
1009	VAFFE	Laurence Lebihan	France	Marseille
1010	WELLI	Elizabeth Lincoln	Canada	Tsawassen
1011	CENTC	Ana Trujilo	Mexico	México D.F.
1012	MORGK	Christina Berglund	Sweden	Luleå
1013	RICSU	Frédérique Citeaux	France	Strasbourg

## Serialization and Deserialization in WPF DataGrid

SfDataGrid allows you to serialize and deserialize the SfDataGrid settings using [DataContractSerializer](#).

### Serialization

You can serialize the SfDataGrid by using [SfDataGrid.Serialize](#) method which exports the current DataGrid control properties to an XML file.

#### C#

```
using (var file = File.Create("DataGrid.xml"))
{
    dataGrid.Serialize(file);
}
```

### Serialize as Stream

You can store the SfDataGrid settings as [Stream](#) using [Serialize](#) method by passing the stream.

#### C#

```
FileStream stream = new FileStream("DataGrid", FileMode.Create);
this.dataGrid.Serialize(stream);
```

### Serialization options

SfDataGrid serialization operation can be customized by passing [SerializationOptions](#) instance as an argument to [Serialize](#) method.

### Serialize sorting

By default, SfDataGrid allows you to serialize the sorting operation. You can disable the sorting serialization by setting the [SerializationOptions.SerializeSorting](#) to `false`.

#### C#

```
using (var file = File.Create("DataGrid.xml"))
{
    SerializationOptions options = new SerializationOptions();
```



```
options.SerializeSorting = false;  
dataGrid.Serialize(file, options);  
}
```

#### *Serialize grouping*

By default, SfDataGrid allows you to serialize the grouping operation. You can disable the grouping serialization by setting the [SerializationOptions.SerializeGrouping](#) to false.

#### **C#**

```
using (var file = File.Create("DataGrid.xml"))  
{  
    SerializationOptions options = new SerializationOptions();  
    options.SerializeGrouping = false;  
    dataGrid.Serialize(file, options);  
}
```

#### *Serialize filtering*

By default, SfDataGrid allows you to serialize the filtering operation. You can disable the filtering serialization by setting the [SerializationOptions.SerializeFiltering](#) to false.

#### **C#**

```
using (var file = File.Create("DataGrid.xml"))  
{  
    SerializationOptions options = new SerializationOptions();  
    options.SerializeFiltering = false;  
    dataGrid.Serialize(file, options);  
}
```

#### *Serialize columns*

By default, SfDataGrid allows you to serialize the columns manipulation operation. You can disable the columns serialization by setting the [SerializationOptions.SerializeColumns](#) to false.

#### **C#**

```
using (var file = File.Create("DataGrid.xml"))  
{  
    SerializationOptions options = new SerializationOptions();  
    options.SerializeColumns = false;  
    dataGrid.Serialize(file, options);  
}
```

#### *Serialize summaries*

By default, SfDataGrid allows you to serialize the caption summary, group summary and table summary settings in SfDataGrid.

You can disable the summaries serialization by setting [SerializationOptions.SerializeCaptionSummary](#) , [SerializationOptions.SerializeGroupSummaries](#) , [SerializationOptions.SerializeTableSummaries](#) properties to false

#### **C#**

```
using (var file = File.Create("DataGrid.xml"))
```

```
{
    SerializationOptions options = new SerializationOptions();
    options.SerializeCaptionSummary = false;
    options.SerializeGroupSummaries = false;
    options.SerializeTableSummaries = false;
    dataGrid.Serialize(file, options);
}
```

#### *Serialize stacked headers*

By default, SfDataGrid allows you to serialize the stack headers operation. You can disable the stack headers serialization by setting the [SerializationOptions.SerializeStackedHeaders](#) to `false`.

#### **C#**

```
using (var file = File.Create("DataGrid.xml"))
{
    SerializationOptions options = new SerializationOptions();
    options.SerializeStackHeaders = false;
    dataGrid.Serialize(file, options);
}
```

#### *Serialize Details View*

By default, SfDataGrid allows you to serialize the DetailsViewDefinition. You can disable the DetailsViewDefinition serialization by setting the [SerializationOptions.SerializeDetailsViewDefinition](#) to `false`.

#### **C#**

```
using (var file = File.Create("DataGrid.xml"))
{
    SerializationOptions options = new SerializationOptions();
    options.SerializeDetailsViewDefinition = false;
    dataGrid.Serialize(file, options);
}
```

#### *Serialize unbound rows*

By default, SfDataGrid allows you to serialize the unbound rows settings. You can disable the unbound rows serialization by setting the [SerializationOptions.SerializeUnBoundRows](#) to `false`.

#### **C#**

```
using (var file = File.Create("DataGrid.xml"))
{
    SerializationOptions options = new SerializationOptions();
    options.SerializeUnBoundRows = false;
    dataGrid.Serialize(file, options);
}
```

#### *Deserialization*

You can deserialize the SfDataGrid setting by using [SfDataGrid.Deserialize](#) method which reconstructs the SfDataGrid based on the setting in the stored XML file.

#### **C#**

```
using (var file = File.Open("DataGrid.xml", FileMode.Open))
{
    dataGrid.Deserialize(file);
}
```

### *Deserialize from Stream*

You can deserialize the SfDataGrid settings from [Stream](#) using [Deserialize](#) method.

#### **C#**

```
FileStream fileStream = new FileStream("DataGrid", FileMode.Open);
this.dataGrid.Deserialize(fileStream);
```

### *Deserialization options*

Deserialization operation can be customized by passing [DeserializationOptions](#) instance as an argument to [Deserialize](#) method.

#### *Deserialize sorting*

By default, SfDataGrid allows you to deserialize the sorting operation. You can disable the sorting deserialization by setting the [DeserializationOptions.DeserializeSorting](#) to `false`.

#### **C#**

```
using (var file = File.Open("DataGrid.xml", FileMode.Open))
{
    DeserializationOptions options = new DeserializationOptions();
    options.DeserializeSorting = false;
    dataGrid.Deserialize(file, options);
}
```

#### *Deserialize grouping*

By default, SfDataGrid allows you to deserialize the grouping operation. You can disable the grouping deserialization by setting the [DeserializationOptions.DeserializeGrouping](#) to `false`.

#### **C#**

```
using (var file = File.Open("DataGrid.xml", FileMode.Open))
{
    DeserializationOptions options = new DeserializationOptions();
    options.DeserializeGrouping = false;
    dataGrid.Deserialize(file, options);
}
```

#### *Deserialize filtering*

By default, SfDataGrid allows you to deserialize the filtering. you can disable the filtering deserialization by setting [DeserializationOptions.DeserializeFiltering](#) to `false`.

#### **C#**

```
using (var file = File.Open("DataGrid.xml", FileMode.Open))
{
    DeserializationOptions options = new DeserializationOptions();
    options.DeserializeFiltering = false;
    dataGrid.Deserialize(file, options);
}
```

```
}
```

#### *Deserialize columns*

By default, SfDataGrid allows you to deserialize the columns manipulation operations. You can disable the columns deserialization by setting the [DeserializationOptions.DeserializeColumns](#) to **false**.

#### **C#**

```
using (var file = File.Open("DataGrid.xml", FileMode.Open))
{
    DeserializationOptions options = new DeserializationOptions();
    options.DeserializeColumns = false;
    dataGrid.Deserialize(file, options);
}
```

#### *Deserialize summaries*

By default, SfDataGrid allows you to deserialize the group summary, caption summary and table summary settings.

You can disable the summaries deserialization by setting [DeserializationOptions.DeserializeCaptionSummary](#), [DeserializationOptions.DeserializeGroupSummaries](#), [DeserializationOptions.DeserializeTableSummaries](#) properties to **false**.

#### **C#**

```
using (var file = File.Open("DataGrid.xml", FileMode.Open))
{
    DeserializationOptions options = new DeserializationOptions();
    options.DeserializeCaptionSummary = false;
    options.DeserializeGroupSummaries = false;
    options.DeserializeTableSummaries = false;
    dataGrid.Deserialize(file, options);
}
```

#### *Deserialize stacked headers*

By default, SfDataGrid allows you to deserialize the stack headers. You can disable the stacked headers deserialization by setting the [DeserializationOptions.DeserializeStackedHeaders](#) to **false**.

#### **C#**

```
using (var file = File.Open("DataGrid.xml", FileMode.Open))
{
    DeserializationOptions options = new DeserializationOptions();
    options.DeserializeStackedHeaders = false;
    dataGrid.Deserialize(file, options);
}
```

#### *Deserialize Details View*

By default, SfDataGrid allows you to deserialize the DetailsViewDefinition. You can disable the DetailsViewDefinition deserialization by setting the [DeserializationOptions.DeserializeDetailsViewDefinition](#) to **false**.

#### **C#**

```
using (var file = File.Open("DataGrid.xml", FileMode.Open))
{
    DeserializationOptions options = new DeserializationOptions();
    options.DeserializeDetailsViewDefinition = false;
    dataGrid.Deserialize(file, options);
}
```

#### *Deserialize unbound rows*

By default, SfDataGrid allows you to deserialize the unbound rows settings. You can disable the unbound rows deserialization by setting the [DeserializationOptions.DeserializeUnBoundRows](#) to `false`.

#### **C#**

```
using (var file = File.Open("DataGrid.xml", FileMode.Open))
{
    DeserializationOptions options = new DeserializationOptions();
    options.DeserializeUnBoundRows = false;
    dataGrid.Deserialize(file, options);
}
```

#### Customizing Serialization and Deserialization Operations

SfDataGrid allows you to customize the serialization and deserialization operations by deriving [SerializationController](#) class and override the necessary virtual methods.

#### *Serialize custom column*

By default, the unknown(custom) column types are serialized as `GridTextColumn` type. If you want to serialize the custom column, you have to add custom column type into predefined types.

In the below code snippet, DatePickerColumn is created. For more information about creating custom column refer [here](#).

#### **C#**

```
public class DatePickerColumn : GridColumn
{
    public DatePickerColumn()
    {
        SetCellType("DatePicker");
    }

    public static readonly DependencyProperty DateMappingNameProperty =
        DependencyProperty.Register("DateMappingName",
            typeof(string), typeof(DatePickerColumn));
    public string DateMappingName
    {
        get { return (string)GetValue(DateMappingNameProperty); }
        set { SetValue(DateMappingNameProperty, value); }
    }

    protected override Freezable CreateInstanceCore()
    {
        return new DatePickerColumn();
    }

    protected override void SetDisplayBindingConverter()
    {
        (this.DisplayBinding as Binding).Converter = new CustomConverter();
    }
}
```

```
}

```

In the below code snippet, the DatePickerColumn is defined in SfDataGrid.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<local:DatePickerColumn HeaderText="OrderDate" MappingName="OrderDate" />
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```

To serialize the above DatePickerColumn, follow the below steps.

1. Create a class derived from [SerializableGridColumn](#) and define the custom column properties in `SerializableCustomGridColumn` class.

### C#

```
[DataContract(Name="SerializableCustomGridColumn")]
public class SerializableCustomGridColumn : SerializableGridColumn
{
    [DataMember]
    public string DateMappingName { get; set; }
}
```

2. Create a new class named as `SerializationControllerExt` by overriding [SerializationController](#) class.

### C#

```
dataGrid.SerializationController = new SerializationControllerExt(dataGrid);
public class SerializationControllerExt : SerializationController
{
    public SerializationControllerExt(SfDataGrid dataGrid)
    : base(dataGrid)
    {
    }
}
```

3. You can get the custom column property settings for serialization by overriding the [GetSerializableGridColumn](#) virtual method.

### C#

```
public class SerializationControllerExt : SerializationController
{
    public SerializationControllerExt(SfDataGrid dataGrid)
    : base(dataGrid)
    {
    }
    protected override SerializableGridColumn
    GetSerializableGridColumn(GridColumn column)
```

```

{
    if (column.MappingName == "OrderDate")
    {
        return new SerializableCustomGridColumn();
    }
    return base.GetSerializableGridColumn(column);
}
}

```

4. Store the custom column property settings during serialization by overriding the [StoreGridColumnProperties](#) virtual method.

#### C#

```

public class SerializationControllerExt : SerializationController
{
    public SerializationControllerExt(SfDataGrid dataGrid)
    : base(dataGrid)
    {
    }

    protected override void StoreGridColumnProperties(GridColumn column,
        SerializableGridColumn serializableColumn)
    {
        base.StoreGridColumnProperties(column, serializableColumn);
        if (column is DatePickerColumn)
            (serializableColumn as SerializableCustomGridColumn).DateMappingName =
            (column as DatePickerColumn).DateMappingName;
    }
}

```

5. Add the custom column in to known column types by overriding the [KnownTypes](#) virtual method.

#### C#

```

public class SerializationControllerExt : SerializationController
{
    public SerializationControllerExt(SfDataGrid dataGrid)
    : base(dataGrid)
    {
    }

    public override Type[] KnownTypes()
    {
        var types = base.KnownTypes();
        var list = types.Cast<Type>().ToList();
        list.Add(typeof(SerializableCustomGridColumn));
        return list.ToArray();
    }
}

```

6. During deserialization, you can get the custom column settings from [SerializableGridColumn](#) by overriding [GetGridColumn](#) virtual method.

#### C#

```

public class SerializationControllerExt : SerializationController
{

```

```

public SerializationControllerExt(SfDataGrid dataGrid)
: base(dataGrid)
{
}

protected override GridColumn GetGridColumn(SerializableGridColumn
serializableColumn)
{
if (serializableColumn is SerializableCustomGridColumn)
return new DatePickerColumn();
return base.GetGridColumn(serializableColumn);
}
}

```

7. Now restore the custom column settings from `SerializableGridColumn` by overriding the `RestoreColumnProperties` virtual method.

### C#

```

public class SerializationControllerExt : SerializationController
{
public SerializationControllerExt(SfDataGrid dataGrid)
: base(dataGrid)
{
}

protected override void RestoreColumnProperties(SerializableGridColumn
serializableColumn, GridColumn column)
{
base.RestoreColumnProperties(serializableColumn, column);
if (column is DatePickerColumn)
(column as DatePickerColumn).DateMappingName = (serializableColumn as
SerializableCustomGridColumn).DateMappingName;
}
}

```

You can download the sample demo [here](#).

### Serializing template column content

By default, you cannot serialize the template content in `SfDataGrid`. This is the default behavior during Serialization and Deserialization operation.

### XML

```

<Application.Resources>
<DataTemplate x:Key="cellTemplate">
<Button Content="{Binding Value}" />
</DataTemplate>
</Application.Resources>
<syncfusion:SfDataGrid x:Name="dataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding Orders}">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTemplateColumn CellTemplate="{StaticResource cellTemplate}"
HeaderText="Order ID"
MappingName="OrderID"
SetCellBoundValue="True" />
</syncfusion:SfDataGrid.Columns>

```



```
</syncfusion:SfDataGrid>
```

If you want to serialize and deserialize the template content, you have to reconstruct the same template during deserialization in [RestoreColumnProperties](#) method.

### C#

```
this.dataGrid.SerializationController = new
SerializationControllerExt(this.dataGrid);
public class SerializationControllerExt : SerializationController
{
    public SerializationControllerExt(SfDataGrid grid)
    : base(grid)
    {
    }
    protected override void RestoreColumnProperties(SerializableGridColumn
serializableColumn, GridColumn column)
    {
        base.RestoreColumnProperties(serializableColumn, column);
        if (column is GridTemplateColumn)
        {
            if (column.MappingName == "OrderID")
            {
                column.CellTemplate = App.Current.Resources["cellTemplate"] as DataTemplate;
            }
        }
    }
}
```

You can download the sample demo [here](#).

### Export To Excel in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) provides support to export data to excel. It also provides support for grouping, filtering, sorting, paging, unbound rows, merged cells, stacked headers and Details View while exporting.

The following assemblies needs to be added for exporting to excel.

- Syncfusion.SfGridConverter.WPF
- Syncfusion.XlsIO.Base

For NuGet package, have to install [Syncfusion.DataGridExcelExport.WPF](#) package. For more details refer this [UG link](#).

You can export SfDataGrid to excel by using the [ExportToExcel](#) extension method present in the [Syncfusion.UI.Xaml.Grid.Converter](#) namespace.

### C#

```
using Syncfusion.UI.Xaml.Grid.Converter;
var options = new ExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

---

**Note:** SfDataGrid exports data to excel by using [XlsIO](#). You can refer [XlsIO documentation](#) for manipulating exported work sheets.

---

#### Excel exporting options

Exporting operation can be customized by passing [ExcelExportingOptions](#) instance as argument to `ExportToExcel` method.

#### Export mode

By default, actual value only will be exported to excel. If you want to export the display text, you need to set [ExportMode](#) property as `Text`.

#### C#

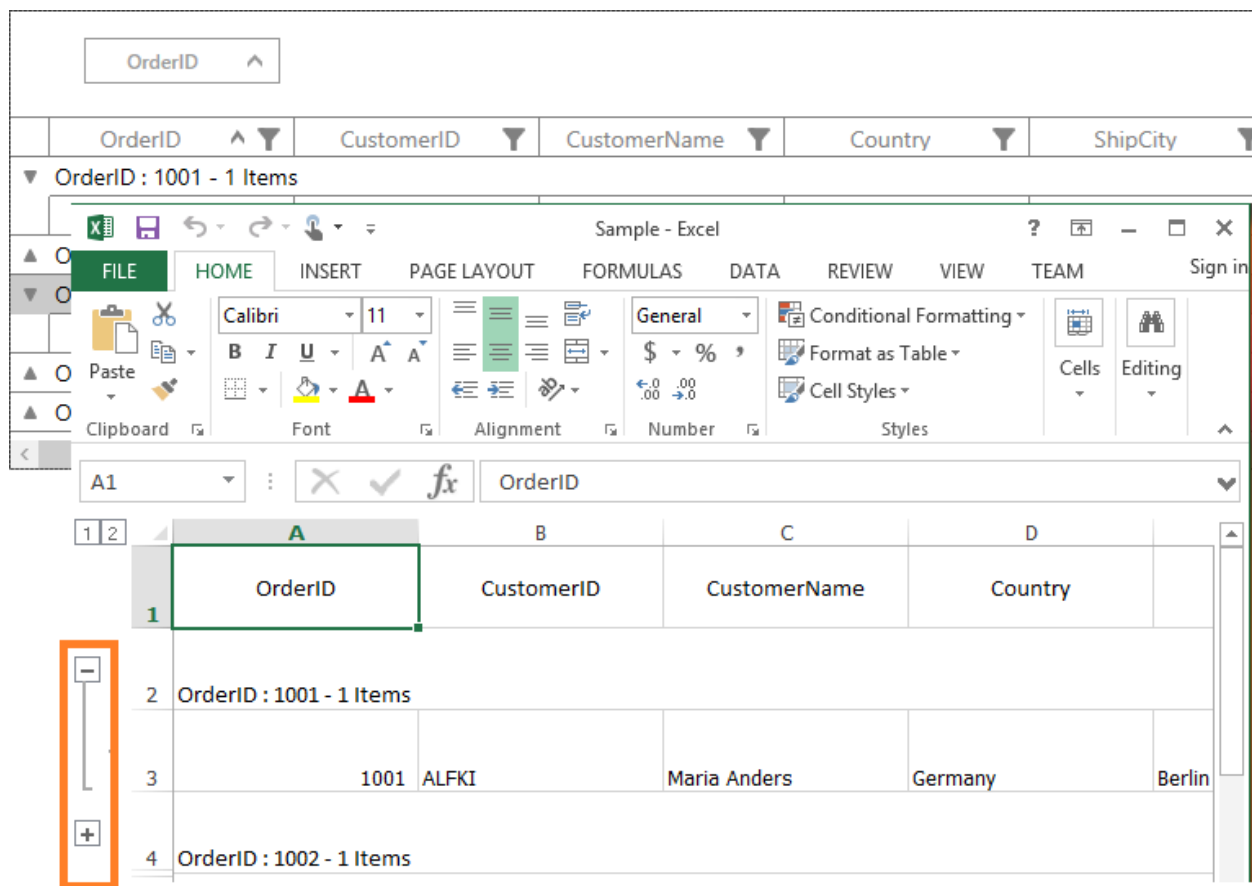
```
var options = new ExcelExportingOptions();
options.ExportMode = ExportMode.Text;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

#### Export groups with outlines

By default, all the groups in dataGrid will be exported in expanded state. You can enable outlines in excel based on groups by setting the [AllowOutlining](#) property as `true` in [ExcelExportingOptions](#).

#### C#

```
var options = new ExcelExportingOptions();
options.AllowOutlining = true;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```



#### Exclude columns while exporting

By default, all the columns (including hidden columns) in SfDataGrid will be exported to Excel. If you want to exclude some columns while exporting to Excel, you can use [ExcludeColumns](#) field in [ExcelExportingOptions](#).

#### C#

```
var options = new ExcelExportingOptions();
options.ExcludeColumns.Add("CustomerName");
options.ExcludeColumns.Add("Country");
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workbook = excelEngine.Excel.Workbooks[0];
workbook.SaveAs("Sample.xlsx");
```

Here, the columns having **CustomerName** and **Country** as MappingName are excluded while exporting.

#### Excel version

While exporting to Excel, you can specify the excel version by using [ExcelVersion](#) property.

#### C#

```
var options = new ExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workbook = excelEngine.Excel.Workbooks[0];
workbook.SaveAs("Sample.xlsx");
```

### Exporting stacked headers

You can export stacked headers to excel by setting [ExportStackedHeaders](#) property to `true`.

#### C#

```
var options = new ExcelExportingOptions();
options.ExportStackedHeaders = true;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

### Exporting merged cells

You can export merged cells to excel by setting [ExportMergedCells](#) property as `true`.

#### C#

```
var options = new ExcelExportingOptions();
options.ExportMergedCells = true;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

### Exporting unbound rows

You can export unbound rows to excel by setting [ExportUnBoundRows](#) property as `true`.

#### C#

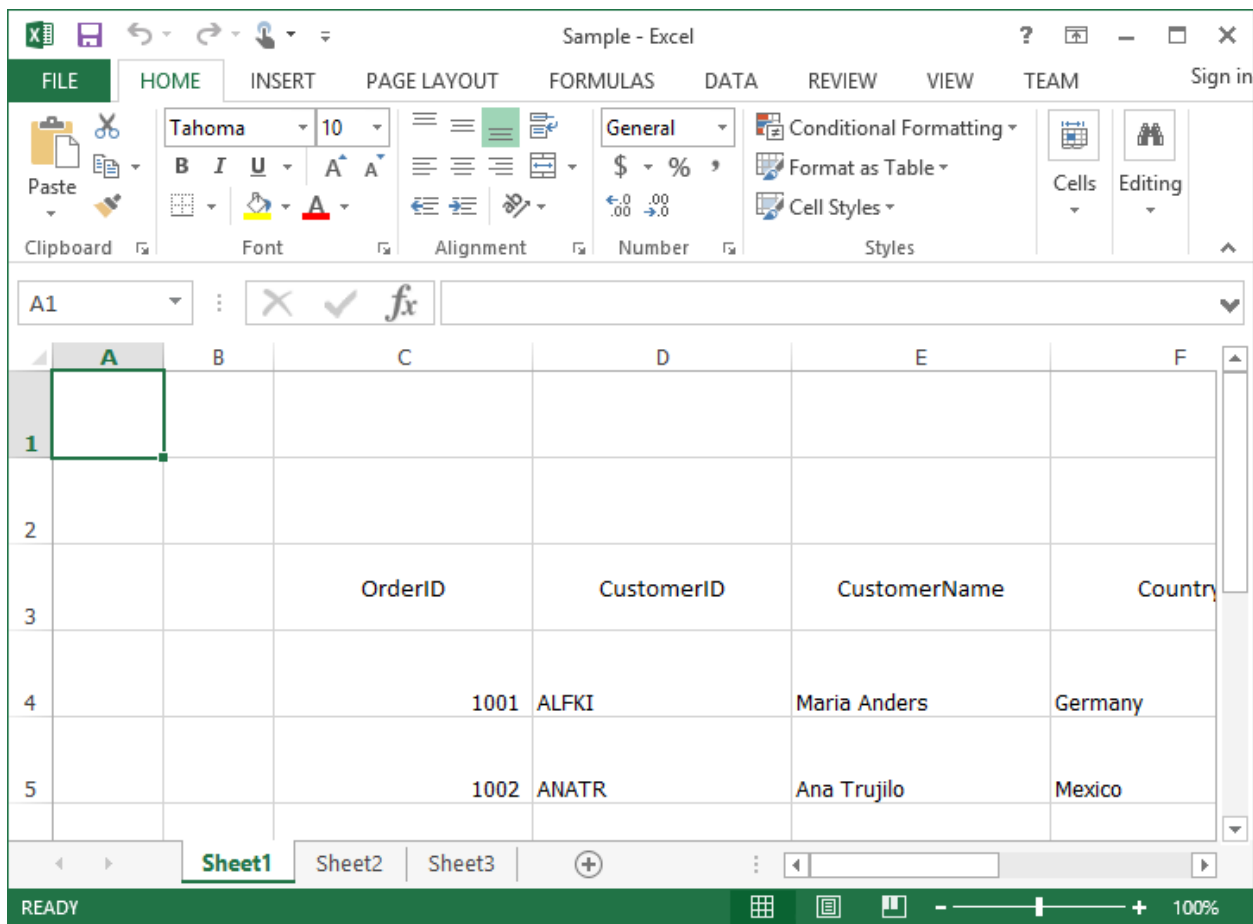
```
var options = new ExcelExportingOptions();
options.ExportUnBoundRows = true;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

### Changing start row and column index while exporting

You can export the data to specified row index and column index in worksheet, by setting [StartRowIndex](#) and [StartColumnIndex](#) properties.

#### C#

```
var options = new ExcelExportingOptions();
options.StartColumnIndex = 3;
options.StartRowIndex = 3;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```



### Saving options

#### Save directly to file

After exporting to excel, you can save exported workbook directly to file system by using [SaveAs](#) method.

#### C#

```
var options = new ExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

#### Save as stream

After exporting to excel, you can save exported workbook to stream by using [SaveAs](#) method.

#### C#

```
var options = new ExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
FileStream fileStream = new FileStream("Output.xlsx", FileMode.Create);
workBook.SaveAs(fileStream);
```

You can refer [XlsIO documentation](#).

#### *Save using File dialog*

After exporting to excel, you can save exported workbook by opening [FileDialog](#).

#### **C#**

```
var options = new ExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
SaveFileDialog sfd = new SaveFileDialog
{
    FilterIndex = 2,
    Filter = "Excel 97 to 2003 Files (*.xls)|*.xls|Excel 2007 to 2010 Files (*.xlsx)|*.xlsx|Excel 2013 File (*.xlsx)|*.xlsx"
};
if (sfd.ShowDialog() == true)
{
    using (Stream stream = sfd.OpenFile())
    {
        if (sfd.FilterIndex == 1)
            workBook.Version = ExcelVersion.Excel97to2003;
        else if (sfd.FilterIndex == 2)
            workBook.Version = ExcelVersion.Excel2010;
        else
            workBook.Version = ExcelVersion.Excel2013;
        workBook.SaveAs(stream);
    }
    //Message box confirmation to view the created workbook.
    if (MessageBox.Show("Do you want to view the workbook?", "Workbook has been created",
        MessageBoxButton.YesNo, MessageBoxImage.Information) ==
        MessageBoxResult.Yes)
    {
        //Launching the Excel file using the default Application.[MS Excel Or Free ExcelViewer]
        System.Diagnostics.Process.Start(sfd.FileName);
    }
}
```

#### *Opening exported excel without saving*

You can open the exported workbook without saving by using [SfSpreadsheet](#) control.

#### **C#**

```
var options = new ExcelExportingOptions();
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
Window window1 = new Window();
SfSpreadsheet spreadsheet = new SfSpreadsheet();
spreadsheet.Open(workBook);
window1.Content = spreadsheet;
window1.Show();
```

	A	B	C	D	E
	OrderID	CustomerID	CustomerName	Country	ShipCity
1					
2	1001	ALFKI	Maria Anders	Germany	Berlin
3	1002	ANATR	Ana Trujilo	Mexico	México D.F.
4	1003	ANTON	Antonio Moreno	Mexico	México D.F.
5	1004	AROUT	Thomas Hardy	UK	London
6	1005	BERGS	Christina Berglund	Sweden	Luleå
7	1006	BLAUS	Hanna Moos	Germany	Mannheim
8	1007	BLOND	Frédérique Citeaux	France	Strasbourg

### Export DataGrid pages to Excel

While exporting data to excel, if paging is used, current page only will be exported, by default. If you want to export all pages, you need to set [ExportAllPages](#) property as `true`.

#### C#

```
var options = new ExcelExportingOptions();
options.ExportAllPages = true;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

By default, all data will be exported to single sheet. If you want to export each page to different sheets, you need to use [ExportPageOptions](#) property.

#### C#

```
var options = new ExcelExportingOptions();
options.ExportAllPages = true;
options.ExportPageOptions = ExportPageOptions.ExportToDifferentSheets;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
```

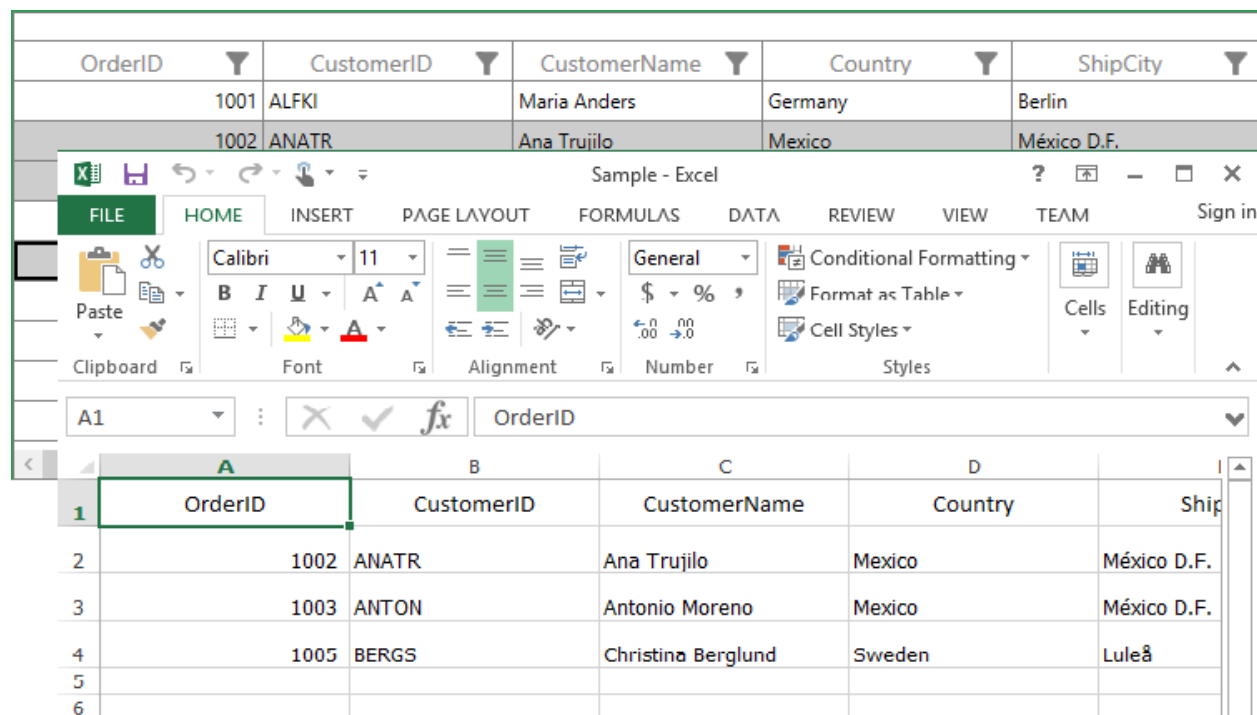
### Export DataGrid SelectedItems to Excel

By default, entire grid will be exported to Excel. You can export selected items only by passing `SelectedItems` to

[ExportToExcel](#) method.

#### C#

```
var options = new ExcelExportingOptions();
ExcelEngine excelEngine = new ExcelEngine();
IWorkbook workBook = excelEngine.Excel.Workbooks.Create();
workBook.Worksheets.Create();
dataGrid.ExportToExcel(dataGrid.SelectedItems, options,
workBook.Worksheets[0]);
workBook.Version = ExcelVersion.Excel2013;
workBook.SaveAs("Sample.xlsx");
```



OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.

### Export DataGrid to HTML

You can save exported workbook as HTML by using [SaveAsHtml](#) method.

#### C#

```
var options = new ExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAsHtml("Sample.html", HtmlSaveOptions.Default);
```

It is also possible to save worksheet as HTML by using [SaveAsHtml](#) method. You can refer [XlsIO documentation](#) for this.



### Export DataGrid to Mail

You can export SfDataGrid to mail by converting it into Excel and save exported worksheet as HTML. Then exported HTML contents is embedded in mail body.

#### C#

```
var options = new ExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2010;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workbook = excelEngine.Excel.Workbooks[0];
workbook.Worksheets[0].UsedRange.BorderInside(ExcelLineStyle.Thick,
ExcelKnownColors.Black);
workbook.Worksheets[0].UsedRange.BorderAround(ExcelLineStyle.Thick,
ExcelKnownColors.Black);
workbook.Worksheets[0].SaveAsHtml("test.htm",
Syncfusion.XlsIO.Implementation.HtmlSaveOptions.Default);
System.Net.Mail.MailMessage myMessage = new System.Net.Mail.MailMessage();
myMessage.To.Add("Support@syncfusion.com");
myMessage.From = new MailAddress("Support@syncfusion.com");
myMessage.Priority = MailPriority.High;
myMessage.Subject = "Order Details";
myMessage.IsBodyHtml = true;
myMessage.Body = new StreamReader("test.htm").ReadToEnd();
SmtpClient client = new SmtpClient("smtp.office365.com", 587);
client.EnableSsl = true;
client.UseDefaultCredentials = false;
client.Credentials = new NetworkCredential("Support@syncfusion.com",
"test");
int count = 0;
while (count < 3)
{
    try
    {
        client.Send(myMessage);
        count = 3;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine(String.Format("Sending Mail Attemp - {0}",
count.ToString()));
        Thread.Sleep(60000);
        count++;
    }
}
Console.WriteLine("Mail has been sent...");
```

### Export DataGrid to XML

You can save exported workbook as Xml file also by using [SaveAsXml](#) methods.

#### C#

```
var options = new ExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workbook = excelEngine.Excel.Workbooks[0];
```

```
workBook.SaveAsXml("Sample.xml", ExcelXmlSaveType.MSExcel);
```

### Export DataGrid to CSV

You can save exported workbook as CSV by using `SaveAs` method.

#### C#

```
var options = new ExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.csv", ",");
```

Similarly, you can save exported worksheet also to CSV. You can refer [XlsIO documentation](#).

### Row Height and Column Width customization

After exporting data to excel, you can set different row height and column width for the columns based on your requirement. You can refer [here](#) for more information.

#### C#

```
var options = new ExcelExportingOptions();
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.Worksheets[0].SetRowHeight(2, 50);
workBook.Worksheets[0].SetColumnWidth(2, 50);
workBook.SaveAs("Sample.xlsx");
```

### Styling cells based on CellType in Excel

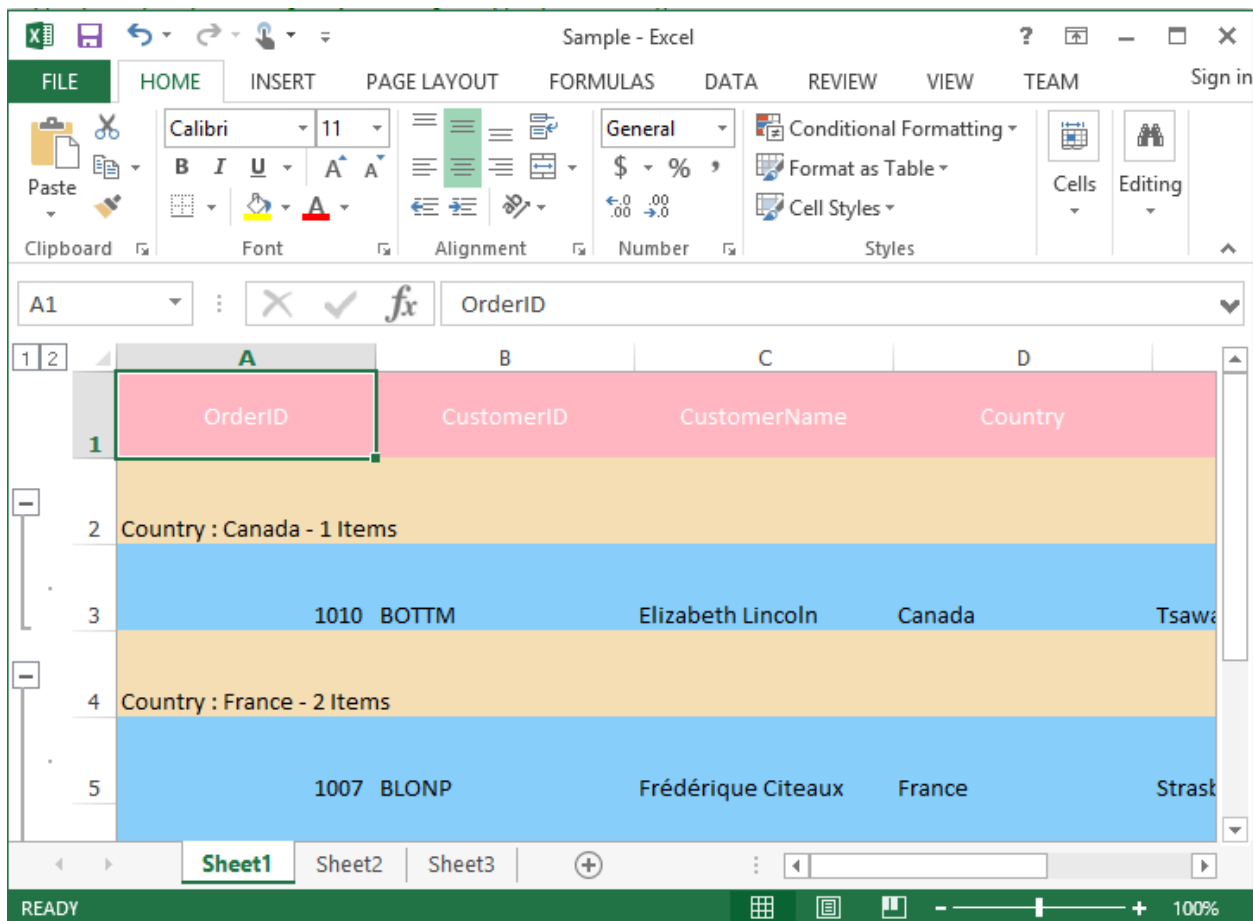
You can customize the cell styles based on `CellType` by using [ExportingEventHandler](#).

#### C#

```
var options = new ExcelExportingOptions();
options.ExportingEventHandler = ExportingHandler;
options.AllowOutlining = true;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
private static void ExportingHandler(object sender,
GridExcelExportingEventArgs e)
{
    if (e.CellType == ExportCellType.HeaderCell)
    {
        e.CellStyle.BackGroundBrush = new SolidColorBrush(Colors.LightPink);
        e.CellStyle.ForeGroundBrush = new SolidColorBrush(Colors.White);
        e.Handled = true;
    }
    else if (e.CellType == ExportCellType.RecordCell)
    {
        e.CellStyle.BackGroundBrush = new SolidColorBrush(Colors.LightSkyBlue);
        e.Handled = true;
    }
    else if (e.CellType == ExportCellType.GroupCaptionCell)
    {

```

```
e.CellStyle.BackGroundBrush = new SolidColorBrush(Colors.Wheat);
e.Handled = true;
}
```



Cell customization in Excel while exporting

You can customize the cells by setting [CellsExportingEventHandler](#) in [ExcelExportingOptions](#).

*Customize cell value while exporting*

You can customize the cell values while exporting to excel by using [CellsExportingEventHandler](#) and [ExcelExportingOptions](#).

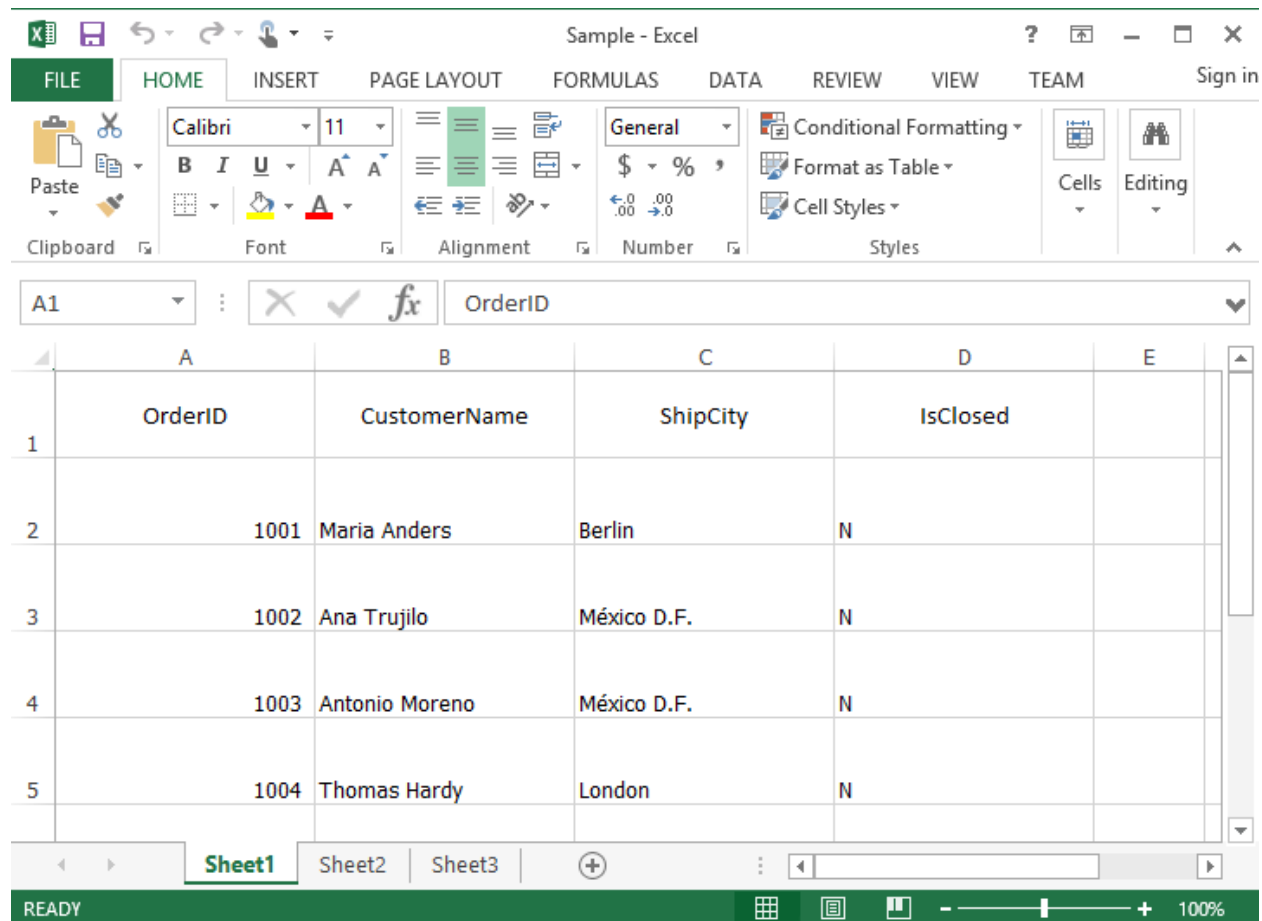
## C#

```
var options = new ExcelExportingOptions();
options.CellsExportingEventHandler = CellExportingHandler;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
private static void CellExportingHandler(object sender,
GridCellExcelExportingEventArgs e)
{
    // Based on the column mapping name and the cell type, we can change the
    cell
    //values while exporting to excel.
```

```

if (e.CellType == ExportCellType.RecordCell && e.ColumnName == "IsClosed")
{
    //if the cell value is True, "Y" will be displayed else "N" will be
    //displayed.
    if (e.CellValue.Equals(true))
    e.Range.Cells[0].Value = "Y";
    else
    e.Range.Cells[0].Value = "N";
    e.Handled = true;
}
}

```



Here, cell values are changed for **IsClosed** column based on custom condition.

*Changing row style in excel based on data*

You can customize the rows based on the record values by using [CellsExportingEventHandler](#).

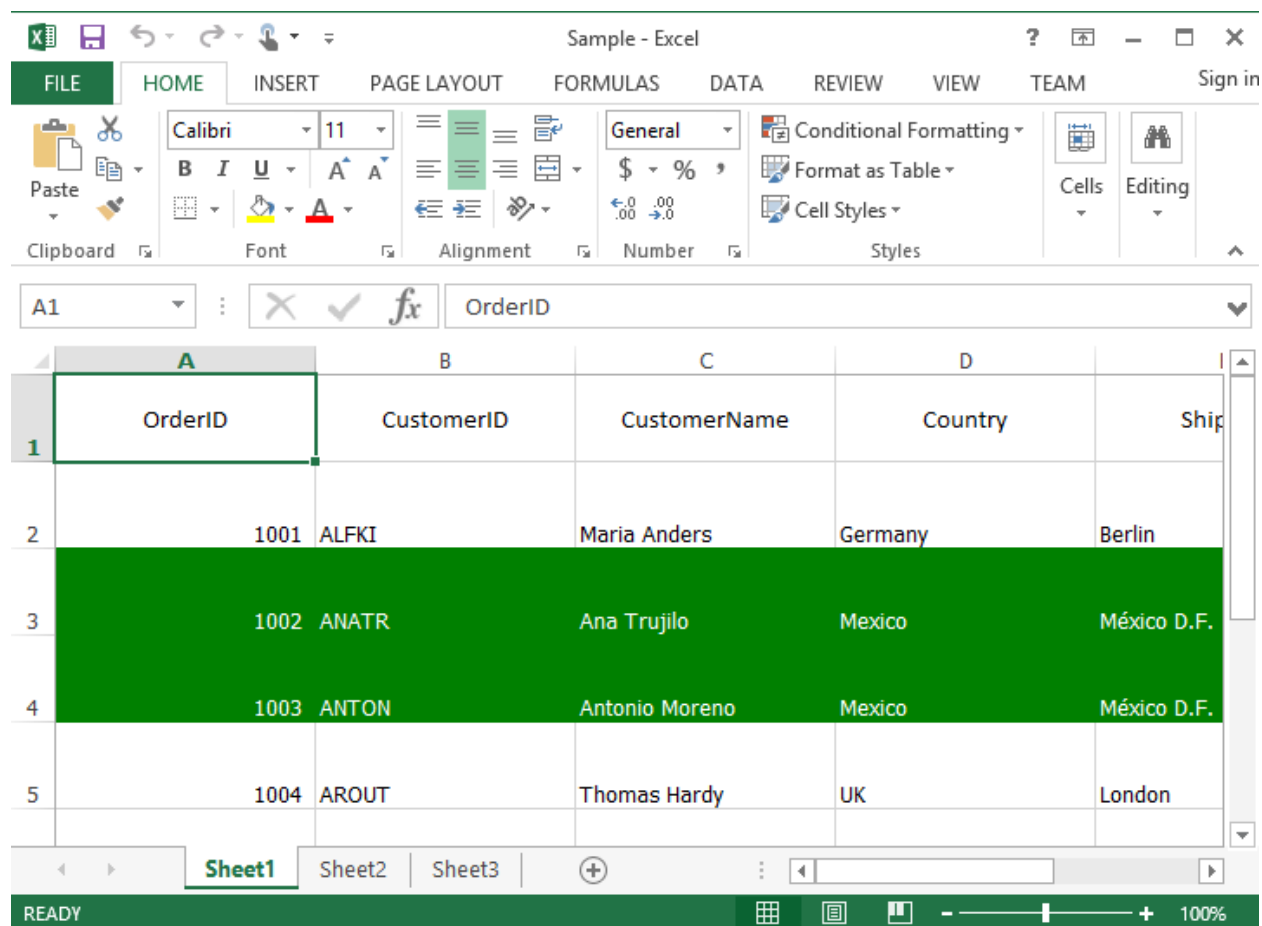
### C#

```

var options = new ExcelExportingOptions();
options.CellsExportingEventHandler = CellExportingHandler;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");

```

```
private static void CellExportingHandler(object sender,
GridCellExcelExportingEventArgs e)
{
    if (!(e.NodeEntry is RecordEntry))
        return;
    var record = e.NodeEntry as RecordEntry;
    if ((record.Data as OrderInfo).Country == "Mexico")
    {
        e.Range.CellStyle.ColorIndex = ExcelKnownColors.Green;
        e.Range.CellStyle.Font.Color = ExcelKnownColors.White;
    }
}
```



Here, records having the **Country** name as **Mexico** are customized.

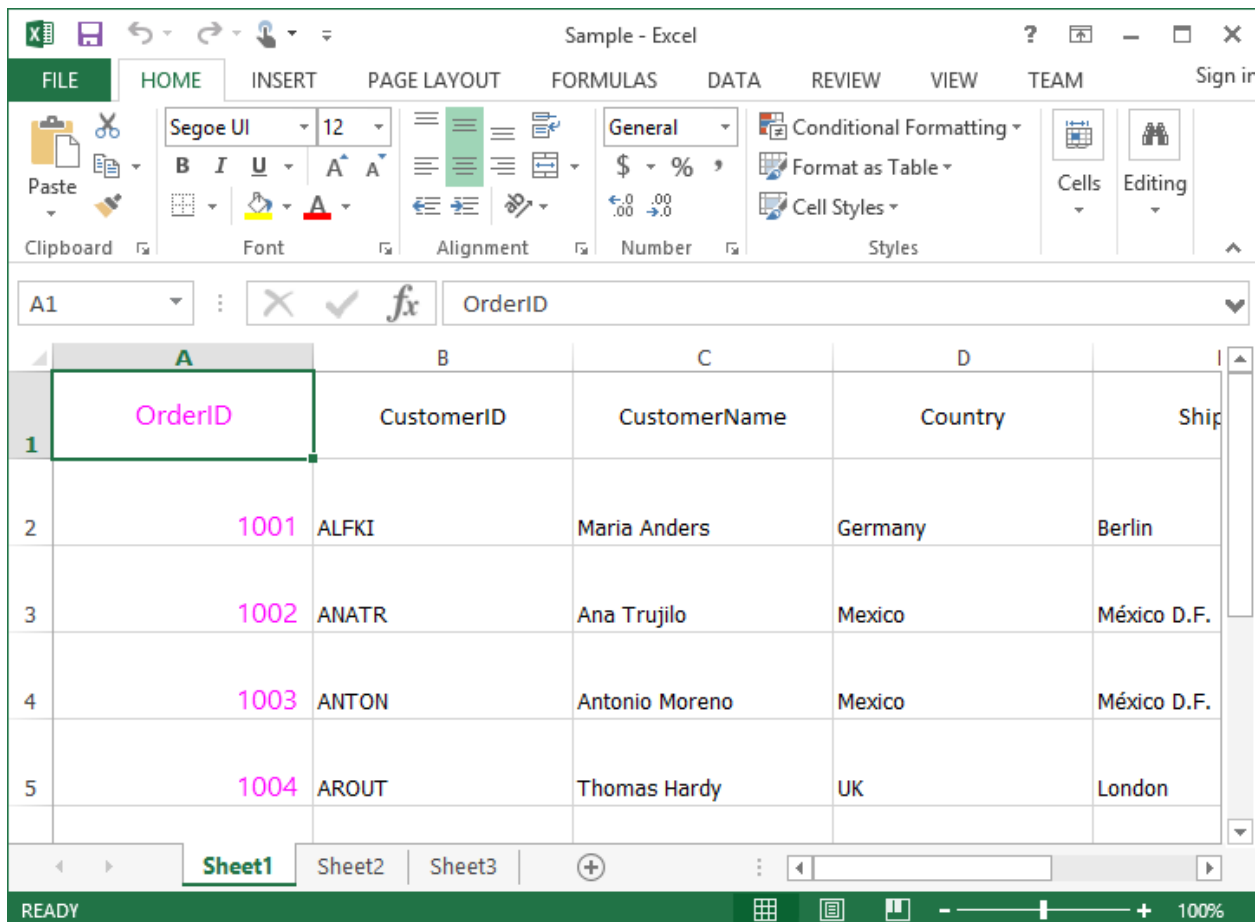
#### *Customize the cells based on Column Name*

You can customize the cells based on [GridCellExcelExportingEventArgs.ColumnName](#) property in the [CellsExportingEventHandler](#).

#### **C#**

```
var options = new ExcelExportingOptions();
options.CellsExportingEventHandler = CellExportingHandler;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
```

```
workBook.SaveAs("Sample.xlsx");
private static void CellExportingHandler(object sender,
GridCellExcelExportingEventArgs e)
{
    if (e.ColumnName != "OrderID")
        return;
    e.Range.CellStyle.Font.Size = 12;
    e.Range.CellStyle.Font.Color = ExcelKnownColors.Pink;
    e.Range.CellStyle.Font.FontName = "Segoe UI";
}
```



Here, **OrderID** column cells are customized while exporting.

Customize exported workbook and worksheet

SfDataGrid exports to excel by using [XlsIO](#). You can refer [XlsIO documentation](#) for manipulating workbook and sheet after exporting.

*Workbook*

SfDataGrid provides option to return [ExcelEngine](#) from that you can get exported workbook. This allows you to protect, encrypt and add worksheet before saving.

**C#**

```
var options = new ExcelExportingOptions();
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
```

```
var workBook = excelEngine.Excel.Workbooks[0];  
workBook.SaveAs("Sample.xlsx");
```

### Worksheet customization

SfDataGrid provides support to export to already existing file or worksheet.

In the below code snippet, worksheet is created and passed to `ExportToExcel` method. In the same way, you can open already existing excel also using `XlsIO`.

### C#

```
var options = new ExcelExportingOptions();  
ExcelEngine excelEngine = new ExcelEngine();  
IWorkbook workBook = excelEngine.Excel.Workbooks.Create();  
dataGrid.ExportToExcel(dataGrid.View, options, workBook.Worksheets[0]);  
workBook.Version = ExcelVersion.Excel2013;  
workBook.SaveAs("Sample.xlsx");
```

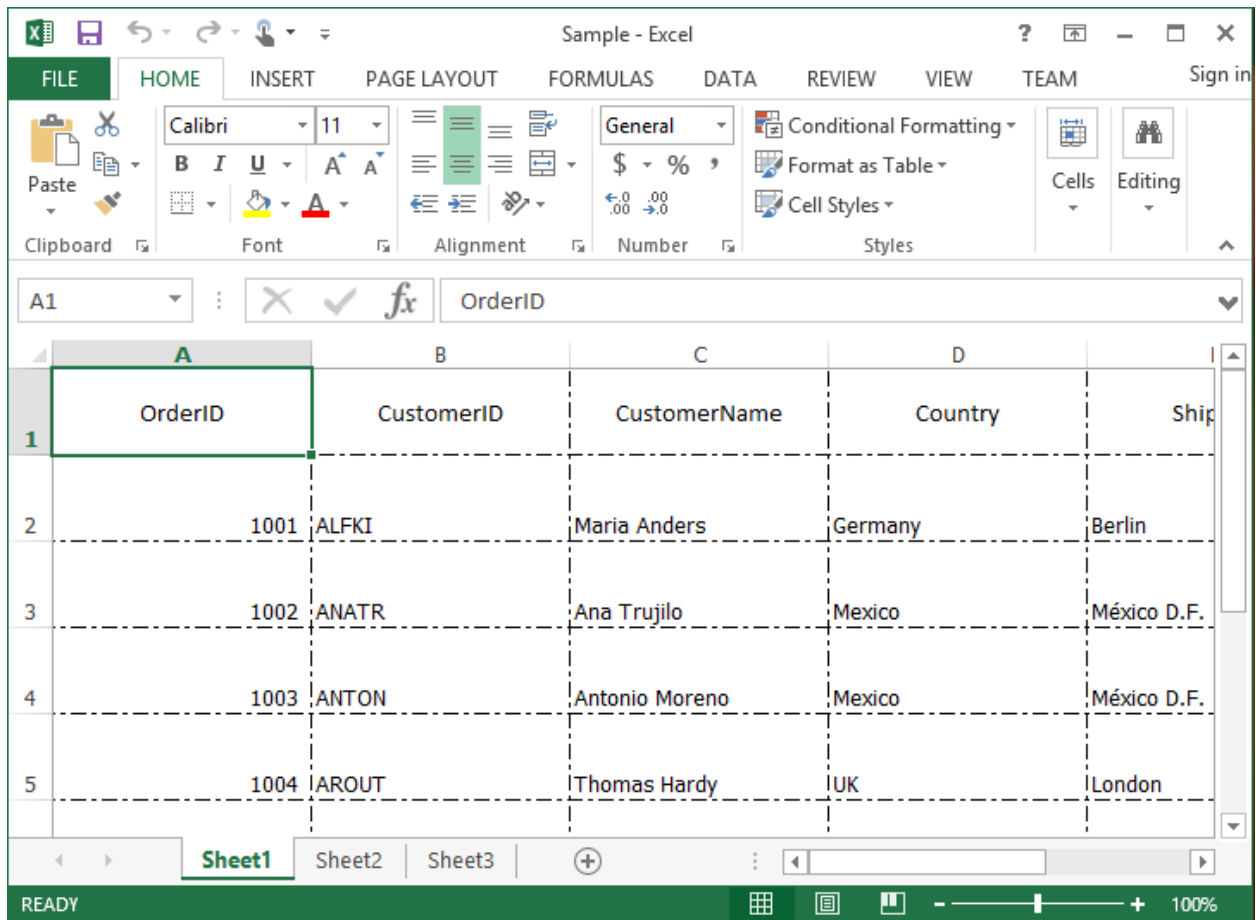
Before saving workbook, you need to set the specific excel version by using [IWorkbook.Version](#) property. Here, you can directly manipulate the data in the worksheet. You can refer [here](#) for more information.

### Setting borders

You can set borders to excel cells by directly accessing worksheet after exporting data.

### C#

```
var options = new ExcelExportingOptions();  
options.ExcelVersion = ExcelVersion.Excel2013;  
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);  
var workBook = excelEngine.Excel.Workbooks[0];  
workBook.Worksheets[0].UsedRange.BorderInside(ExcelLineStyle.Dash_dot,  
ExcelKnownColors.Black);  
workBook.Worksheets[0].UsedRange.BorderAround(ExcelLineStyle.Dash_dot,  
ExcelKnownColors.Black);  
workBook.SaveAs("Sample.xlsx");
```



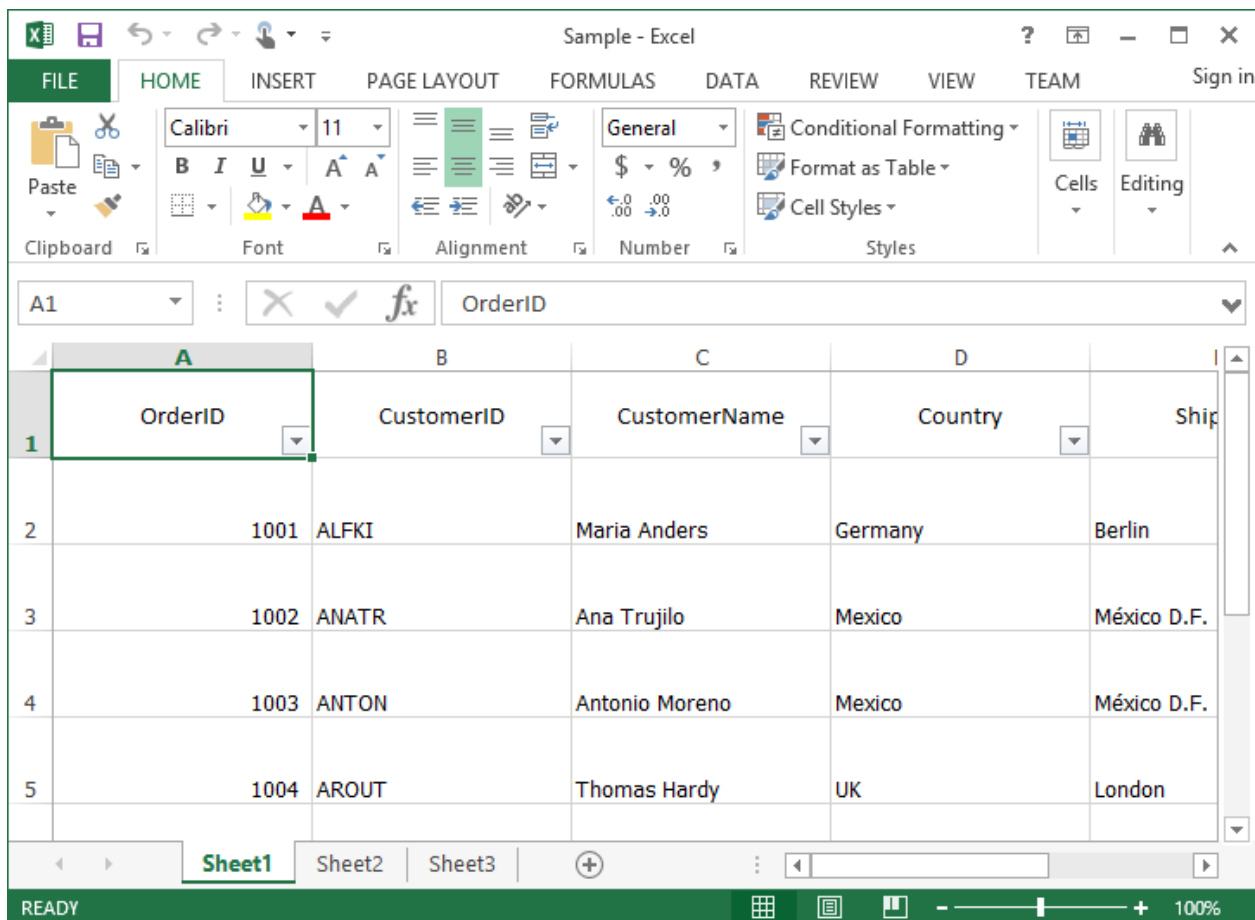
### Enabling Filters

You can show filters in exported worksheet by enabling filter for the exported range in the worksheet.

### C#

```
var options = new ExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.Worksheets[0].AutoFilters.FilterRange =
workBook.Worksheets[0].UsedRange;
workBook.SaveAs("Sample.xlsx");
```





While using **stacked headers**, you can specify the **range** based on Stacked headers count.

#### C#

```
var options = new ExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
options.ExportStackedHeaders = true;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workbook = excelEngine.Excel.Workbooks[0];
var range = "A" + (dataGrid.StackedHeaderRows.Count + 1).ToString() + ":" +
workbook.Worksheets[0].UsedRange.End.AddressLocal;
excelEngine.Excel.Workbooks[0].Worksheets[0].AutoFilters.FilterRange =
workbook.Worksheets[0].Range[range];
workbook.SaveAs("Sample.xlsx");
```

You can refer [XlsIO documentation](#).

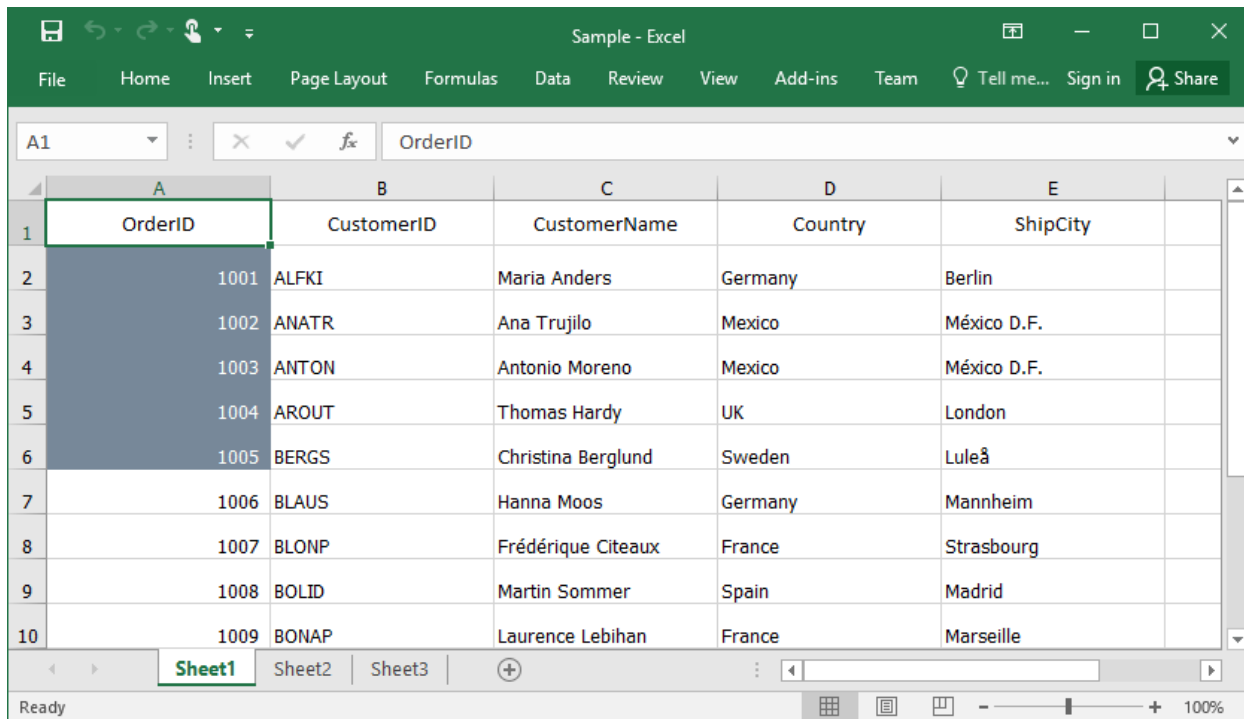
[Customize the range of cells](#)

You can customize the range of cells after exporting to excel by directly manipulating worksheet.

#### C#

```
var options = new ExcelExportingOptions();
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workbook = excelEngine.Excel.Workbooks[0];
```

```
workBook.Worksheets[0].Range["A2:A6"].CellStyle.Color =
System.Drawing.Color.LightSlateGray;
workBook.Worksheets[0].Range["A2:A6"].CellStyle.Font.Color =
ExcelKnownColors.White;
workBook.SaveAs("Sample.xlsx");
```



	A	B	C	D	E
	OrderID	CustomerID	CustomerName	Country	ShipCity
1					
2	1001	ALFKI	Maria Anders	Germany	Berlin
3	1002	ANATR	Ana Trujillo	Mexico	México D.F.
4	1003	ANTON	Antonio Moreno	Mexico	México D.F.
5	1004	AROUT	Thomas Hardy	UK	London
6	1005	BERGS	Christina Berglund	Sweden	Luleå
7	1006	BLAUS	Hanna Moos	Germany	Mannheim
8	1007	BLONP	Frédérique Citeaux	France	Strasbourg
9	1008	BOLID	Martin Sommer	Spain	Madrid
10	1009	BONAP	Laurence Leblan	France	Marseille

### Exporting DetailsView

By default, [DetailsViewDataGrid](#) will be exported to Excel. You can customize its exporting operation by using [ChildExportingEventHandler](#).

### Excluding DetailsViewDataGrid while exporting

You can exclude particular [DetailsViewDataGrid](#) while exporting, by using the [ChildExportingEventHandler](#) and [GridChildExportingEventArgs.Cancel](#).

### C#

```
var options = new ExcelExportingOptions();
options.ChildExportingEventHandler = ChildExportingHandler;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
private static void ChildExportingHandler(object sender,
GridChildExportingEventArgs e)
{
var recordEntry = e.NodeEntry as RecordEntry;
if ((recordEntry.Data as OrderInfo).OrderID == 1002)
e.Cancel = true;
}
```

The screenshot displays the Microsoft Excel application window. The title bar reads "Sample - Excel". The ribbon is set to the "FORMULAS" tab, showing options like "Conditional Formatting", "Format as Table", and "Cell Styles". The active cell is A1, which contains the text "OrderID". Below the ribbon, a data table is visible with columns A through D. The table contains data for orders, including OrderID, CustomerID, CustomerName, and Country. The first row of data (row 2) shows OrderID 1001, CustomerID ALFKI, CustomerName Maria Anders, and Country Germany. The table continues with rows 7, 8, 9, 10, 11, 12, and 13, each containing a unique OrderID and corresponding Customer information.

OrderID	CustomerID	CustomerName	Country
1001	ALFKI	Maria Anders	Germany
1002	ANATR	Ana Trujillo	Mexico
1003	ANTON	Antonio Moreno	Mexico
1003	Bike2		
1003	Bike1		
1004	AROUT	Thomas Hardy	UK

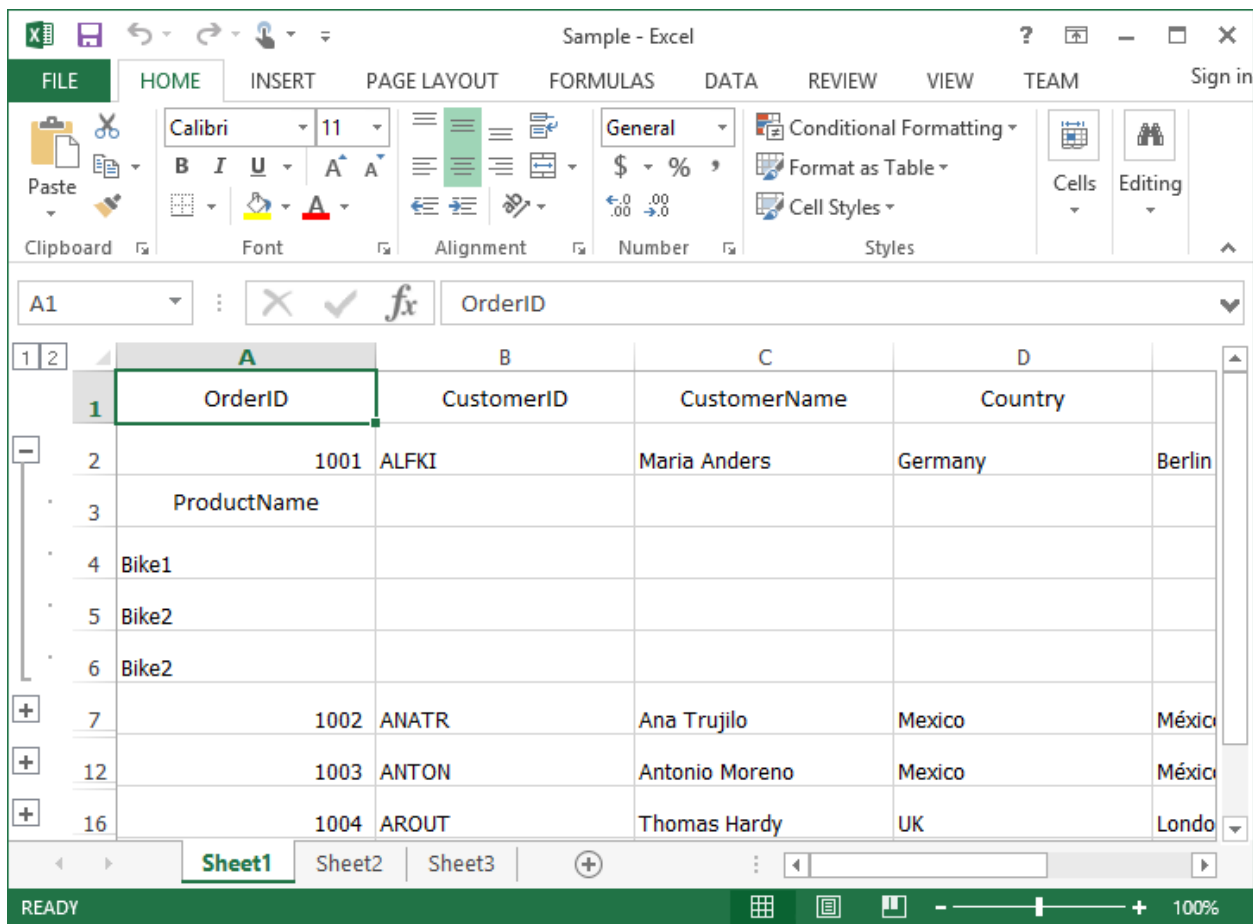
Here, `DetailsViewDataGrid` is not exported for the parent record having `OrderID` as 1002.

### Excluding DataGridView columns from exporting

You can exclude [DetailsViewDataGrid](#) columns while exporting, by using [ChildExportingEventHandler](#) and [GridChildExportingEventArgs.ExcludeColumns](#).

## C#

```
var options = new ExcelExportingOptions();
options.ChildExportingEventHandler = ChildExportingHandler;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
private static void ChildExportingHandler(object sender,
GridChildExportingEventArgs e)
{
e.ExcludeColumns.Add("OrderID");
}
```



Here, **OrderID** column is displayed in **DetailsViewDataGrid** and it is excluded while exporting to excel.

#### *Customizing DetailsViewDataGrid cells*

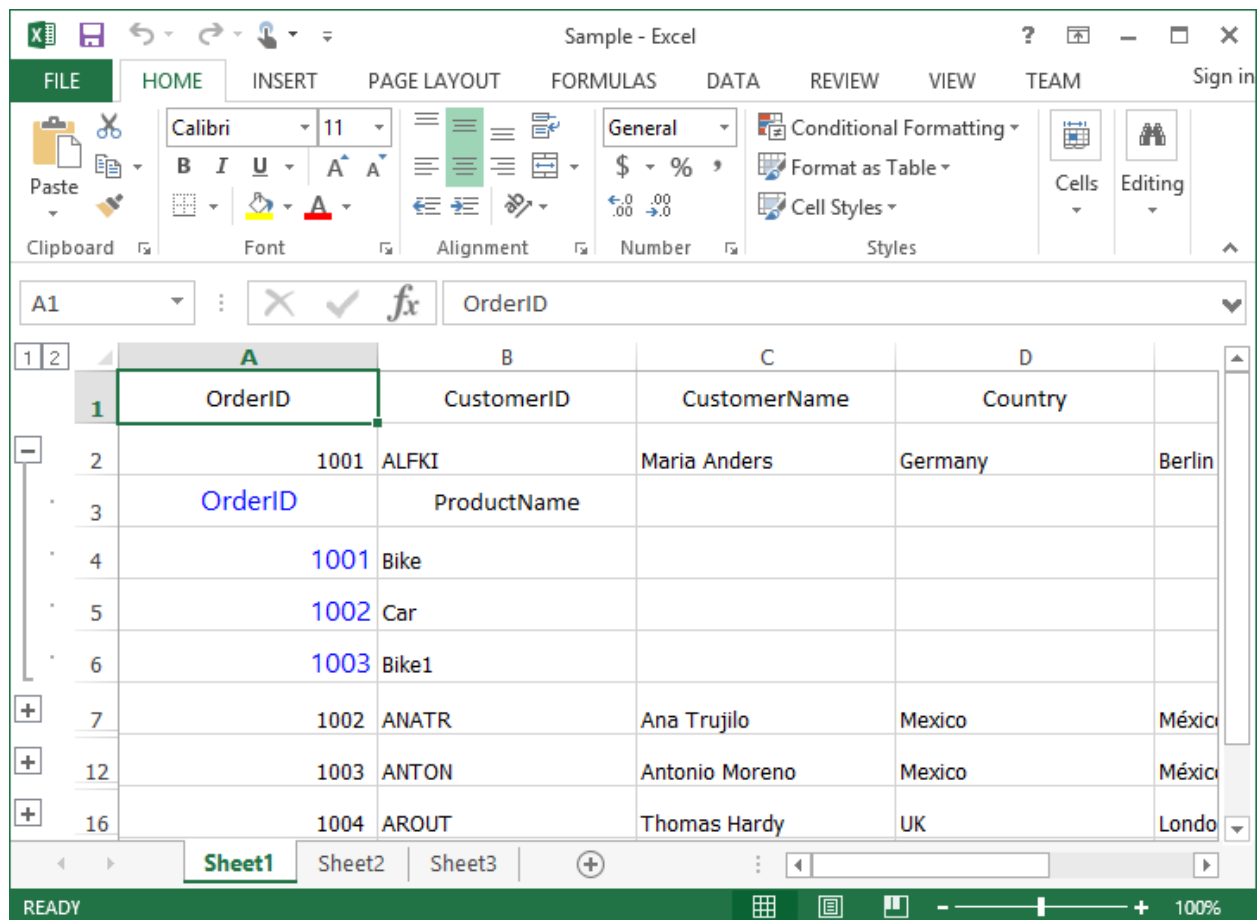
Like parent DataGrid, You can customize the **DetailsViewDataGrid** cells also by using [CellsExportingEventHandler](#). Based on [GridViewDefinition](#) property, you can identify the particular **DetailsViewDataGrid** and customize it.

#### **C#**

```
var options = new ExcelExportingOptions();
options.CellsExportingEventHandler = CellExportingHandler;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
var workBook = excelEngine.Excel.Workbooks[0];
workBook.SaveAs("Sample.xlsx");
private static void CellExportingHandler(object sender,
GridCellExcelExportingEventArgs e)
{
    if (e.GridViewDefinition == null || e.GridViewDefinition.RelationalColumn !=
"ProductDetails")
        return;
    if (e.ColumnName == "OrderID")
    {
        e.Range.CellStyle.Font.Size = 12;
        e.Range.CellStyle.Font.Color = ExcelKnownColors.Blue;
        e.Range.CellStyle.Font.FontName = "Segoe UI";
    }
}
```

```
}

```



### Performance

Using [ExcelExportingOptions.CellsExportingEventHandler](#) and changing settings for each cell will consume more memory and time consumption. So, avoid using `CellsExportingEventHandler` and instead of you can do the required settings in the exported sheet.

### Formatting column without using CellsExportingEventHandler

You can perform cell level customization such as row-level styling, formatting particular column in the exported worksheet.

In the below code snippet, NumberFormat for Unit Price column is changed in the exported sheet after exporting without using `CellsExportingEventHandler`.

Reference:

<http://help.syncfusion.com/file-formats/xlsio/working-with-cell-or-range-formatting>

### C#

```
var options = new ExcelExportingOptions();
options.ExportMode = ExportMode.Value;
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
IWorkbook workBook = excelEngine.Excel.Workbooks[0];
```

```
workBook.ActiveSheet.Columns[4].NumberFormat = "0.0";
```

OrderID	Customer ID	Customer Name	Quantity	Unit Price
1001	ALFKI	Maria Anders	5.00	\$5.00
1002	ANATR	Ana Trujillo	10.00	\$20.00
1003	ANTON	Antonio Moreno	4.00	\$15.00
1004	AROUT	Thomas Hardy	8.00	\$33.00
1005	BERGS	Christina Berglund	15.00	\$50.00
1006	BLONP	Hanna Moos	11.00	\$30.00
1007	BOLID	Frédérique Citeaux		
1008	BONAP	Martin Sommer		
1009	BOTTM	Laurence Lebihan		
1010	BLAUS	Elizabeth Lincoln		

A	B	C	D	E
OrderID	Customer ID	Customer Name	Quantity	Unit Price
1001	ALFKI	Maria Anders	5.00	5.0
1002	ANATR	Ana Trujillo	10.00	20.0
1003	ANTON	Antonio Moreno	4.00	15.0
1004	AROUT	Thomas Hardy	8.00	33.0
1005	BERGS	Christina Berglund	15.00	50.0
1006	BLONP	Hanna Moos	11.00	30.0
1007	BOLID	Frédérique Citeaux	4.00	40.0
1008	BONAP	Martin Sommer	5.00	20.0
1009	BOTTM	Laurence Lebihan	13.00	24.0
1010	BLAUS	Elizabeth Lincoln	5.00	25.0

#### Alternate row styling without using CellsExportingEventHandler

In the below code snippet, the background color of rows in excel is changed based on row index using conditional formatting for better performance.

Reference:

<http://help.syncfusion.com/file-formats/xlsio/working-with-conditional-formatting>

#### C#

```
var options = new ExcelExportingOptions();
options.ExportMode = ExportMode.Value;
options.ExcelVersion = ExcelVersion.Excel2013;
var excelEngine = dataGrid.ExportToExcel(dataGrid.View, options);
IWorkbook workBook = excelEngine.Excel.Workbooks[0];
IConditionalFormats condition =
workBook.ActiveSheet.Range[2,1,this.dataGrid.View.Records.Count+1,this.dataGrid.Columns.Count].ConditionalFormats;
IConditionalFormat condition1 = condition.AddCondition();
condition1.FormatType = ExcelCfType.Formula;
condition1.FirstFormula = "MOD(ROW(),2)=0";
condition1.BackColorRGB = System.Drawing.Color.Pink;
IConditionalFormat condition2 = condition.AddCondition();
condition2.FormatType = ExcelCfType.Formula;
condition2.FirstFormula = "MOD(ROW(),2)=1";
condition2.BackColorRGB = System.Drawing.Color.LightGray;
```

OrderID	Customer ID	Customer Name	Quantity	Unit Price
1001	ALFKI	Maria Anders	5.00	\$5.00
1002	ANATR	Ana Trujillo		
1003	ANTON	Antonio Moreno		
1004	AROUT	Thomas Hardy		
1005	BERGS	Christina Berglund		
1006	BLONP	Hanna Moos		
1007	BOLID	Frédérique Citeaux		
1008	BONAP	Martin Sommer		
1009	BOTTM	Laurence Lebihan		
1010	BLAUS	Elizabeth Lincoln		

OrderID	Customer ID	Customer Name	Quantity	Unit Price
1	1001 ALFKI	Maria Anders	5.00	\$5.00
2	1002 ANATR	Ana Trujillo	10.00	\$20.00
3	1003 ANTON	Antonio Moreno	4.00	\$15.00
4	1004 AROUT	Thomas Hardy	8.00	\$33.00
5	1005 BERGS	Christina Berglund	15.00	\$50.00
6	1006 BLONP	Hanna Moos	11.00	\$30.00
7	1007 BOLID	Frédérique Citeaux	4.00	\$40.00
8	1008 BONAP	Martin Sommer	5.00	\$20.00
9	1009 BOTTM	Laurence Lebihan	13.00	\$24.00
10	1010 BLAUS	Elizabeth Lincoln	5.00	\$25.00

### Export To PDF in WPF DataGrid (SfDataGrid)

DataGrid provides support to export data to PDF file. It also provides support for grouping, filtering, sorting, paging, unbound rows, merged cells, stacked headers and details View while exporting.

The following assemblies needs to be added for exporting to PDF file.

- Syncfusion.SfGridConverter.WPF
- Syncfusion.Pdf.Base

For NuGet package, have to install [Syncfusion.DataGridExcelExport.WPF](#) package. For more details refer this [UG link](#).

You can export SfDataGrid to PDF by using the following extension methods present in the [Syncfusion.UI.Xaml.Grid.Converter](#) namespace.

- [ExportToPdf](#)
- [ExportToPdfGrid](#)

### C#

```
using Syncfusion.UI.Xaml.Grid.Converter;
var document = dataGrid.ExportToPdf();
document.Save("Sample.pdf");
```

**Note:** SfDataGrid exports data to PDF file by using [Essential PDF](#). You can refer [PDF documentation](#) for manipulating.

### PDF exporting options

Exporting operation can be customized by passing [PdfExportingOptions](#) instance as argument to [ExportToPdf](#) and [ExportToPdfGrid](#) method.

#### Auto size column widths in PDF

You can export SfDataGrid to PDF by fitting column widths based on its content by setting [AutoColumnWidth](#) property as `true`.

##### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.AutoColumnWidth = true;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

#### Auto size row heights in PDF

You can export SfDataGrid to PDF by fitting row heights based on its content by setting [AutoRowHeight](#) property as `true`.

##### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.AutoRowHeight = true;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

#### Exclude columns while exporting

By default, all the columns (including hidden columns) in SfDataGrid will be exported to PDF. If you want to exclude some columns while exporting to PDF, you can use [ExcludeColumns](#) property in [PdfExportingOptions](#).

##### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExcludeColumns.Add("CustomerName");
options.ExcludeColumns.Add("Country");
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

Here, the columns having `CustomerName` and `Country` as MappingName are excluded while exporting.

#### Export Format

By default, display text only will be exported to PDF. If you want to export the actual value, you need to set [ExportFormat](#) property as `false`.

##### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportFormat = false;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

#### Column header on each page

Column headers can be exported on each page by setting [RepeatHeaders](#) property.

##### C#



```
PdfExportingOptions options = new PdfExportingOptions();
options.RepeatHeaders = true;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

#### *Export all columns in one page*

While exporting to PDF, you can fit all columns on one page by setting [FitAllColumnsInOnePage](#) property as `true`.

#### **C#**

```
PdfExportingOptions options = new PdfExportingOptions();
options.FitAllColumnsInOnePage = true;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

#### *Export paging*

While exporting data to PDF, if paging is used, current page only will be exported, by default. If you want to export all pages, you need to set [ExportAllPages](#) property as `true`.

#### **C#**

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportAllPages = true;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

#### *Exclude groups while exporting*

By default, all the groups in dataGrid will be exported to PDF. If you want to export without Groups, you need to set [ExportGroups](#) property as `false`.

#### **C#**

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportGroups = false;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

#### *Exclude group Summaries while exporting*

By default, group summaries in dataGrid will be exported to PDF. If you want to export without group summaries, you need to set [ExportGroupSummary](#) property as `false`.

#### **C#**

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportGroupSummary = false;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

#### *Exclude table Summaries while exporting*

By default, table summaries in dataGrid will be exported to PDF. If you want to export without table summaries, you need to set [ExportTableSummary](#) property as `false`.

**C#**

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportTableSummary = false;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

*Exporting unbound rows*

You can export unbound rows to PDF by setting [ExportUnBoundRows](#) property as `true`.

**C#**

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportUnBoundRows = true;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

*Exporting stacked headers*

You can export stacked headers to PDF by setting [ExportStackedHeaders](#) property to `true`.

**C#**

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportStackedHeaders = true;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

*Exporting merged cells*

You can export merged cells to PDF by setting [ExportMergedCells](#) property as `true`.

**C#**

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportMergedCells = true;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

*Setting Header and Footer*

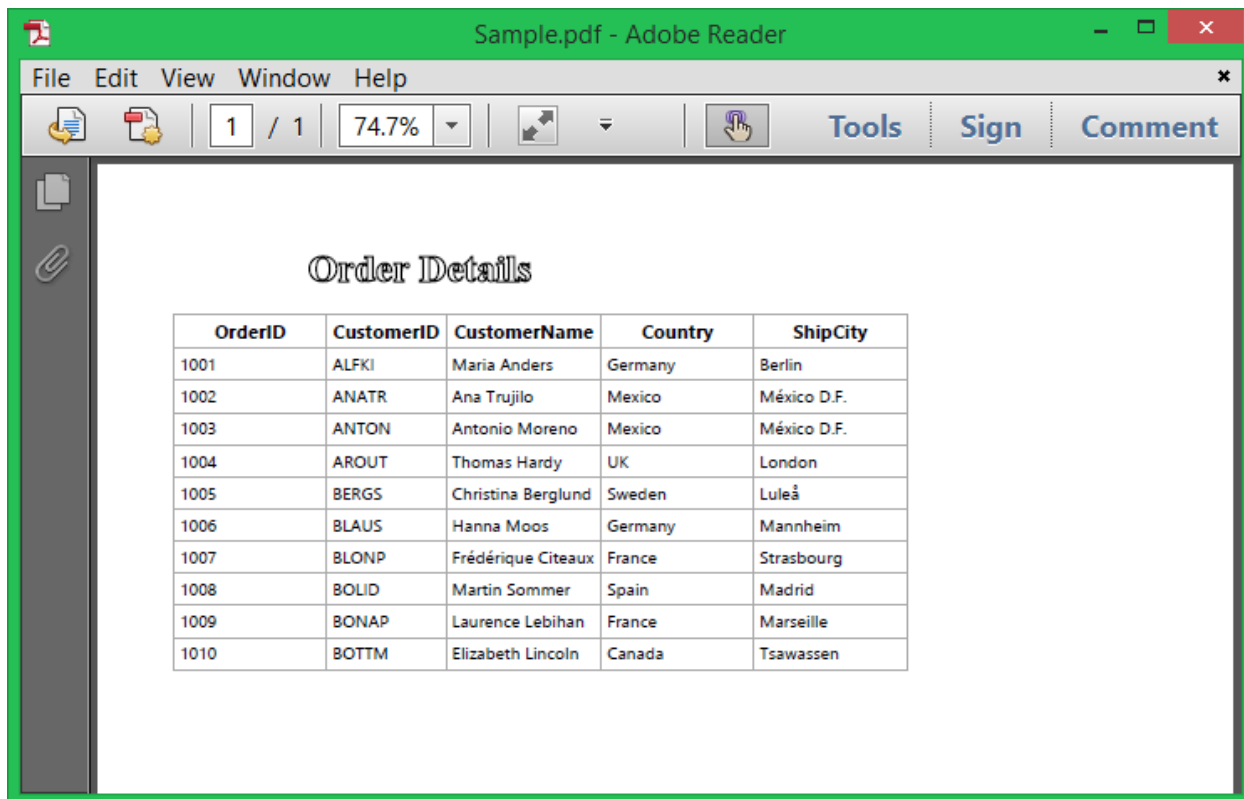
SfDataGrid provides a way to display additional content at the top (Header) or bottom (Footer) of the page while exporting to PDF. This can be achieved by setting [PageHeaderFooterEventHandler](#) in [PdfExportingOptions](#).

You can insert string, image or any drawing in header and footer in PdfHeaderFooterEventHandler. Setting [PdfPageTemplateElement](#) to [PdfHeaderFooterEventArgs.PdfDocumentTemplate.Top](#) loads the content at top of the page and setting the PdfPageTemplateElement to [PdfHeaderFooterEventArgs.PdfDocumentTemplate.Bottom](#) loads the content at bottom of the page.

**C#**

```
PdfExportingOptions options = new PdfExportingOptions();
options.PageHeaderFooterEventHandler = PdfHeaderFooterEventHandler;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

```
static void PdfHeaderFooterEventHandler(object sender,
PdfHeaderFooterEventArgs e)
{
    PdfFont font = new PdfStandardFont(PdfFontFamily.TimesRoman, 20f,
    PdfFontStyle.Bold);
    var width = e.PdfPage.GetClientSize().Width;
    PdfPageTemplateElement header = new PdfPageTemplateElement(width, 38);
    header.Graphics.DrawString("Order Details", font, PdfPens.Black, 70, 3);
    e.PdfDocumentTemplate.Top = header;
}
```



Here, `string` is inserted in the header of exported PDF file using [DrawString](#) method. Similarly, you can insert image, line, etc. using [DrawImage](#), [DrawLine](#) methods respectively.

### Change PDF page orientation

You can change the page orientation of PDF while exporting. The default page orientation is Portrait.

To change the page orientation, you need to get the exported [PdfGrid](#) by using [ExportToPdfGrid](#) method and then draw that [PdfGrid](#) into a [PdfDocument](#) by changing the [PageSettings.Orientation](#) property of [PdfDocument](#).

### C#

```
var options = new PdfExportingOptions();
var document = new PdfDocument();
document.PageSettings.Orientation = PdfPageOrientation.Landscape;
var page = document.Pages.Add();
var PDFGrid = dataGrid.ExportToPdfGrid(dataGrid.View, options);
var format = new PdfGridLayoutFormat()
```

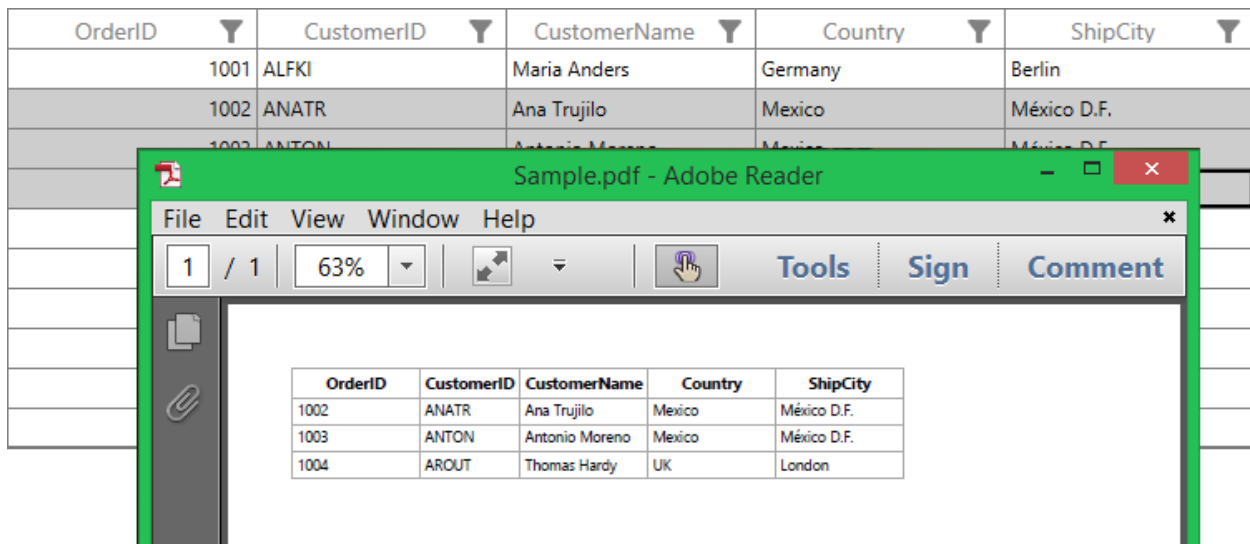
```
{
    Layout = PdfLayoutType.Paginate,
    Break = PdfLayoutBreakType.FitPage
};
PDFGrid.Draw(page, new PointF(), format);
document.Save("Sample.pdf");
```

### Export DataGrid SelectedItems to PDF

By default, entire grid will be exported to PDF. You can export selected items only by passing **SelectedItems** to [ExportToPdf](#) and [ExportToPdfGrid](#) methods.

#### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.AutoColumnWidth = true;
var document = dataGrid.ExportToPdf(dataGrid.SelectedItems, options);
document.Save("Sample.pdf");
```



### Saving options

#### Save directly to file

After exporting to PDF, you can save exported PDF file directly to file system by using [Save](#) method.

#### C#

```
var document = dataGrid.ExportToPdf();
document.Save("Sample.pdf");
```

You can refer [PDF documentation](#).

#### Save as stream

After exporting to PDF, you can save exported PDF file to stream by using [Save](#) method.

#### C#

```
FileStream fileStream = new FileStream("Sample.pdf", FileMode.Create);
var document = dataGrid.ExportToPdf();
```

```
document.Save(fileStream);  
fileStream.Close();
```

You can refer [PDF documentation](#).

*Save using File dialog*

After exporting to PDF, you can save exported PDF file by opening [FileDialog](#).

### C#

```
var document = dataGrid.ExportToPdf();  
SaveFileDialog sfd = new SaveFileDialog  
{  
    Filter = "PDF Files (*.pdf) | *.pdf"  
};  
if (sfd.ShowDialog() == true)  
{  
    using (Stream stream = sfd.OpenFile())  
    {  
        document.Save(stream);  
    }  
    //Message box confirmation to view the created Pdf file.  
    if (MessageBox.Show("Do you want to view the Pdf file?", "Pdf file has been  
created",  
        MessageBoxButton.YesNo, MessageBoxImage.Information) ==  
        MessageBoxResult.Yes)  
    {  
        //Launching the Pdf file using the default Application.  
        System.Diagnostics.Process.Start(sfd.FileName);  
    }  
}
```

Opening exported PDF without saving

You can view exported PDF document without saving by using [PDFViewerControl](#).

### C#

```
var document = this.dataGrid.ExportToPdf();  
MemoryStream stream = new MemoryStream();  
document.Save(stream);  
PdfViewerControl pdfViewer = new PdfViewerControl();  
pdfViewer.Load(stream);  
Window window = new Window();  
window.Content = pdfViewer;  
window.Show();
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujilo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

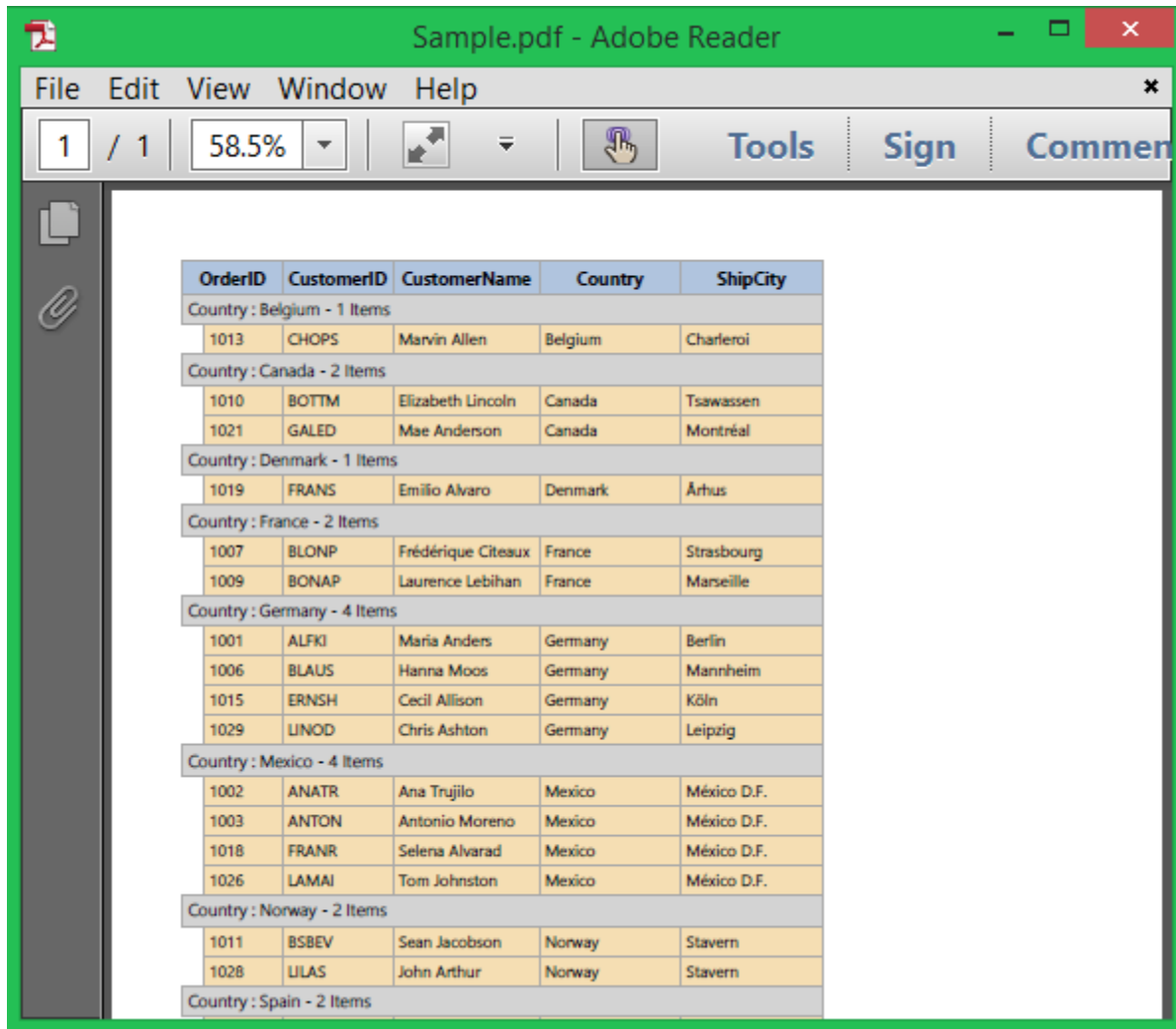
### Exporting Customization

*Styling cells based on CellType in PDF*

You can customize the cell styles based on **CellType** by using [ExportingEventHandler](#).

#### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportingEventHandler = GridPdfExportingEventHandler;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
void GridPdfExportingEventHandler(object sender, GridPdfExportingEventArgs e)
{
    if (e.CellType == ExportCellType.HeaderCell)
        e.CellStyle.BackgroundBrush = PdfBrushes.LightSteelBlue;
    else if (e.CellType == ExportCellType.GroupCaptionCell)
        e.CellStyle.BackgroundBrush = PdfBrushes.LightGray;
    else if (e.CellType == ExportCellType.RecordCell)
        e.CellStyle.BackgroundBrush = PdfBrushes.Wheat;
}
```



### Embedding fonts in PDF file

By default, some fonts (such as Unicode font) are not supported in PDF. In this case, it is possible to embed the font in PDF document with the help of [PdfTrueTypeFont](#).

### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportingEventHandler = GridPdfExportingEventHandler;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
void GridPdfExportingEventHandler(object sender, GridPdfExportingEventArgs e)
{
    if (e.CellType != ExportCellType.RecordCell)
        return;
    //creates a new font from the font file.
    var font = new PdfTrueTypeFont(@"..\..\Resources\segoeui.ttf", 9f,
        PdfFontStyle.Regular);
    e.CellStyle.Font = font;
}
```

Here, new font is created from font file and it is assigned to the **Font** of [PdfGridCell](#).

Cell customization in PDF while exporting

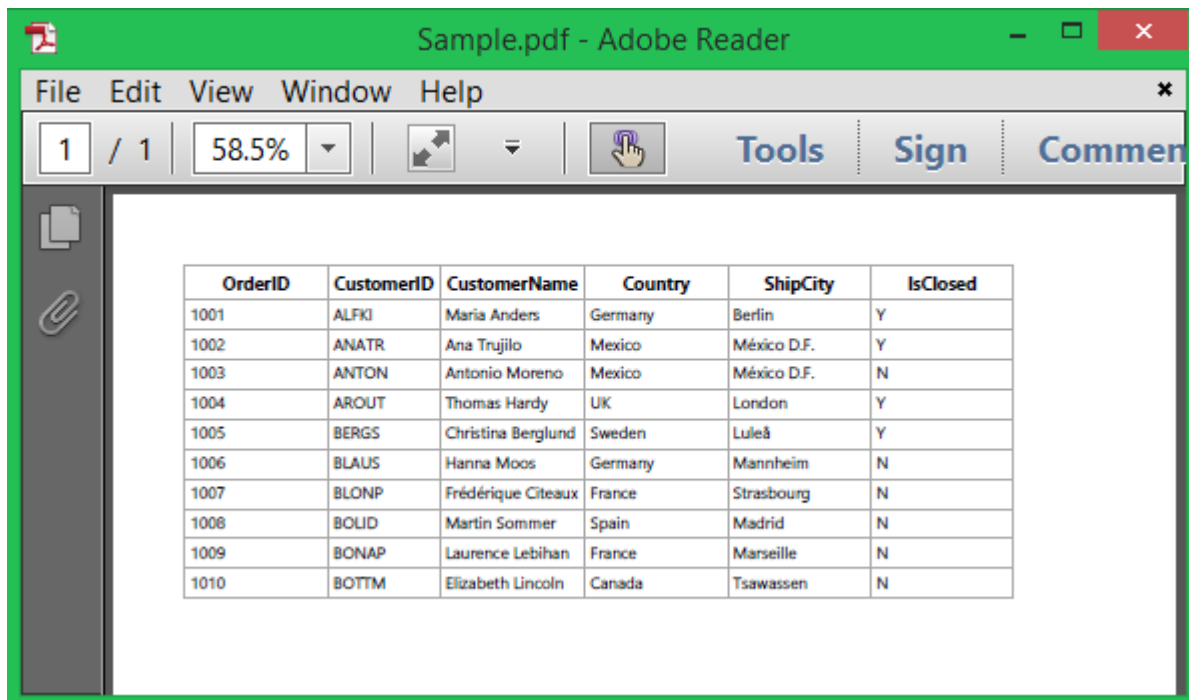
You can customize the cells in the PDF document by setting [CellsExportingEventHandler](#) in [PdfExportingOptions](#).

*Customize cell values while exporting*

You can customize the cell values while exporting to PDF by using [CellsExportingEventHandler](#) and [PdfExportingOptions](#).

## C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.CellsExportingEventHandler = CellsExportingEventHandler;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
private void CellsExportingEventHandler(object sender,
GridCellPdfExportingEventArgs e)
{
    // Based on the column mapping name and the cell type, you can change the
    // cell values while exporting to PDF.
    if (e.CellType == ExportCellType.RecordCell && e.ColumnName == "IsClosed")
    {
        //if the cell value is True, "Y" will be displayed else "N" will be
        //displayed.
        if (e.CellValue.Equals("True"))
            e.CellValue = "Y";
        else
            e.CellValue = "N";
    }
}
```





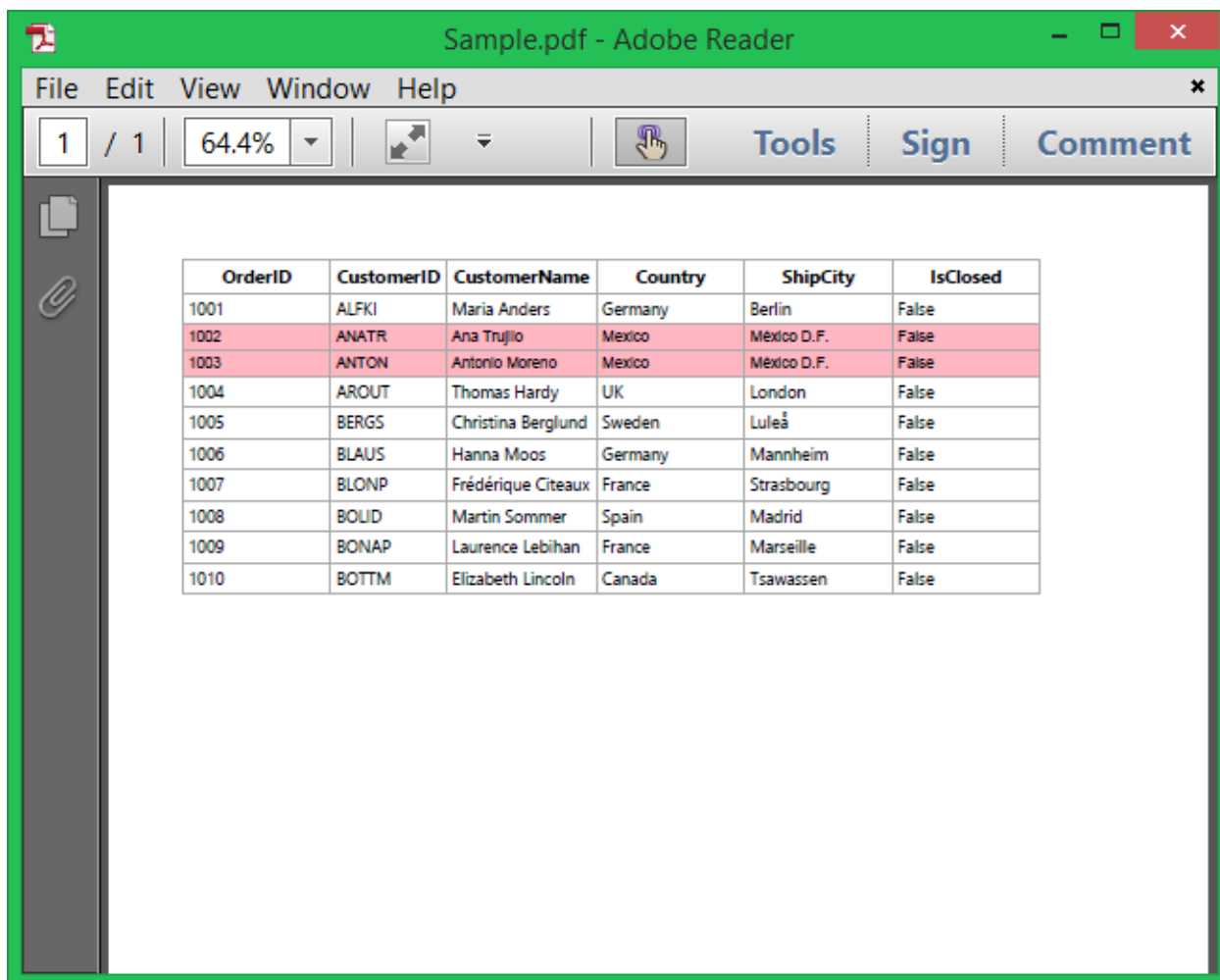
Here, cell values are changed for **IsClosed** column based on custom condition.

*Changing row style in PDF based on data*

You can customize the rows based on the record values by using [CellsExportingEventHandler](#).

#### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.CellsExportingEventHandler = CellsExportingEventHandler;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
private void CellsExportingEventHandler(object sender,
GridCellPdfExportingEventArgs e)
{
    if (!(e.NodeEntry is OrderInfo))
        return;
    if ((e.NodeEntry as OrderInfo).Country == "Mexico")
    {
        var cellStyle = new PdfGridCellStyle();
        cellStyle.BackgroundBrush = PdfBrushes.LightPink;
        cellStyle.Borders.All = new PdfPen(PdfBrushes.DarkGray, 0.2f);
        e.PdfGridCell.Style = cellStyle;
    }
}
```

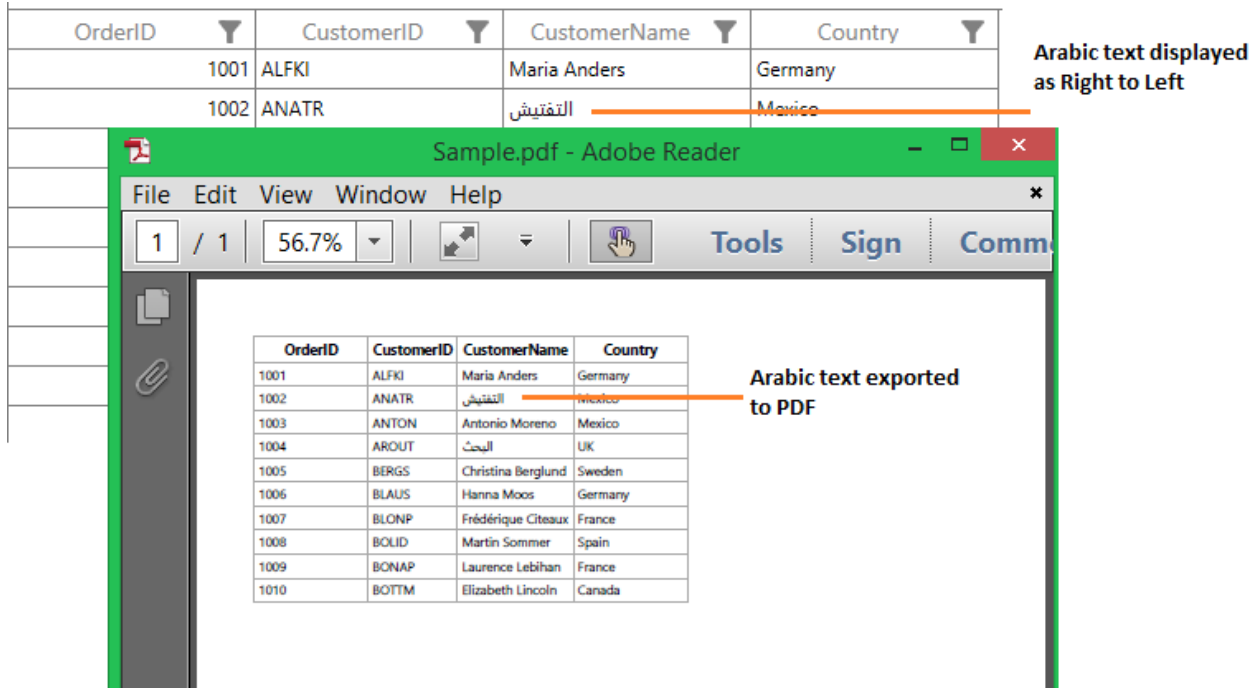


#### Exporting Middle Eastern Languages (Arabic, Hebrew) from SfDataGrid to PDF

By default, [Middle Eastern languages](#) (Arabic, Hebrew) in SfDataGrid are exported as left to right in PDF. You can export them as displayed in SfDataGrid (export from Right to Left) by enabling [RightToLeft](#) property in [PdfStringFormat](#) class and apply the format to the [PdfGridCell](#) by using [CellsExportingEventHandler](#).

#### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.CellsExportingEventHandler = CellsExportingEventHandler;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
private void CellsExportingEventHandler(object sender,
GridCellPdfExportingEventArgs e)
{
    if (e.CellType != ExportCellType.RecordCell)
        return;
    PdfStringFormat format = new PdfStringFormat();
    //format the string from right to left.
    format.RightToLeft = true;
    e.PdfGridCell.StringFormat = format;
}
```



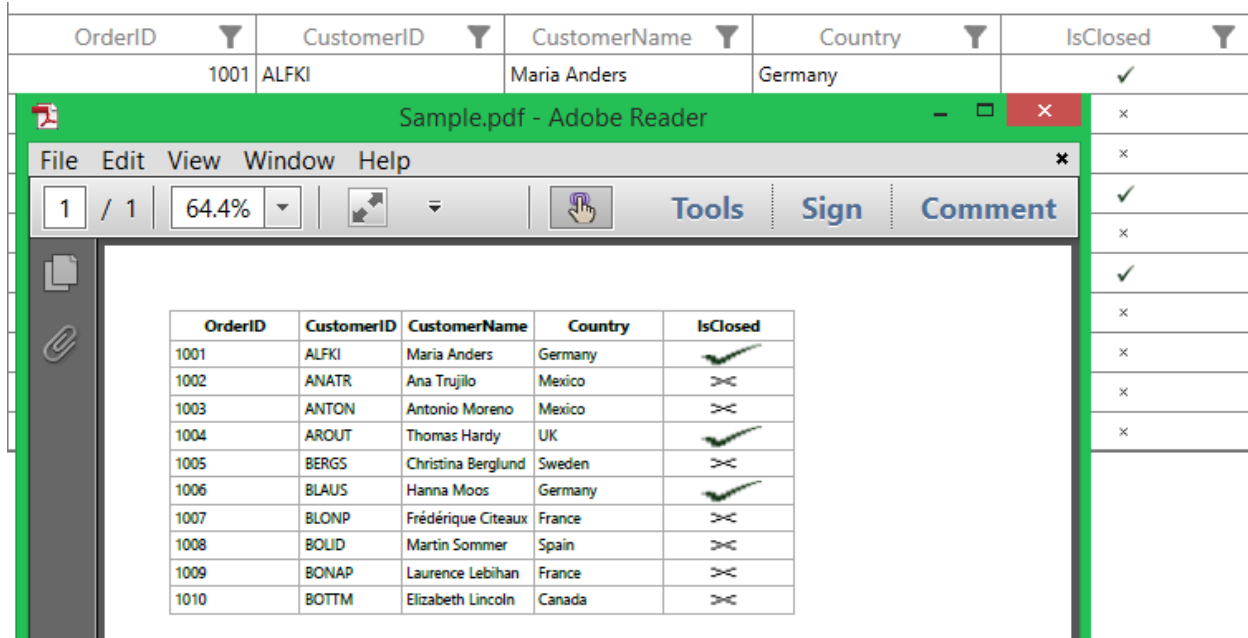
#### Exporting images to PDF document

By default, images which is loaded in the [GridTemplateColumn](#) will not be exported to PDF. You can export it by using [CellsExportingEventHandler](#) in [PdfExportingOptions](#). In [CellsExportingEventHandler](#), image is loaded in [PdfGridCell](#).

#### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.CellsExportingEventHandler = CellsExportingEventHandler;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
private void CellsExportingEventHandler(object sender,
GridCellPdfExportingEventArgs e)
{
    if (e.CellType == ExportCellType.RecordCell && e.ColumnName == "IsClosed")
    {
        var style = new PdfGridCellStyle();
        PdfPen normalBorder = new PdfPen(PdfBrushes.DarkGray, 0.2f);
        System.Drawing.Image image = null;
        //Images are exported based on the CellValue
        if (e.CellValue.Equals("True"))
        {
            //Access the image from the specified path
            image = System.Drawing.Image.FromFile(@"..\..\Images\True.png");
        }
        else
        {
            image = System.Drawing.Image.FromFile(@"..\..\Images\False.png");
        }
        //Create the PDFImage for the specified image and assigned to
        BackgroundImage of the PdfGridCellStyle
        style.BackgroundImage = PDFImage.FromImage(image);
        e.PdfGridCell.ImagePosition = PdfGridImagePosition.Fit;
        e.PdfGridCell.Style = style;
        //customize the Border color of PdfGridCell
    }
}
```

```
e.PdfGridCell.Style.Borders.All = normalBorder;
e.CellValue = null;
}
}
```



### Exporting DetailsView

By default, [DetailsViewDataGrid](#) will not be exported to PDF. You can export [DetailsViewDataGrid](#) by setting [ExportDetailsView](#) property as `true`.

### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportDetailsView = true;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```

	OrderID	CustomerID	CustomerName	Country	ShipCity
-	1001	ALFKI	Maria Anders	Germany	Berlin

	OrderID	ProductName
1001		
1002		
1003		

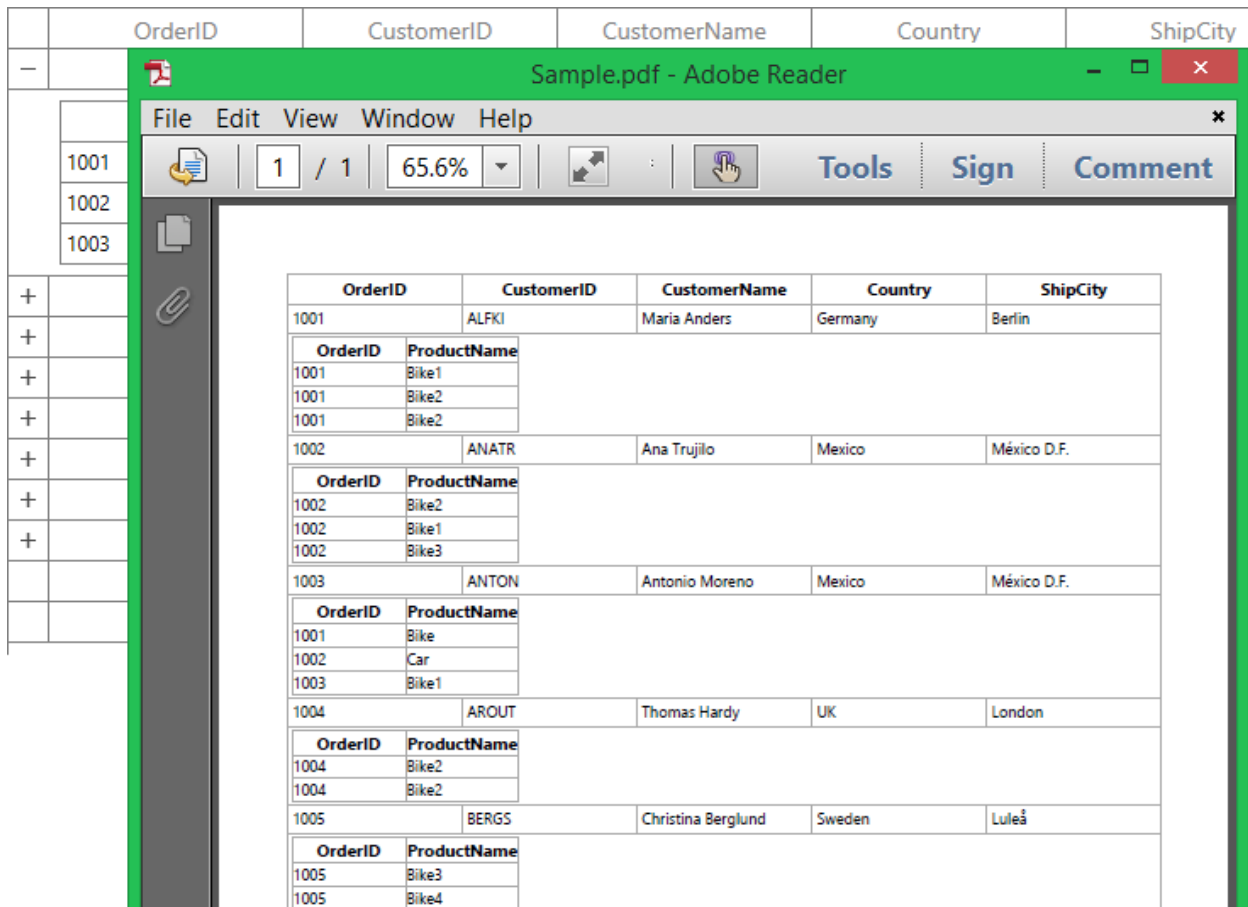
  

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLOPP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Leblond	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

By default, only expanded DetailsViewDataGrids only will be exported to PDF document. If you want to export all the DetailsViewDataGrids, you need to set [ExportAllDetails](#) as `true`.

### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportDetailsView = true;
options.ExportAllDetails = true;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
```



Here, first record only expanded in SfDataGrid. But all the DetailsViewDataGrid's are shown in exported PDF document.

You can customize its exporting operation by using [ChildGridExportingEventHandler](#).

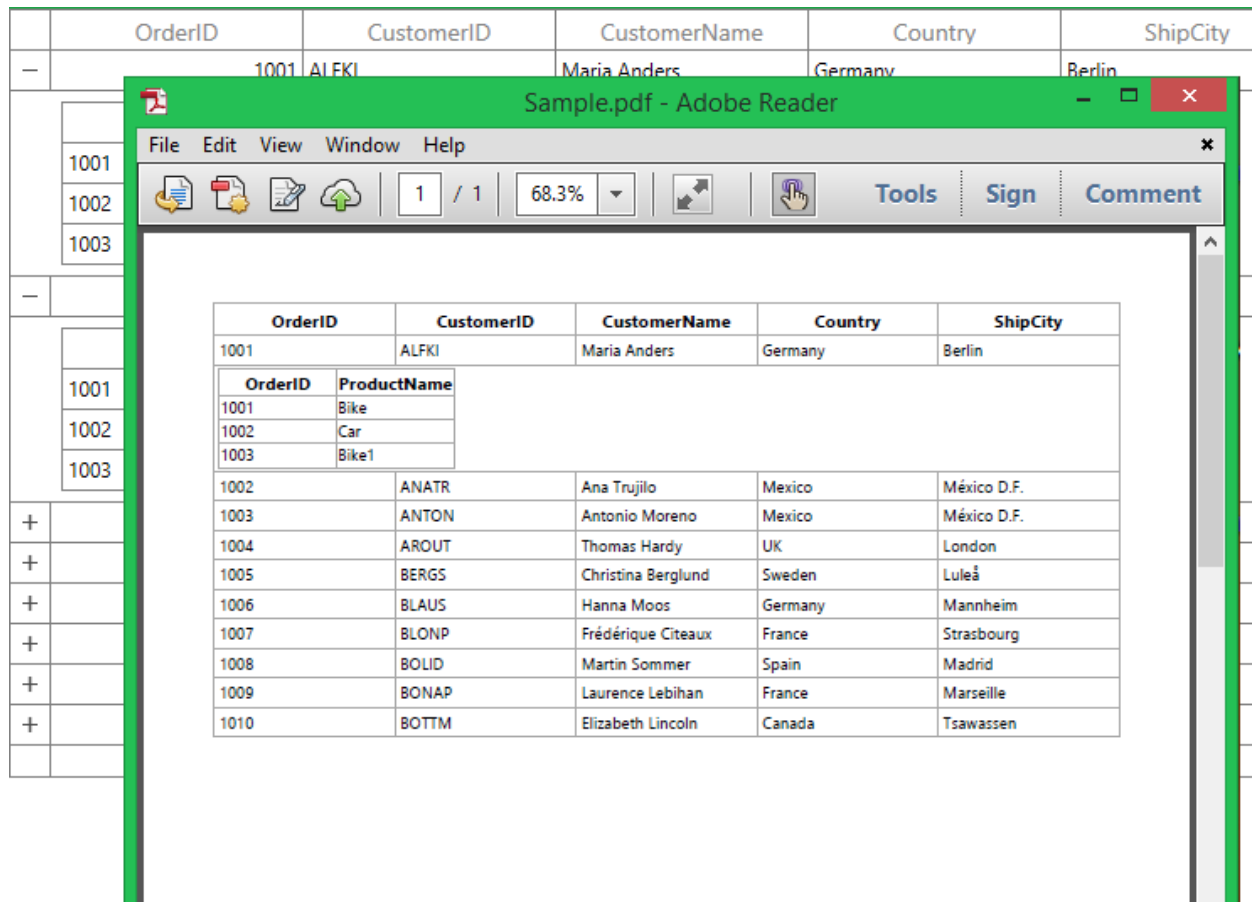
**Note:** While exporting DetailsViewDataGrid, [FitAllColumnsInOnePage](#) is set to **true** internally as horizontal pagination is not supported for DetailsViewDataGrid.

#### *Excluding DetailsViewDataGrid while exporting*

You can exclude particular DetailsViewDataGrid while exporting, by using the [ChildGridExportingEventHandler](#) and [ChildGridPdfExportingEventArgs.Cancel](#).

#### **C#**

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportDetailsView = true;
options.ChildGridExportingEventHandler = ChildGridExportingEventHandler;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
void ChildGridExportingEventHandler(object sender,
ChildGridPdfExportingEventArgs e)
{
var recordEntry = e.NodeEntry as RecordEntry;
if ((recordEntry.Data as OrderInfo).OrderID == 1002)
e.Cancel = true;
}
```



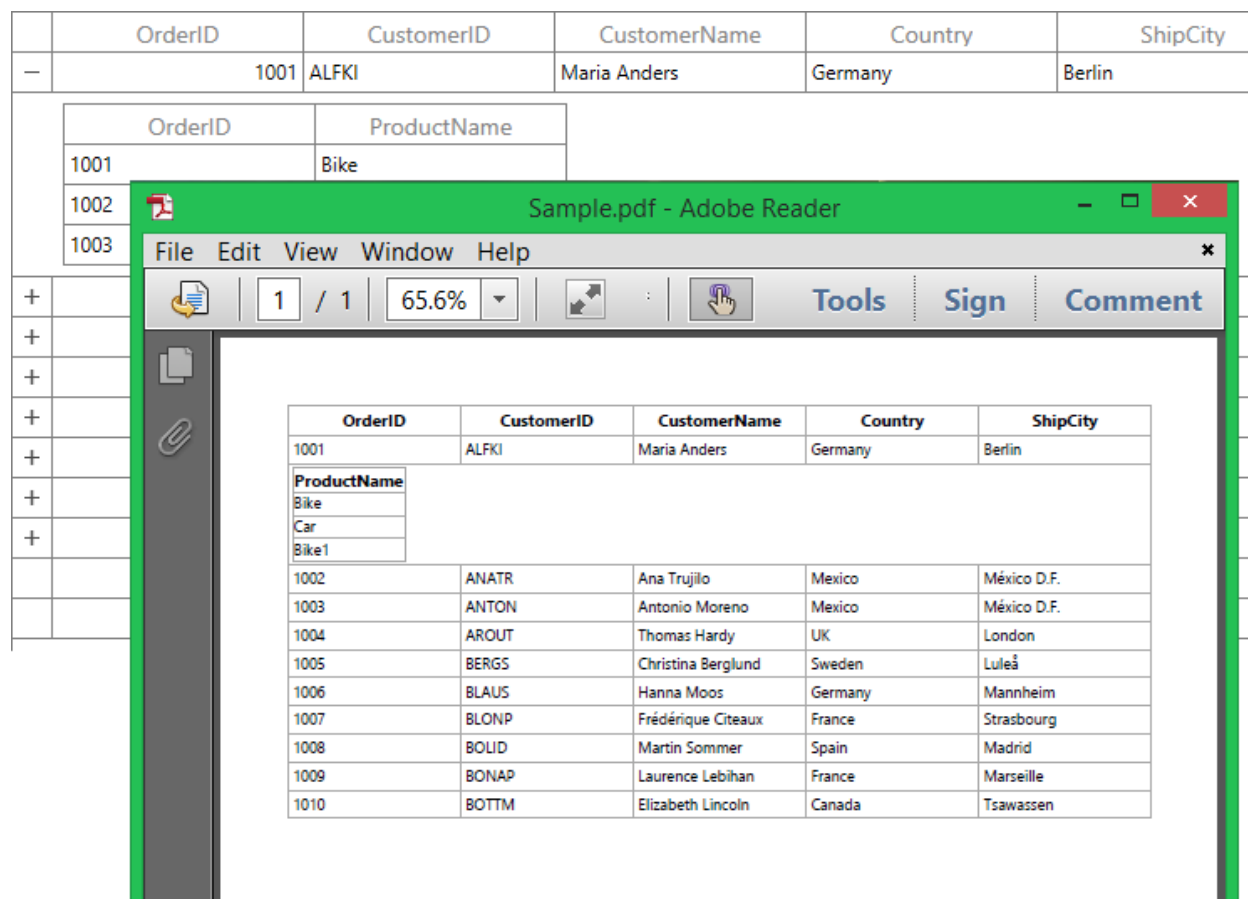
Here, `DetailsViewDataGrid` is not exported for the parent record having `OrderID` as 1002.

*Excluding DetailsViewDataGrid columns from exporting*

You can exclude `DetailsViewDataGrid` columns while exporting, by using `ChildGridExportingEventHandler` and `ChildGridPdfExportingEventArgs.PdfExportingOptions.ExcludeColumns`.

### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportDetailsView = true;
options.ChildGridExportingEventHandler = ChildGridExportingEventHandler;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
void ChildGridExportingEventHandler(object sender,
ChildGridPdfExportingEventArgs e)
{
e.PdfExportingOptions.ExcludeColumns = new List<string>() {
"OrderID" };
}
```



Here, **OrderID** column is displayed in **DetailsViewDataGrid** and it is excluded while exporting to PDF.

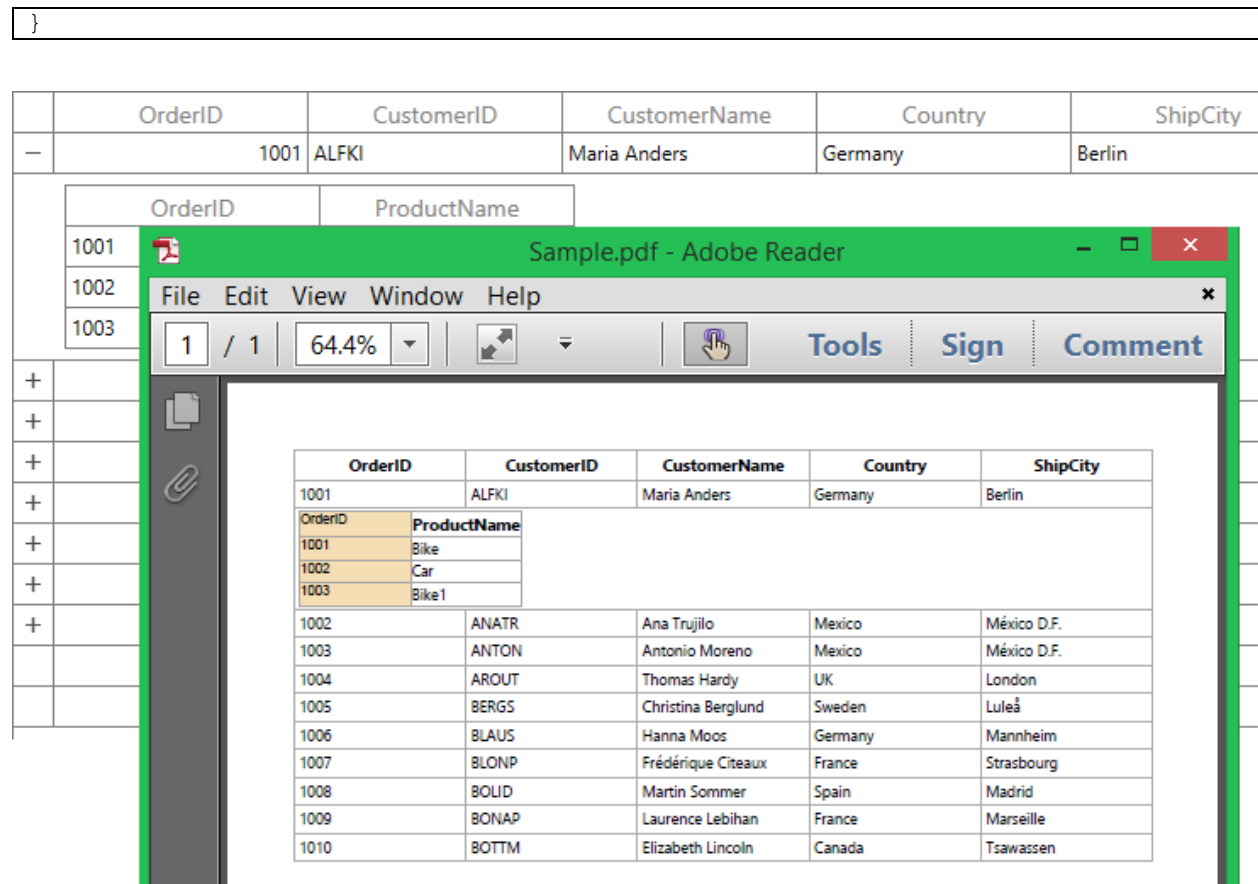
#### Customizing DetailsViewDataGrid cells

Like parent DataGrid, You can customize the **DetailsViewDataGrid** cells also by using **CellsExportingEventHandler**. Based on **GridCellPdfExportingEventArgs.GridViewDefinition** property, you can identify the particular **DetailsViewDataGrid** and customize it.

#### C#

```
PdfExportingOptions options = new PdfExportingOptions();
options.ExportDetailsView = true;
options.CellsExportingEventHandler = CellsExportingEventHandler;
var document = dataGrid.ExportToPdf(options);
document.Save("Sample.pdf");
private void CellsExportingEventHandler(object sender,
GridCellPdfExportingEventArgs e)
{
    if (e.GridViewDefinition == null || e.GridViewDefinition.RelationalColumn !=
"ProductDetails")
        return;
    if (e.ColumnName == "OrderID")
    {
        var cellStyle = new PdfGridCellStyle();
        cellStyle.BackgroundBrush = PdfBrushes.Wheat;
        cellStyle.Borders.All = new PdfPen(PdfBrushes.DarkGray, 0.2f);
        e.PdfGridCell.Style = cellStyle;
    }
}
```





### Printing in WPF DataGrid (SfDataGrid)

[WPF DataGrid](#) (SfDataGrid) provides support to print the data displayed in the DataGrid using [SfDataGrid.Print](#) method. It also provides support to display print preview window by calling [SfDataGrid.ShowPrintPreview](#) method.

#### C#

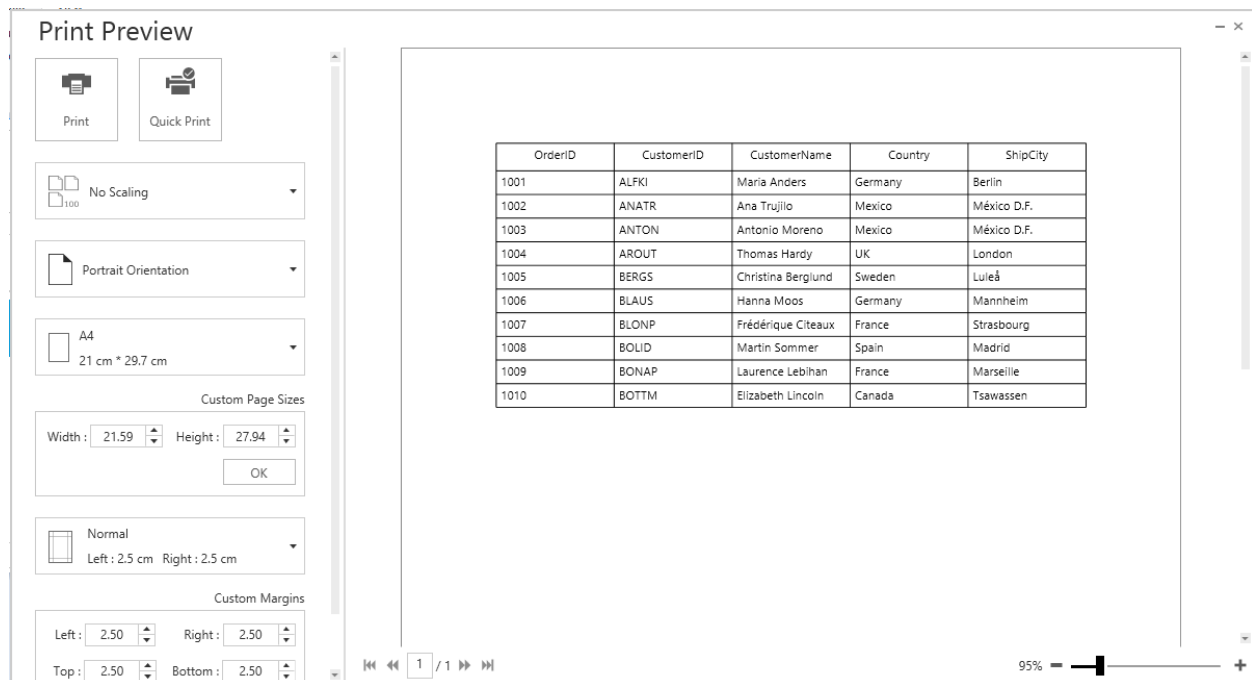
```
dataGrid.Print();
```

#### Print Preview

WPF DataGrid (SfDataGrid) provides option to display print preview to review and customize the document in desired format before printing. Print preview window can be opened by calling [SfDataGrid.ShowPrintPreview](#) method.

#### C#

```
dataGrid.ShowPrintPreview();
```



### Print Settings

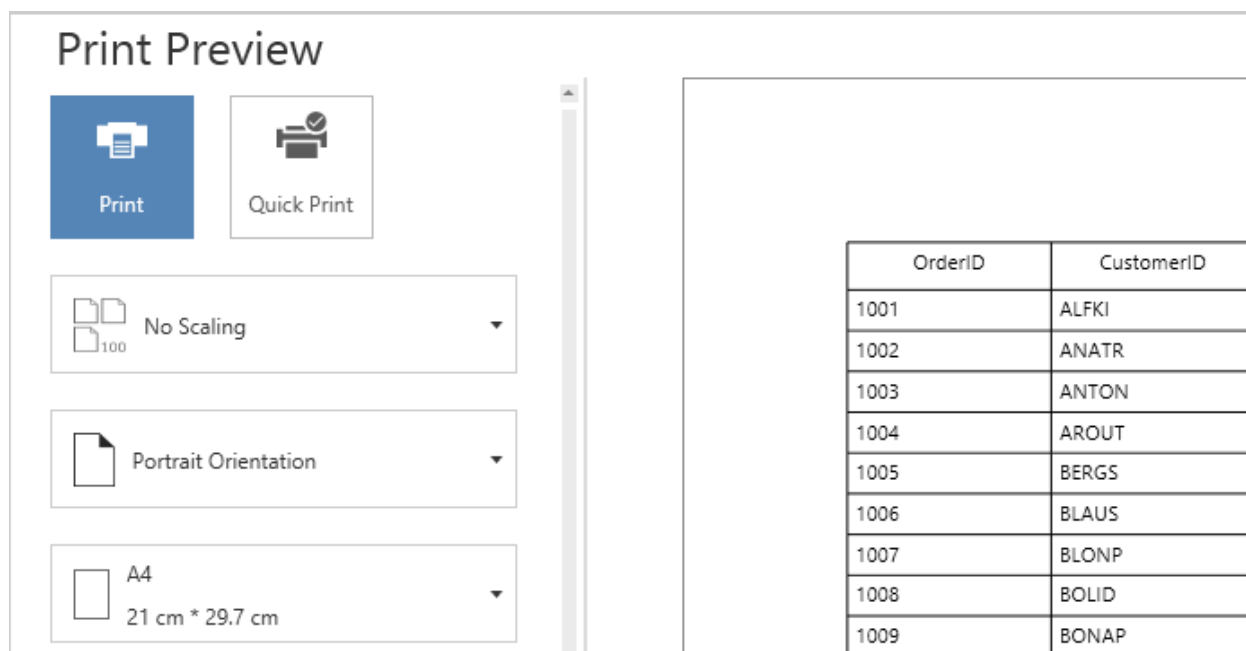
WPF DataGrid (SfDataGrid) provides various options to customize print preview settings using [SfDataGrid.PrintSettings](#) property of type [PrintSettings](#).

### C#

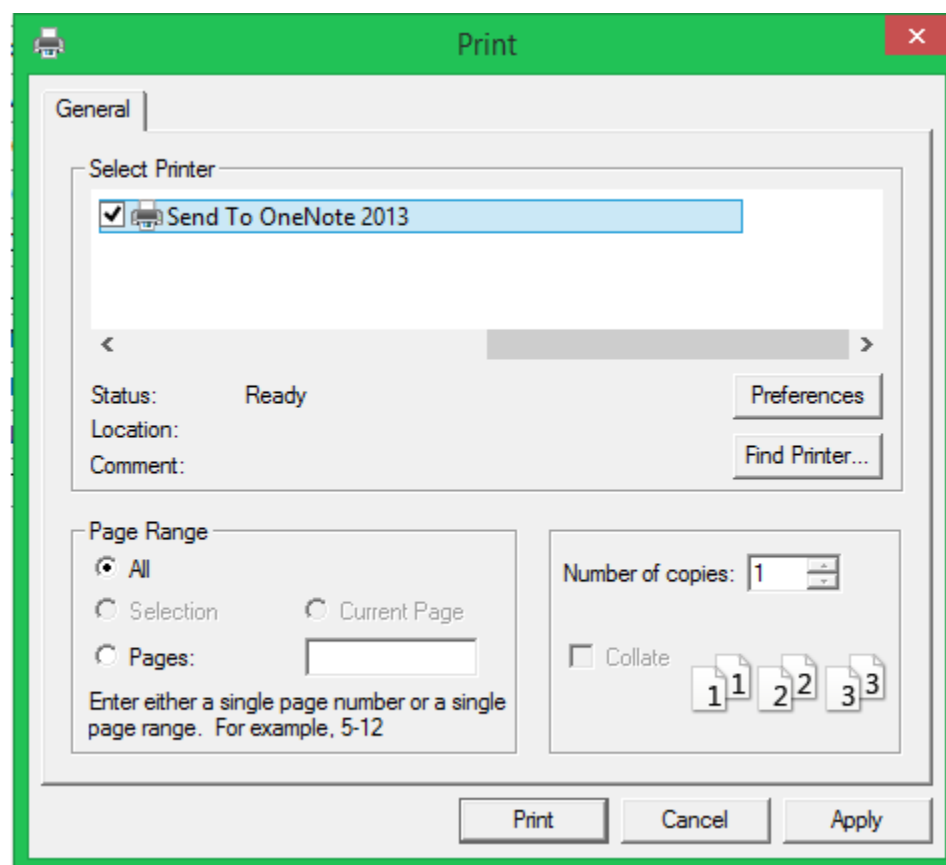
```
dataGrid.PrintSettings = new PrintSettings();
dataGrid.PrintSettings.AllowRepeatHeaders = false;
dataGrid.Print();
```

### Print

Print preview window has Print and Quick Print Buttons which needs to be clicked to print the SfDataGrid.



1. Clicking the Print button opens the System print dialog where user can select the printer and set the number of copies to be printed.



- Clicking the Quick Print button, directly print the pages using default printer without opening the print dialog.

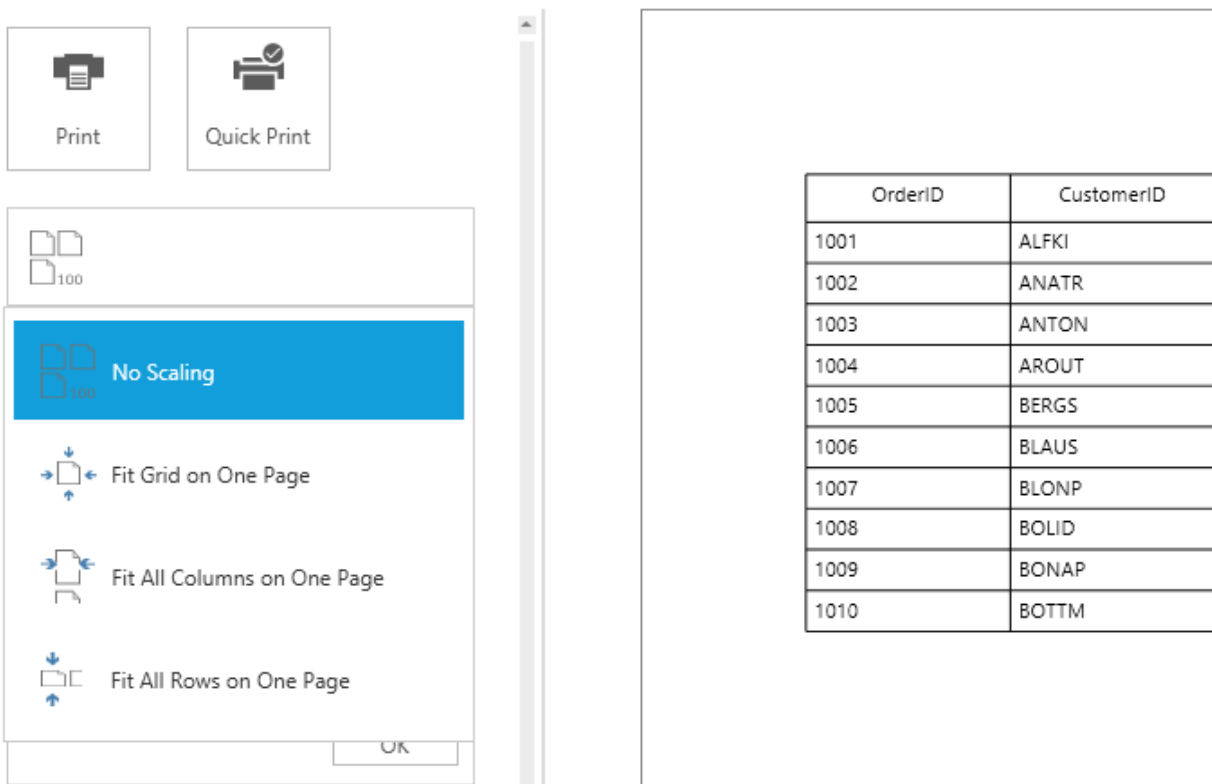
### Scaling

SfDataGrid provides support to scale rows or columns or both while printing to fit on one page. Scaling options can be changed by setting [PrintSettings.PrintScaleOption](#) property.

### C#

```
dataGrid.PrintSettings = new PrintSettings();
dataGrid.PrintSettings.PrintScaleOption =
PrintScaleOptions.FitAllColumnsonOnePage;
dataGrid.ShowPrintPreview();
```

Scaling options can be changed in print preview at runtime by selecting from scaling options drop-down in print preview.



### Column Header on each page

Column headers can be printed on each page by enabling [PrintSettings.AllowRepeatHeaders](#) property while printing.

### C#

```
dataGrid.PrintSettings = new PrintSettings();
dataGrid.PrintSettings.AllowRepeatHeaders = true;
dataGrid.Print();
```

*Changing Flow Direction while printing*

You can change the text direction in print page by using [PrintSettings.PrintFlowDirection](#) property.

**C#**

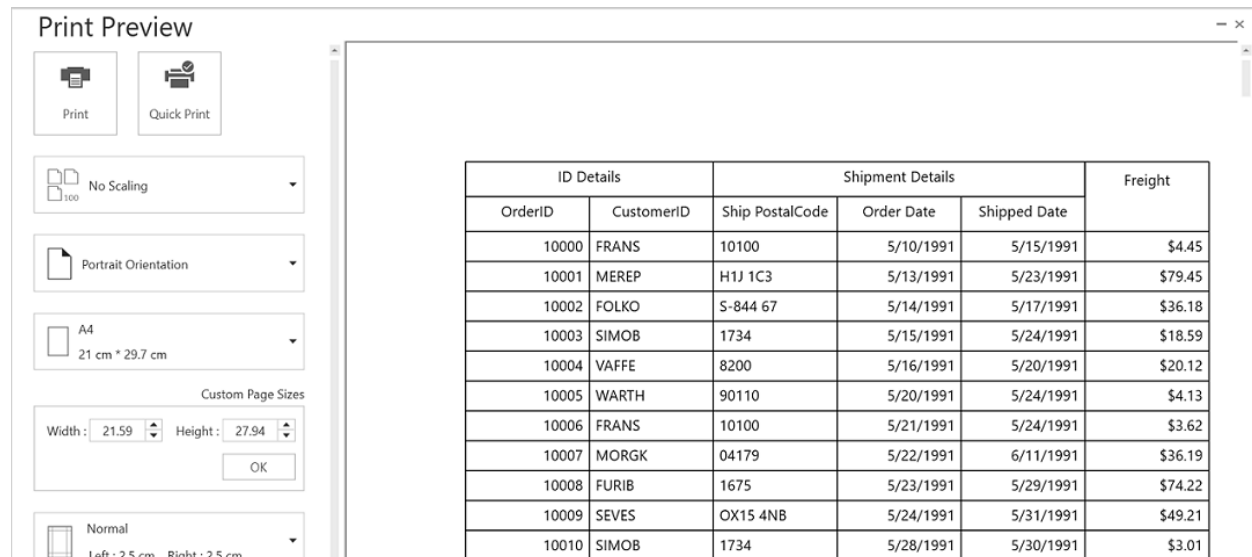
```
dataGrid.PrintSettings = new PrintSettings();
dataGrid.PrintSettings.PrintFlowDirection = FlowDirection.RightToLeft;
dataGrid.Print();
```

*Print with StackedHeaders*

SfDataGrid provides support to print the StackedHeaders by setting the [PrintSettings.CanPrintStackedHeaders](#) as **true**.

**C#**

```
this.sfDataGrid.PrintSettings.CanPrintStackedHeaders = true;
```

*Page Settings*

SfDataGrid provides various options to customize page settings using [SfDataGrid.PrintSettings](#) property of type [PrintSettings](#).

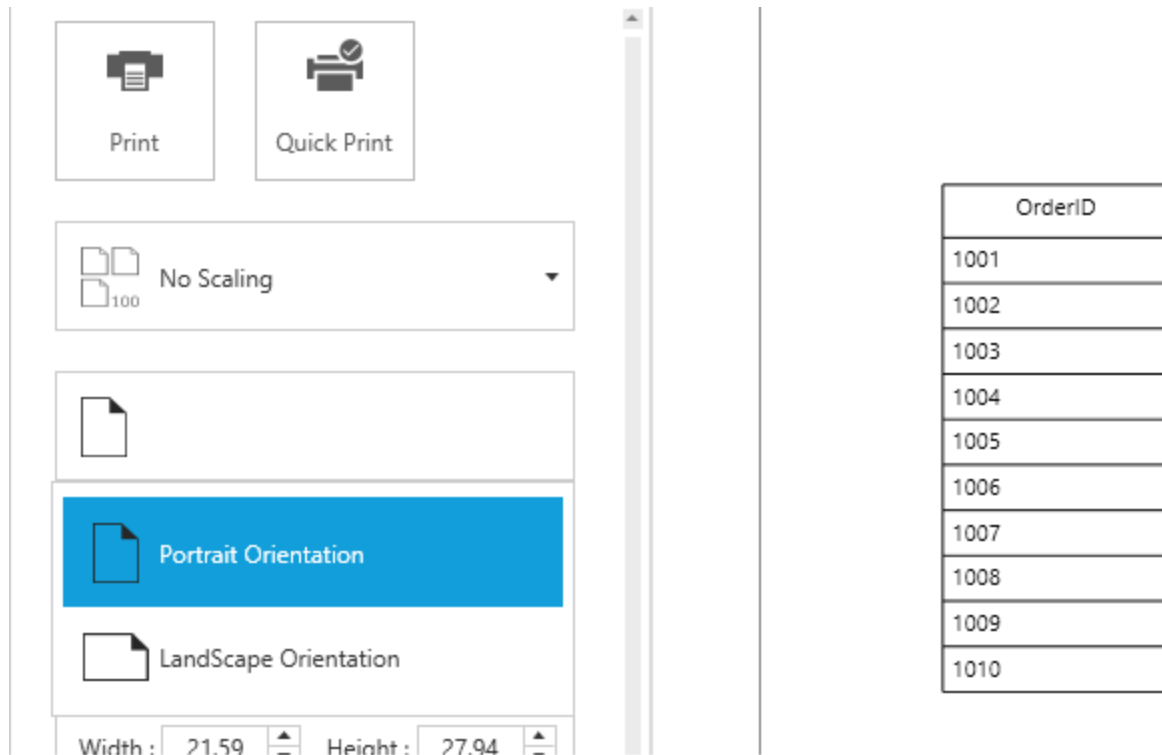
*Orientation*

SfDataGrid provides support to switch between Portrait (more rows but fewer columns) and Landscape (more columns but fewer rows) orientation while printing. Orientation can be changed by setting [PrintSettings.PrintPageOrientation](#) Property.

**C#**

```
dataGrid.PrintSettings = new PrintSettings();
dataGrid.PrintSettings.PrintPageOrientation = PrintOrientation.Landscape;
dataGrid.ShowPrintPreview();
```

Print orientation can be changed in print preview at runtime by selecting from orientation drop-down in print preview.



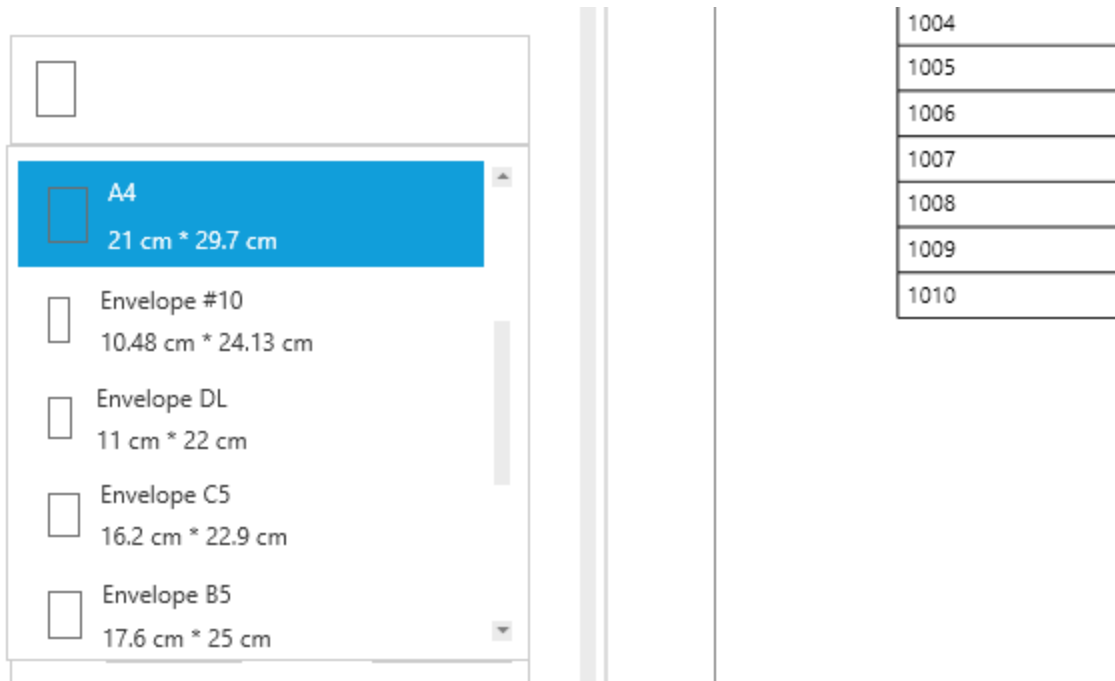
#### Page size

SfDataGrid provides support to change the page size. Page size can be changed by setting [PrintSettings.PrintPageWidth](#) and [PrintSettings.PrintPageHeight](#) properties.

#### C#

```
dataGrid.PrintSettings = new PrintSettings();  
dataGrid.PrintSettings.PrintPageHeight = 800;  
dataGrid.PrintSettings.PrintPageWidth = 800;  
dataGrid.Print();
```

Page size can be changed in print preview also by selecting from page-size drop-down which displays pre-defined page sizes. You can also manually enter custom page width and height in the editors below page-size drop-down and press OK to apply the custom width and height for the page.



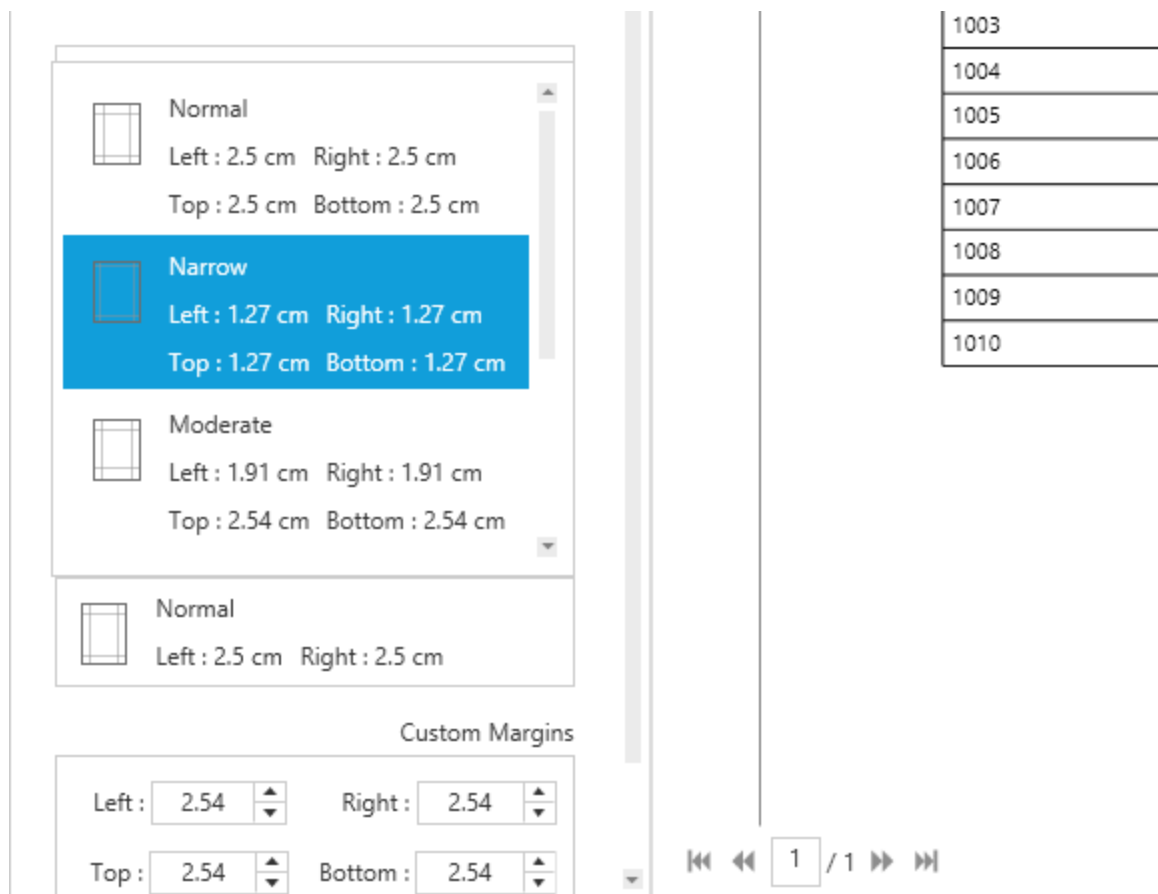
### Page margin

SfDataGrid provides support to change the page margins to adjust content in printed page. Page margin can be changed by setting [PrintSettings.PrintPageMargin](#) property.

### C#

```
dataGrid.PrintSettings = new PrintSettings();
dataGrid.PrintSettings.PrintPageMargin = new Thickness(5);
dataGrid.Print();
```

Page margin can be changed in print preview also by selecting from pre-defined page margin from margin drop-down. You can manually enter custom margins in the editors below margin drop-down and press OK to apply the custom margin.



### Setting Header and Footer

SfDataGrid provides a way to display additional content at the top (Header) or bottom (Footer) of the page while printing. This can be achieved by setting [PrintPageHeaderHeight](#), [PrintPageHeaderTemplate](#), [PrintPageFooterHeight](#), [PrintPageFooterTemplate](#) properties in [PrintSettings](#).

Steps to add page header while printing,

1. Create DataTemplate in `Application.Resources`.

### XML

```
<Application.Resources>
  <DataTemplate x:Key="PageHeaderTempalte">
    <Grid Background="Gray">
      <TextBlock Text="Syncfusion"
        FontSize="18"
        FontWeight="Bold"
        Foreground="White"
        HorizontalAlignment="Center"/>
    </Grid>
  </DataTemplate>
</Application.Resources>
```

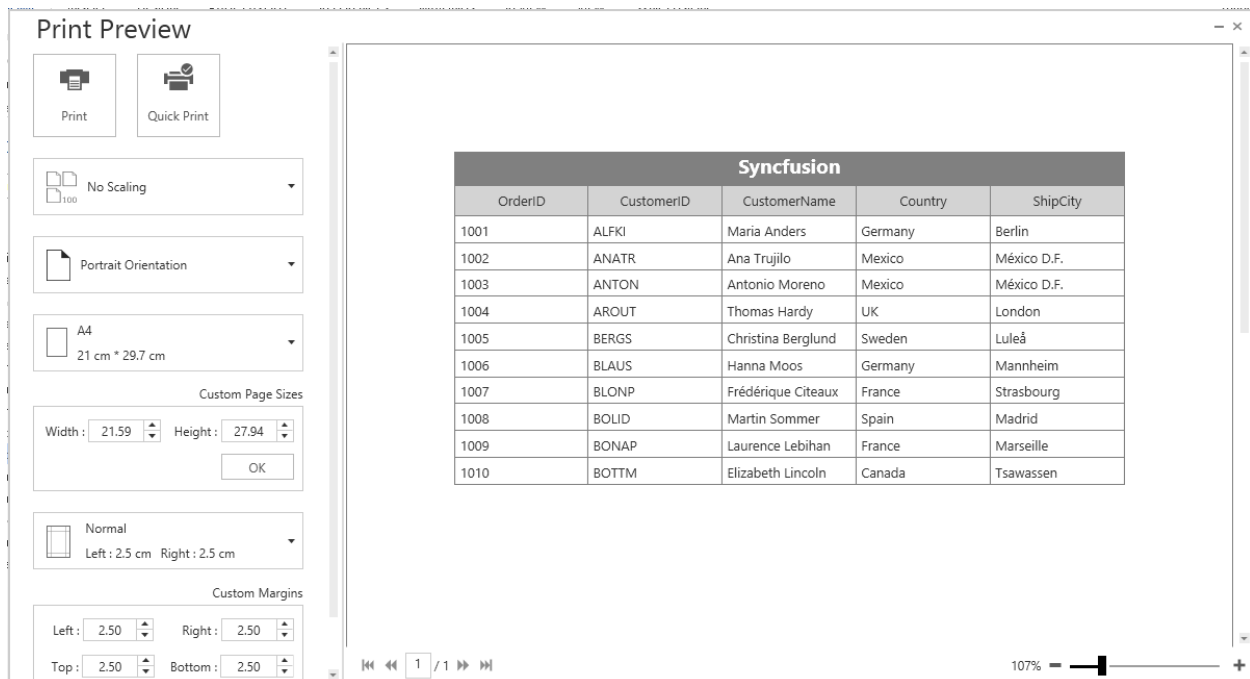


2. Set the above defined DataTemplate to [PrintSettings.PrintPageHeaderTemplate](#) and assign value for [PrintSettings.PrintPageHeaderHeight](#) property also.

### C#

```
dataGrid.PrintSettings = new PrintSettings();
dataGrid.PrintSettings.PrintPageHeaderHeight = 30;
dataGrid.PrintSettings.PrintPageHeaderTemplate =
Application.Current.Resources["PageHeaderTempalte"] as DataTemplate;
dataGrid.ShowPrintPreview();
```

3. Now run the application and you can see page header in all the pages. In the same way, you can set [PrintSettings.PrintPageFooterTemplate](#) also.



**Note:** [PrintManagerBase](#) is the [DataContext](#) for [PrintPageControl](#), where the header and footer templates are loaded.

### Printing Current Date time and Page number

You can print current Date and Time at each page by setting the [PrintPageFooterHeight](#) [PrintPageFooterTemplate](#) properties in [PrintSettings](#).

You can get the page number from [PrintPageControl](#).

### XML

```
<Application.Resources>
<DataTemplate x:Key="PageFooterTempalte">
<Grid>
<TextBlock HorizontalAlignment="Center"
FontSize="20"
Text="{Binding Source={x:Static system:DateTime.Now}}"/>
<TextBlock Margin="0,0,10,0"
HorizontalAlignment="Right"
VerticalAlignment="Center" FontSize="20">
```

```
<TextBlock.Text>
<Binding Path="PageIndex"
RelativeSource="{RelativeSource Mode=FindAncestor,
AncestorType={x:Type syncfusion:PrintPageControl}}"
StringFormat="Page : {0}" />
</TextBlock.Text>
</TextBlock>
</Grid>
</DataTemplate>
</Application.Resources>
```

### C#

```
dataGrid.PrintSettings = new PrintSettings();
dataGrid.PrintSettings.PrintPageFooterHeight = 30;
dataGrid.PrintSettings.PrintPageFooterTemplate =
Application.Current.Resources["PageFooterTempalte"] as DataTemplate;
dataGrid.ShowPrintPreview();
```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	Mexico D.F.
1003	ANTON	Antonio Moreno	Mexico	Mexico D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BONAP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Leblond	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Toronto
1011	BSBEV	Sean Jacobson	Norway	Stavanger
1012	CENTC	Phyllis Allen	Venezuela	Bogotá
1013	CHOPS	Marvin Allen	Belgium	Charleroi
1014	EASTC	Michael Allen	Spain	Madrid
1015	ERNSH	Cecil Allison	Germany	Köln
1016	FOUJG	Oscar Alpuerto	UK	London
1017	FOLKO	Sandra Alaminano	Venezuela	San Cristóbal
1018	FRANR	Selma Alvarad	Mexico	Mexico D.F.
1019	FRANS	Emilio Alvarez	Denmark	Århus
1020	FURUS	Maxwell Amundson	USA	Anchorage
1021	GALED	Mae Anderson	Canada	Montreal
1022	GODOS	Ramona Antrim	USA	Seattle
1023	GROSR	Sabrina Appelbaum	Venezuela	San Cristóbal
1024	HILAA	Hannah Arakawa	Sweden	Bräcke
1025	HUNGO	Kyley Arbelaez	Venezuela	L. de Margarita
1026	LAMAI	Tom Johnston	Mexico	Mexico D.F.
1027	LAZYK	Thomas Armstrong	Sweden	Luleå
1028	LILAS	John Arthur	Norway	Stavanger
1029	LINGD	Chris Ashton	Germany	Leipzig
1030	MAGAA	Teresa Altiman	Venezuela	San Cristóbal

1/28/2016 6:31:50 AM

Page : 1

Different modes of printing for better performance

*Printing using drawing for better performance*

When you require better performance and don't want appearance settings (Background and Foreground) to be exported while printing then you can use this printing option by setting [PrintSettings.AllowPrintByDrawing](#) property to `true`.

---

**Note:** This is default print mode. For appearance customization of rows and cells while printing, refer [printing customization](#) section.

---

*Printing using UIElement Rendering*

When you want to print the SfDataGrid with same appearance settings as in the display (Background and Foreground) or with custom appearance by writing styles, then you can enable this print option by setting [PrintSettings.AllowPrintByDrawing](#) property to `false`.

You can print SfDataGrid as it displayed in View by setting [PrintSettings.AllowPrintStyles](#) to `true`.

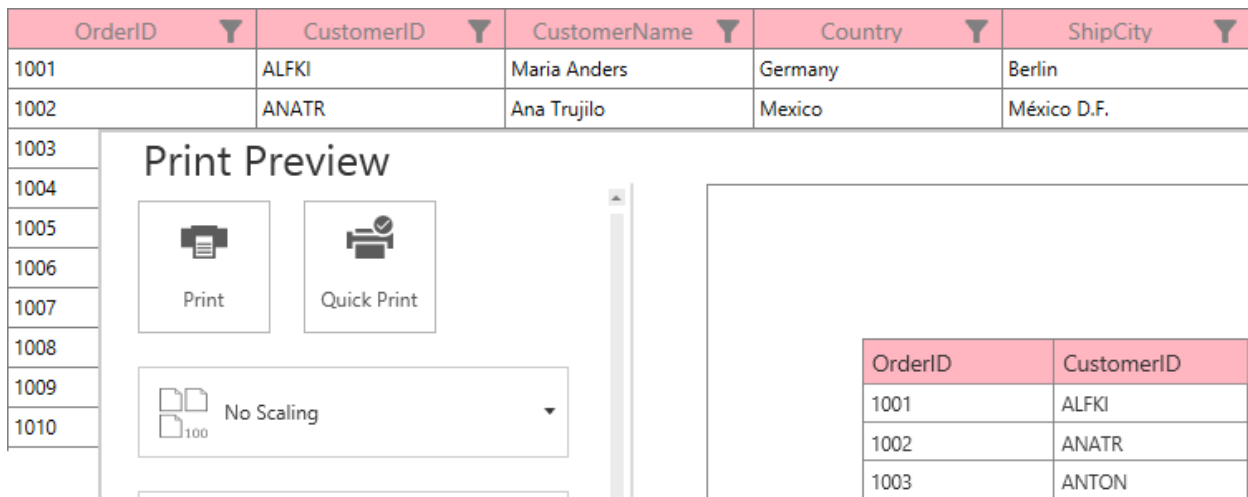
**C#**

```
dataGrid.PrintSettings = new PrintSettings();
dataGrid.PrintSettings.AllowPrintByDrawing = false;
dataGrid.PrintSettings.AllowPrintStyles = true;
dataGrid.ShowPrintPreview();
```

[GridHeaderCellControl](#) style customized and the same style will be exported while printing by setting [PrintSettings.AllowPrintStyles](#) to `true`.

#### XML

```
<Application.Resources>
<Style TargetType="syncfusion:GridHeaderCellControl">
<Setter Property="Background" Value="LightPink"/>
</Style>
</Application.Resources>
```



#### Applying custom style

Custom styles can be applied while printing by setting [PrintSettings.AllowPrintStyles](#) to `false` and writing style for below controls based on your requirement.

Appearance to be customized	TargetType of Style
Header Cell	<a href="#">PrintHeaderCell</a>
Normal Cells	<a href="#">PrintGridCell</a>
Caption summary cells	<a href="#">PrintCaptionSummaryCell</a>
Group summary cells	<a href="#">PrintGroupSummaryCell</a>
Table summary cells	<a href="#">PrintTableSummaryCell</a>
Unbound row cells	<a href="#">PrintUnboundRowCell</a>

To provide custom appearance for Header cells while printing, [PrintHeaderCell](#) style is customized.

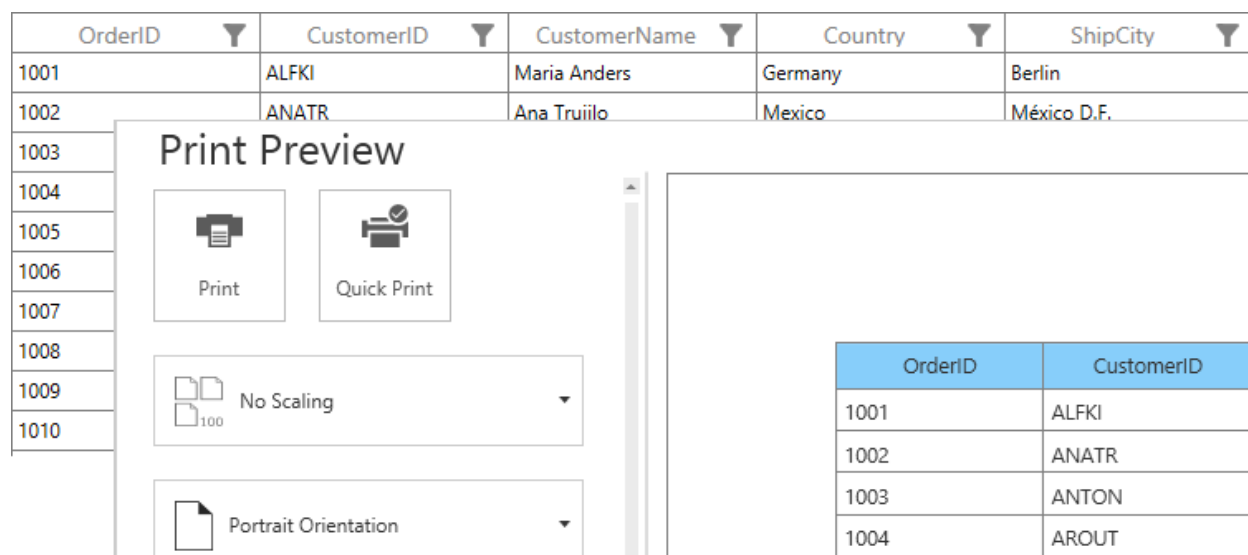
**XML**

```
<Application.Resources>
<Style TargetType="syncfusion:PrintHeaderCell">
<Setter Property="Background" Value="LightSkyBlue"/>
</Style>
</Application.Resources>
```

To print SfDataGrid with custom styles, set [PrintSettings.AllowPrintByDrawing](#) and [PrintSettings.AllowPrintStyles](#) properties to `false`.

**C#**

```
dataGrid.PrintSettings = new PrintSettings();
dataGrid.PrintSettings.AllowPrintByDrawing = false;
dataGrid.PrintSettings.AllowPrintStyles = false;
dataGrid.ShowPrintPreview();
```

**Printing Customization**

Printing operations can be customized by overriding [GridPrintManager](#) and its available methods.

*Setting different Row Height*

SfDataGrid allows you to set different Row height for specific rows while printing. You can achieve this by overriding the [GetRowHeight](#) method in [PrintManagerBase](#) class.

**C#**

```
public class CustomPrintManager : GridPrintManager
{
    public CustomPrintManager(SfDataGrid grid)
    : base(grid)
    {
    }
    protected override double GetRowHeight(object record, int rowIndex, RowType type)
    {
    }
```

```

if (rowIndex != -1 && !(record is Group))
if (rowIndex % 2 != 0)
return 50.0;
return base.GetRowHeight(record, rowIndex, type);
}
}

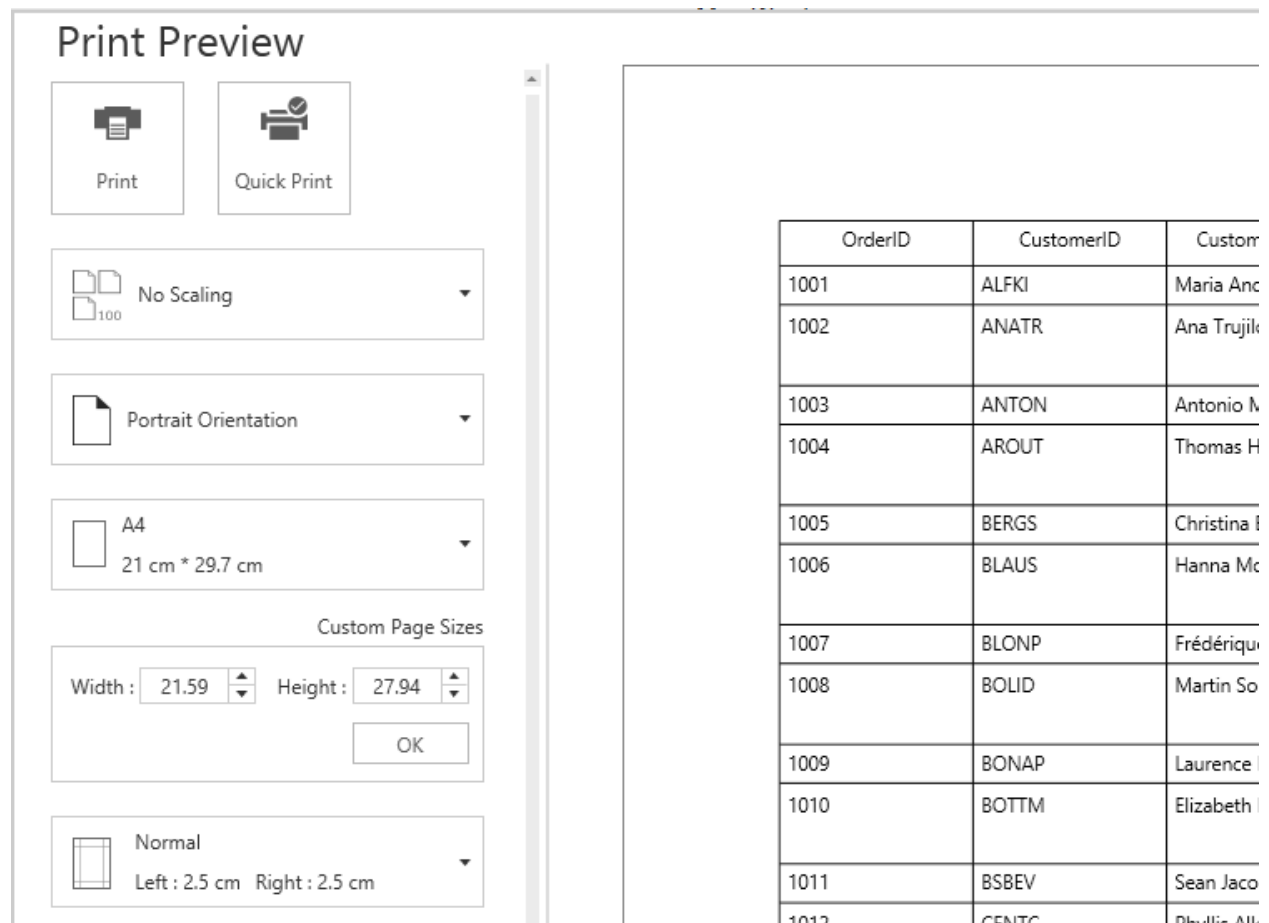
```

**C#**

```

dataGrid.PrintSettings.PrintManagerBase = new
CustomPrintManager(this.dataGrid);
dataGrid.ShowPrintPreview();

```

*Hiding rows while printing*

You can hide specific row by using [GetRowHeight](#) method in [PrintManagerBase](#) class and setting height as 0.

Here, unbound row is excluded while printing. Likewise, you can hide any row based on record and row index.

**C#**

```

public class CustomPrintManager : GridPrintManager
{
public CustomPrintManager(SfDataGrid grid)

```

```
: base(grid)
{
}
protected override double GetRowHeight(object record, int rowIndex, RowType
type)
{
if (record is GridUnBoundRow)
return 0;
return base.GetRowHeight(record, rowIndex, type);
}
}
```

### C#

```
dataGrid.PrintSettings.PrintManagerBase = new
CustomPrintManager(this.dataGrid);
dataGrid.ShowPrintPreview();
```

#### *Setup columns to be printed*

WPF DataGrid (SfDataGrid) allows you to exclude the columns while printing the grid. You can change the column list by overriding the [GetColumnNames](#) method in [PrintManagerBase](#) class.

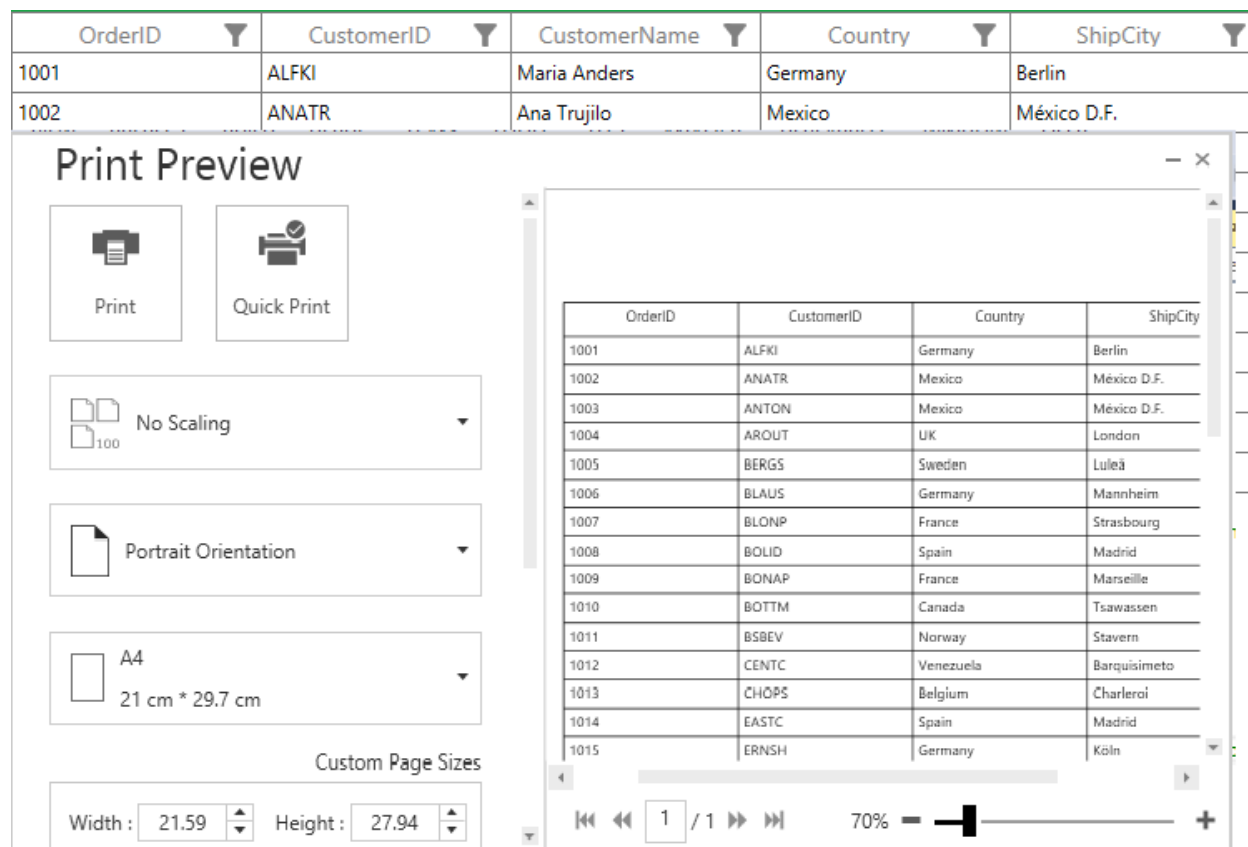
### C#

```
private class CustomPrintManager : GridPrintManager
{
public CustomPrintManager(SfDataGrid grid)
: base(grid)
{
}
protected override List<string> GetColumnNames()
{
List<string> columnList = this.dataGrid.Columns.Select(x =>
x.MappingName).ToList();
columnList.Remove("CustomerName");
return columnList;
}
}
```

### C#

```
dataGrid.PrintSettings.PrintManagerBase = new
CustomPrintManager(this.dataGrid);
dataGrid.ShowPrintPreview();
```

Here, **CustomerName** column is displayed in grid. But it is excluded while printing.



### Customize the header text while printing

SfDataGrid allows you to change column header text while printing the grid. You can change the Column header text by overriding the [GetColumnHeaderText](#) method in [GridPrintManager](#).

### C#

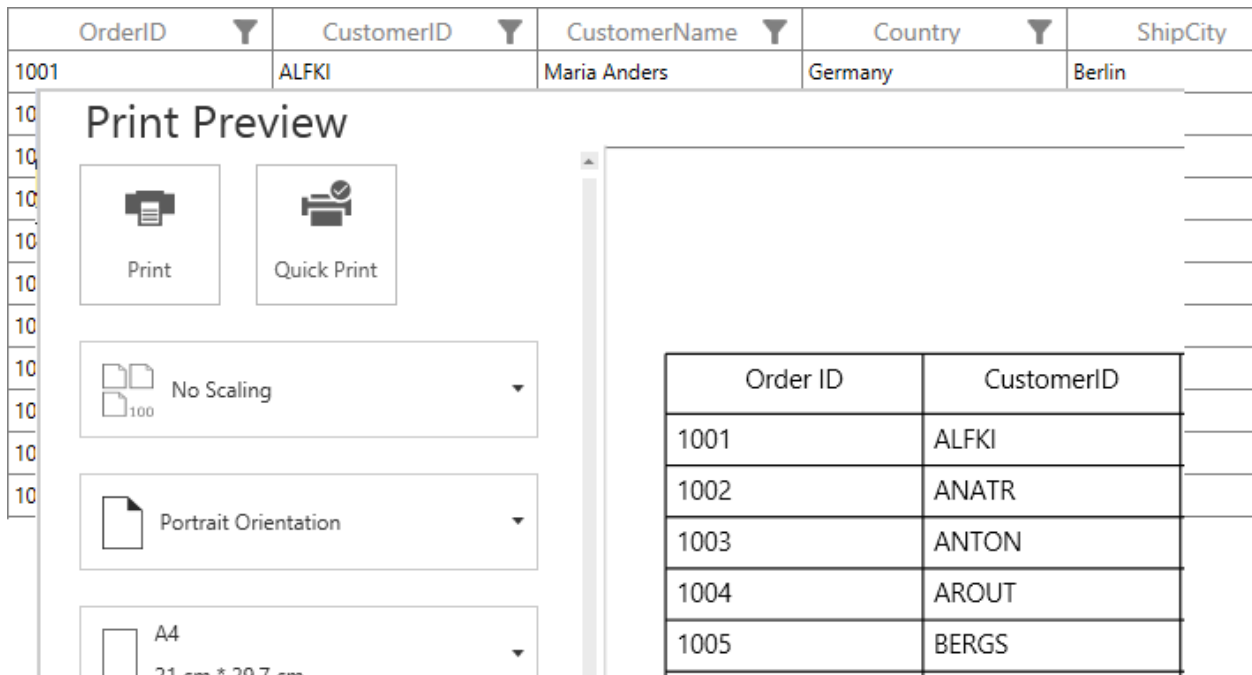
```
private class CustomPrintManager : GridPrintManager
{
    public CustomPrintManager(SfDataGrid grid)
    : base(grid)
    {
    }
    protected override string GetColumnHeaderText(string mappingName)
    {
        if (mappingName == "OrderID")
            return "Order ID";
        return base.GetColumnHeaderText(mappingName);
    }
}
```

### C#

```
dataGrid.PrintSettings.PrintManagerBase = new
CustomPrintManager(this.dataGrid);
dataGrid.ShowPrintPreview();
```



Here, OrderID column Header text is changed as **Order ID** while printing.



#### Styling Rows when AllowPrintByDrawing enabled

You can apply row styles based on custom condition by overriding [OnRenderCell](#) method in [GridPrintManager](#) class.

#### C#

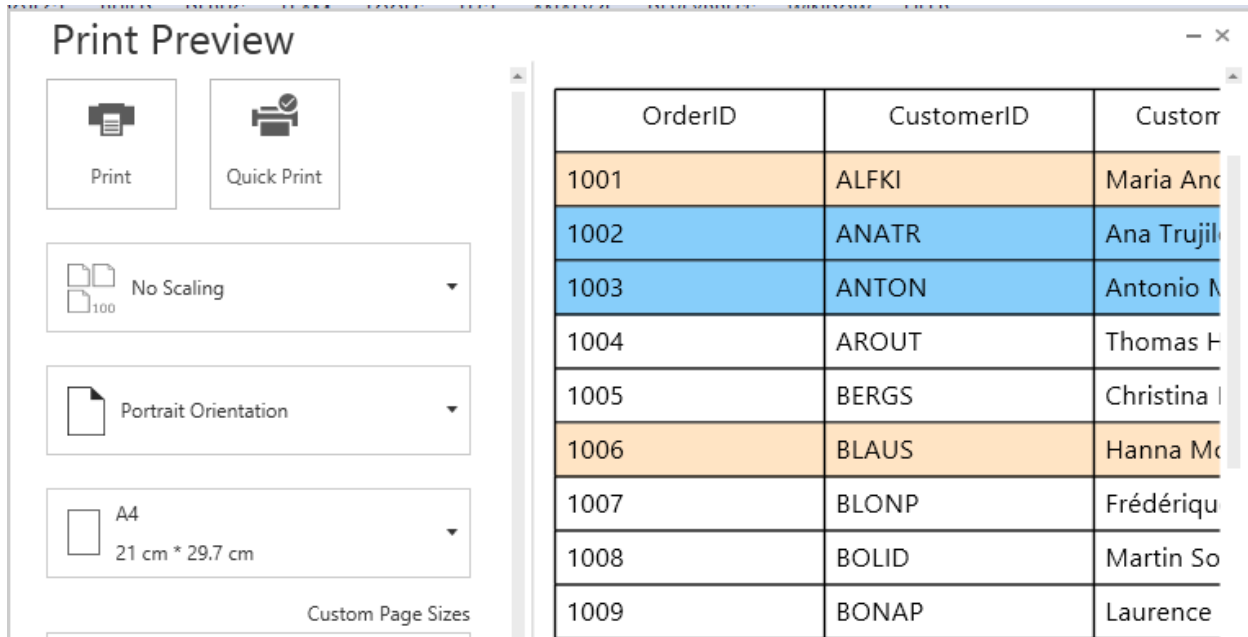
```
private class CustomPrintManager : GridPrintManager
{
    public CustomPrintManager(SfDataGrid grid)
    : base(grid)
    {
    }
    protected override void OnRenderCell(DrawingContext drawingContext, RowInfo rowInfo, CellInfo cellInfo)
    {
        if (!(rowInfo.Record is RecordEntry))
        {
            base.OnRenderCell(drawingContext, rowInfo, cellInfo);
            return;
        }
        var rect = new Rect((cellInfo.CellRect).X, (cellInfo.CellRect).Y + 0.5,
            (cellInfo.CellRect).Width, (cellInfo.CellRect).Height);
        if (((rowInfo.Record as RecordEntry).Data as OrderInfo).Country ==
            "Germany")
            drawingContext.DrawGeometry(new SolidColorBrush(Colors.Bisque), new Pen(),
                new RectangleGeometry(rect));
        else if (((rowInfo.Record as RecordEntry).Data as OrderInfo).Country ==
            "Mexico")
            drawingContext.DrawGeometry(new SolidColorBrush(Colors.LightSkyBlue), new
                Pen(), new RectangleGeometry(rect));
        base.OnRenderCell(drawingContext, rowInfo, cellInfo);
    }
}
```

```
}

```

**C#**

```
dataGrid.PrintSettings.PrintManagerBase = new
CustomPrintManager(this.dataGrid);
dataGrid.ShowPrintPreview();
```

*Styling Rows when AllowPrintByDrawing disabled*

You can apply row styles based on custom logic by overriding [GetPrintGridCell](#) method in [GridPrintManager](#).

**C#**

```
private class CustomPrintManager : GridPrintManager
{
    public CustomPrintManager(SfDataGrid grid)
    : base(grid)
    {
    }

    public override ContentControl GetPrintGridCell(object record, string
mappingName)
    {
        if (!(record is OrderInfo))
            return base.GetPrintGridCell(record, mappingName);
        if ((record as OrderInfo).Country == "Germany")
            return new PrintGridCell() { Background = new SolidColorBrush(Colors.Bisque)
};
        else if ((record as OrderInfo).Country == "Mexico")
            return new PrintGridCell() { Background = new
SolidColorBrush(Colors.LightSkyBlue) };
        return base.GetPrintGridCell(record, mappingName);
    }
}
```

**C#**

```
dataGrid.PrintSettings.PrintManagerBase = new
CustomPrintManager(this.dataGrid);
dataGrid.PrintSettings.AllowPrintByDrawing = false;
dataGrid.ShowPrintPreview();
```

### Print Preview

Print
 Quick Print

No Scaling

Portrait Orientation

A4

OrderID	CustomerID	Custome
1001	ALFKI	Maria Ar
1002	ANATR	Ana Truji
1003	ANTON	Antonio
1004	AROUT	Thomas
1005	BERGS	Christina
1006	BLAUS	Hanna M
1007	BLONP	Frédériqi

Appearance to be customized	Method
Header Cell	<a href="#">GetPrintHeaderCell</a>
Normal Cells	<a href="#">GetPrintGridCell</a>
Caption summary cells	<a href="#">GetPrintCaptionSummaryCell</a>
Group summary cells	<a href="#">GetPrintGroupSummaryCell</a>
Table summary cells	<a href="#">GetPrintTableSummaryCell</a>
Unbound row cells	<a href="#">GetPrintUnboundRowCell</a>

*Setup alternate row style when AllowPrintByDrawing enabled*

SfDataGrid allows you to apply alternative row style by overriding [OnRenderCell](#) method in [GridPrintManager](#).

**C#**

```
private class CustomPrintManager : GridPrintManager
{
    public CustomPrintManager(SfDataGrid grid)
    : base(grid)
    {
    }
    protected override void OnRenderCell(DrawingContext drawingContext, RowInfo
rowInfo, CellInfo cellInfo)
```

```

{
    var index = dataGrid.View.Records.IndexOfRecord(rowInfo.Record);
    var rect = new Rect((cellInfo.CellRect).X, (cellInfo.CellRect).Y + 0.5,
        (cellInfo.CellRect).Width, (cellInfo.CellRect).Height);
    if (index % 2 == 0)
        drawingContext.DrawGeometry(new SolidColorBrush(Colors.Bisque), new Pen(),
            new RectangleGeometry(rect));
    base.OnRenderCell(drawingContext, rowInfo, cellInfo);
}
}

```

**C#**

```

dataGrid.PrintSettings.PrintManagerBase = new
CustomPrintManager(this.dataGrid);
dataGrid.ShowPrintPreview();

```

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.

### Print Preview

Print
 Quick Print

No Scaling

Portrait Orientation

OrderID	CustomerID	CustomerName
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujillo
1003	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglur
1006	BLAUS	Hanna Moos

*Setup alternate row style when AllowPrintByDrawing disabled*

SfDataGrid allows you to apply alternative row style by overriding [GetPrintGridCell](#) method in [GridPrintManager](#).

**C#**

```

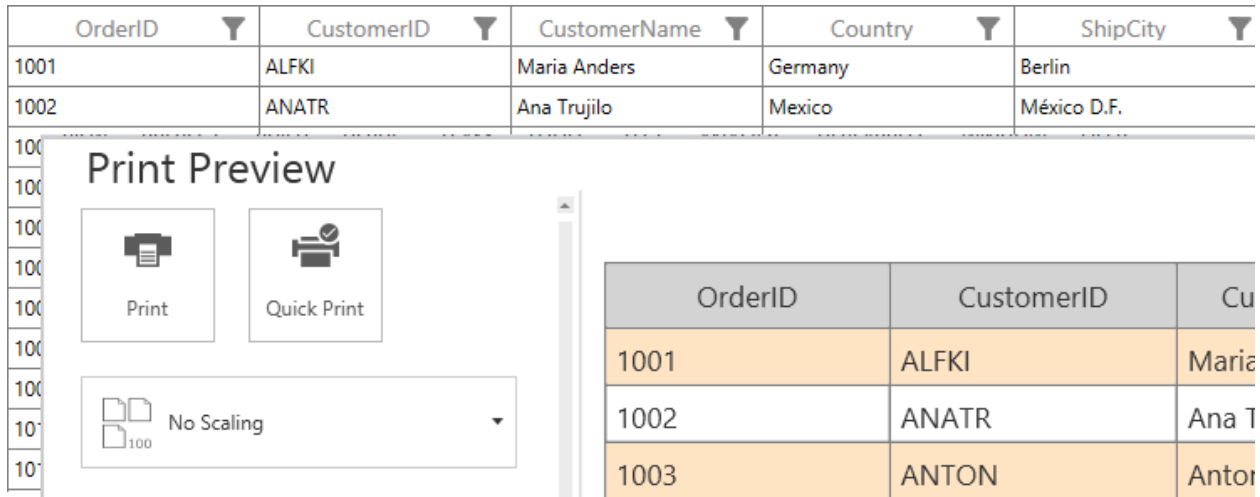
private class CustomPrintManager : GridPrintManager
{
    public CustomPrintManager(SfDataGrid grid)
        : base(grid)
    {
    }
    public override ContentControl GetPrintGridCell(object record, string
mappingName)
    {
        var index = dataGrid.View.Records.IndexOfRecord(record);
        if (index % 2 == 0)
            return new PrintGridCell() { Background = new SolidColorBrush(Colors.Bisque)
};
    }
}

```

```
return base.GetPrintGridCell(record, mappingName);
}
}
```

**C#**

```
dataGrid.PrintSettings.PrintManagerBase = new
CustomPrintManager(this.dataGrid);
dataGrid.PrintSettings.AllowPrintByDrawing = false;
dataGrid.ShowPrintPreview();
```

*Styling Columns when AllowPrintByDrawing enabled*

You can apply column styles based on some conditions by overriding [OnRenderCell](#), [GetFormattedText](#) methods in [GridPrintManager](#).

**C#**

```
private class CustomPrintManager : GridPrintManager
{
    public CustomPrintManager(SfDataGrid grid)
    : base(grid)
    {
    }
    protected override void OnRenderCell(DrawingContext drawingContext, RowInfo
rowInfo, CellInfo cellInfo)
    {
        if (cellInfo.ColumnName == "OrderID" && rowInfo.Record != null)
        {
            var rect = new Rect((cellInfo.CellRect).X, (cellInfo.CellRect).Y + 0.5,
            (cellInfo.CellRect).Width, (cellInfo.CellRect).Height);
            drawingContext.DrawGeometry(new SolidColorBrush(Colors.LightGreen), new
Pen(), new RectangleGeometry(rect));
        }
        base.OnRenderCell(drawingContext, rowInfo, cellInfo);
    }
    protected override FormattedText GetFormattedText(RowInfo rowInfo, CellInfo
cellInfo, string cellValue)
    {
        var formattedText = base.GetFormattedText(rowInfo, cellInfo, cellValue);
    }
}
```

```

if (cellInfo.ColumnName == "OrderID" && rowInfo.Record != null)
formattedText.SetFontStyle(FontStyles.Italic);
return formattedText;
}
}

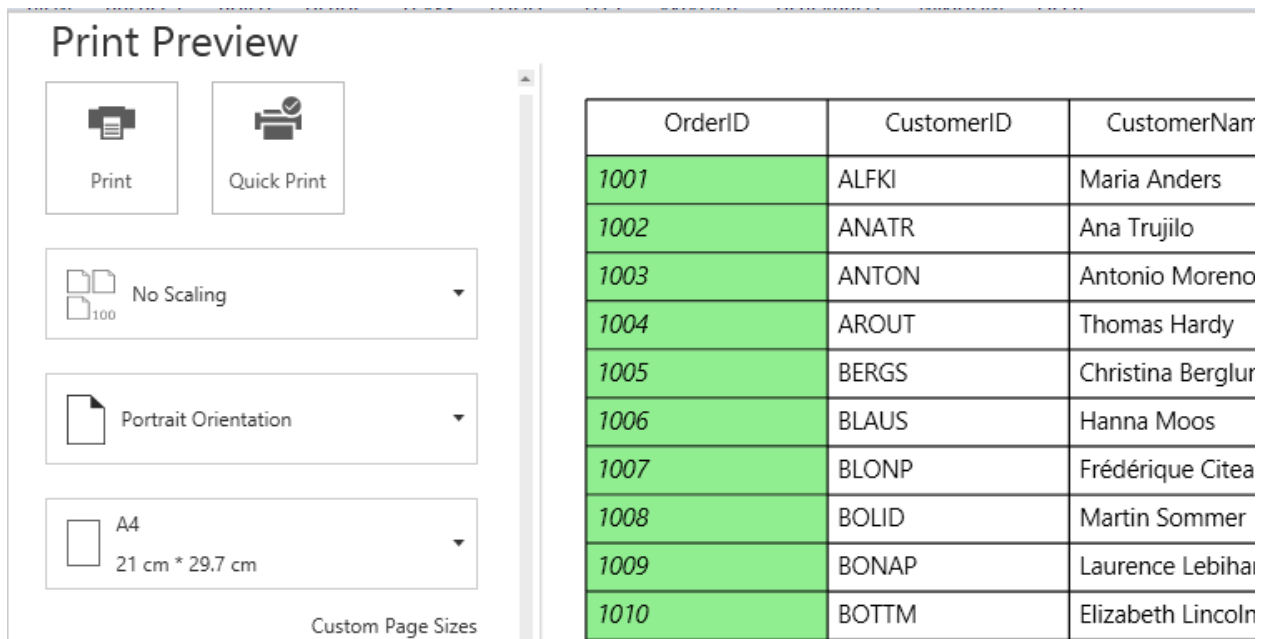
```

**C#**

```

dataGrid.PrintSettings.PrintManagerBase = new
CustomPrintManager(this.dataGrid);
dataGrid.ShowPrintPreview();

```



Here, OrderID column Font Style and Background are changed.

*Styling Columns when AllowPrintByDrawing disabled*

You can apply column styles based on custom logic by overriding [GetPrintGridCell](#) method in [GridPrintManager](#).

**C#**

```

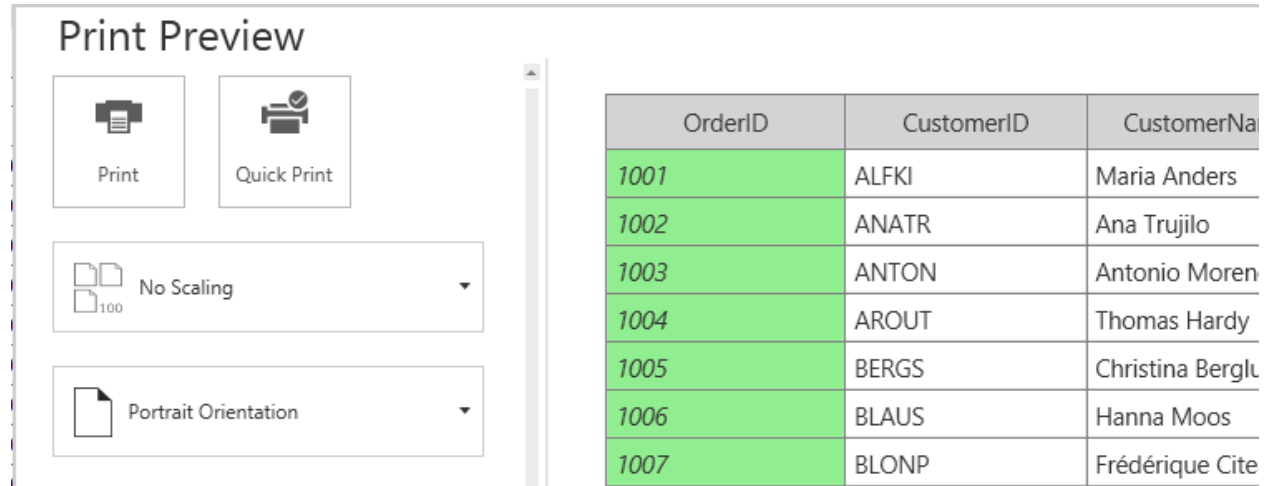
private class CustomPrintManager : GridPrintManager
{
public CustomPrintManager(SfDataGrid grid)
: base(grid)
{
}
public override ContentControl GetPrintGridCell(object record, string
mappingName)
{
if (mappingName == "OrderID")
return new PrintGridCell() { Background = new
SolidColorBrush(Colors.LightGreen), FontStyle = FontStyles.Italic };
return base.GetPrintGridCell(record, mappingName);
}
}

```

```
}
}
```

**C#**

```
dataGrid.PrintSettings.PrintManagerBase = new
CustomPrintManager(this.dataGrid);
dataGrid.PrintSettings.AllowPrintByDrawing = false;
dataGrid.ShowPrintPreview();
```



**Note:** [GetColumnWidth](#), [GetColumnTextWrapping](#), [GetColumnTextAlignment](#) and [GetColumnPadding](#) methods are also used for column customization while printing.

*Printing Selected rows*

Selected rows can be printed by overriding [GetSourceListForPrinting](#) method in [PrintManagerClass](#) class.

**C#**

```
private class CustomPrintManager : GridPrintManager
{
    public CustomPrintManager(SfDataGrid grid)
    : base(grid)
    {
    }
    protected override IList GetSourceListForPrinting()
    {
        List<object> selectedRecords = new List<object>();
        var selectedRows = dataGrid.SelectionController.SelectedRows.ToList();
        foreach (var row in selectedRows)
        {
            if (row.IsAddNewRow || (row.NodeEntry != null &&
                (!row.NodeEntry.IsRecords)))
                continue;
            if (row.IsUnBoundRow)
            {
                var _row = dataGrid.UnBoundRows.FirstOrDefault(r => r.Position ==
                    row.GridUnboundRowInfo.Position && r.ShowBelowSummary ==
```

```

row.GridUnboundRowInfo.ShowBelowSummary && r.RowIndex ==
row.GridUnboundRowInfo.RowIndex);
selectedRecords.Add(_row);
}
else
selectedRecords.Add(row.NodeEntry);
}
return selectedRecords;
}
}

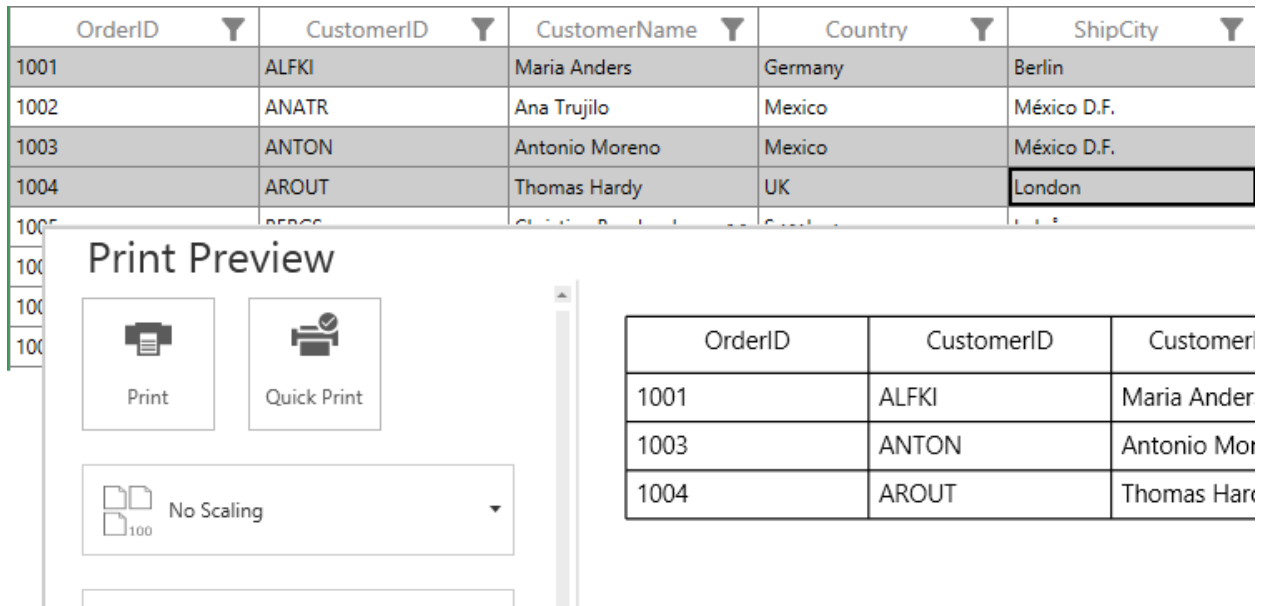
```

## C#

```

dataGrid.PrintSettings.PrintManagerBase = new
CustomPrintManager(this.dataGrid);
dataGrid.ShowPrintPreview();

```



### Creating custom PrintPreview window

You can create custom print preview window by adding [PrintPreviewAreaControl](#) to preview the view. [PrintManagerBase](#) will handle the printing operations and [PrintPreviewAreaControl](#) is responsible for preview.

Steps to create custom print preview window.

1. Add [PrintPreviewAreaControl](#) and required controls to print or customize the print settings.

## XML

```

<Grid>
<Grid Background="#FFF7F7F7">
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>

```



```
<Border Background="#FFDBDBDB" DataContext="{Binding
ElementName=PrintPreviewArea}">
<StackPanel Margin="10,5"
HorizontalAlignment="Center"
Orientation="Horizontal">
<Button Height="38"
Margin="10,3"
Command="{Binding PrintCommand}"
Content="Print"
Style="{StaticResource PrintButtonStyle}"
ToolTip="Print" />
</StackPanel>
</Border>
<syncfusion:PrintPreviewAreaControl x:Name="PrintPreviewArea" Grid.Row="1"
/>
</Grid>
</Grid>
```

2. Assign the instance of [GridPrintManager](#) to [PrintPreviewAreaControl.PrintManagerBase](#) property.

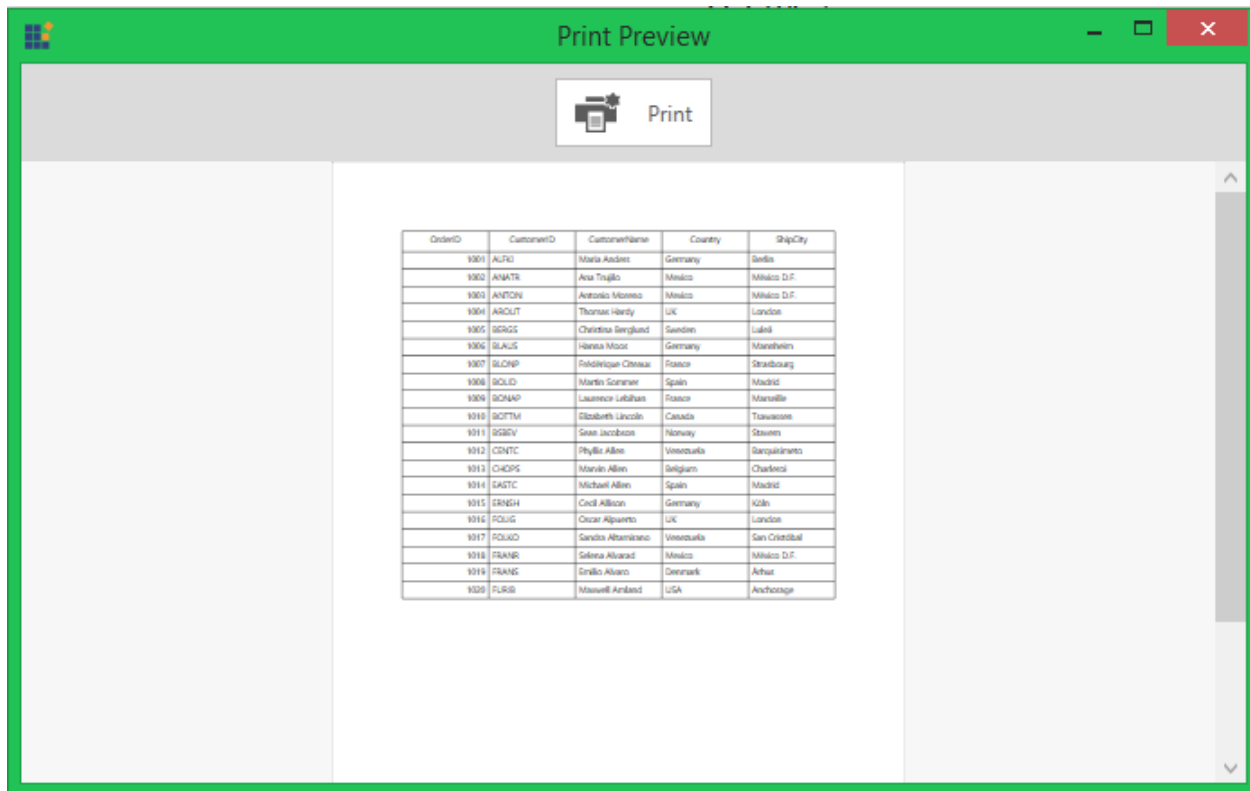
#### C#

```
var window = new PreviewWindow
{
    WindowStartupLocation = WindowStartupLocation.CenterScreen,
};
window.PrintPreviewArea.PrintManagerBase = new GridPrintManager(dataGrid);
window.ShowDialog();
```

3. You can customize print settings like scaling, Orientation at runtime by using [PrintPreviewAreaControl.PrintManagerBase](#) and it will be reflected in custom print preview window.

#### C#

```
this.PrintPreviewArea.PrintManagerBase.PrintScaleOption =
PrintScaleOptions.FitAllRowsonOnePage;
this.PrintPreviewArea.PrintManagerBase.Print();
```



You can get the sample for custom print preview [here](#).

## MVVM in WPF DataGrid (SfDataGrid)

### DataGrid SelectedItem Binding

You can bind the [SelectedItem](#) property directly to the DataGrid by setting the `SfDataGrid.SelectedItem` property.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    AutoGenerateColumns="False"
    SelectedItem="{Binding SelectedItem, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged}"
    ItemsSource="{Binding PersonDetails}"
    NavigationMode="Cell"
    ColumnSizer="Star"
    SelectionMode="Extended"
    ShowRowHeader="True">
```

Whenever the `SelectedItem` is changed, the `ViewModel` property will get notified.

### C#

```
public class PersonModel : INotifyPropertyChanged
{
    private object selectedItem;
    public object SelectedItem
    {
        get
```

```
{
    return selectedItem;
}
set
{
    selectedItem = value;
    RaisePropertyChanged("SelectedItem");
}
}
```

You can download the sample [here](#).

### DataGrid.SelectedItems Binding

You can bind the [SelectedItems](#) property directly to the DataGrid by setting the SfDataGrid.SelectedItems property.

### XML

```
<syncfusion:SfDataGrid x:Name="dataGrid"
    Grid.Row="0"
    AutoGenerateColumns="False"
    SelectedItems="{Binding
    SelectedItems,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"
    ItemsSource="{Binding PersonDetails}"
    NavigationMode="Cell"
    ColumnSizer="Star"
    SelectionMode="Extended"
    ShowRowHeader="True">
```

You can bind the [SelectedItems](#) from the [ViewModel](#) property.

### C#

```
class ViewModel:INotifyPropertyChanged
{
    private ObservableCollection<object> _selectedItems;
    public ObservableCollection<object> SelectedItems
    {
        get
        {
            return _selectedItems;
        }
        set
        {
            _selectedItems = value;
            RaisePropertyChanged("SelectedItem");
        }
    }
}
```

You can download the sample [here](#).

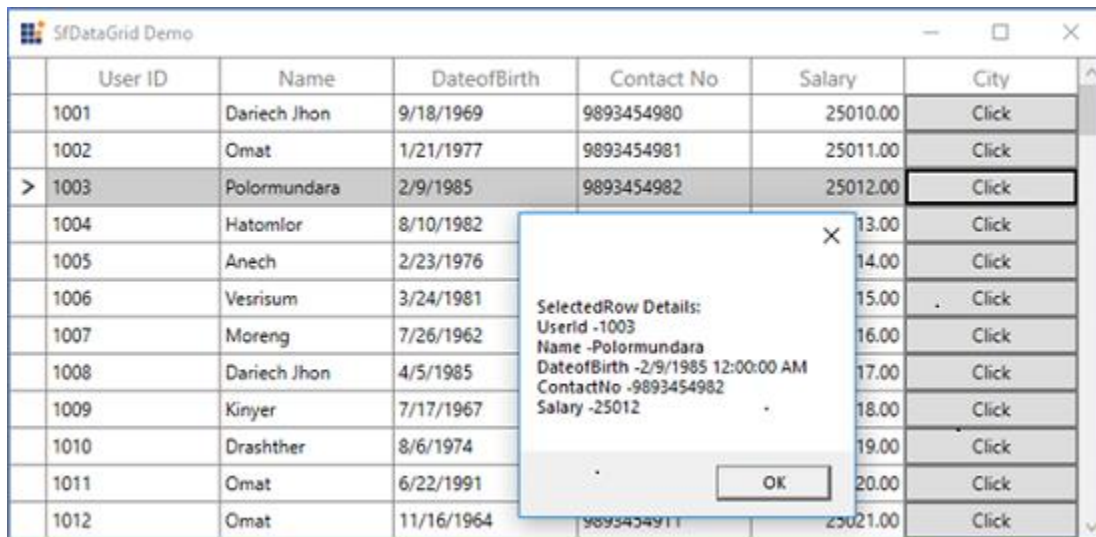
### Button command binding to ViewModel

You can load a button for the columns in the DataGrid by using [GridTemplateColumn](#). When loading the buttons, you can bind command in [ViewModel](#) by using [ElementName](#) binding.

In the following example, [ViewModel](#) command receives the underlying data object as command parameter since the [DataContext](#) is binding as command parameter.

### XML

```
<syncfusion:SfDataGrid x:Name="sfGrid"
Grid.Row="1"
ColumnSizer="Star"
AllowEditing="True"
AutoGenerateColumns="False"
ItemsSource="{Binding UserDetails}"
ShowRowHeader="True">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTemplateColumn MappingName="City"
syncfusion:FocusManagerHelper.WantsKeyInput="True">
<syncfusion:GridTemplateColumn.CellTemplate>
<DataTemplate>
<Button Content="Click" syncfusion:FocusManagerHelper.FocusedElement="True"
Command="{Binding Path=DataContext.RowDataCommand, ElementName=sfGrid}"
CommandParameter="{Binding}"/>
</DataTemplate>
</syncfusion:GridTemplateColumn.CellTemplate>
</syncfusion:GridTemplateColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>
```



You can download the sample [here](#).

### Binding ComboBoxColumn ItemsSource from ViewModel

You can bind the [ItemsSource](#) from [ViewModel](#) to [GridComboBoxColumn](#) or [GridMultiColumnDropDownList](#) by using the [ElementName](#) binding.

### XML

```
<syncfusion:SfDataGrid x:Name="DataGrid"
AutoGenerateColumns="False"
ItemsSource="{Binding OrderList}"
NavigationMode="Cell">
<syncfusion:GridComboBoxColumn AllowEditing="True"
MappingName="Country"
HeaderText="Country"
ItemsSource="{Binding Path=DataContext.CountryList,
ElementName=DataGrid}" />
</syncfusion:SfDataGrid>
```

## C#

```
class ViewModel:INotifyPropertyChanged
{
private ObservableCollection<string> countryList;
public ObservableCollection<string> CountryList
{
get
{
return countryList;
}
set
{
countryList = value;
RaisePropertyChanged("CountryList");
}
}
}
```



Id	Name	DateOfBirth	City	Country
1001	Hanna Moos	12/21/2016	ALFKI	UK
1002	Lie	1/17/2013	London	Canada
1003	Joseph	2/14/2015	BERGS	UK
1004	Thomas Hardy	12/21/2015	BOTTM	Sweden
1005	Thomas Hardy	9/1/2016	NY	America
1006	Laurence Lebihan	3/3/2016	ANTON	Canada
1007	Elizabeth Lincoln	12/6/2015	Berlin	Italy
1008	Martin Sommer	1/23/2012	CA	France
1009	Maria Anders	12/2/2013	Madrid	German
				Mexico

You can download the sample [here](#).

Binding ViewModel ItemsSource to ComboBox inside data template

You can load the **ComboBox** inside the **GridTemplateColumn** and bind the **ItemsSource** from **ViewModel** to **ComboBox** by using the **ElementName** binding.

## XML

```
<syncfusion:SfDataGrid x:Name="DataGrid"
```

```

AutoGenerateColumns="False"
ItemsSource="{Binding OrderList}"
NavigationMode="Cell">
<syncfusion:SfDataGrid.Columns>
<syncfusion:GridTemplateColumn MappingName="Country"
syncfusion:FocusManagerHelper.WantsKeyInput="True">
<syncfusion:GridTemplateColumn.CellTemplate>
<DataTemplate>
<TextBlock Text="{Binding Country}"/>
</DataTemplate>
</syncfusion:GridTemplateColumn.CellTemplate>
<syncfusion:GridTemplateColumn.EditTemplate>
<DataTemplate>
<ComboBox ItemsSource="{Binding Path=DataContext.CountryList,
ElementName=DataGrid}" DisplayMemberPath="{Binding CountryList}"/>
</DataTemplate>
</syncfusion:GridTemplateColumn.EditTemplate>
</syncfusion:GridTemplateColumn>
</syncfusion:SfDataGrid.Columns>
</syncfusion:SfDataGrid>

```



Id	Name	DateOfBirth	City	Country
1001	Hanna Moos	12/21/2016	ALFKI	UK
1002	Lie	1/17/2013	London	Sweden
1003	Joseph	2/14/2015	BERGS	UK
1004	Thomas Hardy	12/21/2015	BOTTM	Sweden
1005	Thomas Hardy	9/1/2016	NY	America
1006	Laurence Lebihan	3/3/2016	ANTON	Canada
1007	Elizabeth Lincoln	12/6/2015	Berlin	Italy
1008	Martin Sommer	1/23/2012	CA	France
1009	Maria Anders	12/2/2013	Madrid	German
				Mexico

You can download the sample [here](#).

#### Binding DataGrid Columns from ViewModel

You can bind the `SfDataGrid.Columns` to a property in the `ViewModel` by having the `binding` property of type `Syncfusion.SfGrid.UI.Xaml.Grid.Columns`. Thus, you can set binding to the `SfDataGrid.Columns` property that provides `DataContext` of the DataGrid is `ViewModel`.

#### XML

```

<Syncfusion:SfDataGrid x:Name="datagrid"
AutoGenerateColumns="False"
Columns="{Binding SfGridColumns, Mode=TwoWay}"
ItemsSource="{Binding ItemsCollection}"
ShowRowHeader="True" />

```

Refer to the following code example in which the `SfGridColumns` is populated with some `GridTextColumn` when creating the `ViewModel` instance.

**C#**

```
public class ViewModel
{
    private Columns sfGridColumnns;
    public ViewModel()
    {
        SetSfGridColumnns();
        this.ItemsCollection = employeeDetails;
    }
    public ObservableCollection<BusinessObjects> ItemsCollection
    {
        get { return itemsCollection; }
        set { itemsCollection = value; }
    }
    public Columns SfGridColumnns
    {
        get { return sfGridColumnns; }
        set { this.sfGridColumnns = value; }
    }
    /// <summary>
    /// To generate the columns for SfDataGrid
    /// </summary>
    protected void SetSfGridColumnns()
    {
        this.sfGridColumnns = new Columns();
        sfGridColumnns.Add(new GridTextColumn() { MappingName = "EmployeeName" });
        sfGridColumnns.Add(new GridTextColumn() { MappingName = "EmployeeAge" });
        sfGridColumnns.Add(new GridTextColumn() { MappingName = "EmployeeSalary" });
        sfGridColumnns.Add(new GridTextColumn() { MappingName = "ExperienceInMonth"
    });
    }
}
```

You can download the sample [here](#).

**Localization in WPF DataGrid (SfDataGrid)**

Localization is the process of translating the application resources into different language for the specific cultures. You can localize the SfDataGrid by adding [resource]

(<https://msdn.microsoft.com/library/aa992030.aspx>) file. Application culture can be changed by setting [CurrentUICulture] (<https://msdn.microsoft.com/en-us/library/system.globalization.cultureinfo.currentuiculture.aspx>) before **InitializeComponent()** method.

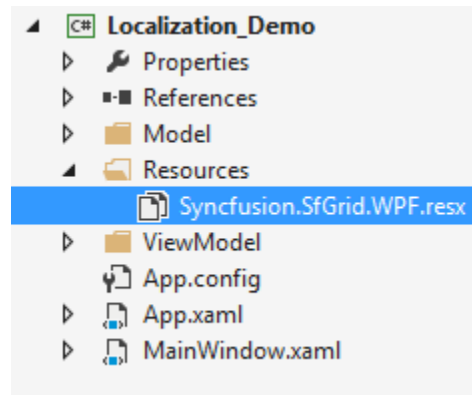
Below application culture changed to German.

**C#**

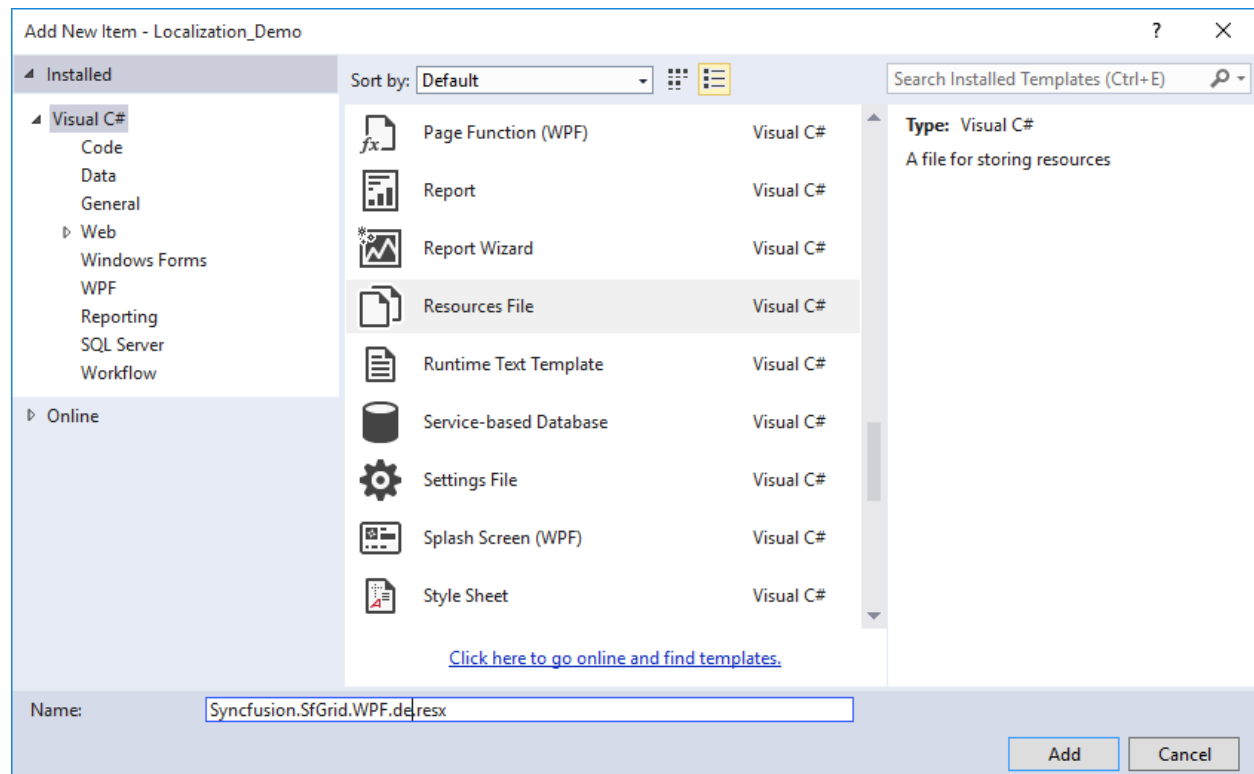
```
public MainWindow()
{
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("de-DE");
    InitializeComponent();
}
```

To localize the SfDataGrid based on `CurrentUICulture` using resource files, follow the below steps.

1. Create new folder and named as **Resources** in your application. 2. Add the default resource file of SfDataGrid into **Resources** folder. You can download the Syncfusion.SfGrid.WPF.resx [here](#).

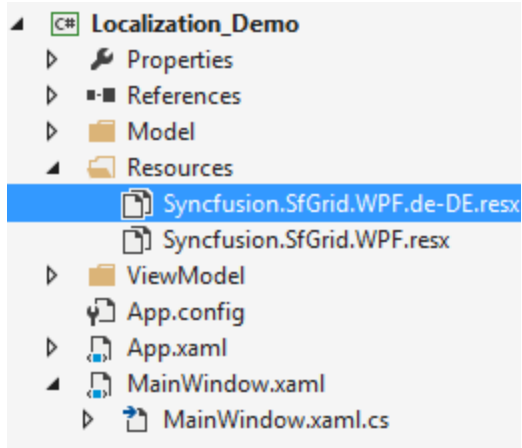


3. Right-click on the Resources folder, select **Add** and then **NewItem**. 4. In Add New Item wizard, select the **Resource File** option and name the filename as **Syncfusion.SfGrid.WPF.<culture name>.resx**. For example, you have to give name as **Syncfusion.SfGrid.WPF.de.resx** for German culture. 5. The culture name that indicates the name of language and country.



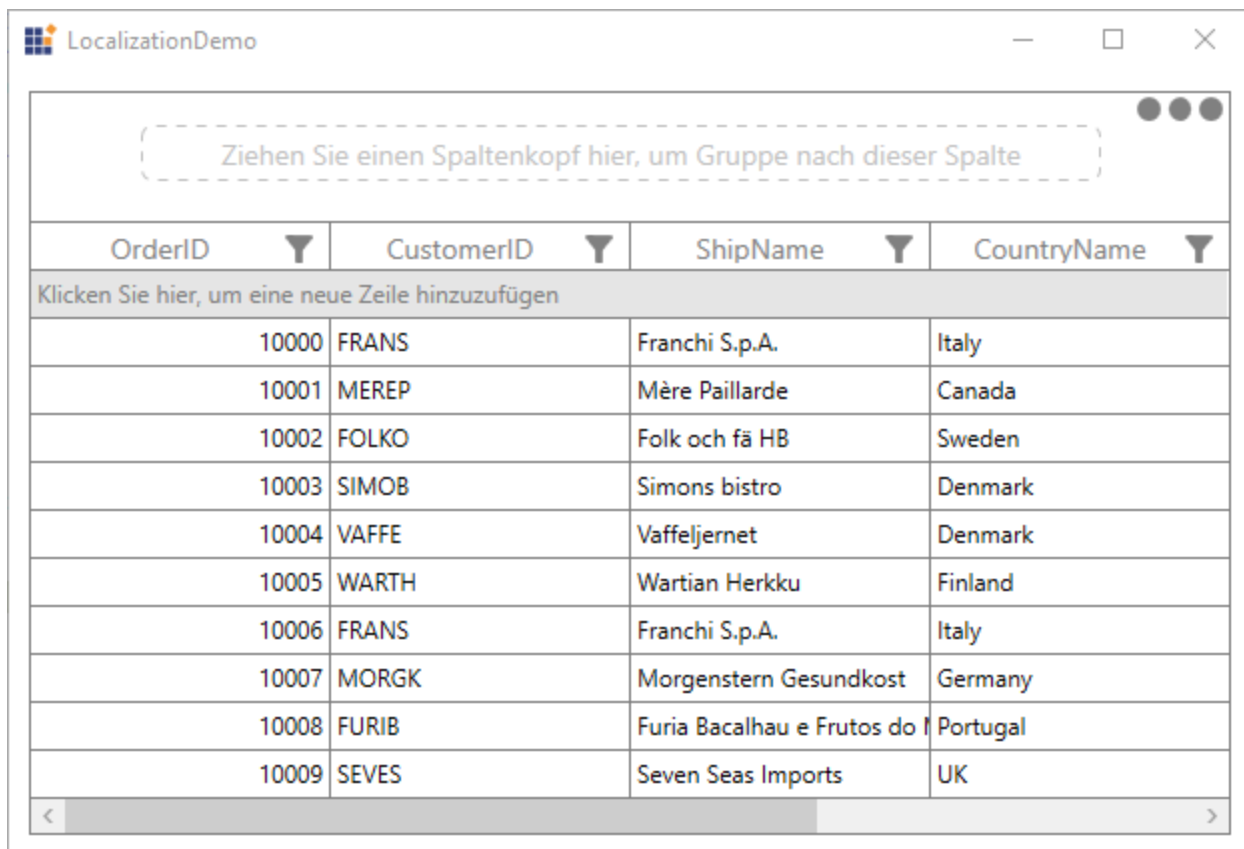
6. Now, select **Add** option to add the resource file in **Resources** folder.





7. Add the Name/Value pair in Resource Designer of **Syncfusion.SfGrid.WPF.de.resx** file and change its corresponding value to corresponding culture.

Name	Value	Comment
AddNewRowText	Click here to add a new row	Add new row water mark text.
AdvancedFiltersButtonText	Advanced Filters	Text for Advanced Filters button
After	After	Text for After
AfterOrEqual	After Or Equal	Text for AfterOrEqual
AND	And	Text for AND Operator
Before	Before	Text for Before
BeforeOrEqual	Before Or Equal	Text for BeforeOrEqual
BeginsWith	Begins With	Text for BeginsWith
Blanks	(Blanks)	Text for Blank filters
Cancel	Cancel	Text for Cancel
ClearFilter	Clear Filter	Text for Clear filters
Close	Close	Tooltip text for Close
ColumnChooserTitle	Column Chooser	Column Chooser Window Title
ColumnChooserWaterMark	(No Fields Available)	Empty Column Chooser Water Mark Text
Contains	Contains	Text for Contains
DateFilters	Date Filters	Text for DateFilters
Done	Done	Text for Done button
Empty	Empty	Text for Empty
EndsWith	Ends With	Text for EndsWith
EnterValidFilterValue	Enter Valid Filter Value	Text for EnterValidFilterValue
Equalss	Equals	Text for Equals
GreaterThan	Greater Than	Text for GreaterThan



You can get the sample from [here](#)

Localize when the resource file present in different assembly or different namespace?

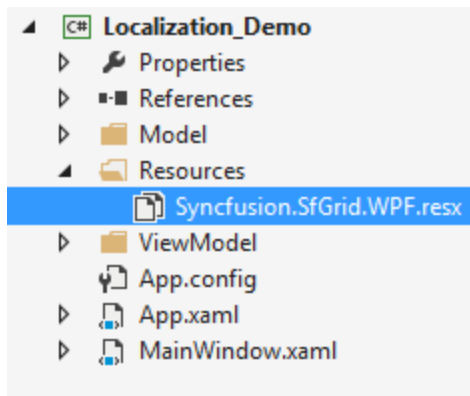
By default, SfDataGrid try to read the resource file from executing assembly and its default namespace by using [Assembly.GetExecuteAssembly](#) method. When the resource file is located at different assembly or namespace, then you can let SfDataGrid know by using [GridResourceWrapper.SetResources](#) method.

### C#

```
public MainWindow()
{
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("de-DE");
    Syncfusion.UI.Xaml.Grid.GridResourceWrapper.SetResources("Assembly_name",
    "namespace_name");
    InitializeComponent();
}
```

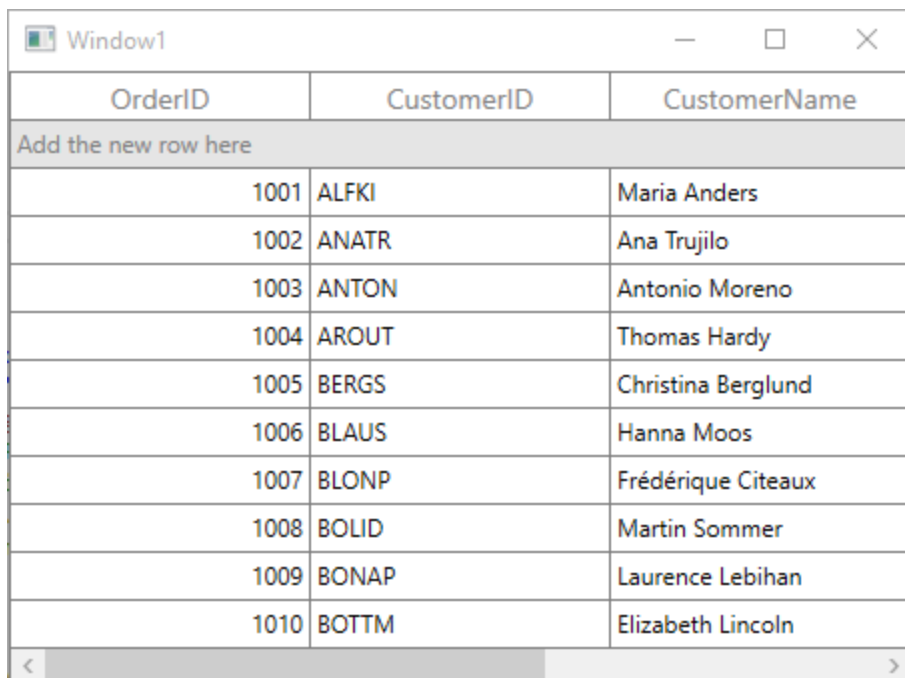
### Editing default culture resource

You can edit default resource file by adding it to **Resources** folder of your application where SfDataGrid reads the static texts from here. You can download the default resource file from [here](#).



Now, change the Name/Value pair in Resource Designer of **Syncfusion.SfGrid.WPF.resx** file.

Name	Value	Comment
AddNewRowText	Click here to add a new row	Add new row water mark text.
AdvancedFiltersButtonText	Advanced Filters	Text for Advanced Filters button
After	After	Text for After
AfterOrEqual	After Or Equal	Text for AfterOrEqual
AND	And	Text for AND Operator
Before	Before	Text for Before
BeforeOrEqual	Before Or Equal	Text for BeforeOrEqual
BeginsWith	Begins With	Text for BeginsWith
Blanks	(Blanks)	Text for Blank filters
Cancel	Cancel	Text for Cancel
ClearFilter	Clear Filter	Text for Clear filters
Close	Close	Tooltip text for Close
ColumnChooserTitle	Column Chooser	Column Chooser Window Title
ColumnChooserWaterMark	(No Fields Available)	Empty Column Chooser Water Mark Text
Contains	Contains	Text for Contains
DateFilters	Date Filters	Text for DateFilters
Done	Done	Text for Done button
Empty	Empty	Text for Empty
EndsWith	Ends With	Text for EndsWith
EnterValidFilterValue	Enter Valid Filter Value	Text for EnterValidFilterValue
Equalss	Equals	Text for Equals
Greater Than	Greater Than	Text for GreaterThan



OrderID	CustomerID	CustomerName
Add the new row here		
1001	ALFKI	Maria Anders
1002	ANATR	Ana Trujilo
1003	ANTON	Antonio Moreno
1004	AROUT	Thomas Hardy
1005	BERGS	Christina Berglund
1006	BLAUS	Hanna Moos
1007	BLONP	Frédérique Citeaux
1008	BOLID	Martin Sommer
1009	BONAP	Laurence Lebihan
1010	BOTTM	Elizabeth Lincoln

### UI Automation in WPF DataGrid (SfDataGrid)

Microsoft UI Automation is the new accessibility Framework for Microsoft Windows, available on all operating systems that support Windows Presentation Foundation (WPF). UI Automation provides accessibility to most UI elements and it provides the information about UI element to the end user. You can interact with the UI by using automated test scripts. To know more about UI Automation, refer the MSDN page [here](#).

SfDataGrid supports the following types of UI Automation,

1. Coded UI
2. Quick Test Professional

### Coded UI Test

Automated tests that drive your application through its user interface (UI) are known as Coded UI Tests (CUITs). These tests include functional testing of the UI controls. SfDataGrid supports CUITs Coded UI automation that helps you create automated tests for inner elements and records the sequence of actions. While dragging the crosshair that is shown in CodedUITestBuilder, on UI elements, it shows the properties of the respective UI elements and you can also add assertion for each of the properties.

[SfDataPager](#) and [SfMultiColumnDropDownControl](#) support Coded UI Test automation.

Provided here are three levels of support in Coded UI Test automation.

### Coded UI Test

Levels	Description
Level 1	Record and Detect the UI Elements when you perform any actions in the Control.
Level 2	Provide custom properties for UI elements when you drag the Crosshair to any UI element.

Level “ 3	Coded UI Test Builder generates code from recorded session and custom class is implemented to access custom properties, so the generated code is simplified.
-----------	--

### Requirements and Configuration

Coded UI provides support only in Visual Studio Ultimate and Visual Studio Premium. For more information about the platforms and configurations that are supported by Coded UI tests, refer this [link](#).

To test SfDataGrid with CUITs, build the Extension Project and place it in the mentioned location. You can get the Extension Project of SfDataGrid from [this](#) location.

1. Open the Extension Project and build it.
2. You can get the `Syncfusion.VisualStudio.TestTools.UITest.SfGridExtension.dll` from bin folder.

The above assembly must be placed into the following directory based on your Visual Studio version.

For **Visual Studio 2010** : C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\10.0\UITestExtensionPackages

For **Visual Studio 2012** : C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\11.0\UITestExtensionPackages

For **Visual Studio 2013** : C:\Program Files (x86)\Common Files\Microsoft Shared\VSTT\12.0\UITestExtensionPackages

**Note:** `Syncfusion.VisualStudio.TestTools.UITest.SfGridExtension.dll` need to be installed in GAC location. Please refer the MSDN link for [GAC](#) installation.

### Getting Started

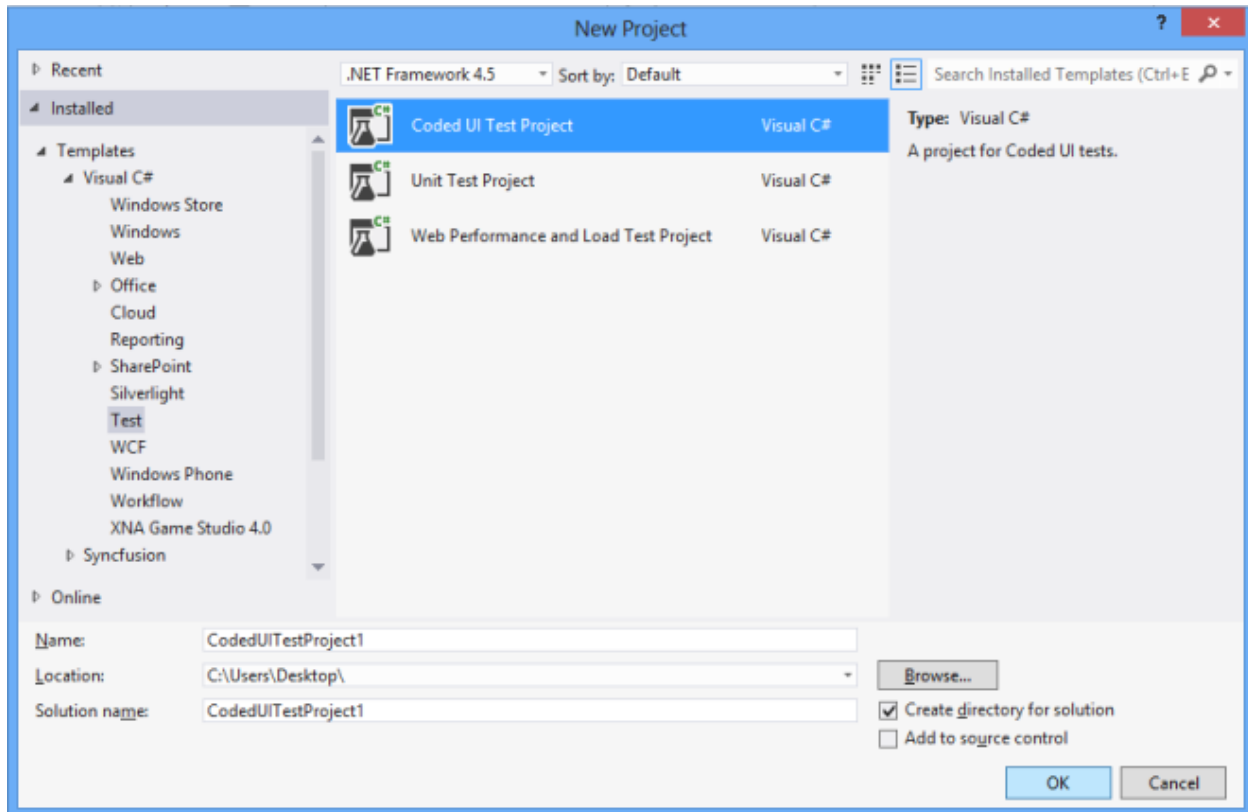
This topic shows you how to create a CodedUITest project and test the SfDataGrid application.

1. Create a new WPF application or open an existing WPF application with SfDataGrid and enable Coded UI Test in SfDataGrid. To enable CUITs, you need to set [AutomationPeerHelper.EnableCodedUI](#) as `True` and access the [AutomationPeerHelper](#) class from [Syncfusion.UI.Xaml.Grid](#) namespace as shown in the following code example,

### C#

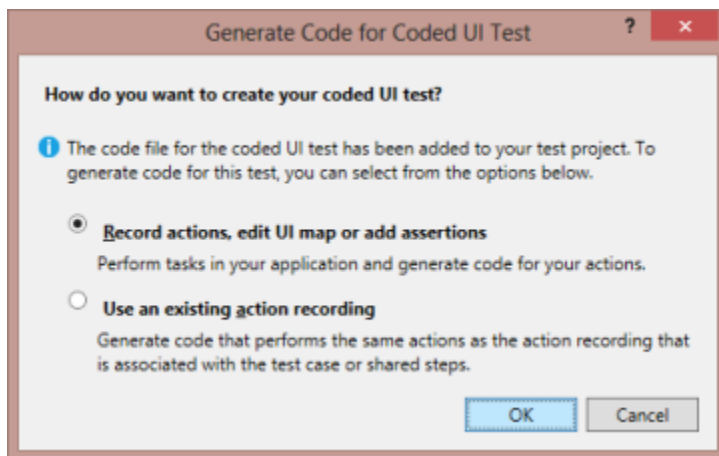
```
using Syncfusion.UI.Xaml.Grid;
public MainWindow ()
{
    InitializeComponent();
    AutomationPeerHelper.EnableCodedUI = true;
}
```

2. Build the application and launch the .exe file from the bin folder.
3. Create a Coded UI Test Project as shown in the following screenshot.



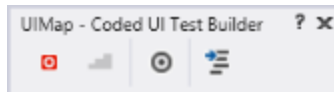
#### Add New Project

4. After you create a new Coded UI project, a CUIT file is added automatically and the Generate Code dialog box appears. In this, choose Record actions, edit UI map or add assertions.



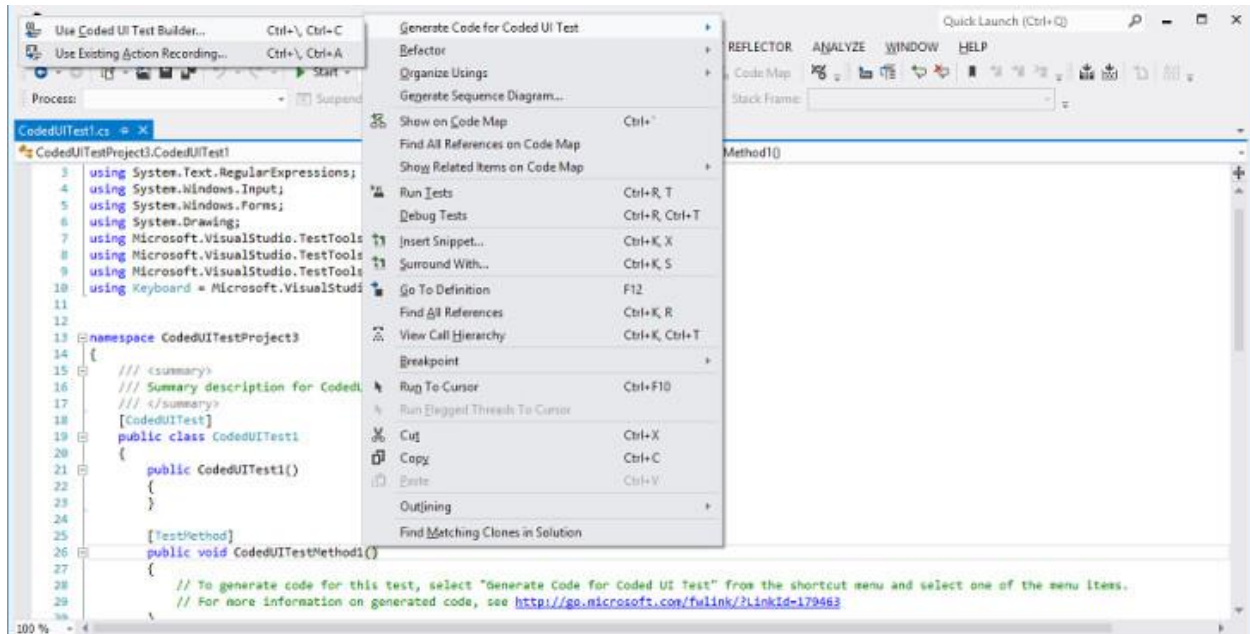
#### Generate Code for Coded UI Test

5. Now the Coded UI project Visual Studio gets minimized and CodedUITestBuilder appears in the bottom right corner of your window. You can record the actions by clicking Start Recording in CodedUITestBuilder.



## CodedUITestBuilder

- You can also open the CodedUITestBuilder from existing Coded UI project by right clicking on the CodedUITestMethod1 in CUIT file and clicking the Generate Code For Coded UI Test as shown in the following screenshot. You can see the same CodedUITestBuilder in the bottom right corner of the window.



## CodedUITestMethod

- Now you can drag the Crosshairs on to the UI elements of your WPF SfDataGrid application and it shows the available properties of the inner UI elements in SfDataGrid.
- You can record the actions made on UI elements by clicking Record button on the CodedUITest builder. For example you can record the action of changing the cell value in SfDataGrid. Click the Pause button to finish the record.

Order ID	Customer ID	Quantity	ProductName	UnitPrice	Discount	Ord
10000	FRANS	27	Alice Mutton	\$4	4.45 %	5/10/1991
10001	MEREP	10	NuNuCa Nuß-Nougat-Cre	\$30	79.45 %	5/13/1991
10001	MEREP	13	Boston Crab Meat	\$40	79.45 %	5/13/1991
10001	MEREP	39	Raclette Courdavault	\$8	79.45 %	5/13/1991
10001	MEREP	50	Wimmers gute Semmelkn	\$15	79.45 %	5/13/1991
10002	FOLKO	8	Gorgonzola Telino	\$35	36.18 %	5/14/1991
10002	FOLKO	13	Chartreuse verte	\$18	36.18 %	5/14/1991
10002	FOLKO	15	Flatemysost	\$15	36.18 %	5/14/1991
10003	SIMOB	44	Carnarvon Tigers	\$12	18.59 %	5/15/1991
10004	VAFFE	86	Thüringer Rostbratwurst	\$35	20.12 %	5/16/1991
10004	VAFFE	31	Veggie-spread	\$6	20.12 %	5/16/1991
10005	WARTH	34	Tarte au sucre	\$6	4.13 %	5/20/1991
10006	FRANS	4	Konbu	\$10	3.62 %	5/21/1991
10006	FRANS	11	Valkoinen suklaa	\$4	3.62 %	5/21/1991
10007	MORGK	26	Queso Manchego La Pastc	\$4	36.19 %	5/22/1991
10007	MORGK	23	Perth Pasties	\$30		
10007	MORGK	31	Veggie-spread	\$20		
10008	FURIB	16	Tofu	\$20		

### CodedUITest

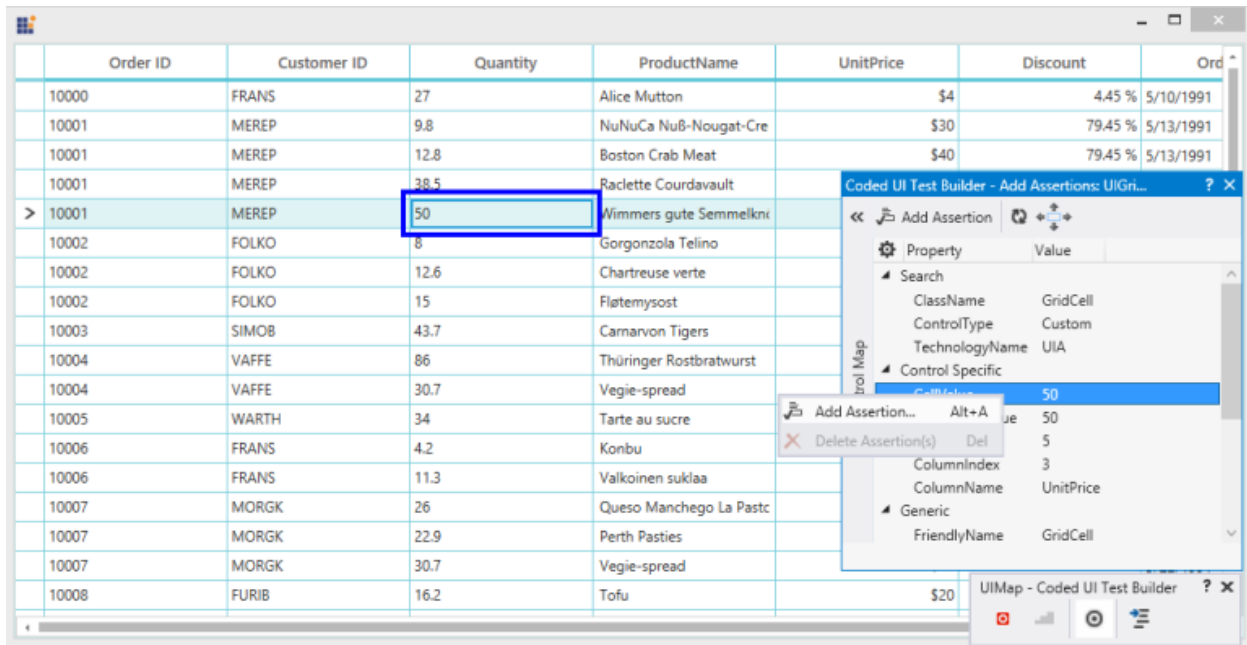
- Once the record is completed, click the GenerateCode icon in CodedUITestBuilder for generate a test method. Then close the CodedUITestBuilder and you can see the generated code for cell value changed action as follows.

### C#

```
public void RecordedMethod1 ()
{
    #region Variable Declarations
    WpfText uIMEREPTText =
        this.UIWpfWindow.UISfDataGridCustom.UIGridCellCustom4.UIMEREPTText;
    WpfEdit uIGridCellEdit = this.UIWpfWindow.UISfDataGridCustom.UIGridCellEdit;
    WpfSfGridCell uIGridCellCustom11 =
        this.UIWpfWindow.UISfDataGridCustom.UIGridCellCustom11;
    WpfSfGridCell uIGridCellCustom12 =
        this.UIWpfWindow.UISfDataGridCustom.UIGridCellCustom12;
    #endregion
}
```

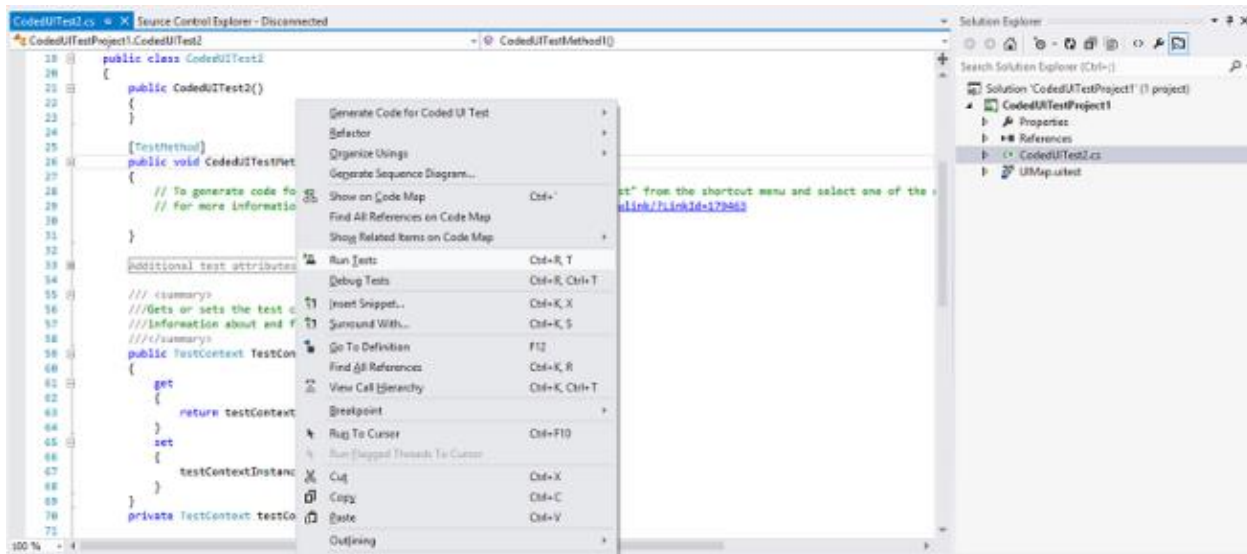
- You can also create an assertion to check the modified cell value. Drag the crosshair to the modified cell, and the Assertion window appears. The properties for control (Cell) is now listed in the Assertion dialog box. You can add assertion by clicking the Generate Code button in CodedUITestBuilder.





Assertion window

11. After all tests and assertion are created, right-click on the Test method and click Run Tests to run the test as follows.



Run Test

### Tables for Properties

The following properties are for each of the UI elements in SfDataGrid.

UI Elements	Properties
DataGrid	RowCountColumnCount SelectionMode SelectionUnit SelectedIndex SelectedItemCount

GridCell	CellValue FormattedValue RowIndex ColumnIndex ColumnName
GridHeaderCellControl	MappingName IsFilterApplied FilterIconVisibility SortDirection SortNumberVisibility
GridRowHeaderCell	RowErrorMessage RowIndexState
GroupDropArea	IsExpanded GroupDropAreaText
DetailsViewDataGrid	RowCountColumnCount SelectionMode SelectionUnit SelectedIndex SelectedItemCount
GroupDropAreaItem	GroupName SortDirection
GridStackedHeaderCellControl	Default Properties
GridTableSummaryCell	Default Properties
GridGroupSummaryCell	Default Properties
GridCaptionSummaryCell	Default Properties
GridDetailsViewExpanderCellControl	Default Properties
GridIndentCell	Default Properties

The following properties are for each of the UI elements in **SfMultiColumnDropDownControl**.

UI Elements	Properties
SfMultiColumnDropDownControl	AllowAutoComplete AllowNullInput AllowImmediatePopup AllowIncrementalFiltering AllowCaseSensitiveFiltering AllowSpinOnMouseWheel DisplayMember IsDropDownOpen SelectedIndex ValueMember

The following properties are for each of the UI elements in **SfDataPager**.

UI Elements	Properties
SfDataPager	AccentBackground AccentForeground AutoEllipsisMode AutoEllipsisText DisplayMode EnableGridPaging NumericButtonCount Orientation PageCount PageSize UseOnDemandPaging

### Limitations

- SfDataGrid UI Automation will not work when you automate the datagrid with editing in Visual Studio 2015. It is a known issue in Visual Studio 2015.

*How To**How to Enable Coded UI Test in SfDataGrid*

To enable Coded UI Test in SfDataGrid, set `AutomationPeerHelper.EnableCodedUI` as `True`. You can use the `AutomationPeerHelper` class from `Syncfusion.UI.Xaml.Grid` namespace.

**C#**

```
using Syncfusion.UI.Xaml.Grid;
public MainWindow()
{
    InitializeComponent();
    AutomationPeerHelper.EnableCodedUI = true;
}
```

## Quick Test Professional (QTP)

You can refer the UFT/QTP document from [here](#)

## Helpers in WPF DataGrid (SfDataGrid)

## IndexResolver

SfDataGrid has `GridIndexResolver` static class present in `Syncfusion.UI.Xaml.Grid` namespace that has some extension methods used to Resolve from row or column index to record or visible column index and vice versa.

Example: You can find the record index from row index using `ResolveToRecordIndex` method.

## Prototype Table

ProtoType	Description
<code>ResolveToRecordIndex(int rowIndex)</code>	Resolves Record index from the RowIndex. When RowIndex does not find any records it returns -1. RecordIndex denotes the index of Record in <code>SfDataGrid.View.Records</code>
<code>ResolveToRowIndex(int recordIndex)</code>	Resolves RowIndex from the record index associated with <code>SfDataGrid.View.Records</code> . When record index is lesser than 0 it returns the -1.
<code>ResolveToRowIndex(object recordItem)</code>	Resolves row index from the record associated with <code>SfDataGrid.View.Records</code> . When the given record is not available in the collection it returns -1.
<code>ResolveStartIndexofGroup(Group group)</code>	Resolves the start index of group in DataGrid associated with <code>SfDataGrid.View.Groups</code> . When there is no group in DataGrid it returns -1.
<code>ResolveToTableSummaryIndex(int rowIndex)</code>	Resolves <code>TableSummaryIndex</code> associated with <code>SfDataGrid.View.Records.TableSummaries</code> . When the row is not a <code>TableSummaryRow</code> , it returns -1.

ResolveToGridVisibleColumnIndex(int visibleColumnIndex)	Resolves the GridColumn index from the visible column index. It excludes the RowHeader and IndentColumn.
ResolveStartIndexBasedOnPosition()	Resolves the start index based on position. It includes the stacked header also. By default it returns 0.
ResolveToScrollColumnIndex(int gridColumnIndex)	Resolves column index from the Grid column index associated with SfDataGrid.Columns. It includes the IndentColumn and RowHeader also.
ResolveToStartColumnIndex()	Returns start column index of the VisibleColumn.
GetGridDetailsViewRowIndex(DetailsViewDataGrid detailsViewDataGrid)	Returns the RowIndex of DetailsViewGrid. You can pass the DetailsViewDataGrid as argument. When the DetailsView is not found it returns the - 1.
GetGridDetailsViewRecord(DetailsViewDataGrid detailsViewDataGrid)	Returns the DetailsView record. You can pass the DetailsViewDataGrid. When there is no item it returns null value.
GetTableSummaryCount(TableSummaryRowPosition position)	Returns the TableSummary count from TableSummaryRowPosition.Position.
GetHeaderIndex()	Returns the header row index.
IsInDetailsViewIndex(int rowIndex)	Decides whether the given row index is DetailsView index or not.
IsAddNewIndex(int rowIndex)	Decides whether the given row index is AddNewRow index or not.
IsTableSummaryIndex(int rowIndex)	Decides whether the given row index is TableSummary index or not.
IsHeaderTableSummaryRow(int rowIndex)	Decides whether the given row index is HeaderTableSummaryRow or not.

### Dispose

The method is associated with relinquishes memory and clears all references associated with SfDataGrid. When you call this method, it releases all the reference for SfDataGrid. So the memory it is occupying using the DataGrid is reclaimed. You have to call SfDataGrid.Dispose method to release the memory.

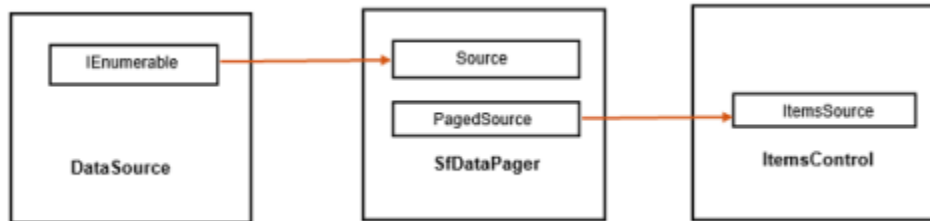
## SfDataPager

### WPF DataPager (SfDataPager) Overview

The SfDataPager control provides a configurable user interface for paging using a data collection. You can bind the SfDataPager to any IEnumerable <a>(http://msdn.microsoft.com/en-

us/library/system.collections.ienumerable(v=vs.95).aspx)collection</a>. The SfDataPager control wraps the collection internally in PagedCollectionView and exposes by using PagedSource property. PagedCollectionView helps to provide the paging functionality. You can apply paging for data bound control by setting PagedSource property as ItemsSource for that control.

The following screenshot displays the basic concept of paging.



### Getting Started with WPF DataPager (SfDataPager)

This section describes about the assembly that is required for SfDataPagerControl in your WPF application. The following assemblies are required in your application.

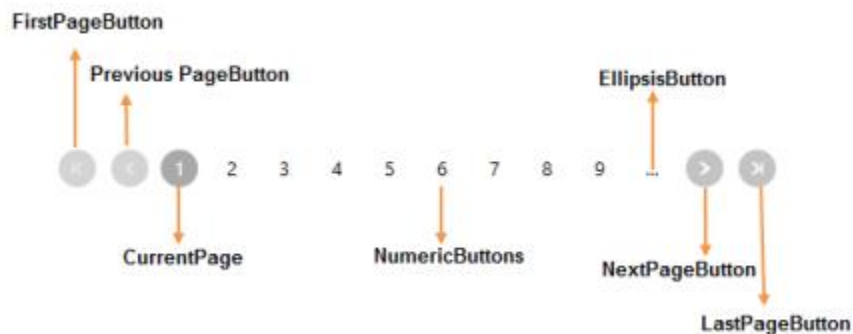
Demanded Assemblies	Description
Syncfusion.SfGrid.WPF	Covers the SfDataPager and other elements.

When you use SfDataPager in the SfDataGrid, You need to add the following assemblies with the default assemblies.

Demanded Assemblies	Description
Syncfusion.Data.WPF	Covers basic processing on data like sorting, grouping, and filtering.
Syncfusion.Shared.WPF	Covers an editors like CurrencyTextBox, PercentEdit, DateTimeEdit.

### Control Structure

The following screenshot describes the elements of the DataPager control.



- **FirstPageButton:** Moves the current page index to the first page and displays the first page data.

- PreviousPageButton: Moves the current page index to the previous page and displays the previous page data.
- LastPageButton: Moves the current page index to the last page and displays the last page data.
- NextPageButton: Moves the current page index to the next page and displays the next page data.
- NumericButtons: Denotes the available pages. You can directly navigate to the page by clicking the corresponding button.
- EllipsisButton: Displayed when AutoEllipsis mode is set. This button displays the next set of numeric page buttons are displayed.

### Create Simple Application with SfDataPager

The following steps help you to use the SfDataPager in an application:

1. Create a new WPF application
2. Open a VisualStudio toolbox. Navigate to SyncfusionControls and drag the SfDataPager to the design window.



3. When you drag the SfDataPager to the window, it automatically adds the required references to the current application. To add the SfDataPager using code, you can add the following assemblies to the project.

I. Syncfusion.Data.WPF

II. Syncfusion.SfGrid.WPF

4. You can either drag the control from Visual Studio or Expression Blend, or add the control to your project manually. You need to add the namespace to make use of SfDataPager in your application.

### XML

```
<Window x:Class="SfDataPagerDemo.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:SfDataPagerDemo"
```

```
xmlns:datapager="clr-
namespace:Syncfusion.UI.Xaml.Controls.DataPager;assembly=Syncfusion.SfGrid.W
PF"
mc:Ignorable="d"
Title="MainWindow" Height="350" Width="525">
<Grid>
<datapager:SfDataPager x:Name="sfDataPager"/>
</Grid>
</Window>
```

5.Create Business object class named as OrderInfo

### C#

```
public class OrderInfo
{
    int orderID;
    string customerId;
    string country;
    string customerName;
    string shippingCity;
    public int OrderID
    {
        get { return orderID; }
        set { orderID = value; }
    }
    public string CustomerID
    {
        get { return customerId; }
        set { customerId = value; }
    }
    public string CustomerName
    {
        get { return customerName; }
        set { customerName = value; }
    }
    public string Country
    {
        get { return country; }
        set { country = value; }
    }
    public string ShipCity
    {
        get { return shippingCity; }
        set { shippingCity = value; }
    }
    public OrderInfo(int orderId, string customerName, string country, string
customerId, string shipCity)
    {
        this.OrderID = orderId;
        this.CustomerName = customerName;
        this.Country = country;
        this.CustomerID = customerId;
        this.ShipCity = shipCity;
    }
}
```

6.Add the following code in ViewModel class

#### C#

```
public class ViewModel
{
    ObservableCollection<OrderInfo> orderCollection;
    public ObservableCollection<OrderInfo> OrderInfoCollection
    {
        get { return orderCollection; }
        set { orderCollection = value; }
    }
    public ViewModel()
    {
        orderCollection = new ObservableCollection<OrderInfo>();
        this.GenerateOrders();
    }
    private void GenerateOrders()
    {
        orderCollection.Add(new OrderInfo(1001, "Maria Anders", "Germany", "ALFKI", "Berlin"));
        orderCollection.Add(new OrderInfo(1002, "Ana Trujilo", "Mexico", "ANATR", "Mexico D.F.));
        orderCollection.Add(new OrderInfo(1003, "Antonio Moreno", "Mexico", "ANTON", "Mexico D.F.));
        orderCollection.Add(new OrderInfo(1004, "Thomas Hardy", "UK", "AROUT", "London"));
        orderCollection.Add(new OrderInfo(1005, "Christina Berglund", "Sweden", "BERGS", "Lula"));
        orderCollection.Add(new OrderInfo(1006, "Hanna Moos", "Germany", "BLAUS", "Mannheim"));
        orderCollection.Add(new OrderInfo(1007, "Frederique Citeaux", "France", "BLONP", "Strasbourg"));
        orderCollection.Add(new OrderInfo(1008, "Martin Sommer", "Spain", "BOLID", "Madrid"));
        orderCollection.Add(new OrderInfo(1009, "Laurence Lebihan", "France", "BONAP", "Marseille"));
        orderCollection.Add(new OrderInfo(1010, "Elizabeth Lincoln", "Canada", "BOTTM", "Tsawassen"));
    }
}
```

7.Set the ViewModel instance as DataContext to window. Now, you can bind the data collection to the SfDataPagerSource property.

#### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<datapager:SfDataPager x:Name="sfDataPager"
Grid.Row="1"
AccentBackground="DodgerBlue"
NumericButtonCount="5"
```



```

    PageSize="5"
    Source="{Binding OrderInfoCollection}" />
</Grid>

```

8. Then bind the PagedSource property of the SfDataPager control into the SfDataGridItemsSource property.

#### XML

```

<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<sfGrid:SfDataGrid AutoGenerateColumns="True"
ItemsSource="{Binding ElementName=sfDataPager, Path=PagedSource}" />
<datapager:SfDataPager x:Name="sfDataPager"
Grid.Row="1"
NumericButtonCount="10"
PageSize="10"
Source="{Binding OrdersDetails}" />
</Grid>

```

The following screenshot displays the output for Implementation of the SfDataPager in the SfDataGrid Control.

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå
1006	BLAUS	Hanna Moos	Germany	Mannheim
1007	BLONP	Frédérique Citeaux	France	Strasbourg
1008	BOLID	Martin Sommer	Spain	Madrid
1009	BONAP	Laurence Lebihan	France	Marseille
1010	BOTTM	Elizabeth Lincoln	Canada	Tsawassen

⏪
⏴
1
2
3
4
5
6
7
8
9
...
⏵
⏩

#### Theme

SfDataPager supports various built-in themes. Refer to the below links to apply themes for the SfDataPager,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	Mexico D.F.
1003	ANTON	Antonio Moreno	Mexico	Mexico D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Lula
1006	BLAUS	Hanna Moos	Germany	Mannheim

< < 1 > >

### DataBinding in WPF DataPager (SfDataPager)

Data binding is the master feature of SfDataPager. SfDataPager is bound to an external data source to display the data.

#### Source and PagedSource

SfDataPager exposes the Source property, where you can pass your collection of data for Paging. SfDataPager automatically wrap the collection in the PagedCollectionView and exposes to the PagedSource property. You can pass the PagedSource property to any ItemsControl's ItemsSource property.

The following code example explains how to use the Source and PagedSource property.

#### XML

```

<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<sfGrid:SfDataGrid AutoGenerateColumns="True"
ItemsSource="{Binding ElementName=sfDataPager, Path=PagedSource}" />
<datapager:SfDataPager x:Name="sfDataPager"
Grid.Row="1"
NumericButtonCount="10"
PageSize="16"
Source="{Binding OrdersDetails}" />
</Grid>

```

The following screenshot displays the output of the above code.

*PageIndex*

OrderID	CustomerID	EmployeeID	Ship Name	Ship Address	Ship City
10000	FRANS	6	Franchi S.p.A.	Via Monte Bianco 34	Torino
10001	MEREP	8	Mère Paillarde	43 rue St. Laurent	Montréal
10002	FOLKO	3	Folk och få HB	Åkergatan 24	Bräcke
10003	SIMOB	8	Simons bistro	Vinbæltet 34	København
10004	VAFFE	3	Vaffeljernet	Smagsløget 45	Århus
10005	WARTH	5	Wartian Herkku	Torikatu 38	Oulu
10006	FRANS	8	Franchi S.p.A.	Via Monte Bianco 34	Torino
10007	MORGK	4	Morgenstern Gesundkos	Heerstr. 22	Leipzig
10008	FURIB	3	Furia Bacalhau e Frutos	Jardim das rosas n. 32	Lisboa
10009	SEVES	8	Seven Seas Imports	90 Wadhurst Rd.	London
10010	SIMOB	8	Simons bistro	Vinbæltet 34	København
10011	WELLI	6	Wellington Importadora	Rua do Mercado, 12	Resende
10012	LINOD	6	LINO-Delicateses	Ave. 5 de Mayo Porlamar	I. de Margarita
10013	RICSU	3	Richter Supermarkt	Starenweg 5	Genève
10014	GROSR	4	GROSELLA-Restaurante	5ª Ave. Los Palos Grande	Caracas
10015	PICCO	6	Piccolo und mehr	Geichweg 14	Salzburg

SfDataPager exposes the PageIndex property. It contains the index of the currently selected page. You can use this property to set or get the current page of the SfDataPager.

**Events**

Event	Parameters	Description
PageIndexChanging	OldPageIndex NewPageIndex Cancel	Event is triggered when the current page index is changing. By using this event, you can cancel the page navigation operation by setting Cancel to true.
PageIndexChanged	OldPageIndex NewPageIndex	Event is triggered after the current page index is changed.

*How to bind the PageCollection to the other controls?*

SfDataPager automatically wraps the collection in PagedCollectionView and exposes to the PagedSource property. You can pass the PagedSource property to any ItemsControl's ItemsSource property. Here, the PagedSource property is bound to the ListBox.

The following code explains how to use Source and PagedSource property in ListBox.

**XML**

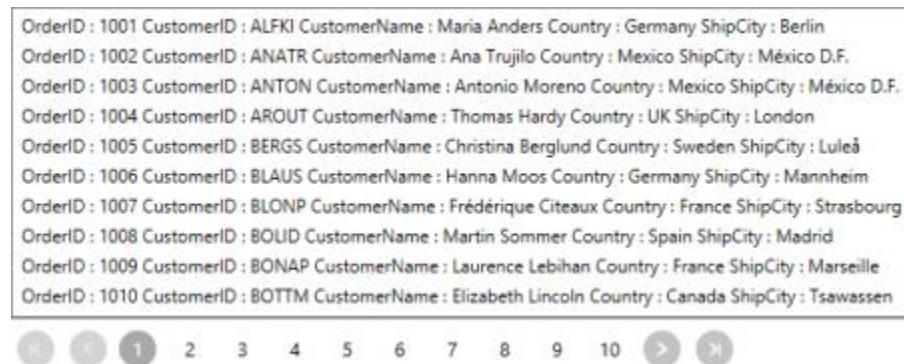
```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
```

```

<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<ListBox ItemsSource="{Binding ElementName=sfDataPager, Path=PagedSource}">
<ListBox.ItemTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="{Binding Data.OrderID}" Grid.Column="0" />
<TextBlock Text="{Binding Data.CustomerName}" Grid.Column="1" />
<TextBlock Text="{Binding Data.Country}" Grid.Column="2" />
<TextBlock Text="{Binding Data.CustomerID}" Grid.Column="3" />
<TextBlock Text="{Binding Data.ShipCity}" Grid.Column="4" />
</Grid>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
<datapager:SfDataPager x:Name="sfDataPager"
Grid.Row="1"
NumericButtonCount="10"
PageSize="10"
Source="{Binding OrdersDetails}" />
</Grid>

```

The following screenshot displays the output for ListView bound with PagedCollection.



### On DemandPaging

In normal paging, the entire data collection is initially loaded into the SfDataPager control. However, the SfDataPager control allows you to load the data for the current page dynamically. To enable on demand paging, set UseOnDemandPaging to true.

OnDemandPaging can be achieved by using the OnDemandLoading event and LoadDynamicItems method.

The OnDemandLoading event is triggered when the pager moves to the corresponding page. The OnDemandLoading event contains the following event arguments,

- StartIndex: Corresponding page start index.
- PageSize: Number of items to be load for that page.

In an OnDemandLoading event, you can use the LoadDynamicItems method to load the data for the corresponding page.

---

**Note:** In OnDemandPaging, you cannot assign a value for the Source property.

---

The following steps help you to achieve an OnDemandLoading with the SfDataPager control.

1. Set the UseOnDemandPaging property to true.
2. Set the PageCount value for the SfDataPager control.

### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<syncfusion:SfDataGrid x:Name="dataGrid"
AllowResizingColumns="True"
ColumnSizer="Star"
ItemsSource="{Binding Path=PagedSource,ElementName=sfDataPager}" />
<datapager:SfDataPager x:Name="sfDataPager"
OnDemandLoading="OnDemandDataLoading"
PageCount="50"
PageSize="16"
UseOnDemandPaging="True" />
</Grid>
```

3. Wire up the OnDemandLoading event of SfDataPager.

### C#

```
void OnDemandLoading(object sender, OnDemandLoadingEventArgs args)
{
    AssociatedObject.sfDataPager.LoadDynamicItems(args.StartIndex, source.Skip(ar
gs.StartIndex).Take(args.PageSize));
}
```

The following screenshot displays the output for OnDemandPaging,

OrderID	CustomerID	EmployeeID	Ship Name	Ship Address	Ship City
10000	FRANS	6	Franchi S.p.A.	Via Monte Bianco 34	Torino
10001	MEREP	8	Mère Paillarde	43 rue St. Laurent	Montréal
10002	FOLKO	3	Folk och få HB	Åkergatan 24	Bräcke
10003	SIMOB	8	Simons bistro	Vinbæltet 34	København
10004	VAFFE	3	Vaffeljernet	Smagsløget 45	Århus
10005	WARTH	5	Wartian Herkku	Torikatu 38	Oulu
10006	FRANS	8	Franchi S.p.A.	Via Monte Bianco 34	Torino
10007	MORGK	4	Morgenstern Gesundkos	Heerstr. 22	Leipzig
10008	FURIB	3	Furia Bacalhau e Frutos	Jardim das rosas n. 32	Lisboa
10009	SEVES	8	Seven Seas Imports	90 Wadhurst Rd.	London
10010	SIMOB	8	Simons bistro	Vinbæltet 34	København
10011	WELLI	6	Wellington Importadora	Rua do Mercado, 12	Resende
10012	LINOD	6	LINO-Delicateses	Ave. 5 de Mayo Porlamar	I. de Margarita
10013	RICSU	3	Richter Supermarkt	Starenweg 5	Genève
10014	GROSR	4	GROSELLA-Restaurante	5ª Ave. Los Palos Grande	Caracas
10015	PICCO	6	Piccolo und mehr	Geichweg 14	Salzburg

### Page Size

SfDataPager splits the data into separate pages based on the PageSize. In order to specify the size of the page, you can use the PageSize property. By defaults, it is set to 0 and all the data is displayed in a single page.

The following code example explains how to use PageSize property in SfDataPager.

### XML

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="Auto" />
  </Grid.RowDefinitions>
  <syncfusion:SfDataGrid AutoGenerateColumns="True"
    ItemsSource="{Binding ElementName=sfDataPager, Path=PagedSource}" />
  <datapager:SfDataPager x:Name="sfDataPager"
    Grid.Row="1"
    NumericButtonCount="10"
    PageSize="10"
    Source="{Binding OrdersDetails}" />
</Grid>
```

The following screenshot displays the output for PageSize set as 5.

OrderID	CustomerID	CustomerName	Country	ShipCity
1001	ALFKI	Maria Anders	Germany	Berlin
1002	ANATR	Ana Trujillo	Mexico	México D.F.
1003	ANTON	Antonio Moreno	Mexico	México D.F.
1004	AROUT	Thomas Hardy	UK	London
1005	BERGS	Christina Berglund	Sweden	Luleå


## How To

### *How to change the PageSize in Runtime*

In general, the size of the page is defined in the PageSize property. In some cases, you may want to change it during the run time. The following code example explains how to change the size of the page during runtime for a ComboBox.

## XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<StackPanel Grid.Column="1" Grid.Row="1" Height="20" Margin="3,0"
Orientation="Horizontal">
<ComboBox Name="combobox" SelectedIndex="0" ItemsSource="{Binding
ComboBoxitem}" />
</StackPanel>
<sfgrid:SfDataGrid AutoGenerateColumns="True"
ItemsSource="{Binding ElementName=sfDataPager,Path =PagedSource}"/>
<datapager:SfDataPager x:Name="sfDataPager"
AccentBackground="DodgerBlue"
NumericButtonCount="10"
Grid.Row="1"
PageSize="{Binding Path =SelectedValue,ElementName=combobox}"
Source="{Binding OrdersDetails}" />
</Grid>
```

The following screenshot displays the output for changing the PageSize value during runtime.

OrderID	CustomerID	EmployeeID	Ship Name	Ship Address	Ship City	Ship Country	Freight
10000	FRANS	6	Franchi S.p.A.	Via Monte Bianco	Torino	Italy	\$4.43
10001	MEREP	8	Mère Pailarde	43 rue St. Laurent	Montreal	Canada	\$79.43
10002	FOLKO	3	Folk och få HB	Åkergatan 24	Bräcke	Sweden	\$36.18
10003	SINOB	8	Simons bistro	Vindbæltet 34	København	Denmark	\$18.59
10004	VAFFE	3	Vallejermet	Smagblæget 43	Århus	Denmark	\$20.12
10005	WARTH	5	Wartan Herku	Torikatu 38	Oulu	Finland	\$4.13
10006	FRANS	8	Franchi S.p.A.	Via Monte Bianco	Torino	Italy	\$3.62
10007	MORGK	4	Morgenstern Ges	Heenstr. 22	Leipzig	Germany	\$36.19
10008	FURIB	3	Furia Bacalhau e	Jardim das rosas	Lisboa	Portugal	\$74.22
10009	SEVES	8	Seven Seas Impo	90 Wadhurst Rd.	London	UK	\$49.21
10010	SINOB	8	Simons bistro	Vindbæltet 34	København	Denmark	\$3.01
10011	WELLI	6	Wellington Impor	Rua do Mercado	Resende	Brazil	\$31.54
10012	LINDO	6	LINDO-Delicatess	Ave. 5 de Mayo P	de Margarita	Venezuela	\$102.59
10013	RICSU	3	Richter Supermar	Stammweg 5	Genève	Switzerland	\$50.87
10014	GROSR	4	GROSELLA-Reser	59 Ave. Los Palos	Caracas	Venezuela	\$17.67
10015	PICCO	6	Piccolo und mehr	Gessingweg 14	Salzburg	Austria	\$22.10
10016	FOUGS	3	Folies gourmand	154, chaussée de	Lille	France	\$113.01
10017	BLONP	4	Blondel père et	fil 24, place Kleber	Strasbourg	France	\$111.81
10018	RATTIC	4	Rattlesnake Cany	2817 Milton Dr.	Albuquerque	USA	\$65.46

### Appearance in WPF DataPager (SfDataPager)

SfDataPager supports appearance styles by using the following properties.

- AutoEllipsisMode
- AccentBrush
- DisplayMode
- Orientation

#### AutoEllipsisMode

The AutoEllipsis button is displayed when the page count is greater than numeric button count. The SfDataPager control allows you to define the AutoEllipsis button by using the AutoEllipsisMode property which is the Enum type.

- AutoEllipsisMode – This Property is used to set the AutoEllipsisMode. By default, it is set to None.
- AutoEllipsisText– This property is used to change the AutoEllipsisButton Text.

The following table explains the different AutoEllipsisModes.

Enum Values	Description
After	Displays the ellipsis button after the numeric buttons. 
Before	Displays the ellipsis button before the numeric buttons. 
Both	Displays the ellipsis button before and after the numeric buttons. 
None	It does not display the AutoEllipsisButton.



OrderID	CustomerID	EmployeeID	Ship Name	Ship Address	Ship City
10000	FRANS	6	Franchi S.p.A.	Via Monte Bianco 34	Torino
10001	MEREP	8	Mière Paillarde	43 rue St. Laurent	Montréal
10002	FOLKO	3	Folk och få HB	Åkergatan 24	Bräcke
10003	SIMOB	8	Simons bistro	Vinbæltet 34	København
10004	VAFFE	3	Vaffeljernet	Smagsløget 45	Århus
10005	WARTH	5	Wartian Herkku	Torikatu 38	Oulu
10006	FRANS	8	Franchi S.p.A.	Via Monte Bianco 34	Torino
10007	MORGK	4	Morgenstern Gesund	Heerstr. 22	Leipzig
10008	FURIB	3	Furia Bacalhau e Fru	Jardim das rosas n. 3	Lisboa
10009	SEVES	8	Seven Seas Imports	90 Wadhurst Rd.	London
10010	SIMOB	8	Simons bistro	Vinbæltet 34	København
10011	WELLI	6	Wellington Importad	Rua do Mercado, 12	Resende
10012	LINOD	6	LINO-Delicateses	Ave. 5 de Mayo Por	I. de Margarita
10013	RICSU	3	Richter Supermarkt	Starenweg 5	Genève
10014	GROSR	4	GROSELLA-Restaura	5ª Ave. Los Palos Gri	Caracas
10015	PICCO	6	Piccolo und mehr	Geiselweg 14	Salzburg

Navigation buttons: < < 1 2 3 4 5 6 7 8 9 ...etc > >

**Note:** By Default

AutoEllipsisMode is set to None.

The following code example explains how to change the AutoEllipsisText.

**XML**

```

<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<sfgrid:SfDataGrid AutoGenerateColumns="True"
Grid.Row="0"
ItemsSource="{Binding ElementName=sfDataPager,Path=PagedSource}"/>
<datapager:SfDataPager x:Name="sfDataPager"
Grid.Row="1"
AutoEllipsisMode="After"
NumericButtonCount="10"
AutoEllipsisText="...etc"
PageSize="16"
Source="{Binding OrdersDetails}" />
</Grid>

```

The following screenshot displays the output for AutoEllipsisText changed as ...etc.

OrderID	CustomerID	EmployeeID	Ship Name	Ship Address	Freight	
10000	FRANS	6	Franchi S.p.A.	Via Monte Bianco 34	\$4.45	
10001	MEREP	8	Mère Paillardie	43 rue St. Laurent	\$79.45	
10002	FOLKO	3	Folk och få HB	Åkergatan 24	\$36.18	
10003	SIMOB	8	Simons bistro	Vinbæltet 34	\$18.59	
10004	VAFFE	3	Vaffeljernet	Smagsløget 45	\$20.12	
10005	WARTH	5	Wartian Herkku	Torikatu 38	\$4.13	
10006	FRANS	8	Franchi S.p.A.	Via Monte Bianco 34	\$3.62	
10007	MORGK	4	Morgenstern Gesundkost	Heerstr. 22	\$36.19	
10008	FURIB	3	Furia Bacalhau e Frutos do Mar	Jardim das rosas n. 32	\$74.22	
10009	SEVES	8	Seven Seas Imports	90 Wadhurst Rd.	\$49.21	
10010	SIMOB	8	Simons bistro	Vinbæltet 34	\$3.01	
10011	WELLI	6	Wellington Importadora	Rua do Mercado, 12	\$31.54	
10012	LINOD	6	LINO-Delicateses	Ave. 5 de Mayo Porlamar	\$102.59	
10013	RICSU	3	Richter Supermarkt	Starenweg 5	\$50.87	
10014	GROSR	4	GROSELLA-Restaurante	5ª Ave. Los Palos Grandes	\$17.67	
10015	PICCO	6	Piccolo und mehr	Geislweg 14	\$22.10	

![[Features

images9]](Featuresimages/Features\_img9.png)

### AccentBrush

AccentBrush properties are used to decorate the SfDataPager control with a solid color. There are two AccentBrush properties in the SfDataPager control:

- AccentBackground – Property that is applied to the background color for NavigationButtons and current selected numeric page button. By default, it set to DarkGray.
- AccentForeground – Property that is applied to the foreground color for the current selected numeric page button. By default, it set to White.
- NumericButtonStyle – Property that is applied to the Style of Numeric Button. This is the Style type property. By default, it set to Null.

The following code example explains how to apply the AccentBackground and AccentForeground properties for the SfDataPager control.

### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<sfgrid:SfDataGrid AutoGenerateColumns="True"
ItemsSource="{Binding ElementName=sfDataPager, Path =PagedSource}"/>
<datapager:SfDataPager x:Name="sfDataPager"
AccentBackground="#FF8CBF26"
AccentForeground="White"
NumericButtonCount="10"
PageSize="16"
Source="{Binding OrdersDetails}"/>
</Grid>
```

The following screenshot displays the output for AccentBackground and AccentForeground applied to the SfDataPager.

OrderID	CustomerID	EmployeeID	Ship Name	Ship Address	Ship City
10000	FRANS	6	Franchi S.p.A.	Via Monte Bianco 34	Torino
10001	MEREP	8	Mère Paillarde	43 rue St. Laurent	Montréal
10002	FOLKO	3	Folk och få HB	Åkergatan 24	Bräcke
10003	SIMOB	8	Simons bistro	Vinbæltet 34	København
10004	VAFFE	3	Vaffeljernet	Smagsløget 45	Århus
10005	WARTH	5	Wartian Herkku	Torikatu 38	Oulu
10006	FRANS	8	Franchi S.p.A.	Via Monte Bianco 34	Torino
10007	MORGK	4	Morgenstern Gesund	Heerstr. 22	Leipzig
10008	FURIB	3	Furia Bacalhau e Fru	Jardim das rosas n.3	Lisboa
10009	SEVES	8	Seven Seas Imports	90 Wadhurst Rd.	London
10010	SIMOB	8	Simons bistro	Vinbæltet 34	København
10011	WELLI	6	Wellington Importac	Rua do Mercado, 12	Resende
10012	LINOD	6	LINO-Delicateses	Ave. 5 de Mayo Por	L. de Margarita
10013	RICSU	3	Richter Supermarkt	Starenweg 5	Genève
10014	GROSR	4	GROSELLA-Restaura	5ª Ave. Los Palos Gr	Caracas
10015	PICCO	6	Piccolo und mehr	Geiselweg 14	Salzburg

![[Features

images10](Featuresimages/Features\_img10.png)

The following code example explains how to use NumericButtonStyle in SfDataPager.

### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Window.Resources>
<Style TargetType="datapager:NumericButton">
<Setter Property="BorderBrush" Value="Blue"/>
<Setter Property="BorderThickness" Value="2"/>
</Style>
</Window.Resources>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="50" />
</Grid.RowDefinitions>
<sfgrid:SfDataGrid AutoGenerateColumns="True"
Grid.Row="0"
ItemsSource="{Binding ElementName=sfDataPager, Path=PagedSource}" />
<datapager:SfDataPager x:Name="sfDataPager"
PageCount="16"
Grid.Row="1"
AutoEllipsisMode="Both"
AccentBackground="DodgerBlue"
```

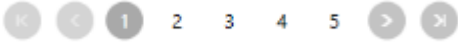

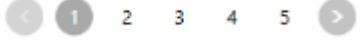



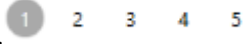




```
NumericButtonCount="10"
Source="{Binding OrdersDetails}"/>
</Grid>
```

The following screenshot displays the output of NumericButtonStyle.

![[Featuresimages11]](Featuresimages/Features\_img11.png)

### Display Modes

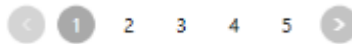
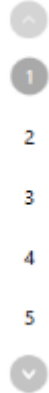
The SfDataPager control allows you to choose the elements that are visible in the control. This can be achieved by using the DisplayMode property. The following table explains the PageDisplayMode enum values.

Enum Values	Description
FirstLastPreviousNextNumeric	Displays all the navigation buttons and numeric page buttons. 
FirstLastNumeric	Displays the first page, last page navigation button and numeric page buttons. 
PreviousNextNumeric	Displays the previous, next page navigation buttons and numeric page buttons. 
FirstLastPreviousNext	Displays only the page navigation buttons. Numeric page buttons are not displayed. 
FirstLast	Displays only the first and last page navigation buttons. 
PreviousNext	Displays only the previous and next page navigation buttons. 
Numeric	Displays only the numeric page buttons. 
First	Displays only the first page navigation button. 
Last	Displays only the last page navigation button. 
Previous	Displays only the previous page navigation button. 
Next	Displays only the next page navigation button. 
None	It does not display anything

**Note:** By Default, DisplayMode is set to FirstLastPreviousNextNumeric.

### Orientation

SfDataPager allows you to arrange the child elements either horizontally or vertically. This can be achieved by using the Orientation Property. Orientation is an Enum type. The following table describes the Orientation enum values.

Enum Value	Description
Horizontal	This is the default enum value for Orientation. Arranges all the Navigation Buttons and Numeric Buttons Horizontally. 
Vertical	Arranges all the Navigation Buttons and Numeric Buttons Vertically. 

### PageNavigation in WPF DataPager (SfDataPager)

SfDataPager allows you to move from the current Page to various Pages. For example, when you want to move the CurrentPage to the last page directly, you can use the method MoveToLastPage(). When this method is called, the current page moves to the last page.

SfDataPager provides the following methods for page navigations.

Method	Prototype	Description
MoveToFirstPage	MoveToFirstPage()	This method moves the current page index to the first page and displays the first page data.
MoveToLastPage	MoveToLastPage()	This method moves the current page index to the last page and displays the last page data.
MoveToNextPage	MoveToNextPage()	This method moves the current page index to the next page and displays the next page data.
MoveToPreviousPage	MoveToPreviousPage()	This method moves the current page index to the previous page and displays the previous page data.
MoveToPage	MoveToPage(int pageIndex)	This method moves the current page index to the corresponding page index that is passed as an argument.

## How To

### How to Interact with User before Page Changes

When you are working with Paging, you may be in Edit mode or in CurrentPage. In this case, you can stop navigating the Paging by using the PageIndexChanging event before changing the page.

The following example displays the MessageBox before the PageChanging,

#### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<sfgrid:SfDataGrid AutoGenerateColumns="True"
Grid.Row="0"
ItemsSource="{Binding ElementName=sfDataPager,Path=PagedSource}"/>
<datapager:SfDataPager x:Name="sfDataPager"
NumericButtonCount="10"
Grid.Row="1"
PageSize="16"
PageIndexChanging="sfDataPager_PageIndexChanging"
Source="{Binding OrdersDetails}" />
</Grid>
```

#### C#

```
private void sfDataPager_PageIndexChanging(object sender,
Syncfusion.UI.Xaml.Controls.DataPager.PageIndexChangingEventArgs e)
{
    MessageBoxResult result = MessageBox.Show("Do you want to change the page?",
"Confirm", MessageBoxButton.YesNo);
    if (result == MessageBoxResult.No)
    {
        e.Cancel = true;
    }
}
```

## UIAutomation in WPF DataPager (SfDataPager)

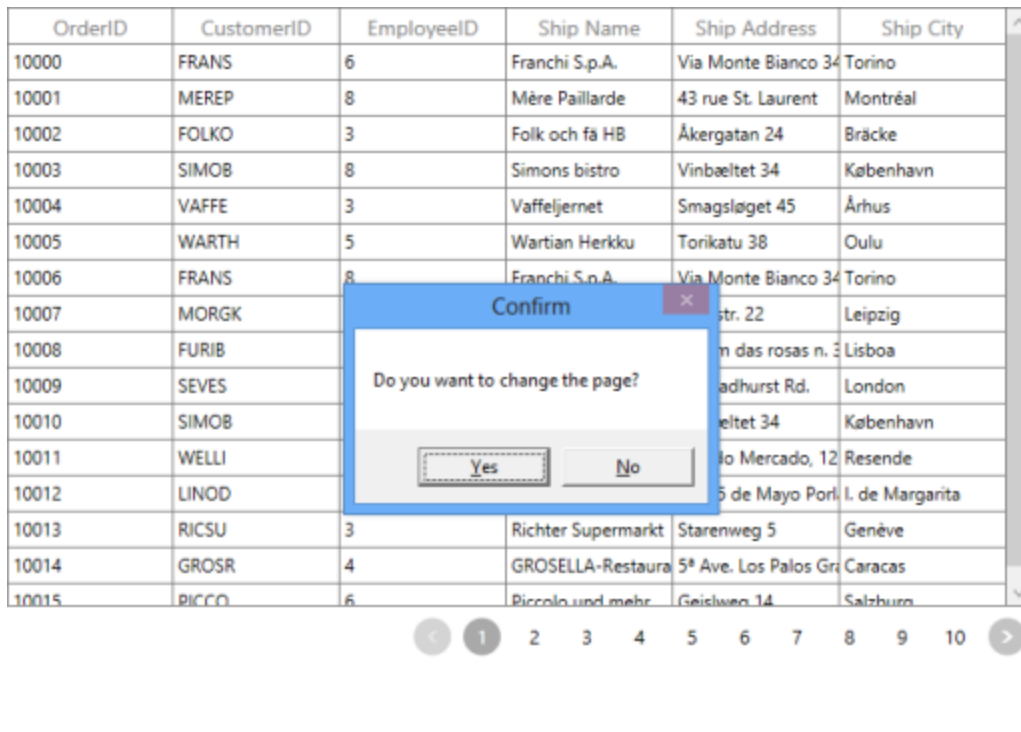
SfDataPager supports the following UIAutomation,

- Coded UI
- Quick Test Professional

### Coded UI

SfDataPager supports CodedUITest automation that helps you to create an automation test with SfDataPager elements and record the sequence of actions.

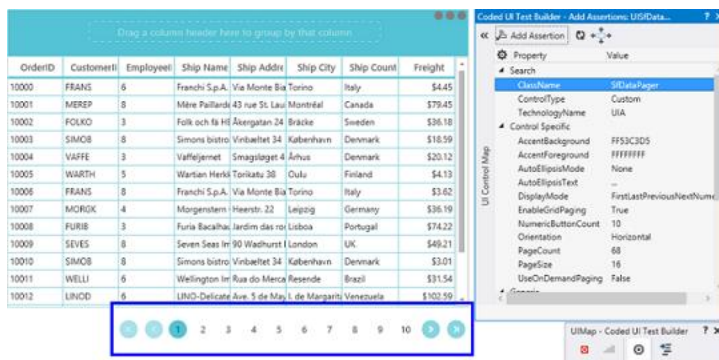
There are three levels of support in CodedUITest for SfDataPager



Levels	Description
Level 1	Record and Detects the UI Elements when the actions in the Control is performed.
Level 2	Provides custom properties for UI elements when you drag the Cross hair to any UI element.
Level 3	CodedUITest Builder generates code from the recorded session and you need to implement custom class to access custom properties, so the generated code is simplified.

To know more about CodedUITest, refer to the [link](#).

The following screenshot displays the SfDataPager properties when you drag the crosshair to the SfDataPager



The following table describes the properties of SfDataPager.

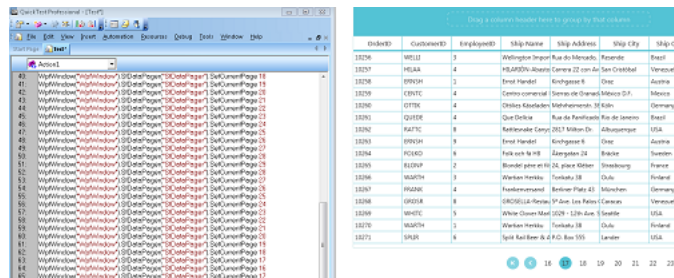
UI Element	Properties
SfDataPager	<i>AccentBackground</i> <i>AccentForeground</i> <i>AutoEllipsisMode</i> <i>AutoEllipsisText</i> <i>DisplayMode</i> <i>EnableGridPaging</i> <i>NumericButtonCount</i> <i>Orientation</i> <i>PageCount</i> <i>PageSize*</i> <i>UseOnDemandPaging</i>

## Quick Test Professional

SfDataPager supports QTP test. You can record the actions performed in the control by the corresponding method name

with Syncfusion namespace. To know more about QTP test, refer to the [link](#)

The following screenshot displays the QTP Test for SfDataPager



The following table describe the methods of SfDataPager.

Method	Description	Parameters	Return Type
void SetCurrentPage(int pageIndex)	To set the current page in SfDataPager	Int pageIndex	Void

## Styles and Templates in WPF DataPager (SfDataPager)

WPF Styles and Templates is a suite of features (styles and templates) that allow developers and designers to create visual compelling effects and consistent appearance of the products.

This section explains the information to change the visual appearance of the DataPager. In addition, you can edit the structure of the DataPager by using Blend and VisualStudio that helps you to customize their appearances.

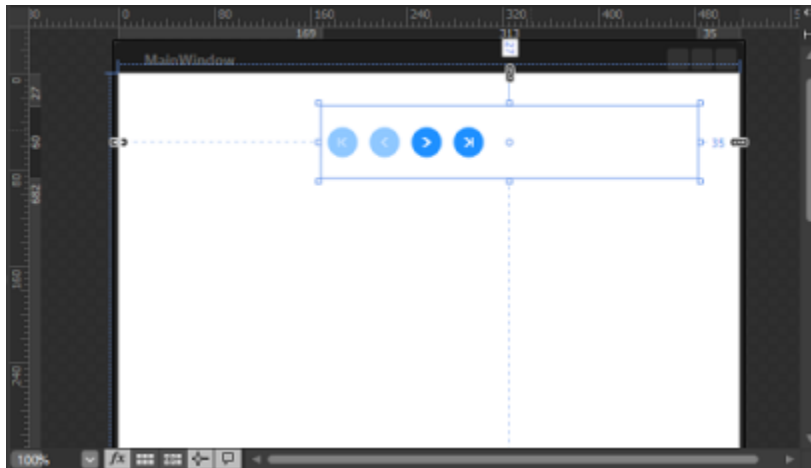
- Edit Appearance in Expression Blend
- Edit Appearance in VisualStudio

## Edit Appearance in Expression Blend

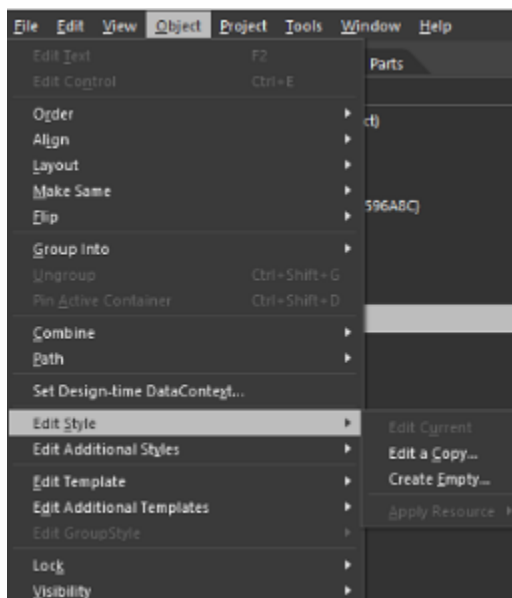
The section explains how to edit a SfDataPager style in Expression Blend. The following steps helps to Edit the style of the control in Expression Blend.

- Open your application in Expression Blend
- Select the SfDataPager control from the window.

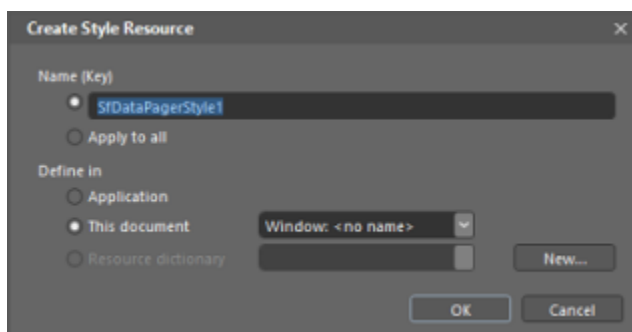




- Go to the Menu bar Choose Object > EditStyle



- There are two options in submenu,
- Edit a Copy –Edits a copy of the default style. When you select this option, a new dialog window opens as follows.



The Create style Resources dialog prompts you to enter the name or change the name of your style, as well as to choose the location for the style.

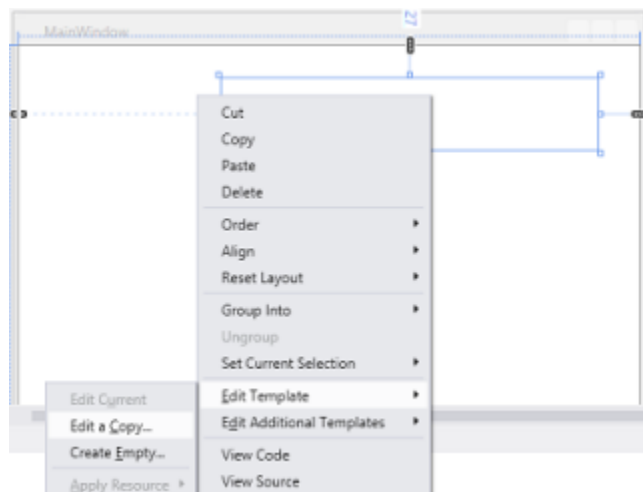
When OK is pressed, the Expression Blend generates the style of the SfDataPager control in the Resource section. You can edit the generated XAML in the XAML view or in VisualStudio

- Create Empty- create an empty style for the SfDataPager. When you select this option, the Create style Resources dialog is opened. You can enter the name or change the name of style and choose the location for the style.

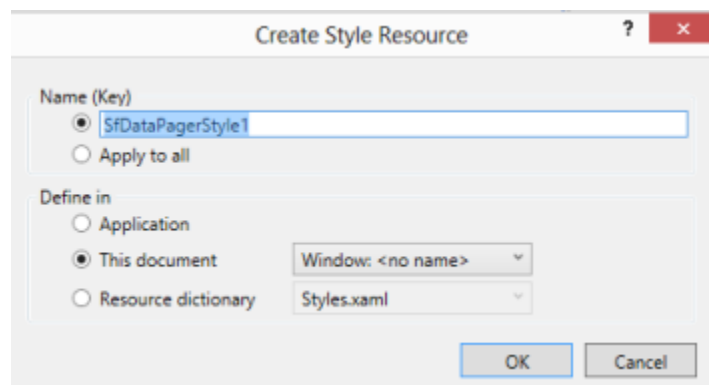
### Edit Appearance in Visual Studio

This section explains how to edit the style of SfDataPager in Visual Studio Designing View. To Edit the control style in Visual Studio, refer to the following steps,

- Open your application in Visual Studio.
- Open Design view>> Select SfDataPager Control >Right Click on SfDataPagerControl, Menu options are displayed.



- Click EditTemplate, you have the following two options.
- Edit a Copy –Edit a copy of the default style. When you select this option, a new dialog window opens as follows.



The Create style Resources dialog prompts you to enter the name or change the name for your style, as well as to choose the location for the style.

When OK is pressed, Visual Studio generate the style of SfDataPager in the Resource section. The control style of the SfDataPager control is loaded in the XAML . You can edit the generated XAML in the XAML view.

- Create Empty- create an empty style for the SfDataPager. When you select this option, the Create style Resources dialog opens. You can enter the name or change the name of style and choose the location where your style is defined.

## SfSkinManager

### Getting Started with WPF Skin Manager

The [SfSkinManager](#) helps you to apply the themes for both Syncfusion and Framework controls. There are 27 built-in themes that can be applied using the [SfSkinManager](#) for a rich user interface experience. Some of the built-in themes color derivations can be customized using [WPF Theme Studio](#).

**Note:** Theme Studio-based themes provide improved consistency and uniqueness among various controls when compared to other themes. It is preferable to use Theme Studio-based themes in the application over other themes.

#### Themes list

The following table lists the available themes as well as the assembly or NuGet reference to be used in the application.

Styles	Assembly	NuGet package	Supported in Theme Studio	Alternative theme suggestion to use
FluentLight	Syncfusion.Themes.FluentLight.Wpf.dll	<a href="#">Syncfusion.Themes.FluentLight.WPF</a>	Yes	-
FluentDark	Syncfusion.Themes.FluentDark.Wpf.dll	<a href="#">Syncfusion.Themes.FluentDark.WPF</a>	Yes	-
MaterialLight	Syncfusion.Themes.MaterialLight.Wpf.dll	<a href="#">Syncfusion.Themes.MaterialLight.WPF</a>	Yes	-
MaterialDark	Syncfusion.Themes.MaterialDark.Wpf.dll	<a href="#">Syncfusion.Themes.MaterialDark.WPF</a>	Yes	-
MaterialLightBlue	Syncfusion.Themes.MaterialLightBlue.Wpf.dll	<a href="#">Syncfusion.Themes.MaterialLightBlue.WPF</a>	Yes	-
MaterialDarkBlue	Syncfusion.Themes.MaterialDarkBlue.Wpf.dll	<a href="#">Syncfusion.Themes.MaterialDarkBlue.WPF</a>	Yes	-
Office2019Colorful	Syncfusion.Themes.Office2019Colorful.Wpf.dll	<a href="#">Syncfusion.Themes.Office2019Colorful.WPF</a>	Yes	-

Office2019Black	Syncfusion.Themes.Office2019Black.Wpf.dll	<a href="#">Syncfusion.Themes.Office2019Black.WPF</a>	Yes	-
Office2019White	Syncfusion.Themes.Office2019White.Wpf.dll	<a href="#">Syncfusion.Themes.Office2019White.WPF</a>	Yes	-
Office2019DarkGray	Syncfusion.Themes.Office2019DarkGray.Wpf.dll	<a href="#">Syncfusion.Themes.Office2019DarkGray.WPF</a>	Yes	-
Office2019HighContrast	Syncfusion.Themes.Office2019HighContrast.Wpf.dll	<a href="#">Syncfusion.Themes.Office2019HighContrast.WPF</a>	Yes	-
Office2019HighContrastWhite	Syncfusion.Themes.Office2019HighContrastWhite.Wpf.dll	<a href="#">Syncfusion.Themes.Office2019HighContrastWhite.WPF</a>	Yes	-
SystemTheme	Syncfusion.Themes.SystemTheme.Wpf.dll	<a href="#">Syncfusion.Themes.SystemTheme.WPF</a>	Yes	-
Metro	Syncfusion.Themes.Metro.Wpf.dll	<a href="#">Syncfusion.Themes.Metro.WPF</a>	-	FluentLight , Office2019 Colorful
Lime	Syncfusion.Themes.Lime.Wpf.dll	<a href="#">Syncfusion.Themes.Lime.WPF</a>	-	FluentLight , Office2019 Colorful
Saffron	Syncfusion.Themes.Saffron.Wpf.dll	<a href="#">Syncfusion.Themes.Saffron.WPF</a>	-	FluentLight , Office2019 Colorful
Blend	Syncfusion.Themes.Blend.Wpf.dll	<a href="#">Syncfusion.Themes.Blend.WPF</a>	-	FluentDark, MaterialDark, Office2019 Black
Office2013White	Syncfusion.Themes.Office2013White.Wpf.dll	<a href="#">Syncfusion.Themes.Office2013White.WPF</a>	-	Office2019 White
Office2013LightGray	Syncfusion.Themes.Office2013LightGray.Wpf.dll	<a href="#">Syncfusion.Themes.Office2013LightGray.WPF</a>	-	Office2019 Colorful
Office2013DarkGray	Syncfusion.Themes.Office2013DarkGray.Wpf.dll	<a href="#">Syncfusion.Themes.Office2013DarkGray.WPF</a>	-	Office2019 DarkGray
VisualStudio2013	Syncfusion.Themes.VisualStudio2013.Wpf.dll	<a href="#">Syncfusion.Themes.VisualStudio2013.WPF</a>	-	-

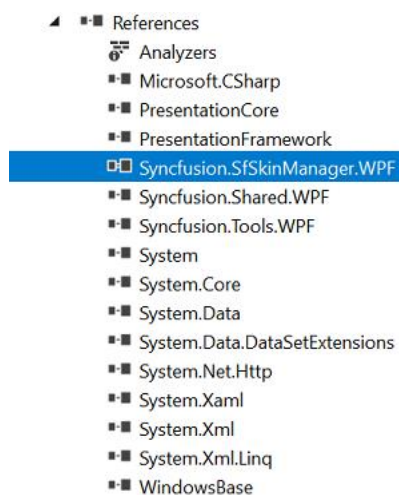
Office2010Black	Syncfusion.Themes.Office2010Black.Wpf.dll	<a href="#">Syncfusion.Themes.Office2010Black.WPF</a>	-	-
Office2010Blue	Syncfusion.Themes.Office2010Blue.Wpf.dll	<a href="#">Syncfusion.Themes.Office2010Blue.WPF</a>	-	-
Office2010Silver	Syncfusion.Themes.Office2010Silver.Wpf.dll	<a href="#">Syncfusion.Themes.Office2010Silver.WPF</a>	-	-
Office365	Syncfusion.Themes.Office365.Wpf.dll	<a href="#">Syncfusion.Themes.Office365.WPF</a>	-	Office2019 Colorful
Office2016Colorful	Syncfusion.Themes.Office2016Colorful.Wpf.dll	<a href="#">Syncfusion.Themes.Office2016Colorful.WPF</a>	-	Office2019 Colorful
Office2016White	Syncfusion.Themes.Office2016White.Wpf.dll	<a href="#">Syncfusion.Themes.Office2016White.WPF</a>	-	Office2019 White
Office2016DarkGray	Syncfusion.Themes.Office2016DarkGray.Wpf.dll	<a href="#">Syncfusion.Themes.Office2016DarkGray.WPF</a>	-	Office2019 DarkGray
VisualStudio2015	Syncfusion.Themes.VisualStudio2015.Wpf.dll	<a href="#">Syncfusion.Themes.VisualStudio2015.WPF</a>	-	-

Apply a theme to a control

[Add SkinManager reference](#)

There are several ways for including the Syncfusion [SfSkinManager](#) reference in the Visual Studio WPF project. The following steps will help you to add by XAML Code:

1) Add a reference to the `Syncfusion.SfSkinManager.WPF` assembly or [Syncfusion.SfSkinManager.WPF nuget package](#) to the project. 2) Import Syncfusion WPF schema `http://schemas.syncfusion.com/wpf` or the assembly namespace `Syncfusion.SfSkinManager` into a XAML page.

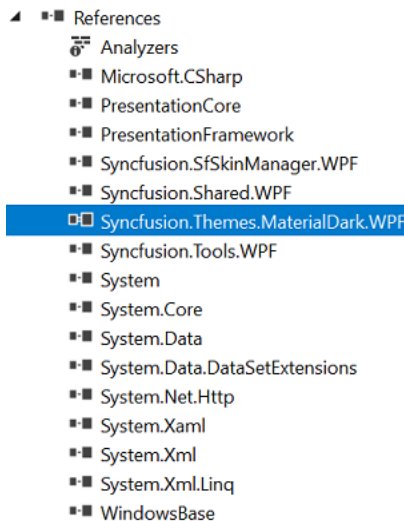


**XAML**

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf" />
```

### Add a theme assembly reference

The [SfSkinManager](#) supports to apply themes listed in [themes list](#). To use a theme in the application, add Reference to the corresponding theme assembly. For example, to apply **MaterialDark** theme, attach **Syncfusion.Themes.MaterialDark.Wpf** assembly or [NuGet](#) reference to the project. While applying a theme to a Window, SkinManager inherits the same theme to all the elements inside the Window.



### Set theme

Themes will be applied to both Syncfusion and Framework controls by using [Theme](#) attached property of the [SfSkinManager](#). The theme assemblies have resource dictionaries with styles of controls. Thus, when the **Theme** property is set, the skin manager merges the theme resource dictionaries of an element to which the theme is applied and its descendants into the resource dictionary of the element to which the theme is applied or **Application.Current.Resource**.

**Note:** While applying the theme to a Window or any element, **SkinManager** inherits the same theme to all its descendants.

### XML

```
<syncfusion:ChromelessWindow x:Class="DataGrid_Themes.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:DataGrid_Themes"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
Icon="App.ico"
Title="Getting Started"
WindowStartupLocation="CenterScreen"
syncfusionskin:SfSkinManager.Theme="{syncfusionskin:SkinManagerExtension
ThemeName=MaterialDark}">
```

```
<Grid DataContext="{StaticResource viewmodel}">
<syncfusion:SfDataGrid Name="sfgrid" Margin="5"
AutoGenerateColumns="False"
AllowDraggingColumns="True"
AllowEditing="True"
LiveDataUpdateMode="AllowDataShaping"
AllowFiltering="True"
HeaderRowHeight="26"
SelectionMode="Extended"
ColumnSizer="Auto"
ItemsSource="{Binding EmployeeDetails}">
</syncfusion:SfDataGrid>
</Grid>
</syncfusion:ChromelessWindow>
```

## C#

```
SfSkinManager.SetTheme(this, new Theme("MaterialDark"));
```

Employee Name	Designation	Mail	Location	Status	Trust Worthiness	Rating
Buchanan	CEO	buchanan@py.com	USA	Available	Insufficient	★★★★★
Margaret	CEO	margaret@yourapide.com	Sweden	Available	Sufficient	★★★★★
Zachery	Developer	zachery@sample.com	UK	Available	Perfect	★★★★★
Laura	System Analyst	laura@sample.com	Germany	Available	Sufficient	★★★★★
Zachery	CEO	zachery@py.com	Argentina	Available	Perfect	★★★★★
Nancy	Manager	nancy@yourapide.com	Germany	Available	Perfect	★★★★★
Jack	Program Director	jack@sample.com	Sweden	Available	Perfect	★★★★★
Alex	CEO	alex@py.com	USA	Available	Perfect	★★★★★
Janet	System Analyst	janet@py.com	Argentina	Available	Insufficient	★★★★★
Berly	Developer	berly@yourapide.com	France	Available	Sufficient	★★★★★
Marlin	Developer	marlin@py.com	USA	Available	Sufficient	★★★★★
Tim	CEO	tim@yourapide.com	Sweden	Available	Perfect	★★★★★
Janet	Developer	janet@py.com	Canada	Available	Insufficient	★★★★★
Vinod	Program Director	vinod@py.com	France	Available	Sufficient	★★★★★
Jack	CEO	jack@yourapide.com	Sweden	Available	Perfect	★★★★★
Laura	Manager	laura@yourapide.com	France	Available	Perfect	★★★★★
Janet	CEO	janet@yourapide.com	USA	Available	Sufficient	★★★★★

**Note:** [View sample in GitHub.](#)

Apply a theme globally in the application

By default, [SfSkinManager](#) merges the required resource files from the theme assembly to the element to which the theme is applied. To apply a theme globally in an application, set the `ApplyStylesOnApplication` property to `True`. It merges all the theme resource files to `Application.Current.Resources`.

**Note:** The `SfSkinManager.ApplyStylesOnApplication` static property should be set before `InitializeComponent` of the window or during application start up, when applying for multiple windows.

## C#

```
SfSkinManager.ApplyStylesOnApplication = true;
```

Customize theme colors and fonts in the application

To customize the theme colors and fonts in the application, call [RegisterThemeSettings](#) method and pass the theme name and respective theme setting instance as parameters.

Each theme supported by the theme studio has its own theme settings class, which begins with the prefix of the themes' name. For example, if the theme name is **MaterialDark**, then there will be theme settings class called **MaterialDarkThemeSettings**.

**Note:** Need to register theme settings before setting respective theme for window or control.

Please find the complete list of theme names, respective theme settings class, and supported palette.

Styles/Theme name	Respective theme settings class to customize	Supported palette
FluentLight	<a href="#">FluentLightThemeSettings</a>	<a href="#">FluentPalette</a>
FluentDark	<a href="#">FluentDarkThemeSettings</a>	<a href="#">FluentPalette</a>
MaterialLight	<a href="#">MaterialLightThemeSettings</a>	<a href="#">MaterialPalette</a>
MaterialDark	<a href="#">MaterialDarkThemeSettings</a>	<a href="#">MaterialPalette</a>
MaterialLightBlue	<a href="#">MaterialLightBlueThemeSettings</a>	<a href="#">MaterialPalette</a>
MaterialDarkBlue	<a href="#">MaterialDarkBlueThemeSettings</a>	<a href="#">MaterialPalette</a>
Office2019Colorful	<a href="#">Office2019ColorfulThemeSettings</a>	<a href="#">Office2019Palette</a>
Office2019Black	<a href="#">Office2019BlackThemeSettings</a>	<a href="#">Office2019Palette</a>
Office2019White	<a href="#">Office2019WhiteThemeSettings</a>	<a href="#">Office2019Palette</a>
Office2019DarkGray	<a href="#">Office2019DarkGrayThemeSettings</a>	<a href="#">Office2019Palette</a>
Office2019HighContrast	<a href="#">Office2019HighContrastThemeSettings</a>	<a href="#">HighContrastPalette</a>
Office2019HighContrastWhite	<a href="#">Office2019HighContrastWhiteThemeSettings</a>	<a href="#">HighContrastPalette</a>
SystemTheme	<a href="#">SystemThemeThemeSettings</a>	-

Customize theme colors and fonts in the application

### C#

```
FluentDarkThemeSettings themeSettings = new FluentDarkThemeSettings();
themeSettings.PrimaryBackground = new SolidColorBrush(Colors.Red);
themeSettings.PrimaryForeground = new SolidColorBrush(Colors.AntiqueWhite);
themeSettings.BodyFontSize = 15;
themeSettings.HeaderFontSize = 18;
themeSettings.SubHeaderFontSize = 17;
themeSettings.TitleFontSize = 17;
themeSettings.SubTitleFontSize = 16;
themeSettings.BodyAltFontSize = 15;
themeSettings.FontFamily = new FontFamily("Callibri");
SfSkinManager.RegisterThemeSettings("FluentDark", themeSettings);
```



Getting Started										
Employee Name	Designation	Mail	Location	Status	Trust Worthiness	Rating	Salary	Address	Software Proficiency	
Edward	Manager	edward@rpy.com	Sweden	Active	Perfect	★★★★★	\$159,101.00	Carrera 22 con Ave.	5.00%	
Bergs	Manager	bergs@arpy.com	Germany	Active	Sufficient	★★★★★	\$358,806.00	Carrera 22 con Ave.	47.00%	
Tamer	Manager	tamer@jourrapide.c	Sweden	Active	Sufficient	★★★★★	\$324,370.00	P.O. Box 555	72.00%	
Tamer	System Analyst	tamer@jourrapide.c	France	Inactive	Perfect	★★★★★	\$143,381.00	Åkergatan 24	69.00%	
Martin	System Analyst	martin@jourrapide.c	UK	Inactive	Perfect	★★★★★	\$284,356.00	Torikatu 38	73.00%	
Anto	Program Directory	anto@arpy.com	UK	Active	Sufficient	★★★★★	\$144,609.00	2 rue du Commerce	30.00%	
Tamer	Program Directory	tamer@sample.com	USA	Active	Sufficient	★★★★★	\$263,264.00	Berliner Platz 43	60.00%	
Van	Program Directory	van@sample.com	USA	Active	Sufficient	★★★★★	\$340,638.00	Luisenstr. 48	89.00%	
Vinet	Program Directory	vinet@arpy.com	Germany	Inactive	Sufficient	★★★★★	\$216,007.00	Rua da Panificadora 1	92.00%	
Kathryn	Designer	kathryn@rpy.com	UK	Inactive	Perfect	★★★★★	\$327,761.00	Torikatu 38	71.00%	
Jack	CFO	jack@arpy.com	Austria	Inactive	Insufficient	★★★★★	\$337,608.00	Torikatu 38	6.00%	
Laura	CFO	laura@rpy.com	UK	Inactive	Perfect	★★★★★	\$393,191.00	P.O. Box 555	31.00%	
Kathryn	CFO	kathryn@arpy.com	Germany	Active	Insufficient	★★★★★	\$271,132.00	Torikatu 38	30.00%	
Buchanan	Developer	buchanan@arpy.com	UK	Inactive	Sufficient	★★★★★	\$134,116.00	Hauptstr. 31	30.00%	
Zachery	Developer	zachery@arpy.com	France	Inactive	Insufficient	★★★★★	\$293,283.00	Rua do mailPaço 67	43.00%	
Zachery	Designer	zachery@rpy.com	Austria	Inactive	Insufficient	★★★★★	\$123,762.00	Sierras de Granada 95	21.00%	
Anto	CFO	anto@sample.com	UK	Active	Perfect	★★★★★	\$119,394.00	5ª Ave. Los Palos Gra	45.00%	
Van	CFO	van@arpy.com	Austria	Active	Sufficient	★★★★★	\$103,217.00	Torikatu 38	75.00%	
Margaret	Program Directory	margaret@rpy.com	Sweden	Inactive	Insufficient	★★★★★	\$203,173.00	Starenweg 5	37.00%	
Laura	CFO	laura@arpy.com	Argentina	Active	Perfect	★★★★★	\$394,943.00	1029 - 12th Ave. S.	12.00%	
Bergs	Developer	bergs@sample.com	Austria	Active	Sufficient	★★★★★	\$289,467.00	Rua do Mercado, 12	97.00%	
Bergs	CFO	bergs@arpy.com	USA	Inactive	Sufficient	★★★★★	\$133,832.00	Carlos Soubllette #8-	13.00%	
Leverling	Program Directory	leverling@arpy.com	Austria	Active	Sufficient	★★★★★	\$195,463.00	Sierras de Granada 95	57.00%	
Fuller	System Analyst	fuller@jourrapide.c	UK	Active	Sufficient	★★★★★	\$118,856.00	Rua do Mercado, 12	84.00%	
Van	Program Directory	van@rpy.com	UK	Active	Perfect	★★★★★	\$258,298.00	Kirchgasse 6	7.00%	
Martin	Manager	martin@arpy.com	Canada	Active	Perfect	★★★★★	\$262,463.00	Åkergatan 24	57.00%	
Callahan	Developer	callahan@rpy.com	USA	Active	Sufficient	★★★★★	\$377,724.00	Starenweg 5	8.00%	
Laura	Program Directory	laura@arpy.com	Austria	Active	Perfect	★★★★★	\$170,591.00	1029 - 12th Ave. S.	7.00%	
Andrew	System Analyst	andrew@arpy.com	Austria	Inactive	Sufficient	★★★★★	\$255,357.00	Rua do Mercado, 12	20.00%	

Customize theme colors using the predefined palette

### C#

```
FFluentDarkThemeSettings themeSettings = new FluentDarkThemeSettings();
themeSettings.Palette = FluentPalette.PinkRed;
SfSkinManager.RegisterThemeSettings("FluentDark", themeSettings);
```

**Note:** [View sample in GitHub.](#)

Apply themes to the controls derived from Syncfusion controls

To apply themes to the derived control using SfSkinManager, call [SetResourceReference](#) method and, pass the [StyleProperty](#) and derived control type as parameters.

### XML

```
<local:SfDataGridExt x:Name="grid"
AllowGrouping="True"
AutoGenerateColumns="False"
ItemsSource="{Binding EmployeeDetails}"
ShowGroupDropArea="True">
<local:SfDataGridExt.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.SfGrid.WPF;component/Styles/Styles.xaml" />
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</local:SfDataGridExt.Resources>
<local:SfDataGridExt.Columns>
<syncfusion:GridNumericColumn MappingName="EmployeeAge" />
<syncfusion:GridTextColumn MappingName="EmployeeName" />
<syncfusion:GridTextColumn MappingName="EmployeeGender" />
<syncfusion:GridTextColumn AllowEditing="True" MappingName="Country" />
```

```
<syncfusion:GridNumericColumn MappingName="EmployeeSalary" />
</local:SfDataGridExt.Columns>
</local:SfDataGridExt>
```

### C#

```
public class SfDataGridExt : SfDataGrid
{
    public SfDataGridExt()
    {
        SetResourceReference(StyleProperty, typeof(SfDataGrid));
    }
}
```

### Clearing SkinManager instance in an application

The **SfSkinManager** will hold some instances to use it further when applying the theme. However, this can be cleared using the function named **Dispose(object)**, which must be called when the theme applied by **SfSkinManager** is to be cleared, as shown in the following code. **Object** refers to the element whose instance needs to be cleared.

### C#

```
private void Window_Closed(object sender, EventArgs e)
{
    SfSkinManager.Dispose(this);
}
```

### How to

#### *Apply custom theme in the application*

To apply a custom theme in the application, export the custom theme project from ThemeStudio using [this reference](#).

For demonstration purposes, the exported theme name has been used as **MaterialDarkYellow** and assembly name as **Syncfusion.Themes.MaterialDarkYellow.WPF**.

Now, for the control used in the application, set the **SfSkinManager** attached property **Theme** to **MaterialDarkYellow;MaterialDark**. Since, custom theme name should be updated in the following format: **CustomTheme1;BaseThemeName**, where **CustomTheme1** denotes the custom theme name and **BaseThemeName** denotes the theme name from which it is derived. For example, **MaterialDarkYellow;MaterialDark**.

### C#

```
SfSkinManager.SetTheme(this, new Theme("MaterialDarkYellow;MaterialDark"));
```

#### *Override syncfusion themes in the application*

All Syncfusion themes are [supported in theme studio](#). A common naming convention can be used to override control styles. A unique key is given to every style so that the styles can be overridden using the **BasedOn** property.

The naming convention of a control style will be like `Syncfusion-ControlName-Style`. For example, `MaterialDarkButtonAdvStyle`.

The following steps explain how to override the Syncfusion themes:

#### Step1:

Add respective resource dictionary from the themes assembly to the application.

The resource dictionary will be in this format:

`/Syncfusion.Themes.<ThemeName>.WPF;component/<ControlName>/<ControlName>.xml`,

where `ThemeName` denotes the theme name for which overridden style is going to be applied and `ControlName` denotes the control name. For example,

`/Syncfusion.Themes.MaterialDark.WPF;component/ButtonAdv/ButtonAdv.xml`.

#### XML

```
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/Syncfusion.Themes.MaterialDark.WPF;component/ButtonAdv/ButtonAdv.xml"/>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

#### Step2:

Define the new style using the `BasedOn` property.

The following code sample overrides the Syncfusion style for the `ButtonAdv` Control.

#### XML

```
<Grid>
<Grid.Resources>
<Style x:Key="CustomButtonAdvStyle" TargetType="syncfusion:ButtonAdv"
BasedOn="{StaticResource SyncfusionButtonAdvStyle}" >
<Setter Property="Foreground" Value="Red"/>
</Style>
</Grid.Resources>
<syncfusion:ButtonAdv Name="buttonAdv" Content="Testing" Width="150"
Height="30" SmallIcon="{x:Null}" Style="{StaticResource
CustomButtonAdvStyle}"></syncfusion:ButtonAdv>
</Grid>
```



#### *Change visual style at runtime*

Themes for application can be changed at runtime by changing `VisualStyle` property. Make sure that the new theme assembly is attached as reference in the application when applying theme.

- Syncfusion.Linq.Base
- Syncfusion.SfGrid.WPF
- Syncfusion.SfInput.WPF
- Syncfusion.SfShared.WPF
- Syncfusion.SfSkinManager.WPF
- Syncfusion.Shared.WPF
- Syncfusion.Themes.Blend.WPF
- Syncfusion.Themes.Lime.WPF
- Syncfusion.Themes.MaterialDark.WPF
- Syncfusion.Themes.MaterialDarkBlue.WPF
- Syncfusion.Themes.MaterialLight.WPF
- Syncfusion.Themes.MaterialLightBlue.WPF
- Syncfusion.Themes.Metro.WPF
- Syncfusion.Themes.Office2010Black.WPF
- Syncfusion.Themes.Office2010Blue.WPF
- Syncfusion.Themes.Office2010Silver.WPF
- Syncfusion.Themes.Office2013DarkGray.WPF
- Syncfusion.Themes.Office2013LightGray.WPF
- Syncfusion.Themes.Office2013White.WPF
- Syncfusion.Themes.Office2016Colorful.WPF
- Syncfusion.Themes.Office2016DarkGray.WPF
- Syncfusion.Themes.Office2016White.WPF
- Syncfusion.Themes.Office2019Black.WPF

## XML

```
<syncfusion:ChromelessWindow x:Class="DataGrid_Themes.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="clr-namespace:DataGrid_Themes"
xmlns:system="clr-namespace:System;assembly=microsoftcorlib"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
syncfusionskin:SfSkinManager.VisualStyle="{Binding
ElementName=comboVisualStyle, Path=SelectedValue, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">
<syncfusion:ChromelessWindow.Resources>
<ObjectDataProvider
x:Key="Themes"
MethodName="GetValues"
ObjectType="{x:Type system:Enum}">
<ObjectDataProvider.MethodParameters>
<x:Type TypeName="syncfusionskin:VisualStyles" />
</ObjectDataProvider.MethodParameters>
</ObjectDataProvider>
</syncfusion:ChromelessWindow.Resources>
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="40"/>
<RowDefinition Height="*" />
</Grid.RowDefinitions>
<ComboBox
x:Name="comboVisualStyle"
Grid.Row="0"
Width="250"
Margin="5"
ItemsSource="{Binding Source={StaticResource Themes}}"
SelectedIndex="18" />
```

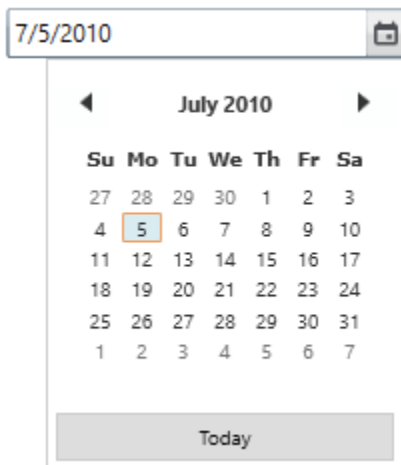
```
<Grid Grid.Row="1" DataContext="{StaticResource viewmodel}">
<syncfusion:SfDataGrid Name="sfgrid" Margin="5"
AutoGenerateColumns="False"
AllowDraggingColumns="True"
AllowEditing="True"
LiveDataUpdateMode="AllowDataShaping"
AllowFiltering="True"
HeaderRowHeight="26"
SelectionMode="Extended"
ColumnSizer="Auto"
ItemsSource="{Binding EmployeeDetails}">
</syncfusion:SfDataGrid>
</Grid>
</Grid>
</syncfusion:ChromelessWindow>
```

**Note:** [View sample in GitHub.](#)

## DateTimeEdit

### WPF DateTimePicker (DateTimeEdit) Overview

The [DateTimeEdit](#) allows you to quickly navigate and select dates using months, years, and decades of calendar. The `DateTimeEdit` comprises text box and a dropdown with calendar and clock to pick or edit a date with time. It supports data binding, null value, maximum and minimum dates, date validation, watermark, culture and much more. It also provides flexible options for displaying the date-time according to the format required, as well as many customization options to enhance its appearance.



#### Key features

**Editing mode** - Supports the default text editing and mask mode that helps to restrict the date input in formatted values based on pre-defined or custom date-time pattern.

**Date-range support** - Supports the maximum and minimum dates in order to prevent users from setting a date or time within a given range.

**Globalization** - Supports different date-time formats and patterns based on cultures.

**Date validation** - Supports date validation when date range constraints are met.

**Accessibility** - Provides touch, keyboard, and mouse support to make applications available to a wide variety of users.

**Watermark** - Supports to display watermark text when a selected date is null.

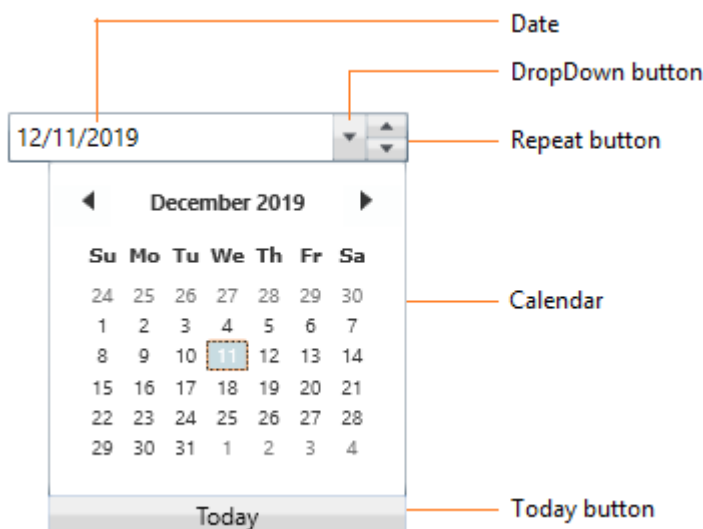
**Styles** - Provides a rich set of built-in themes and customizes the style of each part of `DateTimeEdit`.

**Testing** - Provides QTP add-in that contains custom libraries, which helps [QTP](#) to recognize `DateTimeEdit`.

### Getting Started with WPF DateTimePicker (DateTimeEdit)

This section provides a quick overview of working with the WPF [DateTimeEdit](#).

#### Visual Structure



#### Assembly deployment

Refer to the [Control Dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the `DateTimeEdit` control in any application.

You can find more details about installing the NuGet package in a WPF application in the following link: [How to install nuget packages](#)

#### Creating an application with the DateTimeEdit control

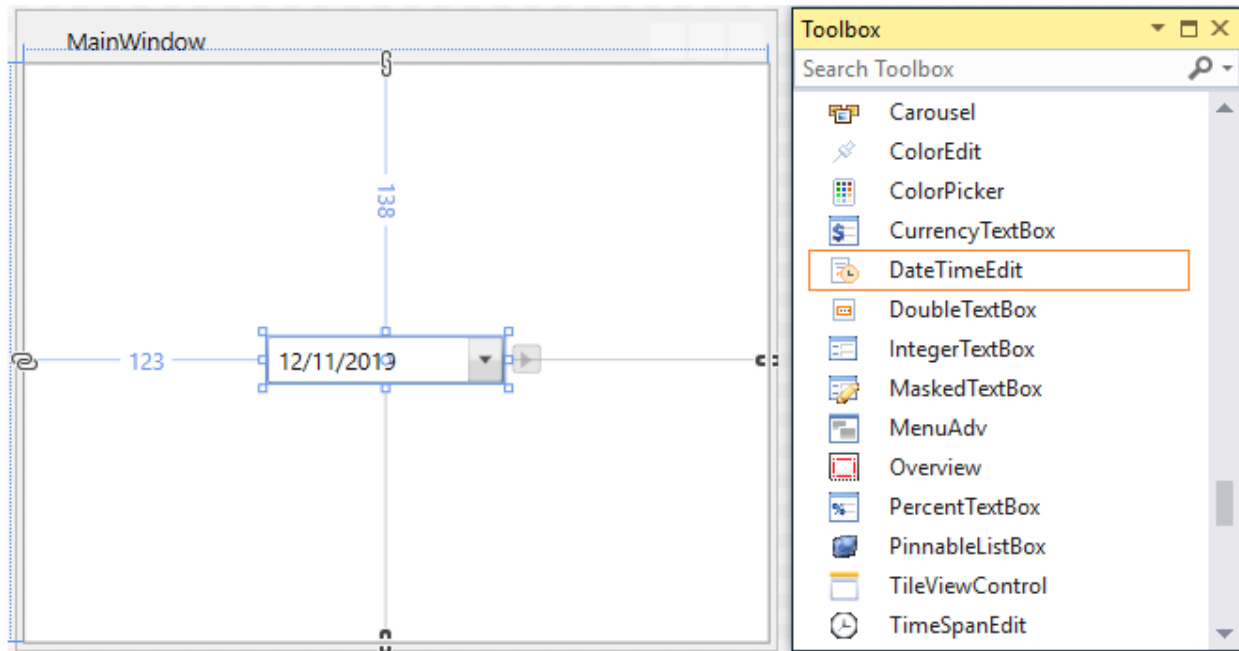
In this walkthrough, you will create a WPF application that contains the `DateTimeEdit` control.

#### Creating a project

Create a new WPF project to show the `DateTimeEdit` control in Visual Studio.

#### Adding control via designer

The `DateTimeEdit` control can be added to the application by dragging it from Toolbox and dropping it in the designer. The required [assemblies](#) will be added automatically.



**Note:** You can customize the properties of DateTimeEdit control using the SmartTag.

#### Adding control manually in XAML

To add the control manually in XAML page, follow the given steps:

1. Add the following required assembly references to the project, \* Syncfusion.Shared.WPF 2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in XAML page. 3. Declare the DateTimeEdit control in XAML page.

#### XML

```
<Window x:Class="DateTimeEdit_sample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:DateTimeEdit_sample"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid Name="grid">
<syncfusion:DateTimeEdit x:Name="dateTimeEdit" Height="25" Width="120"
VerticalAlignment="Center"/>
</Grid>
</Window>
```

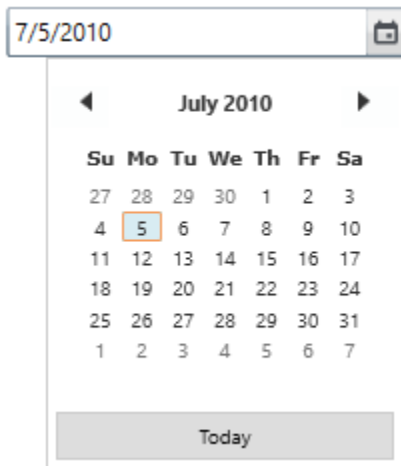
#### Adding control manually in C#

To add the control manually in C#, follow the given steps:

1. Add the following required assembly references to the project, \* Syncfusion.Shared.WPF 2. Import the DateTimeEdit namespace **Syncfusion.Windows.Shared**. 3. Create the DateTimeEdit control instance and add it to the page.

**C#**

```
// Creating an instance of the DateTimeEdit
DateTimeEdit dateTimeEdit = new DateTimeEdit();
// Setting height and width to DateTimeEdit
dateTimeEdit.Height = 25;
dateTimeEdit.Width = 120;
// Adding control into the main window
this.Content = dateTimeEdit;
```



## Setting date time value

You can set the date using the [DateTime](#) property of the `DateTimeEdit` control.

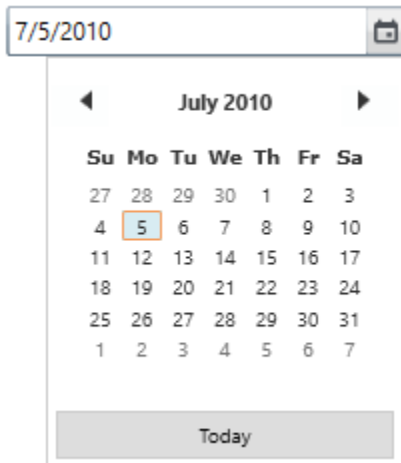
**XML**

```
<!--Setting date -->
<syncfusion:DateTimeEdit DateTime="07/05/2010"
Name="dateTimeEdit"
Height="25"
Width="120" />
```

**C#**

```
//Setting date
dateTimeEdit.DateTime = new DateTime(2010, 07, 05);
```





**Note:** [View sample in GitHub.](#)

Binding date time value

You can bind the selected date time value by using the `DateTime` property.

The following code snippets illustrate the value binding from one `DateTimeEdit` to another.

#### C#

```
//ViewModel.cs
class ViewModel : NotificationObject {
private DateTime selectedDate = DateTime.Now;
public DateTime SelectedDate {
get {
return selectedDate;
}
set {
selectedDate = value;
this.RaisePropertyChanged(nameof(SelectedDate));
}
}
}
```

#### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<StackPanel HorizontalAlignment="Center"
VerticalAlignment="Center">
<syncfusion:DateTimeEdit Name="dateTimeEdit1"
Height="25"
Width="200"
DateTime="{Binding SelectedDate, Mode=TwoWay}"/>
<syncfusion:DateTimeEdit Name="dateTimeEdit2"
Height="25"
Width="200"
DateTime="{Binding SelectedDate, Mode=TwoWay}"
Margin="10"/>
</StackPanel>
```



**Note:** [View sample in GitHub.](#)

### Value Changed Notification

The [DateTime](#) property value changed in the `DateTimeEdit` can be examined using the [DateTimeChanged](#) event. The `DateTimeChanged` event contains the old and newly selected date time values in the `OldValue` and `NewValue` properties.

### XML

```
<syncfusion:DateTimeEdit DateTimeChanged="dateTimeEdit_DateTimeChanged"
Name="dateTimeEdit"/>
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.DateTimeChanged += dateTimeEdit_DateTimeChanged;
```

You can handle the event as follows,

### C#

```
private void dateTimeEdit_DateTimeChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
    //Get old and new values
    var oldValue = e.OldValue;
    var newValue = e.NewValue;
}
```

### Applying built-in pattern

You can change the date-time pattern using the [Pattern](#) property to the following formats in the `DateTimeEdit` control.

- LongDate
- LongTime
- ShortDate
- ShortTime
- FullDateTime
- MonthDay
- CustomPattern
- ShortableDateTime
- UniversalSortableDateTime
- RFC1123
- YearMonth

### XML

```
<!--Setting ShortDate Pattern-->
<syncfusion:DateTimeEdit x:Name="dateTimeEdit"
DateTime="07/15/2010"
Pattern="ShortDate"/>
```

**C#**

```
//Setting predefined ShortDate pattern
dateTimeEdit.Pattern = DateTimePattern.ShortDate;
```

<input type="text" value="7/15/2010"/>	<input type="text" value="12:00 AM"/>
ShortDate	ShortTime
<input type="text" value="Thursday, July 15, 2010"/>	<input type="text" value="12:00:00 AM"/>
LongDate	LongTime
<input type="text" value="July 15"/>	<input type="text" value="July 2010"/>
MonthDay	YearMonth
<input type="text" value="Thursday, July 15, 2010 12:00:00 AM"/>	<input type="text" value="Thu, 15 Jul 2010 00:00:00 GMT"/>
FullDateTime	RFC1123
<input type="text" value="2010-07-15T00:00:00"/>	<input type="text" value="2010-07-15 00:00:00Z"/>
SortableDateTime	UniversalSortableDateTime

**Note:** [View sample in GitHub.](#)

**Applying custom pattern**

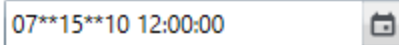
You can display the date and time in any format in the DateTimeEdit control by using the [CustomPattern](#) property. The CustomPattern property support can be enabled by setting the [Pattern](#) property to CustomPattern.

**XML**

```
<syncfusion:DateTimeEdit CustomPattern="MM*dd*yy hh:mm:ss"
Pattern="CustomPattern"
DateTime="07/15/2010"
Name="dateTimeEdit"/>
```

**C#**

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.CustomPattern = "MM*dd*yy hh:mm:ss";
dateTimeEdit.Pattern = DateTimePattern.CustomPattern;
dateTimeEdit.DateTime = new DateTime(2019, 07, 15);
```



**Note:** [View sample in GitHub.](#)

#### Editing date time

The [DateTimeEdit](#) control supports both free form editing (like normal textbox editing) and mask based editing. You can edit the date-time in the [DateTimeEdit](#) control by setting the [CanEdit](#) property to `true`. For every user input, the value will be validated after the Enter key is pressed, or focus is lost when the [CanEdit](#) property is enabled.

#### XML

```
<syncfusion:DateTimeEdit CanEdit="True"
    DateTime="07/15/2010"
    Name="dateTimeEdit"/>
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.CanEdit = true;
```



**Note:** [View sample in GitHub.](#)

#### Restrict date range

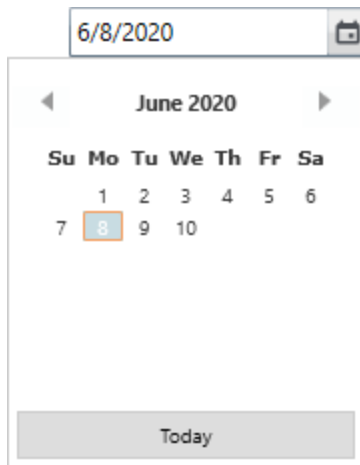
You can restrict the users from selecting a date and time in a particular range by using the [MinimumDateTime](#) and [MaximumDateTime](#) properties in the [DateTimeEdit](#) control.

#### XML

```
<!--Setting date range -->
<syncfusion:DateTimeEdit MinDateTime="06/1/2020"
    MaxDateTime="06/10/2020"
    Name="dateTimeEdit"/>
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.MinDateTime = new DateTime(2020, 06, 01);
dateTimeEdit.MaxDateTime = new DateTime(2020, 06, 10);
```



**Note:** [View sample in GitHub.](#)

#### Show watermark when value is null

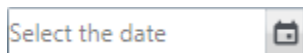
If you want to display the watermark text instead of null value, use the [NoneDateText](#) property to setting the watermark text. You can enable it only by setting the [IsEmptyDateEnabled](#) property to `true`, [ShowMaskOnNullValue](#) property to `false` and `NullValue` property as `null`. The default value of `NoneDateText` property is `string.Empty`.

#### XML

```
<syncfusion:DateTimeEdit NoneDateText="Select the date"
ShowMaskOnNullValue="True"
NullValue="{x:Null}"
IsEmptyDateEnabled="True"
Name="dateTimeEdit"/>
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.NoneDateText= "Select the date";
dateTimeEdit.ShowMaskOnNullValue= true;
dateTimeEdit.IsEmptyDateEnabled= true;
dateTimeEdit.NullValue = null;
```



**Note:** [View sample in GitHub.](#)

#### Change month names

You can change the popup calendar month names in the `DateTimeEdit` control by adding the respective new names to the [AbbreviatedMonthNames](#) property. The default value of the `AbbreviatedMonthNames` property is `null`.

#### XML

```
<syncfusion:DateTimeEdit Name="dateTimeEdit" >
<syncfusion:DateTimeEdit.AbbreviatedMonthNames>
<x:Array Type="sys:String"
xmlns:sys="clr-namespace:System;assembly=mscorlib">
```

```

<sys:String>[1] Jan</sys:String>
<sys:String>[2] Feb</sys:String>
<sys:String>[3] Mar</sys:String>
<sys:String>[4] Apr</sys:String>
<sys:String>[5] May</sys:String>
<sys:String>[6] Jun</sys:String>
<sys:String>[7] Jul</sys:String>
<sys:String>[8] Aug</sys:String>
<sys:String>[9] Sep</sys:String>
<sys:String>[10] Oct</sys:String>
<sys:String>[11] Nov</sys:String>
<sys:String>[12] Dec</sys:String>
</x:Array>
</syncfusion:DateTimeEdit.AbbreviatedMonthNames>
</syncfusion:DateTimeEdit>

```

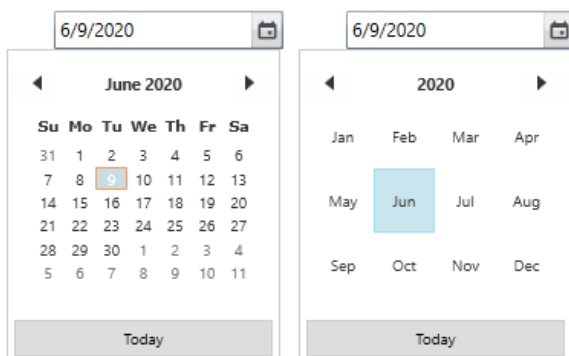
**C#**

```

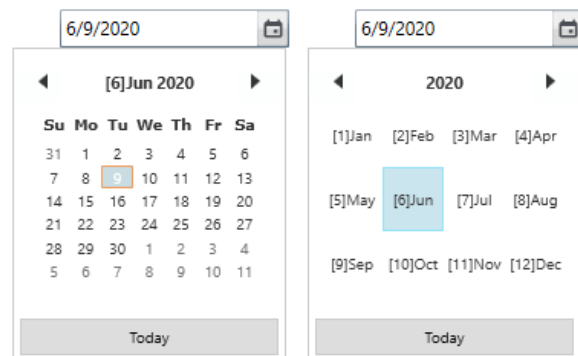
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.AbbreviatedMonthNames = new string[]
{
    "[1] Jan",
    "[2] Feb",
    "[3] Mar",
    "[4] Apr",
    "[5] May",
    "[6] Jun",
    "[7] Jul",
    "[8] Aug",
    "[9] Sep",
    "[10] Oct",
    "[11] Nov",
    "[12] Dec"
};

```

Default month names



Changed month names



**Note:** [View sample in GitHub.](#)

### Change week day names

You can change the popup calendar week day names in DateTimeEdit control by adding the respective new names to the [ShortestDayNames](#) property. The default value of the [ShortestDayNames](#) property is `null`.

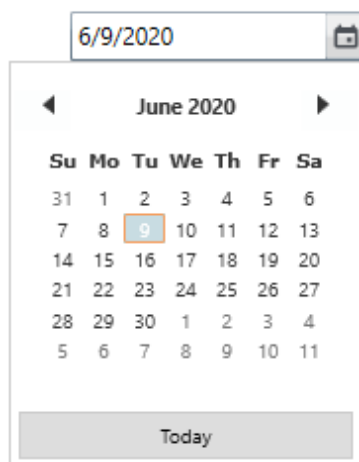
#### XML

```
<syncfusion:DateTimeEdit Name="dateTimeEdit" >
  <syncfusion:DateTimeEdit.ShortestDayNames>
    <x:Array Type="sys:String"
      xmlns:sys="clr-namespace:System;assembly=mscorlib">
      <sys:String>Sunday</sys:String>
      <sys:String>Monday</sys:String>
      <sys:String>Tuesday</sys:String>
      <sys:String>Wednesday</sys:String>
      <sys:String>Thursday</sys:String>
      <sys:String>Friday</sys:String>
      <sys:String>Saturday</sys:String>
    </x:Array>
  </syncfusion:DateTimeEdit.ShortestDayNames>
</syncfusion:DateTimeEdit>
```

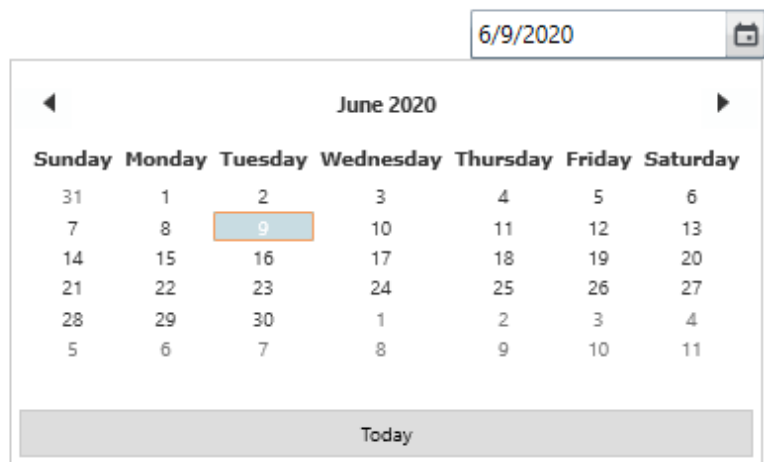
#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.ShortestDayNames= new string[]
{
    "Sunday",
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday"
};
```

Default weekday names



Changed weekday names



**Note:** [View sample in GitHub.](#)

### Block particular dates

You can restrict the user to select the date within some range by blocking the particular date in the DateTimeEdit. If you try to set the blackout dates as the selected datetime through editing or when pressing down arrow, it will reset the previous valid date from **StartDate** of blackout dates in DateTimeEdit. If you try to set the blackout dates as the selected datetime by pressing down arrow, it will reset the next valid date from **EndDate** of blackout dates in DateTimeEdit.

**Note:** If you try to set the DateTime value which contained in blackout dates collection or try to add the selected DateTime value in blackout dates collection, it throws the **Specified argument was out of the range of valid values**.

### XML

```
<syncfusion:DateTimeEdit Loaded="DateTimeEdit_Loaded"
Name="dateTimeEdit"/>
```

### XML

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.Loaded += DateTimeEdit_Loaded;
```

### C#

```
//Setting the blackout days
private void DateTimeEdit_Loaded(object sender, RoutedEventArgs e)
{
    //Setting start and end range for blocking dates
    DateTime StartDate = new DateTime(DateTime.Now.Year, DateTime.Now.Month, 1);
    DateTime EndDate = new DateTime(DateTime.Now.Year, DateTime.Now.Month,
    DateTime.Now.Day - 2);
    CalendarDateRange blackOutDays = new CalendarDateRange()
    {
        Start = StartDate,
        End = EndDate
    };
    Calendar calendar = new Calendar();
    calendar.BlackoutDates.Add(blackOutDays);
    dateTimeEdit.DateTimeCalender = calendar;
}
```





---

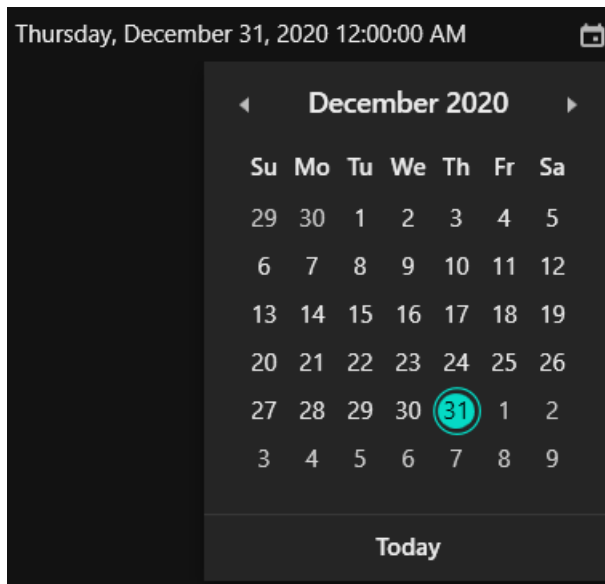
**Note:** [View sample in GitHub.](#)

---

### Theme

The WPF DateTimePicker (DateTimeEdit) supports various built-in themes. Refer to the below links to apply themes for the DateTimeEdit,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



---

**Note:** [View sample in GitHub.](#)

---

### DateTime Editing in WPF DateTimePicker (DateTimeEdit)

The [DateTimeEdit](#) control provides support for changing date time using text box. It supports both free form editing and mask based editing.

#### Mask editing

Mask editing is the default editing mode. In this mode, you can provide only valid input. Any invalid input given will be automatically corrected or ignored. In mask editing mode, the date will be separated

into different fields such as date, month, year, minutes, hours, and seconds. The field can be updated by selecting the field and edit the value for the respectively fields or pressing the up (or) down arrow keys to increase or decrease the selected field respectively.

### XML

```
<syncfusion:DateTimeEdit Name="dateTimeEdit"
    Height="25"
    Width="200"/>
```

### C#



```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.Width = 150;
dateTimeEdit.Height = 25;
```



**Note:** View [Sample](#) in GitHub

### Validation

In mask editing mode, value will be validated and corrected immediately based on the edited input value. The following table demonstrates some of the validation in `DateTimeEdit` control:

Validation	Example
If the Space or next field character is pressed, focus will automatically navigated to next valid date / time field that can be edited.	In short pattern, "/" character is used as the date separators: MM/dd/yyyy. On pressing "/" the focus will automatically navigated to next valid field.
Field focus will be automatically navigated when value of the selected field is filled.	
DateTimeEdit value will be validated and corrected immediately based on the edited input value.	 Ex: If the month field is February, then if we type 29 in day field year field will be automatically validated and moved to next leap year based on corresponding date.
Day field of DateTimeEdit can not be selected as it cannot be edited.	-











**Note:** View [Sample](#) in GitHub

### DateTime field navigation

By default, the focus field will be navigated automatically after the value has been validated to the corresponding field. If you want to manually change the date, month, year, hour or minute values, before that you navigate to the respective field by using the mouse or move the `Left-Right` keys in the keyboard.

*Keyboard Navigation between datetime fields*

The following table explains how the navigation performed between datetime fields,

S.No	Key	Description	Image
1	Up	Increase selected date, month or year value by 1.	
2	Down	Decrease selected date, month or year value by 1.	
3	Right	Navigate to the previous field from the currently selected field.	
4	Left	Navigate to the next field from the currently selected field.	
5	Ctrl + Up	Move forward from current selection view to next month or year selection view.	
6	Ctrl + Down	Move backward from current selection view to previous month or date selection view.	
7	PageUp	Move to previous month, date or year from the corresponding selection view.	
8	PageDown	Move to next month, date or year from the corresponding selection view.	
9	Home	Select first date or month or year or decade of the current selection view.	
10	End	Select last date or month or year or decade of the current selection view.	

**Note:** View [Sample](#) in GitHub

*Change date time programmatically*

You can set or change the selected datetime of the `DateTimeEdit` programmatically by setting the value to the `DateTime` property.

**XML**

```
<syncfusion:DateTimeEdit DateTime="08/06/2020"
Name="dateTimeEdit"/>
```

**C#**

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.DateTime = new DateTime(2020, 06, 30);
```



**Note:** View [Sample](#) in GitHub

### Binding date time value

You can bind the selected date time value by using the `DateTime` property.

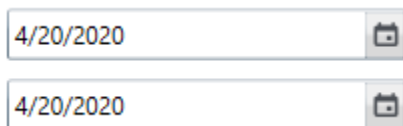
The following code snippets illustrate the value binding from one `DateTimeEdit` to another.

#### C#

```
//ViewModel.cs
class ViewModel : NotificationObject {
private DateTime selectedDate = DateTime.Now;
public DateTime SelectedDate {
get {
return selectedDate;
}
set {
selectedDate = value;
this.RaisePropertyChanged(nameof(SelectedDate));
}
}
}
```

#### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<StackPanel HorizontalAlignment="Center"
VerticalAlignment="Center">
<syncfusion:DateTimeEdit Name="dateTimeEdit1"
Height="25"
Width="200"
DateTime="{Binding SelectedDate, Mode=TwoWay}"/>
<syncfusion:DateTimeEdit Name="dateTimeEdit2"
Height="25"
Width="200"
DateTime="{Binding SelectedDate, Mode=TwoWay}"
Margin="10"/>
</StackPanel>
```



**Note:** View [Sample](#) in GitHub

### Free form editing

You can change the `DateTime` value like a normal textbox editing by setting the `CanEdit` property value as true. Input given by an end-user, will be validated when pressing Enter key or if control lost its focus. If the entered value is invalid, it set the previously selected date as `DateTime` value. Otherwise, it will accept the given input.

#### XML

```
<syncfusion:DateTimeEdit Name="dateTimeEdit"
```

```
CanEdit="True"/>
```

### C#

```
DateTimeEdit dateTimeEdit= new DateTimeEdit();  
dateTimeEdit.CanEdit = true;
```



**Note:** View [Sample](#) in GitHub

### Change date time on mouse wheel

You can increase or decrease the datetime by select the respective fields and mouse scrolling over the DateTimeEdit. If you want to restrict the user to change datetime by using mouse scrolling, use the [EnableMouseWheelEdit](#) property value as `false`. The default value of [EnableMouseWheelEdit](#) property is `true`.

### XML

```
<syncfusion:DateTimeEdit EnableMouseWheelEdit="True"  
Name="dateTimeEdit"/>
```

### C#

```
DateTimeEdit dateTimeEdit= new DateTimeEdit();  
dateTimeEdit.EnableMouseWheelEdit = true;
```



**Note:** View [Sample](#) in GitHub

### Change date time using up-down button

You can change the value of the datetime by selecting the respective date, month, year, minutes, hours, or seconds field and pressing the up or down button to increase or decrease the selected field respectively. By default, the updown button are hidden. you can show the UpDown button by setting the [IsVisibleRepeatButton](#) property value as `true`.

### XML

```
<syncfusion:DateTimeEdit IsVisibleRepeatButton="True"  
IsEnabledRepeatButton="True"  
Name="dateTimeEdit"/>
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();  
dateTimeEdit.IsVisibleRepeatButton= true;  
dateTimeEdit.IsEnabledRepeatButton= true;
```



---

**Note:** View [Sample](#) in GitHub

---

#### Disable up-down button

You can disable the updown button by setting the [IsEnabledRepeatButton](#) property as `false`. The default value of [IsEnabledRepeatButton](#) property is `true`.

#### XML

```
<syncfusion:DateTimeEdit IsEnabledRepeatButton="False"
    IsVisibleRepeatButton="True"
    Name="dateTimeEdit"/>
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.IsEnabledRepeatButton= false;
dateTimeEdit.IsVisibleRepeatButton= true;
```



---

**Note:** View [Sample](#) in GitHub

---

#### Show only the up-down button

You can show only the updown button by hiding the dropdown button. You can hide the dropdown button by setting the [IsButtonPopUpEnabled](#) property value as `false`.

#### XML

```
<syncfusion:DateTimeEdit IsButtonPopUpEnabled="False"
    IsVisibleRepeatButton="True"
    IsEnabledRepeatButton="True"
    Name="dateTimeEdit"/>
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.IsVisibleRepeatButton= true;
dateTimeEdit.IsEnabledRepeatButton= true;
dateTimeEdit.IsButtonPopUpEnabled = false;
```



---

**Note:** View [Sample](#) in GitHub

---

#### Customize up-down appearance

You can change the background and UI of the updown button.

#### Change updown background

If you want to change the updown button background, use the [RepeatButtonBackground](#) property. You can also change the margin of the up and down button separately by using the

[DownRepeatButtonMargin](#) and [UpRepeatButtonMargin](#) properties. The default value of RepeatButtonBackground property is Gainsboro and UpRepeatButtonMargin & DownRepeatButtonMargin properties is {0,0,0,0}.

### XML

```
<syncfusion:DateTimeEdit RepeatButtonBackground="Red"
DownRepeatButtonMargin="1"
UpRepeatButtonMargin="1"
IsVisibleRepeatButton="True"
Width="150"
Height="25"
Name="dateTimeEdit"/>
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.RepeatButtonBackground= Brushes.Red;
dateTimeEdit.DownRepeatButtonMargin= new Thickness(1);
dateTimeEdit.UpRepeatButtonMargin = new Thickness(1);
dateTimeEdit.IsVisibleRepeatButton= true;
```



**Note:** View [Sample](#) in GitHub

*Custom UI for up and down button*

You can customize the up and down button appearance separately by using the [UpRepeatButtonTemplate](#) and [DownRepeatButtonTemplate](#) properties.

### XML

```
<syncfusion:DateTimeEdit IsVisibleRepeatButton="True"
Name="dateTimeEdit">
  <!--Custom UI for up button-->
  <syncfusion:DateTimeEdit.UpRepeatButtonTemplate>
    <ControlTemplate>
      <TextBlock Foreground="Yellow"
        TextAlignment="Center"
        FontSize="15"
        FontWeight="ExtraBold"
        Text="+"
        Background="Green"/>
    </ControlTemplate>
  </syncfusion:DateTimeEdit.UpRepeatButtonTemplate>
  <!--Custom UI for down button-->
  <syncfusion:DateTimeEdit.DownRepeatButtonTemplate>
    <ControlTemplate>
      <TextBlock Foreground="Yellow"
        TextAlignment="Center"
        FontSize="15"
        FontWeight="ExtraBold"
        Text="-"
        Background="Red"/>
    </ControlTemplate>
  </syncfusion:DateTimeEdit.DownRepeatButtonTemplate>
</syncfusion:DateTimeEdit>
```

```
</syncfusion:DateTimeEdit.DownRepeatButtonTemplate>
</syncfusion:DateTimeEdit>
```



**Note:** View [Sample](#) in GitHub

### Change month using alpha keys

You can change the month which is in abbreviated form or digit form, can be edited by select the month field and pressing the initial letter of respective month. You can enable it by setting the [EnableAlphaKeyNavigation](#) value as `true` and `CanEdit` property as `false`. The default value of `EnableAlphaKeyNavigation` property is `false`.

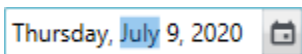
For example, If you pressing the `j` key, then it will selects `January`. On subsequent presses of the `j` key selects `June` and then `July`.

### XML

```
<syncfusion:DateTimeEdit EnableAlphaKeyNavigation="True"
CanEdit="False"
Pattern="LongDate"
Name="dateTimeEdit"/>
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.EnableAlphaKeyNavigation = true;
dateTimeEdit.CanEdit = false;
dateTimeEdit.Pattern = DateTimePattern.LongDate;
```



**Note:** View [Sample](#) in GitHub

### Delete and edit the date time value

By default, date time field will be selected and you can only override the date time field value without deleting the old value. You can delete the existing field values by pressing the `BackSpace` or `Delete` key and enter the new values. You can enable it by setting the [EnableBackspaceKey](#) and [EnableDeleteKey](#) properties as `true`. The default value of `EnableBackspaceKey` and `EnableDeleteKey` property is `false`.

### XML

```
<syncfusion:DateTimeEdit EnableBackspaceKey="True"
EnableDeleteKey="True"
Name="dateTimeEdit"/>
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.EnableBackspaceKey = true;
dateTimeEdit.EnableDeleteKey = true;
```





**Note:** View [Sample](#) in GitHub

### Change date time using custom calendar and clock

You can add your own calendar and clock control in `DateTimeEdit` to selected the date and time.

#### Custom calendar

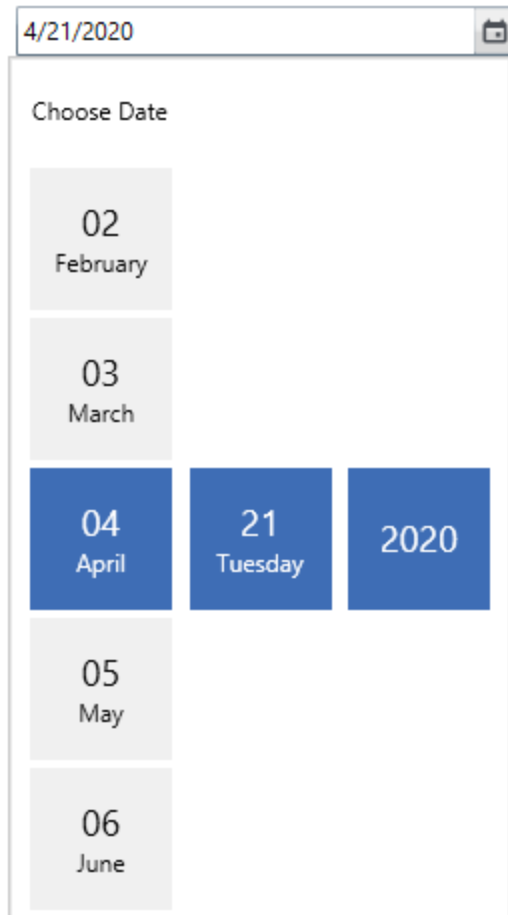
If you want to select the Date from the own calendar, use the [DateTimeCalender](#) property and set the custom date selector. You can enable the calendar by setting [DropDownView](#) property as `Calendar`.

#### C#

```
//ViewModel.cs
class ViewModel : NotificationObject {
private DateTime selectedDate = DateTime.Now;
public DateTime SelectedDate {
get {
return selectedDate;
}
set {
selectedDate = value;
this.RaisePropertyChanged(nameof(SelectedDate));
}
}
}
```

#### XML

```
<Window.DataContext>
<local:ViewModel/>
</Window.DataContext>
<syncfusion:DateTimeEdit DateTime="{Binding SelectedDate, Mode=TwoWay}"
DropDownView="Calendar"
Pattern="ShortDate"
Name="dateTimeEdit">
<syncfusion:DateTimeEdit.DateTimeCalender>
<syncfusion:SfDateSelector SelectorItemWidth="80"
SelectorItemHeight="80"
ShowDoneButton="False"
SelectedDateTime="{Binding SelectedDate, Mode=TwoWay}"/>
</syncfusion:DateTimeEdit.DateTimeCalender>
</syncfusion:DateTimeEdit>
```



---

**Note:** View [Sample](#) in GitHub

---

#### *Custom clock*

If you want to select the time from the own clock, use the [Clock](#) property and set the custom time selector. You can enable the custom clock by setting `DropDownView` property as `Clock`.

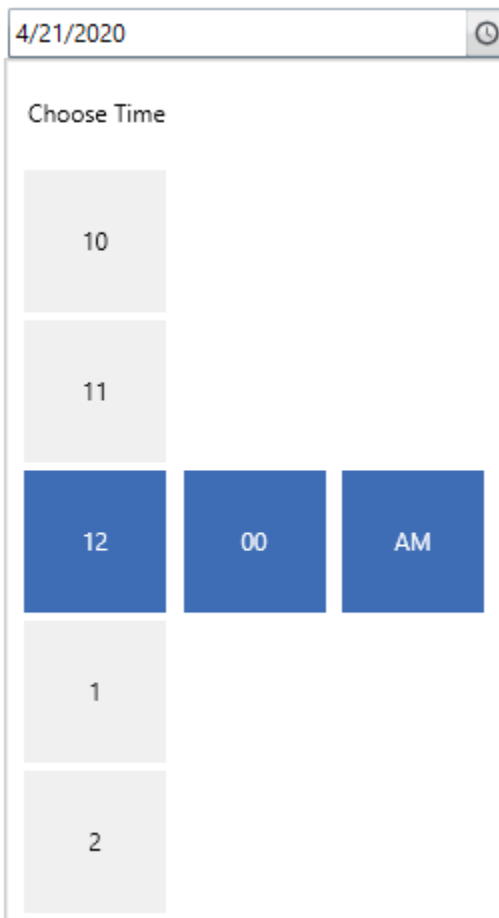
#### **C#**

```
//ViewModel.cs
class ViewModel : NotificationObject {
    private DateTime selectedTime = DateTime.Now;
    public DateTime SelectedTime {
        get {
            return selectedTime;
        }
        set {
            selectedTime = value;
            this.RaisePropertyChanged(nameof(SelectedTime));
        }
    }
}
```

#### **XML**

```
<syncfusion:DateTimeEdit DateTime="{Binding SelectedTime, Mode=TwoWay}"
```

```
DropDownView="Clock"  
Pattern="ShortDate"  
Name="dateTimeEdit">  
<syncfusion:DateTimeEdit.Clock>  
<syncfusion:SfTimeSelector SelectorItemWidth="70"  
SelectorItemHeight="70"  
ShowCancelButton="False"  
ShowDoneButton="False"  
SelectedTime="{Binding SelectedTime, Mode=TwoWay}"/>  
</syncfusion:DateTimeEdit.Clock>  
</syncfusion:DateTimeEdit>
```



**Note:** View [Sample](#) in GitHub

**Note:** You can also use both custom clock and calendar in the full datetime pattern by setting the `DropDownView` property as `Combined`. View [Sample](#) in GitHub

### Setting the null value

If you want to set or change the null value for the `DateTimeEdit`, use the `[NullValue]()` property. you can enable it only by setting the `IsEmptyDateEnabled` property as `true` and `ShowMaskOnNullValue` property as `false`. If `IsEmptyDateEnabled` property is `false`, then the default date time is displayed. If the `ShowMaskOnNullValue` property true, then datetime mask is shown.

### XML

```
<syncfusion:DateTimeEdit ShowMaskOnNullValue="False"
NullValue="{x:Null}"
IsEmptyDateEnabled="True"
x:Name="dateTimeEdit"
Height="25"
Width="200" />
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.ShowMaskOnNullValue= false;
dateTimeEdit.IsEmptyDateEnabled= true;
dateTimeEdit.NullValue = null;
```



**Note:** View [Sample](#) in GitHub

#### Show mask on null value

you can show only the mask on null value by setting the `ShowMaskOnNullValue` property value as `true`.

### XML

```
<syncfusion:DateTimeEdit ShowMaskOnNullValue="True"
NullValue="{x:Null}"
IsEmptyDateEnabled="True"
Name="dateTimeEdit" />
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.ShowMaskOnNullValue= true;
dateTimeEdit.IsEmptyDateEnabled= true;
dateTimeEdit.NullValue = null;
```



#### Show watermark when value is null

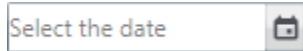
If you want to display any watermark text instead of null value, use the [NoneDateText](#) property to setting the watermark text. you can enable it only by setting the `IsEmptyDateEnabled` property as `true`, `ShowMaskOnNullValue` property as `false` and `NullValue` property as `null`. The default value of `NoneDateText` property is `string.Empty`.

### XML

```
<syncfusion:DateTimeEdit NoneDateText="Select the date"
ShowMaskOnNullValue="True"
NullValue="{x:Null}"
IsEmptyDateEnabled="True"
Name="dateTimeEdit" />
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();  
dateTimeEdit.NoneDateText= "Select the date";  
dateTimeEdit.ShowMaskOnNullValue= true;  
dateTimeEdit.IsEmptyDateEnabled= true;  
dateTimeEdit.NullValue = null;
```



**Note:** View [Sample](#) in GitHub

#### Select the date time field on focus

By default, the first field of the date time is selected to edit when control got focus. you can set the last field of date time as selected field when control got focus by setting the [OnFocusBehavior](#) property as [CursorAtEnd](#). The default value of [OnFocusBehavior](#) property is [CursorOnFirstCharacter](#).

**Note:** If control got focus by directly clicking the date time field, [OnFocusBehavior](#) property value not take effects.

### XML

```
<syncfusion:DateTimeEdit OnFocusBehavior="CursorAtEnd"  
Name="dateTimeEdit" />
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();  
dateTimeEdit.OnFocusBehavior = OnFocusBehavior.CursorAtEnd;
```

OnFocusBehavior = "CursorAtEnd"



OnFocusBehavior = "CursorOnFirstCharacter"



**Note:** View [Sample](#) in GitHub

#### Restrict automatic focus to next field

You can prevent the focus in [DateTimeEdit](#) automatically moving from one field to another by setting the [AutoForwarding](#) property value as [false](#). By default, the value of [AutoForwarding](#) property is [true](#).

### XML

```
<syncfusion:DateTimeEdit x:Name="dateTimeEdit" AutoForwarding="False" />
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();  
dateTimeEdit.AutoForwarding = false;
```

### Value Changed Notification

The selected datetime changed in `DateTimeEdit` can be examined using [DateTimeChanged](#) event. The `DateTimeChanged` event contains the old and newly selected date time values in the `OldValue` and `NewValue` properties.

#### XML

```
<syncfusion:DateTimeEdit DateTimeChanged="dateTimeEdit_DateTimeChanged"
Name="dateTimeEdit"/>
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.DateTimeChanged += dateTimeEdit DateTimeChanged;
```

You can handle the event as follows,

#### C#

```
private void dateTimeEdit_DateTimeChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
    //Get old and new values
    var oldValue = e.OldValue;
    var newValue = e.NewValue;
}
```

### Restricting date value in WPF DateTimePicker (DateTimeEdit)

This section explains how to select a date and time in a particular range by specifying minimum and maximum dates and formatting the datetime in [DateTimeEdit](#) control.

#### Restrict the datetime within minimum and maximum datetime

The selecting datetime in `DateTimeEdit` can be restricted within the maximum and minimum time span limits. Once the selected time has reached the minimum or maximum time span limits, the selected time does not exceed the limit. You can change the minimum and maximum time span limits by using [MinDateTime](#) and [MaxDateTime](#) properties

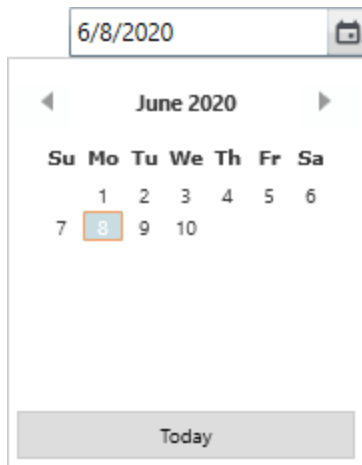
**Note:** The `MaxDateTime` should be greater than `MinDateTime` of the `DateTimeEdit`. If the `MinDateTime` property is greater than the new `MaxDateTime`, then the `MinDateTime` will be reset to the `MaxDateTime`.

#### XML

```
<!--Setting date range -->
<syncfusion:DateTimeEdit MinDateTime="06/1/2020"
MaxDateTime="06/10/2020"
Name="dateTimeEdit"/>
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.MinDateTime = new DateTime(2020, 06, 01);
dateTimeEdit.MaxDateTime = new DateTime(2020, 06, 10);
```



**Note:** View [Sample](#) in GitHub

#### *Minimum and maximum value change notification*

The `DateTimeEdit` notifies that the minimum and maximum value is changed through the `MinDateTimeChanged` and `MaxDateTimeChanged` event. You can use the `OldValue` and `NewValue` properties to get the old and new minimum / maximum date time value.

#### **C#**

```
<syncfusion:DateTimeEdit
MinDateTimeChanged="DateTimeEdit_MinDateTimeChanged"
MaxDateTimeChanged="DateTimeEdit_MaxDateTimeChanged"
Name="dateTimeEdit"/>
```

#### **C#**

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.MinDateTimeChanged += DateTimeEdit_MinDateTimeChanged;
dateTimeEdit.MaxDateTimeChanged += DateTimeEdit_MaxDateTimeChanged;
```

You can handle the event as follows,

#### **C#**

```
private void DateTimeEdit_MinDateTimeChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
    //Get old and new minimum date values
    var oldMinDate = e.OldValue;
    var newMinDate = e.NewValue;
}
private void DateTimeEdit_MaxDateTimeChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
    //Get old and new maximum date values
    var oldMaxDate = e.OldValue;
    var newMaxDate = e.NewValue;
}
```

### Restrict date selection

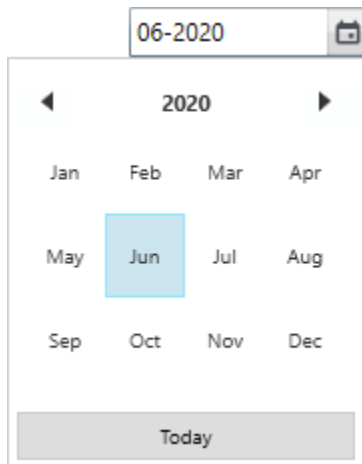
You can restrict the user to select the date from the popup calendar by setting the [DisableDateSelection](#) property value as true. You can select only month and year from the popup calendar. The default value of [DisableDateSelection](#) property is false.

#### XML

```
<syncfusion:DateTimeEdit DisableDateSelection="true"
Pattern="CustomPattern"
CustomPattern="MM-yyyy"
Name="dateTimeEdit"/>
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.DisableDateSelection = true;
dateTimeEdit.Pattern = DateTimePattern.CustomPattern;
dateTimeEdit.CustomPattern = "MM-yyyy";
```



**Note:** View [Sample](#) in GitHub

### Block particular dates

You can restrict the user to select the date within some range by blocking the particular date in the [DateTimeEdit](#). If you try to set the blackout dates as the selected datetime through editing or when pressing down arrow, it will reset the previous valid date from [StartDate](#) of blackout dates in [DateTimeEdit](#). If you try to set the blackout dates as the selected datetime by pressing down arrow, it will reset the next valid date from [EndDate](#) of blackout dates in [DateTimeEdit](#).

**Note:** If you try to set the [DateTime](#) value which contained in blackout dates collection or try to add the selected [DateTime](#) value in blackout dates collection, it throws the [Specified argument was out of the range of valid values](#).

#### XML

```
<syncfusion:DateTimeEdit Loaded="DateTimeEdit_Loaded"
Name="dateTimeEdit"/>
```



### XML

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();  
dateTimeEdit.Loaded += DateTimeEdit_Loaded;
```

### C#

```
//Setting the blackout days  
private void DateTimeEdit_Loaded(object sender, RoutedEventArgs e) {  
    //Setting start and end range for blocking dates  
    DateTime StartDate = new DateTime(DateTime.Now.Year, DateTime.Now.Month, 1);  
    DateTime EndDate = new DateTime(DateTime.Now.Year, DateTime.Now.Month,  
    DateTime.Now.Day - 2);  
    Syncfusion.Windows.Controls.CalendarDateRange blackOutDays = new  
    Syncfusion.Windows.Controls.CalendarDateRange()  
    {  
        Start = StartDate,  
        End = EndDate  
    };  
    Syncfusion.Windows.Controls.Calendar calendar =  
    dateTimeEdit.DateTimeCalendar as Syncfusion.Windows.Controls.Calendar;  
    calendar.BlackoutDates.Add(blackOutDays);  
}
```

5/18/2020



**Note:** View [Sample](#) in GitHub

### ReadOnly support

If you want to restrict the inputs from the user, use the `IsReadOnly` property value as `true`. However, value can be changed programmatically in readonly mode and the user can still select text. The default value of `IsReadOnly` property is `false`.

### XML

```
<syncfusion:DateTimeEdit IsReadOnly="True"  
    DateTime="06/20/2020"  
    Name="dateTimeEdit" />
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();  
dateTimeEdit.IsReadOnly = true;  
dateTimeEdit.DateTime = new DateTime(2020, 06, 20);
```



**Note:** View [Sample](#) in GitHub

### DateTime formatting in WPF DateTimePicker (DateTimeEdit)

you can specify the selected datetime display format in the [WPF DateTimePicker](#) (DateTimeEdit) by using the predefined patterns and custom patterns.

#### Predefined display patterns

You can display the selected date time in the predefined patterns

by setting the pattern value to the [Pattern](#) property. The default value of `Pattern` property is `ShortDate`.

The `DateTimeEdit` control supports the following patterns:

- LongDate
- LongTime
- ShortDate
- ShortTime
- FullDateTime
- MonthDay
- CustomPattern
- ShortableDateTime
- UniversalSortableDateTime
- RFC1123
- YearMonth


<input type="text" value="7/15/2010"/>	<input type="text" value="12:00 AM"/>
ShortDate	ShortTime
<input type="text" value="Thursday, July 15, 2010"/>	<input type="text" value="12:00:00 AM"/>
LongDate	LongTime
<input type="text" value="July 15"/>	<input type="text" value="July 2010"/>
MonthDay	YearMonth
<input type="text" value="Thursday, July 15, 2010 12:00:00 AM"/>	<input type="text" value="Thu, 15 Jul 2010 00:00:00 GMT"/>
FullDateTime	RFC1123
<input type="text" value="2010-07-15T00:00:00"/>	<input type="text" value="2010-07-15 00:00:00Z"/>
SortableDateTime	UniversalSortableDateTime

**XML**

```
<syncfusion:DateTimeEdit Pattern="FullDateTime"
Name="dateTimeEdit" />
```

**C#**

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.Pattern = DateTimePattern.FullDateTime;
```



**Note:** View [Sample](#) in GitHub

*Pattern Changed notification*

The **DateTimeEdit** notifies that the pattern is changed through the [PatternChanged](#) event. You can get the old and new pattern by using the **OldValue** and **NewValue** properties in the **PatternChanged** event.

**C#**

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.PatternChanged += DateTimeEdit_PatternChanged;
private void DateTimeEdit_PatternChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
//Get old and new pattern values
var oldPattern = e.OldValue;
var newPattern = e.NewValue;
}
```

### Change datetime format

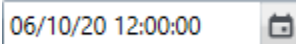
you can change the date time format by using the [DateTimeFormat](#) property. The default value of property is .

#### XML

```
<syncfusion:DateTimeEdit Name="dateTimeEdit">
<syncfusion:DateTimeEdit.DateTimeFormat>
<global:DateTimeFormatInfo ShortDatePattern="MM/dd/yy hh:mm:ss"/>
</syncfusion:DateTimeEdit.DateTimeFormat>
</syncfusion:DateTimeEdit>
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.DateTimeFormat = new DateTimeFormatInfo()
{
    ShortDatePattern = "MM/dd/yy hh:mm:ss"
};
```



**Note:** View [Sample](#) in GitHub

### Custom display pattern

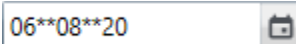
You can also set the custom pattern for displaying the datetime in the `DateTimeEdit` control by using the [CustomPattern](#) property. You can enable the custom display pattern by setting the `Pattern` property value as `CustomPattern`.

#### XML

```
<syncfusion:DateTimeEdit CustomPattern="MM**dd**yy"
Pattern="CustomPattern"
Name="dateTimeEdit" />
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.CustomPattern = "MM**dd**yy";
dateTimeEdit.Pattern = DateTimePattern.CustomPattern;
```



**Note:** View [Sample](#) in GitHub

### Custom pattern Changed notification

The `DateTimeEdit` notifies that the custom pattern is changed through the [CustomPatternChanged](#) event. You can get the old and new custom pattern by using the `OldValue` and `NewValue` properties in the `CustomPatternChanged` event.

#### C#

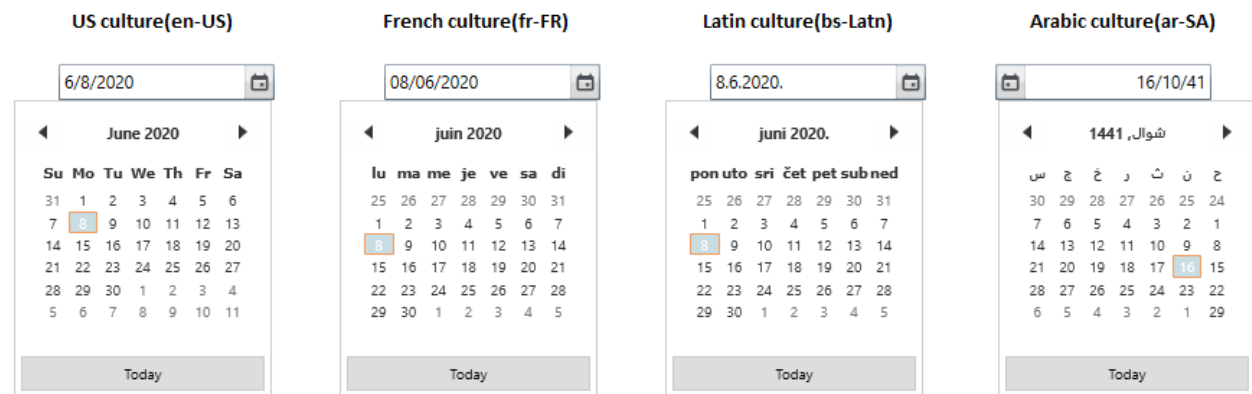
```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
```

```
dateTimeEdit.CustomPatternChanged += dateTimeEdit_CustomPatternChanged;
private void dateTimeEdit_CustomPatternChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
    //Get old and new custom pattern values
    var oldCustomPattern = e.OldValue;
    var newCustomPattern = e.NewValue;
}
```

**Note:** View [Sample](#) in GitHub

### Change culture

By default, the **DateTimeEdit** supports system's current culture. You can change the culture of **DateTimeEdit** by using the [CultureInfo](#) property. Based on the value of **CultureInfo** property, **DateTimeEdit** control elements localized.

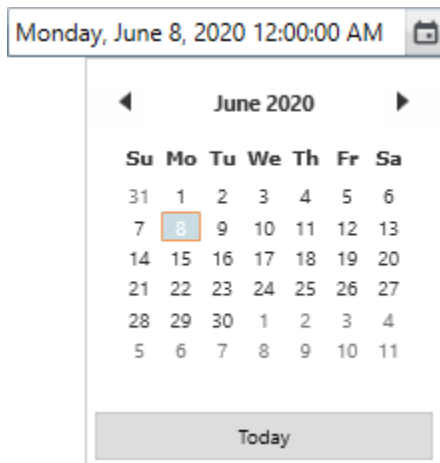


### XML

```
<syncfusion:DateTimeEdit CultureInfo="en-US"
Pattern="FullDateTime"
Name="dateTimeEdit" />
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.CultureInfo = new CultureInfo("fr-FR");
dateTimeEdit.Pattern = DateTimePattern.FullDateTime;
```



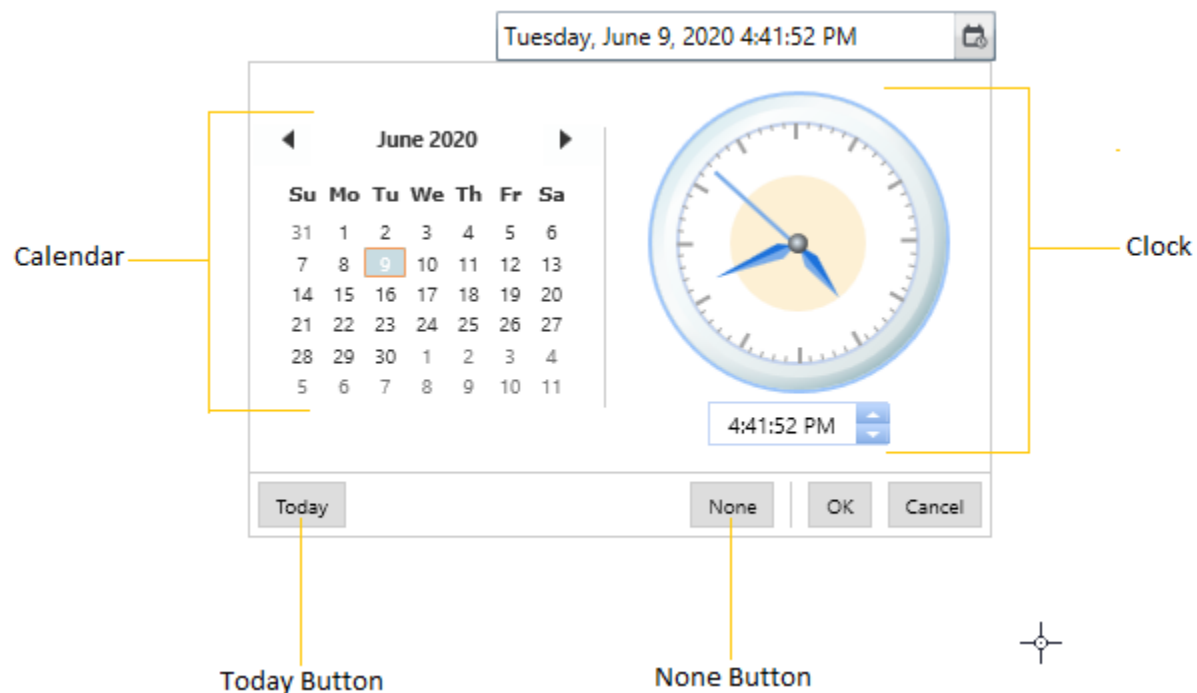
**Note:** View [Sample](#) in GitHub

### Dropdown Popup in WPF DateTimePicker (DateTimeEdit)

You can display the dropdown popup by clicking the dropdown button. You can select the datetime by using the dropdown popup calendar and clock. The dropdown pop-up in [DateTimeEdit](#) controls contains the following parts:

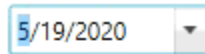
- Calendar
- Clock
- Today Button
- None Button

DropDown\_Structure



Open and close the datetime selector popup using short-cut keys

You can open or close the popup datetime selector popup by pressing the **Alt + Down** key and **F4** key.



### Dropdown date time selector

You can select the date and time using **Calendar** and **Clock**. By default, the **Calendar** is displayed in the drop down popup.

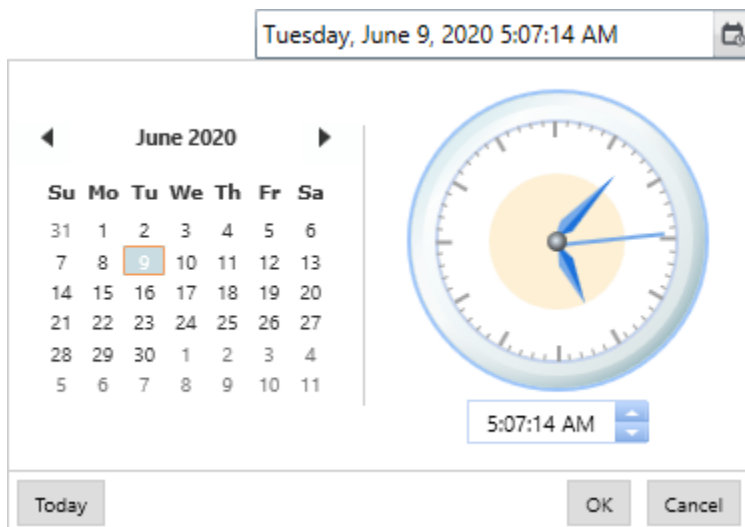
You can use the [DropDownView](#) property to display either **Calendar** or **Clock** or both in the drop down to select the datetime.

### XML

```
<syncfusion:DateTimeEdit DropDownView="Combined"
Pattern="FullDateTime"
Name="dateTimeEdit"/>
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.DropDownView = DropDownViews.Combined;
dateTimeEdit.Pattern = DateTimePattern.FullDateTime;
```



---

**Note:** View [Sample](#) in GitHub

---

#### Custom drop down date time selector

You can add your own calendar and clock control in `DateTimeEdit` drop down to selected the date and time.

Please refer the [Change date time using custom calendar and clock](#) topic to know more about the custom calendar and clock.

---

**Note:** View [Sample](#) in GitHub

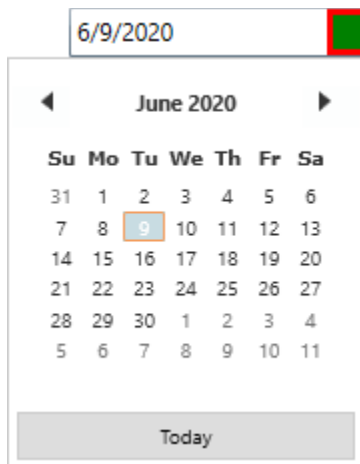
---

#### Custom UI for drop down button

You can customize the dropdown button appearance by using the [DropDownButtonTemplate](#) property.

#### XML

```
<syncfusion:DateTimeEdit DropDownView="Calendar"
Name="dateTimeEdit">
<syncfusion:DateTimeEdit.DropDownButtonTemplate>
<ControlTemplate>
<Border Background="Red">
<Border Background="Green"
Margin="3"/>
</Border>
</ControlTemplate>
</syncfusion:DateTimeEdit.DropDownButtonTemplate>
</syncfusion:DateTimeEdit>
```



---

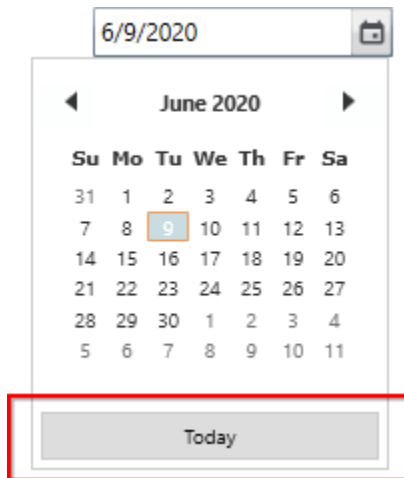
**Note:** View [Sample](#) in GitHub

---

#### Select today date

You can easily select the today date by clicking the dropdown popup `Today` button.





*Select today date and time*

You can select the today date and time by setting the [TodayButtonAction](#) property as `DateTime` and clicking the dropdown popup `Today` button. The default value of `TodayButtonAction` property is `Date`.

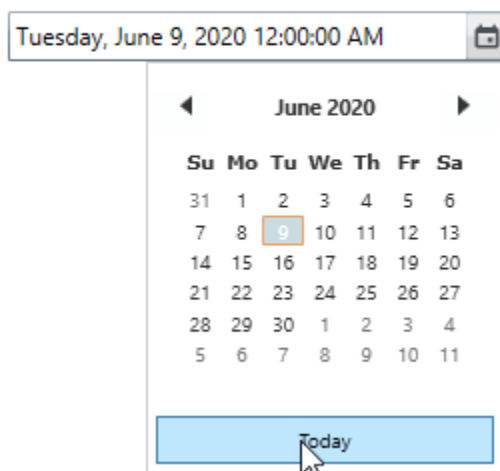
#### XML

```
<syncfusion:DateTimeEdit TodayButtonAction="DateTime"
Pattern="FullDateTime"
Name="dateTimeEdit"/>
```

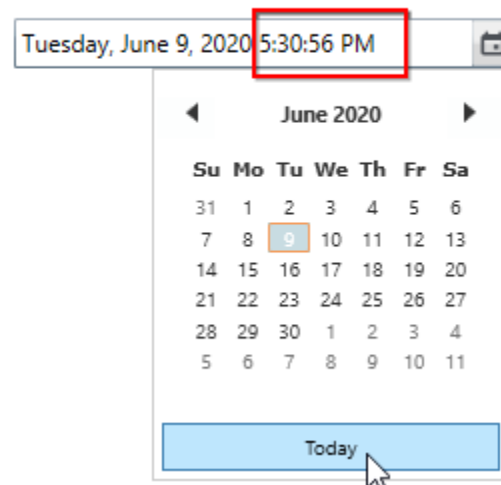
#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.TodayButtonAction = TodayButtonAction.DateTime;
dateTimeEdit.Pattern = DateTimePattern.FullDateTime;
```

TodayButtonAction = "Date"



TodayButtonAction = "DateTime"



**Note:** View [Sample](#) in GitHub

### Reset the selected date

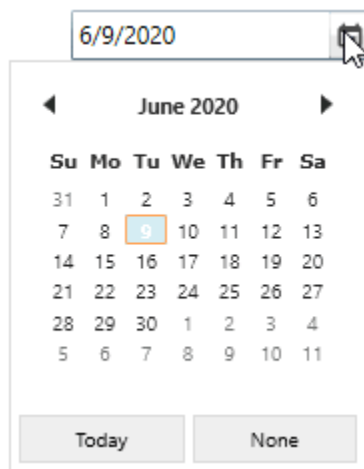
You can easily reset the selected datetime value with null value by clicking the dropdown popup **None** button. You can enable it by setting the [IsEmptyDateEnabled](#) property as **true**. The default value of [IsEmptyDateEnabled](#) property is **false**.

### XML

```
<syncfusion:DateTimeEdit IsEmptyDateEnabled="True"
Name="dateTimeEdit"/>
```

### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.IsEmptyDateEnabled = true;
```



**Note:** View [Sample](#) in GitHub

### Change month names

You can change the popup calendar month names by adding the respective new names to the [AbbreviatedMonthNames](#) property. The default value of [AbbreviatedMonthNames](#) property is **null**.

### XML

```
<syncfusion:DateTimeEdit Name="dateTimeEdit" >
<syncfusion:DateTimeEdit.AbbreviatedMonthNames>
<x:Array Type="sys:String"
xmlns:sys="clr-namespace:System;assembly=mscorlib">
<sys:String>[1] Jan</sys:String>
<sys:String>[2] Feb</sys:String>
<sys:String>[3] Mar</sys:String>
<sys:String>[4] Apr</sys:String>
<sys:String>[5] May</sys:String>
<sys:String>[6] Jun</sys:String>
<sys:String>[7] Jul</sys:String>
<sys:String>[8] Aug</sys:String>
<sys:String>[9] Sep</sys:String>
<sys:String>[10] Oct</sys:String>
<sys:String>[11] Nov</sys:String>
<sys:String>[12] Dec</sys:String>
```

```

</x:Array>
</syncfusion:DateTimeEdit.AbbreviatedMonthNames>
</syncfusion:DateTimeEdit>

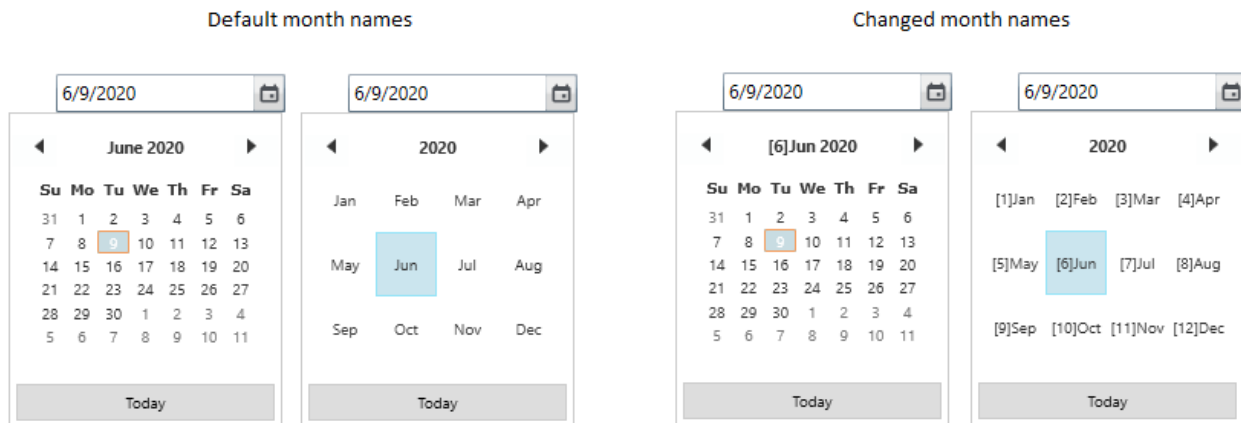
```

## C#

```

DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.AbbreviatedMonthNames = new string[]
{
    "[1] Jan",
    "[2] Feb",
    "[3] Mar",
    "[4] Apr",
    "[5] May",
    "[6] Jun",
    "[7] Jul",
    "[8] Aug",
    "[9] Sep",
    "[10] Oct",
    "[11] Nov",
    "[12] Dec"
};

```



**Note:** View [Sample](#) in GitHub

## Change weekday names

You can change the popup calendar week day names by adding the respective new names to the [AbbreviatedMonthNames](#) property. The default value of [AbbreviatedMonthNames](#) property is null.

## XML

```

<syncfusion:DateTimeEdit Name="dateTimeEdit" >
<syncfusion:DateTimeEdit.ShortestDayNames>
<x:Array Type="sys:String"
xmlns:sys="clr-namespace:System;assembly=mscorlib">
<sys:String>Sunday</sys:String>
<sys:String>Monday</sys:String>
<sys:String>Tuesday</sys:String>
<sys:String>Wednesday</sys:String>
<sys:String>Thursday</sys:String>
<sys:String>Friday</sys:String>

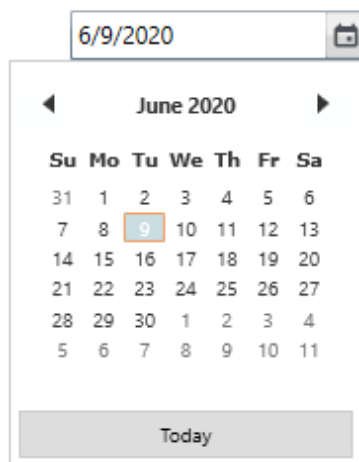
```

```
<sys:String>Saturday</sys:String>
</x:Array>
</syncfusion:DateTimeEdit.ShortestDayNames>
</syncfusion:DateTimeEdit>
```

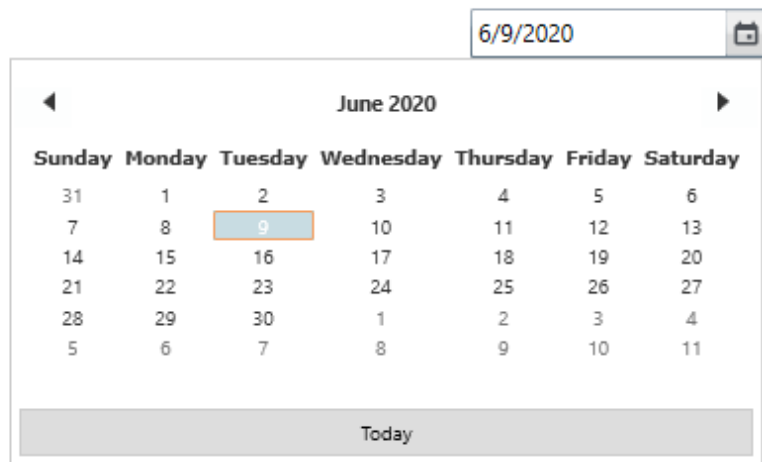
**C#**

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.ShortestDayNames= new string[]
{
    "Sunday",
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday"
};
```

Default weekday names



Changed weekday names



**Note:** View [Sample](#) in GitHub

Open the popup date time selector with time delay

You can open the dropdown popup with some delay after clicking the dropdown button by setting the time span to the [PopupDelay](#) property. The default value of `PopupDelay` property is `{00:00:00}`.

**XML**

```
<syncfusion:DateTimeEdit PopupDelay="0:0:2"
Name="dateTimeEdit"/>
```

**C#**

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.PopupDelay = new TimeSpan(0, 0, 2);
```



### Disable dropdown date time selector

You can restrict the user to select the datetime from the dropdown popup calendar and clock by hiding the dropdown button. You can hide the dropdown button by setting the [IsButtonPopUpEnabled](#) property value as `false`. The default value of `IsButtonPopUpEnabled` property is `true`.

#### XML

```
<syncfusion:DateTimeEdit IsButtonPopUpEnabled="False"
Name="dateTimeEdit"/>
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.IsButtonPopUpEnabled = false;
```



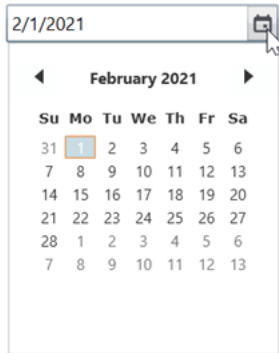
**Note:** View [Sample](#) in GitHub

### Hide Today button

Today button in the dropdown of DateTimeEdit can be collapsed by setting the visibility by retrieving it using code in the loaded event.

#### C#

```
private void DateTimeEdit_Loaded(object sender, RoutedEventArgs e)
{
    Button button = dateTimeEdit.Template.FindName("todayButton", dateTimeEdit)
    as Button;
    if (button != null)
    {
        button.Visibility = Visibility.Collapsed;
    }
}
```



### Appearance in WPF DatePicker (DateTimeEdit)

This section explains different UI customization options available in [DateTimeEdit](#) control.

#### Setting the Foreground

You can change the foreground color for `dateTimeEdit` by setting the `Foreground` property. The default color value of `Foreground` property is `Black`.

##### XML

```
<syncfusion:DateTimeEdit Foreground="Red"
Name="dateTimeEdit"/>
```

##### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.Foreground = Brushes.Red;
```



**Note:** View [Sample](#) in GitHub

#### Setting the Background

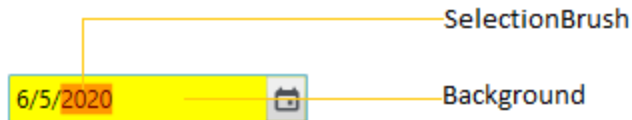
You can change the background color and selection color of `DateTimeEdit` by using the `Background` and `SelectionBrush` property. The default value of `Background` property is `White` and `SelectionBrush` property is `Royal Blue`.

##### XML

```
<syncfusion:DateTimeEdit Background="Yellow"
SelectionBrush="Red"
Name="dateTimeEdit"/>
```

##### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.Background = Brushes.Yellow;
dateTimeEdit.SelectionBrush = Brushes.Red;
```



**Note:** View [Sample](#) in GitHub

#### Change focus border color

You can change the focus border color of the `DateTimeEdit` by setting the [FocusedBorderBrush](#) property. The default value of `FocusedBorderBrush` property is `Medium Aquamarine`.

#### XML

```
<syncfusion:DateTimeEdit FocusedBorderBrush="Red"
Name="dateTimeEdit"/>
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.FocusedBorderBrush = Brushes.Red;
```



**Note:** View [Sample](#) in GitHub

#### Change flow direction

You can change the flow direction of the `DateTimeEdit` layout from right to left by setting the `FlowDirection` property value as `RightToLeft`. The default value of `FlowDirection` property is `LeftToRight`.

#### XML

```
<syncfusion:DateTimeEdit FlowDirection="RightToLeft"
Name="dateTimeEdit"/>
```

#### C#

```
DateTimeEdit dateTimeEdit = new DateTimeEdit();
dateTimeEdit.FlowDirection = FlowDirection.RightToLeft;
```

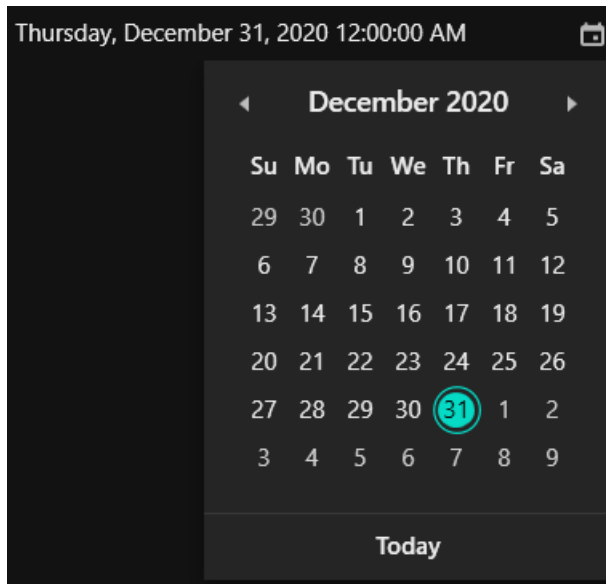


**Note:** View [Sample](#) in GitHub

#### Theme

`DateTimeEdit` supports various built-in themes. Refer to the below links to apply themes for the `DateTimeEdit`,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Localization in WPF DateTimePicker (DateTimeEdit)

Localization is the process of translating the application resources into different language for the specific cultures. You can localize the [DateTimeEdit](#) by [adding resource file](#). Application culture can be changed by setting [CurrentUICulture](#) after `InitializeComponent` method.

Below application culture changed to French.

#### C#

```
public MainWindow()  
{  
    InitializeComponent();  
    System.Threading.Thread.CurrentThread.CurrentUICulture = new  
    System.Globalization.CultureInfo("fr-FR");  
}
```

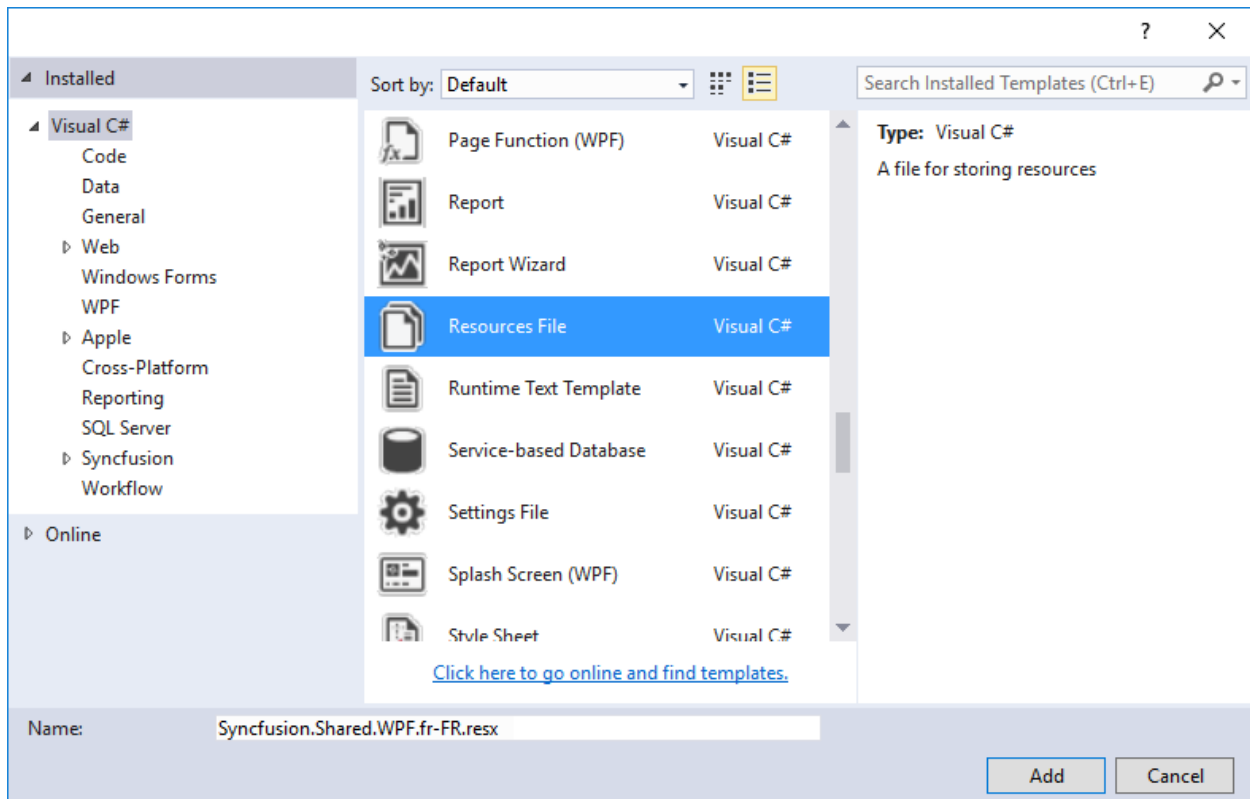
To localize the DateTimeEdit based on [CurrentUICulture](#) using resource files, follow the below steps.

**Step 1:** Create new folder and named as **Resources** in your application.

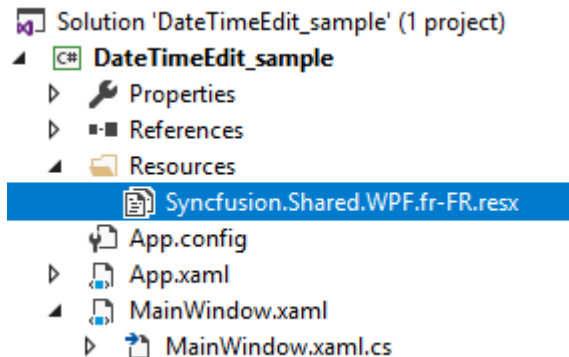
**Step 2:** Right-click on the Resources folder, select **Add** and then **NewItem**.

**Step 3:** In `Add New Item` wizard, select the **Resource File** option and name the filename as **Syncfusion.Shared.Wpf.<culture name>.resx**. For example, you have to give name as **Syncfusion.Shared.WPF.fr-FR.resx** for French culture.

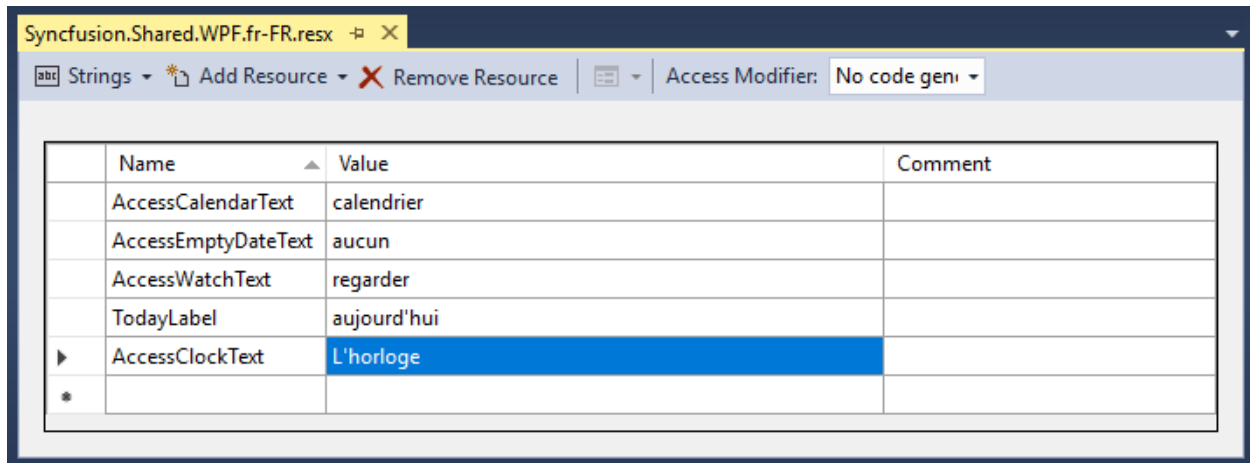




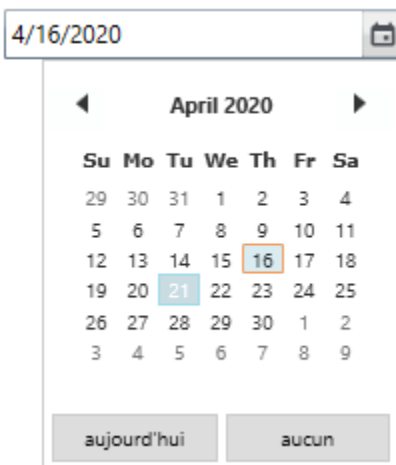
**Step 4:** Now, select **Add** option to add the resource file in **Resources** folder.



**Step 5:** Add the Name/Value pair in Resource Designer of **Syncfusion.Shared.Wpf.fr-FR.resx** file and change its corresponding value to corresponding culture.



The following screenshot shows the localized DateTimeEdit control.



**Note:** View [Sample](#) in GitHub

### UI Automation in WPF DateTimePicker (DateTimeEdit)

Microsoft UI Automation is the new accessibility Framework for Microsoft Windows, available on all operating systems that support Windows Presentation Foundation (WPF). UI Automation provides accessibility to most UI elements and it provides the information about UI element to the end user. You can interact with the UI by using automated test scripts. To know more about UI Automation, refer the MSDN page [here](#).

[DateTimeEdit](#) supports the following types of UI Automation,

1. Quick Test Professional
2. Coded UI

### Quick Test Professional (QTP)

The QTP tool can recognize the `DateTimeEdit` control. Hence with the `DateTimeEdit` control, it is possible to record interactions such as opening the calendar popup, closing the calendar popup, choosing the date from the calendar, setting the DateTime value and and so on, for testing using QTP.

### *Mapping the control to the custom server*

QTP Tool interacts with the control through CustomServerBase extension through the RunInterface. The custom server must be mapped to the control type for the tool to identify the control. A conventional configuration file (\*.cfg) will resemble the one shown in the following code snippet. When the configuration file and the assembly containing the server type are loaded, QTP will recognize the control and record it.

### **XML**

```
<?xml version="1.0" encoding="UTF-8"?>
<Controls>
  <Control Type="Syncfusion.Windows.Shared.DateTimeEdit"
  MappedTo="SfDateTimeEdit">
    <CustomServer>
      <Component>
        <DllName>Syncfusion.QtpDateTimeEdit.Wpf.dll</DllName>
        <TypeName>Syncfusion.Windows.Controls.QTP.DateTimeEditServer</TypeName>
      </Component>
    </CustomServer>
  </Control>
</Controls>
```

This XML configuration file should contain the fully qualified type name of the control, fully qualified name of the custom server and the name of the assembly containing the custom server. MappedTo attribute, points to the name of the ClassInfo (an XML file containing the API information about the RunInterface).

### Coded UI

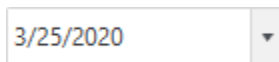
You can refer the Coded UI document from [here](#)

## SfDatePicker

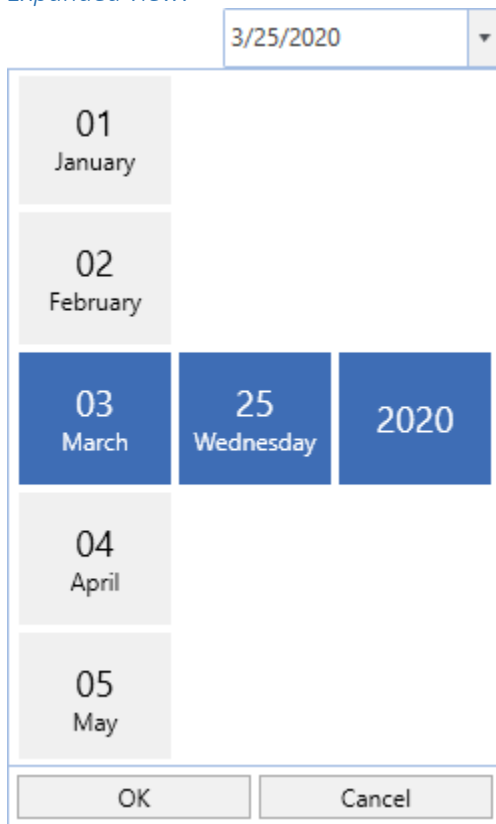
### WPF DatePicker (SfDatePicker) Overview

The [SfDatePicker](#) control allows the user to select date values in a touch friendly manner.

#### *Normal view:*



Expanded view:



#### Key Features

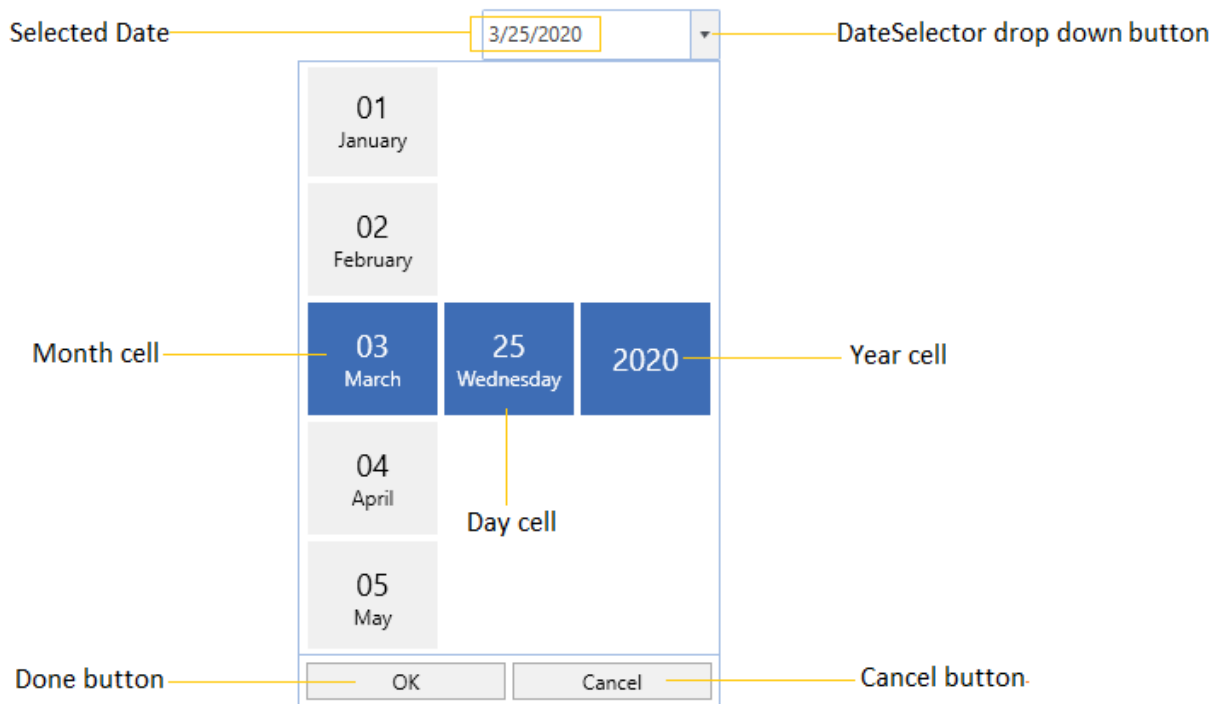
**Formatting** – The Control displays the selected Date value in a various formats.

**Date Selector** – The drop-down portion used for selecting the date can be customized.

#### Getting Started with WPF DatePicker (SfDatePicker)

This section explains how to create a [WPF DatePicker](#) and explains about its structure.

## Structure of SfDatePicker



## Assembly deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

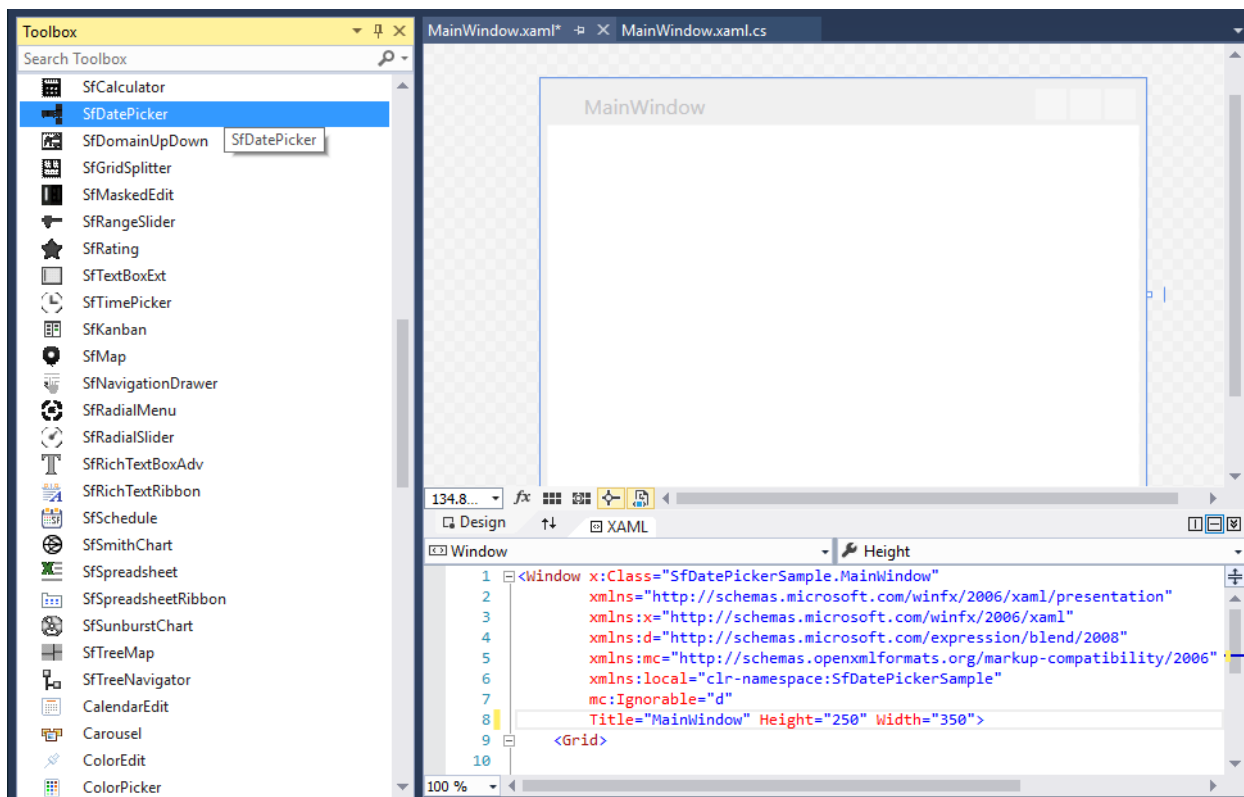
You can find more details about installing the NuGet package in a WPF application in the following link:

[How to install nuget packages](#)

## Add control through designer

The **SfDatePicker** control can be added to an application by dragging it from the toolbox to a designer view. The following required assembly references will be added automatically:

- Syncfusion.SfInput.WPF
- Syncfusion.SfShared.WPF



### Adding control manually in XAML

To add the control manually in XAML, follow the given steps:

1. Add the following required assembly references to the project: *Syncfusion.SfInput.WPF* and *Syncfusion.SfShared.WPF*
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page.
3. Declare the **SfDatePicker** control in the XAML page.

### XAML

```

<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="SfDatePickerSample.MainWindow"
Title="SfDatePicker Sample" Height="350" Width="525">
<Grid>
<!-- Adding SfDatePicker control -->
<syncfusion:SfDatePicker x:Name="sfDatePicker"
Width="200"/>
</Grid>
</Window>

```

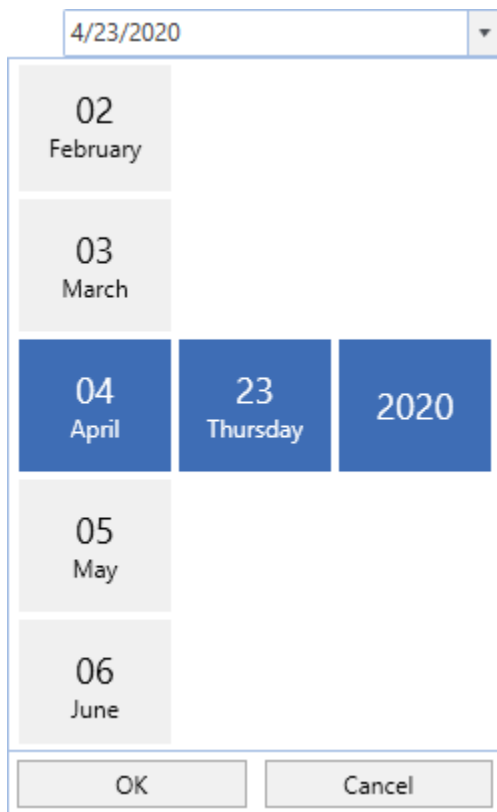
### Add control manually in C#

To add the control manually in C#, follow the given steps:

1. Add the following required assembly references to the project: *Syncfusion.SfInput.WPF* and *Syncfusion.SfShared.WPF*
2. Import the **SfDatePicker** namespace using **Syncfusion.Windows.Controls.Input**;
3. Create an **SfDatePicker** instance, and add it to the window.

### C#

```
using Syncfusion.Windows.Controls.Input;
namespace SfDatePickerSample {
public partial class MainWindow : Window {
public MainWindow() {
InitializeComponent();
//Creating an instance of SfDatePicker control
SfDatePicker sfDatePicker = new SfDatePicker();
//Adding SfDatePicker as window content
this.Content = sfDatePicker;
}
}
}
```



### Setting the Date

We can set or change the selected date by using [Value](#) property. If we not assign any value for the [Value](#) property, it will automatically assign the current system date as [Value](#) property value.

### XML

```
<syncfusion:SfDatePicker Value="5/30/2021"
Name="sfDatePicker" />
```

### C#

```
SfDatePicker sfDatePicker= new SfDatePicker();
sfDatePicker.Value = new DateTime(2021, 5, 30);
```



### Date changed notification

When the selected date of SfDatePicker is changed, it will be notified by using the [ValueChanged](#) event. You can get the details about the checked item in [ItemCheckedEventArgs](#).

- **OldValue** : Gets a date which is previously selected.
- **NewValue** : Gets a date which is currently selected.

### XML

```
<syncfusion:SfDatePicker ValueChanged="Sfdatepicker_ValueChanged"  
Name="sfDatePicker"/>
```

### C#

```
SfDatePicker sfDatePicker = new SfDatePicker();  
sfDatePicker.ValueChanged += Sfdatepicker_ValueChanged;
```

You can handle the event as follows:

### C#

```
private void Sfdatepicker_ValueChanged(DependencyObject d,  
DependencyPropertyChangedEventArgs e) {  
    Console.WriteLine("The Old selected Date: " + e.OldValue.ToString());  
    Console.WriteLine("The Newly selected Date: " + e.NewValue.ToString());  
}
```

### Display the date using the FormatString

We can edit and display the selected date with various formatting like date, month and year formats by using the [FormatString](#) property. The default value of [FormatString](#) property is "d".

### XML

```
<syncfusion:SfDatePicker x:Name="sfDatePicker"  
FormatString="M"/>
```

### C#

```
SfDatePicker sfDatePicker = new SfDatePicker();  
sfDatePicker.FormatString = "M";
```



### Specifying format for the DateSelector

We can allow the user to select the pair of date, month and year selector or any single selector cell from the [SfDateSelector](#) by using the [SelectorFormatString](#) property. The default value of [SelectorFormatString](#) property is "M/d/yyyy" and the date, time and year value selector is enabled in the [SfDateSelector](#).

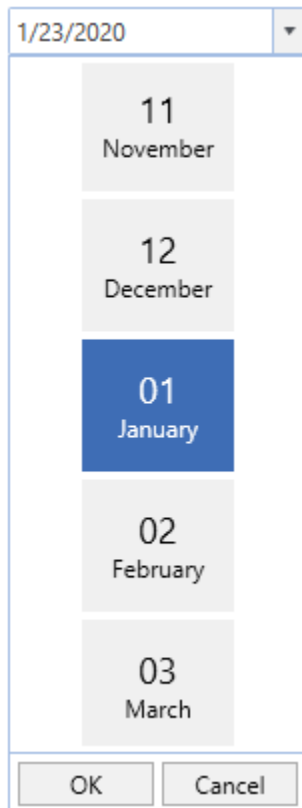


### XML

```
<syncfusion:SfDatePicker x:Name="sfDatePicker"
SelectorFormatString="M"/>
```

### C#

```
SfDatePicker sfDatePicker = new SfDatePicker();
sfDatePicker.SelectorFormatString = "M";
```



Here, we can only able to select the month value from the **SfDateSelector**

Click [here](#) to download the sample that showcases the display date formatting and date selection formatting by the **SfDatePicker**.

#### Set selected value on lost focus

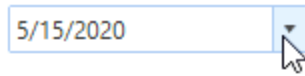
If we want to update the selected date of **SfDateSelector** to the **SfDatePicker.Value** property by moving the focus from **SfDateSelector** to anywhere, use the [SetValueOnLostFocus](#) property value as **true**. By default, the selected date of **SfDateSelector** can be sets to the **SfDatePicker.Value** property only by clicking the **OK** button, otherwise the selected value not updated by the move focus.

### XML

```
<syncfusion:SfDatePicker SetValueOnLostFocus="True"
Name="sfDatePicker" />
```

### C#

```
SfDatePicker sfDatePicker= new SfDatePicker();  
sfDatePicker.SetValueOnLostFocus = true;
```



Click [here](#) to download the sample that showcases the value setting support in the SfDatePicker.

#### Localization support

Localization is the process of translating the application resources into different language for the specific cultures. You can localize the **Ok** and **Cancel** button text in SfDatePicker control by adding resource file for each language.

---

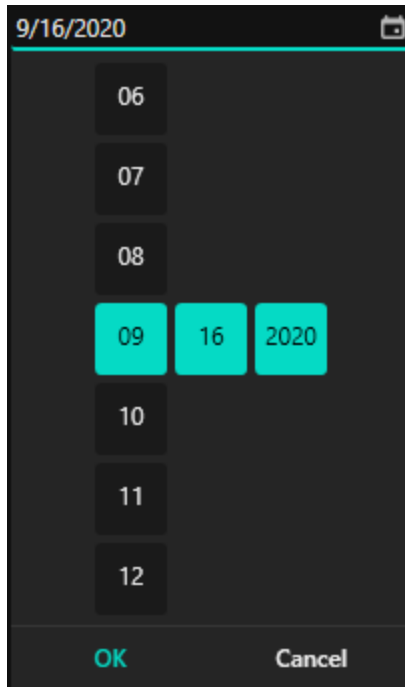
**Note:** Refer [Localization](#) page to know more about how to provide a localization support for the SfDatePicker.

---

#### Theme

SfDatePicker supports various built-in themes. Refer to the below links to apply themes for the SfDatePicker,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Setting Date in WPF DatePicker (SfDatePicker)

We can change the value of [SfDatePicker](#) by using the [SfDateSelector](#) and keyboard interaction.

#### Setting Date using property

We can set or change the selected date by using [Value](#) property. If we not assign any value for the [Value](#) property, it will automatically assign the current system date as [Value](#) property value.

#### XML

```
<syncfusion:SfDatePicker Value="5/30/2021"
Name="sfDatePicker" />
```

#### C#

```
SfDatePicker sfDatePicker= new SfDatePicker();
sfDatePicker.Value = new DateTime(2021, 5, 30);
```



#### Setting Null Value

If we want to set null value for the SfDatePicker, set the [AllowNull](#) property as true and [Value](#) property as null. If [AllowNull](#) property is false, then the current system date is updated in [Value](#) property and displayed instead of null.

#### XML

```
<syncfusion:SfDatePicker AllowNull="True"
Value="{x:Null}"
Name="sfDatePicker" />
```

### C#

```
SfDatePicker sfDatePicker= new SfDatePicker();  
sfDatePicker.AllowNull = true;  
sfDatePicker.Value = null;
```



#### Setting WaterMark text

We can prompt the user with some information by using the [Watermark](#) property. This will apply on when the SfDatePicker contains the [Value](#) property as null and [AllowNull](#) property as true . If [AllowNull](#) property is false, then the current system date is updated in [Value](#) property and displayed instead of [Watermark](#) text.

### XML

```
<syncfusion:SfDatePicker Watermark="Select the Date"  
AllowNull="True"  
Value="{x:Null}"  
Name="sfDatePicker" >  
</syncfusion:SfDatePicker>
```

### C#

```
SfDatePicker sfDatePicker = new SfDatePicker();  
sfDatePicker.Watermark = "Select the Date";  
sfDatePicker.AllowNull = true;  
sfDatePicker.Value = null;
```

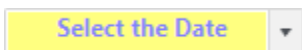


#### Setting WaterMark Template

We can change the template of the [Watermark](#) by using the [WatermarkTemplate](#) property.

### XML

```
<syncfusion:SfDatePicker Name="sfDatePicker"  
AllowNull="True"  
Value="{x:Null}"  
Watermark="Select the Date" >  
  <syncfusion:SfDatePicker.WatermarkTemplate >  
    <DataTemplate>  
      <Border Background="Yellow">  
        <TextBlock Foreground="Blue"  
          FontWeight="Bold"  
          Text="{Binding}"  
          TextAlignment="Center"/>  
      </Border>  
    </DataTemplate>  
  </syncfusion:SfDatePicker.WatermarkTemplate>  
</syncfusion:SfDatePicker>
```



### Set selected value on lost focus

If we want to update the selected date of `SfDateSelector` to the `SfDatePicker.Value` property by moving the focus from `SfDateSelector` to anywhere, use the [SetValueOnLostFocus](#) property value as `true`. By default, the selected date of `SfDateSelector` can be sets to the `SfDatePicker.Value` property only by clicking the `OK` button, otherwise the selected value not updated by the move focus.

### XML

```
<syncfusion:SfDatePicker SetValueOnLostFocus="True"
Name="sfDatePicker" />
```

### C#

```
SfDatePicker sfDatePicker= new SfDatePicker();
sfDatePicker.SetValueOnLostFocus = true;
```



### Setting the date using editing

If we want to perform the validation after the user completely entering their date inputs, use the [AllowInlineEditing](#) property value as `true`. Then the entered date value is validated with the [FormatString](#) property value by pressing the `Enter` key or lost focus. If entered value is not suit with `FormatString` property, the selected date will be set as default format value.

By default, the user entering each input numbers are automatically validated with the `FormatString` formats and assigned the proper value for it, then it will move to next input part of the date format.

### XML

```
<syncfusion:SfDatePicker Name="sfDatePicker"
AllowInlineEditing="True" />
```

### C#

```
SfDatePicker sfDatePicker= new SfDatePicker();
sfDatePicker.AllowInlineEditing = true;
```

Selected Date : 4/7/2020



### Setting the Input Scope for the On-Screen Keyboard

We can change the input type of the on-screen keyboard by using the [InputScope](#) property. When the [InputScope](#) property set to [Number](#), only the numeric keypad will be visible in the on-screen keyboard.

**Note:** The [AllowInlineEditing](#) property must be set to [True](#) for this property to take effect.

### XML

```
<syncfusion:SfDatePicker Name="sfDatePicker"
AllowInlineEditing="True"
InputScope="Number" />
```

### C#

```
SfDatePicker sfDatePicker= new SfDatePicker();
sfDatePicker.AllowInlineEditing = true;
sfDatePicker.InputScope = InputScopeNameValue.Date;
```



### Restrict selecting date limit

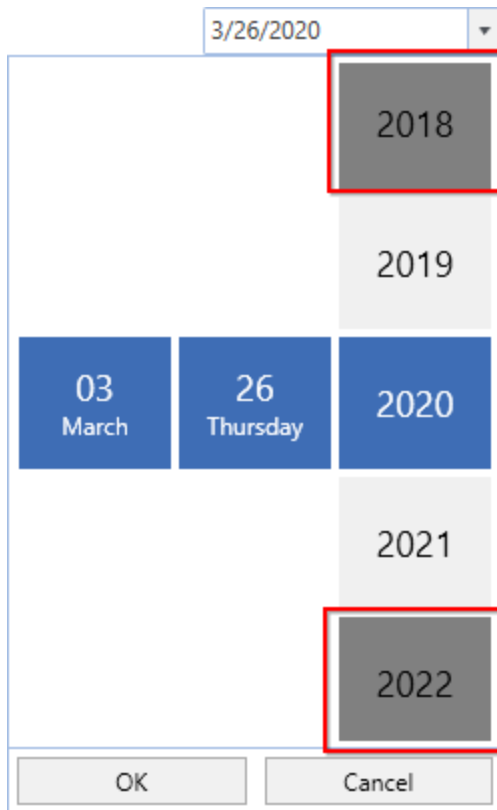
we can restrict the user to select a date in the specific date limit by setting the value for the [MinDate](#) and [MaxDate](#) properties. If we assign the value for the [Value](#) property lower than [MinDate](#), then [MinDate](#) will be the selected date. If we assign the value for the [Value](#) property higher than [MaxDate](#), then [MaxDate](#) will be the selected date.

### XML

```
<syncfusion:SfDatePicker MinDate="1/1/2020"
MaxDate="6/30/2020"
Name="sfDatePicker" />
```

### C#

```
SfDatePicker sfDatePicker = new SfDatePicker();
sfDatePicker.MinDate = new DateTime(2020, 1, 1);
sfDatePicker.MaxDate = new DateTime(2020, 6, 30);
```



Here, the users can select the year from 2019 to 2021 only.

#### Date changed notification

When the selected date of SfDatePicker is changed, it will be notified by using the [ValueChanged](#) event. You can get the details about the checked item in [ItemCheckedEventArgs](#).

- **OldValue** : Gets a date which is previously selected.
- **newValue** : Gets a date which is currently selected.

#### XML

```
<syncfusion:SfDatePicker ValueChanged="Sfdatepicker_ValueChanged"
Name="sfDatePicker"/>
```

#### C#

```
SfDatePicker sfDatePicker = new SfDatePicker();
sfDatePicker.ValueChanged += Sfdatepicker_ValueChanged;
```

You can handle the event as follows:

#### C#

```
private void Sfdatepicker_ValueChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
Console.WriteLine("The Old selected Date: " + e.OldValue.ToString());
Console.WriteLine("The Newly selected Date: " + e.NewValue.ToString());
}
```

```
}
```

Click [here](#) to download the sample that showcases the input types and selected date with its notification supports.

### Date Formatting in WPF DatePicker (SfDatePicker)

The [SfDatePicker](#) control allows the user to select and display the date in various formats.

Display the date using the `FormatString`

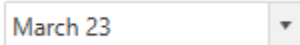
We can edit and display the selected date with various formatting like date, month and year formats by using the [FormatString](#) property. The default value of `FormatString` property is `"d"`.

#### XML

```
<syncfusion:SfDatePicker x:Name="sfDatePicker"
FormatString="M"/>
```

#### C#

```
SfDatePicker sfDatePicker = new SfDatePicker();
sfDatePicker.FormatString = "M";
```



### Specifying format for the DateSelector

We can allow the user to select the pair of date, month and year selector or any single selector cell from the [SfDateSelector](#) by using the [SelectorFormatString](#) property. The default value of `SelectorFormatString` property is `"M/d/yyyy"` and the date, time and year value selector is enabled in the `SfDateSelector`.

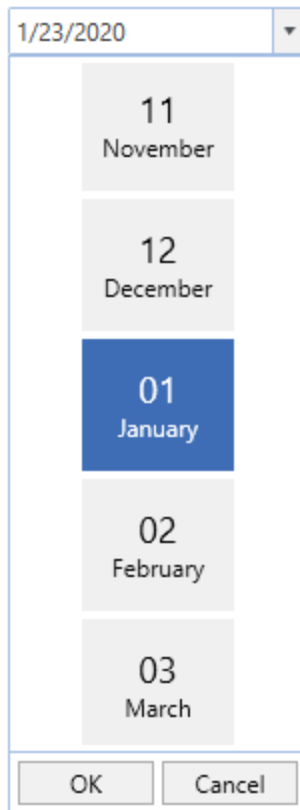
#### XML

```
<syncfusion:SfDatePicker x:Name="sfDatePicker"
SelectorFormatString="M"/>
```

#### C#

```
SfDatePicker sfDatePicker = new SfDatePicker();
sfDatePicker.SelectorFormatString = "M";
```





Here, we can only able to select the month value from the **SfDateSelector**

Click [here](#) to download the sample that showcases the edit, display date formatting and date selection formatting by the **SfDatePicker**.

---

**Note:** A detailed explanation of standard date time formatting is available [here](#). The result string produced by these format specifiers are influenced by the settings in the Regional Options control panel. Computers with different cultures or different date and time settings will generate different result strings.

---

### Customizing DropDown in WPF DatePicker (SfDatePicker)

We can customize the **SfDateSelector** visibility, drop down button visibility and height of the **SfDateSelector**.

#### Change DropDown height

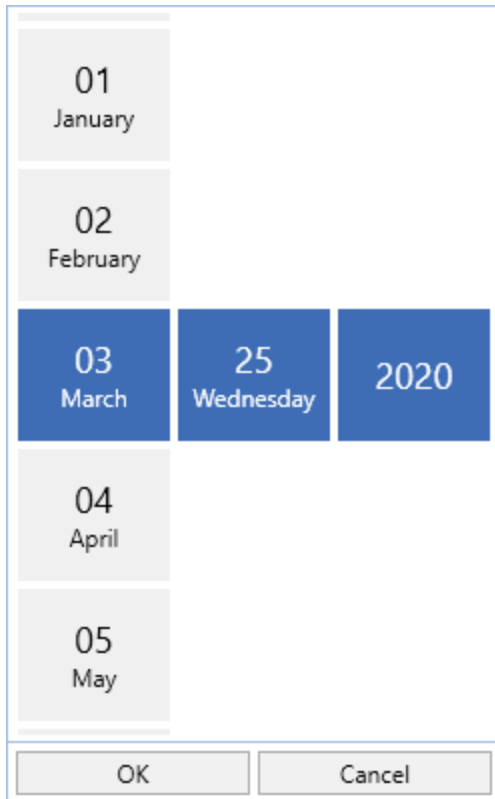
The height of drop down can be changed using **DropDownHeight** property.

#### XML

```
<syncfusion:SfDatePicker DropDownHeight="300"
x:Name="sfDatePicker"/>
```

#### C#

```
SfDatePicker sfDatePicker = new SfDatePicker();
sfDatePicker.DropDownHeight = 300;
```



#### Show or hide DropDown button

If we want to restrict the user to selecting a date from a drop down date selector, we can hide the drop down button by using the [ShowDropDownButton](#) property value as `false`. The default value of `ShowDropDownButton` property is `true`.

#### XML


```
<syncfusion:SfDatePicker ShowDropDownButton="False"
x:Name="sfDatePicker"/>
```

#### C#

```
SfDatePicker sfDatePicker = new SfDatePicker();
sfDatePicker.ShowDropDownButton = false;
```

**ShowDropDownButton = false**    **ShowDropDownButton = true**

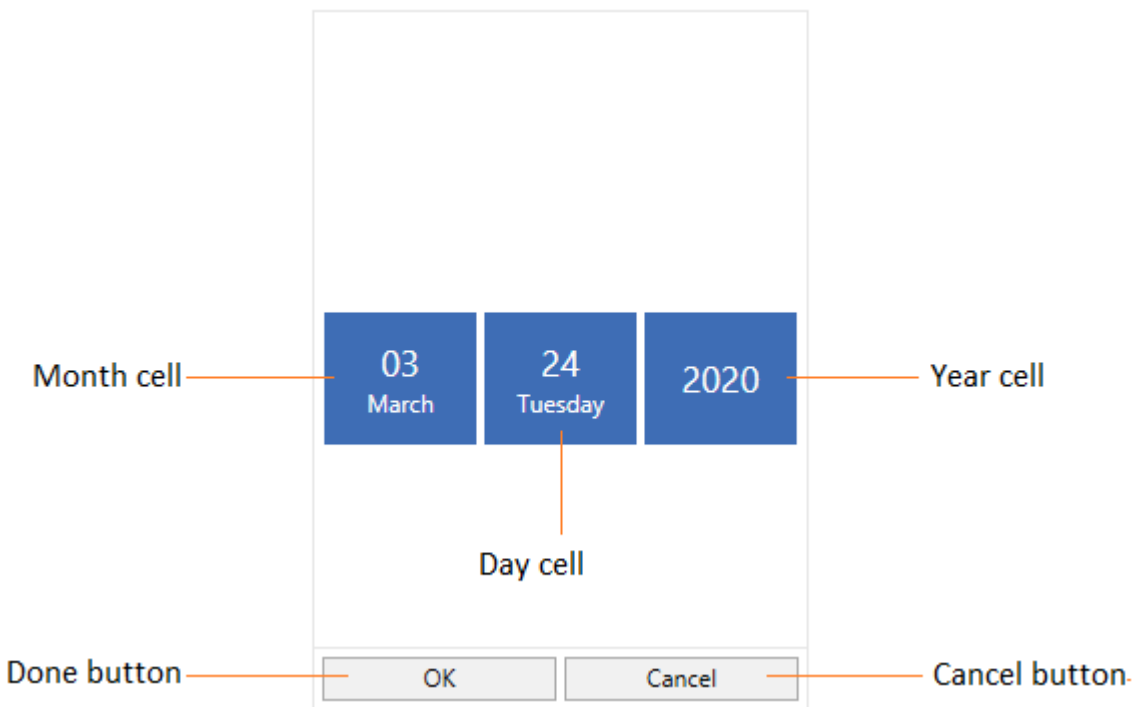
3/25/2020

3/25/2020 

Click [here](#) to download the sample that showcases the DropDown customization support.

#### Date Selector in WPF DatePicker (SfDatePicker)

The [SfDateSelector](#) is a sub-control of [SfDatePicker](#) which opens inside the drop-down popup and used to select the date for the [SfDatePicker](#). It contains the date, month and year selection cells for select the date. The selected date of the [SfDateSelector](#) is assigned to the [SfDatePicker.Value](#) property.



The visual elements of the date selector can be customized using the [SelectorStyle](#) property.

#### Change the Cell templates

We can change the template for the Date, Month or Year selector by using the [DayCellTemplate](#), [MonthCellTemplate](#) or [YearCellTemplate](#) which are available in the [SfDateSelector](#).

**Note:** The DataContext of Month, Date and Year selection cell is [DateTimeWrapper](#).

#### Change the DayCell Template

We can change the day selector template by using the [DayCellTemplate](#) property. In that, we can add like image, icon or text with the day numbers.

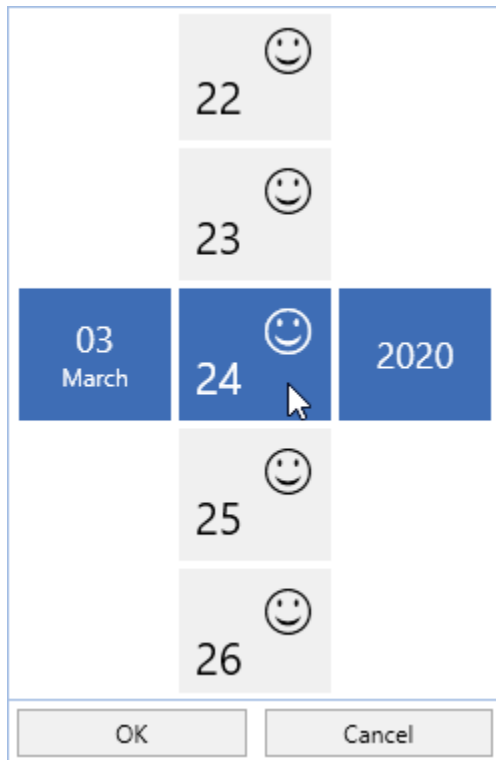
#### XML

```
<syncfusion:SfDatePicker VerticalAlignment="Center"
    HorizontalAlignment="Center"
    Width="200"
    Name="sfDatePicker">
    <syncfusion:SfDatePicker.SelectorStyle>
    <Style TargetType="syncfusion:SfDateSelector">
    <Setter Property="DayCellTemplate">
    <Setter.Value>
    <DataTemplate>
    <Grid>
    <TextBlock VerticalAlignment="Top"
        HorizontalAlignment="Right"
        Margin="5"
        FontSize="22"
        FontFamily="Segoe UI Symbol"
        Text="&#xE170;" />
    <TextBlock Text="{Binding DayNumber}"
        VerticalAlignment="Bottom"
```

```

Margin="5"
FontSize="22"/>
</Grid>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
</syncfusion:SfDatePicker.SelectorStyle>
</syncfusion:SfDatePicker>

```



### Change the MonthCell Template

We can change the month selector template by using the [MonthCellTemplate](#) property. In that, we can add like image, icon or text with the month numbers.

### XML

```

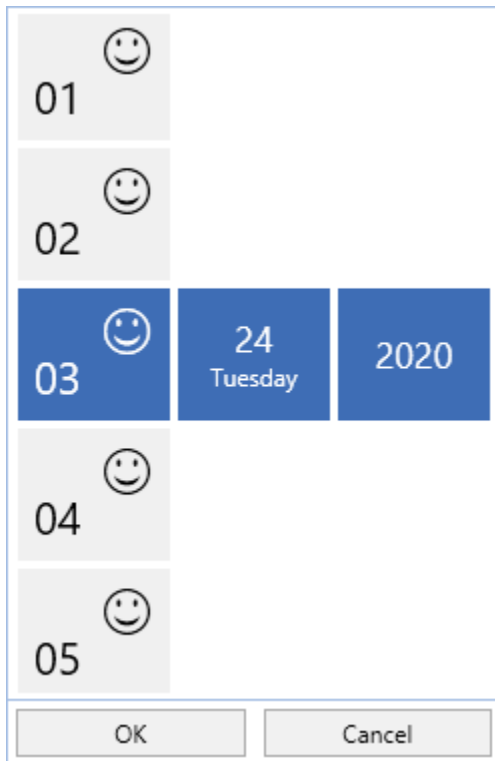
<syncfusion:SfDatePicker VerticalAlignment="Center"
HorizontalAlignment="Center"
Width="200"
Name="sfDatePicker">
<syncfusion:SfDatePicker.SelectorStyle>
<Style TargetType="syncfusion:SfDateSelector">
<Setter Property="MonthCellTemplate">
<Setter.Value>
<DataTemplate>
<Grid>
<TextBlock VerticalAlignment="Top"
HorizontalAlignment="Right"
Margin="5"
FontSize="22"

```

```

FontFamily="Segoe UI Symbol"
Text="&#xE170;" />
<TextBlock Text="{Binding MonthNumber}"
VerticalAlignment="Bottom"
Margin="5"
FontSize="22" />
</Grid>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
</syncfusion:SfDatePicker.SelectorStyle>
</syncfusion:SfDatePicker>

```



### Change the YearCell Template

We can change the year selector template by using the [YearCellTemplate](#) property. In that, we can add like image, icon or text with the year numbers.

### XML

```

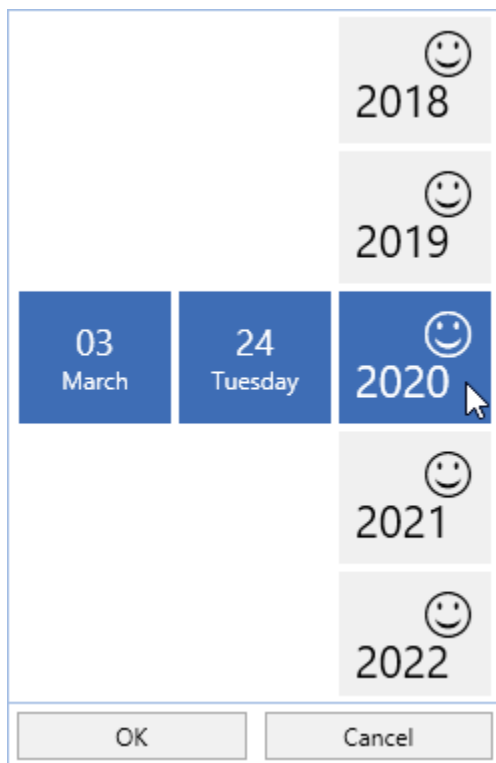
<syncfusion:SfDatePicker VerticalAlignment="Center"
HorizontalAlignment="Center"
Width="200"
Name="sfDatePicker">
<syncfusion:SfDatePicker.SelectorStyle>
<Style TargetType="syncfusion:SfDateSelector">
<Setter Property="YearCellTemplate">
<Setter.Value>
<DataTemplate>
<Grid>

```

```

<TextBlock VerticalAlignment="Top"
HorizontalAlignment="Right"
Margin="5"
FontSize="22"
FontFamily="Segoe UI Symbol"
Text="&#xE170;" />
<TextBlock Text="{Binding YearNumber}"
VerticalAlignment="Bottom"
Margin="5"
FontSize="22" />
</Grid>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
</syncfusion:SfDatePicker.SelectorStyle>
</syncfusion:SfDatePicker>

```



Change size of cells

We can change the cell size in the `SfDateSelector` control by setting the [SelectorItemWidth](#) and [SelectorItemHeight](#) properties. The default value of the `SelectorItemWidth` and `SelectorItemHeight` properties is 80 and 70.

#### XML

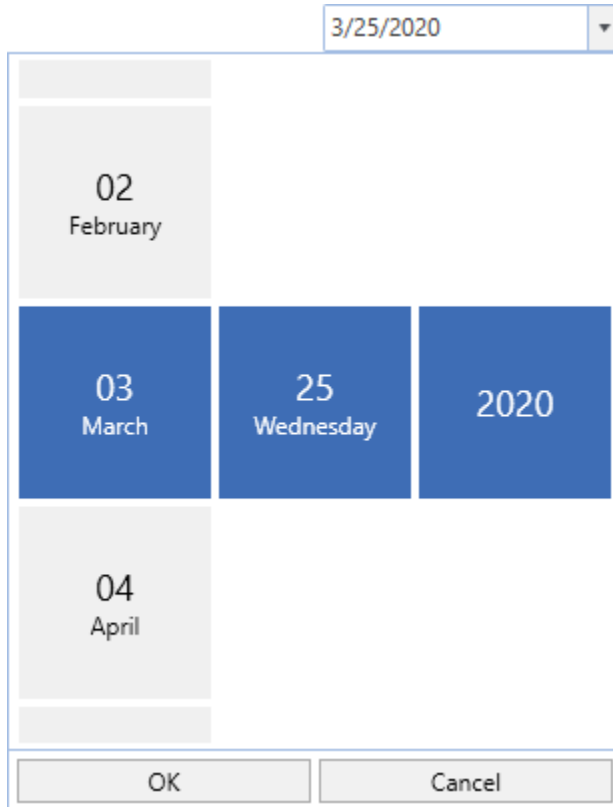
```

<syncfusion:SfDatePicker SelectorItemWidth="100"
SelectorItemHeight="100"
x:Name="sfDatePicker" />

```

**C#**

```
SfDatePicker sfDatePicker = new SfDatePicker();  
sfDatePicker.SelectorItemWidth = 100;  
sfDatePicker.SelectorItemHeight = 100;
```

**DateSelector item spacing**

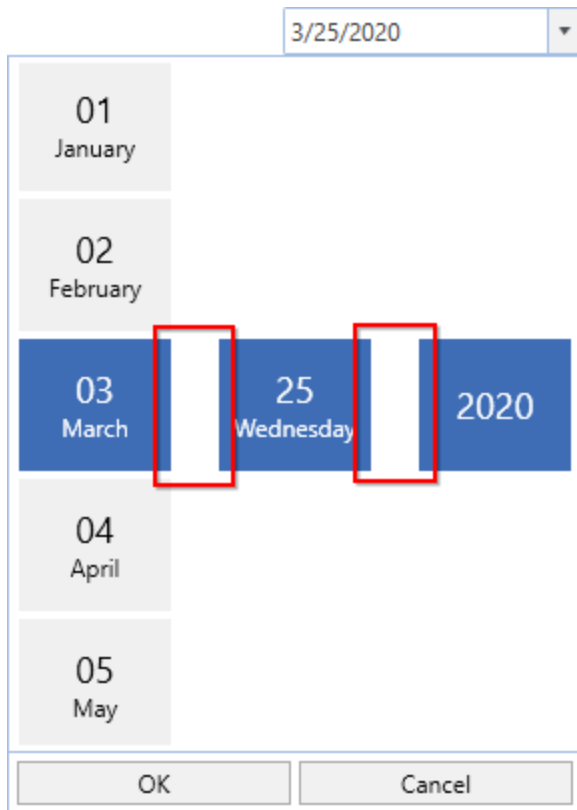
We can change the space between SfDateSelector date, month and year items by using the [SelectorItemSpacing](#) property. The default value of the SelectorItemSpacing property is 4.

**XML**

```
<syncfusion:SfDatePicker SelectorItemSpacing="50"  
x:Name="sfDatePicker"/>
```

**C#**

```
SfDatePicker sfDatePicker = new SfDatePicker();  
sfDatePicker.SelectorItemSpacing = 50;
```



Click [here](#) to download the sample that showcases the SfDateSelector template customization.

### Appearance in WPF DatePicker (SfDatePicker)

This section explains different UI customization, styling, theming options available in [SfDatePicker](#) control.

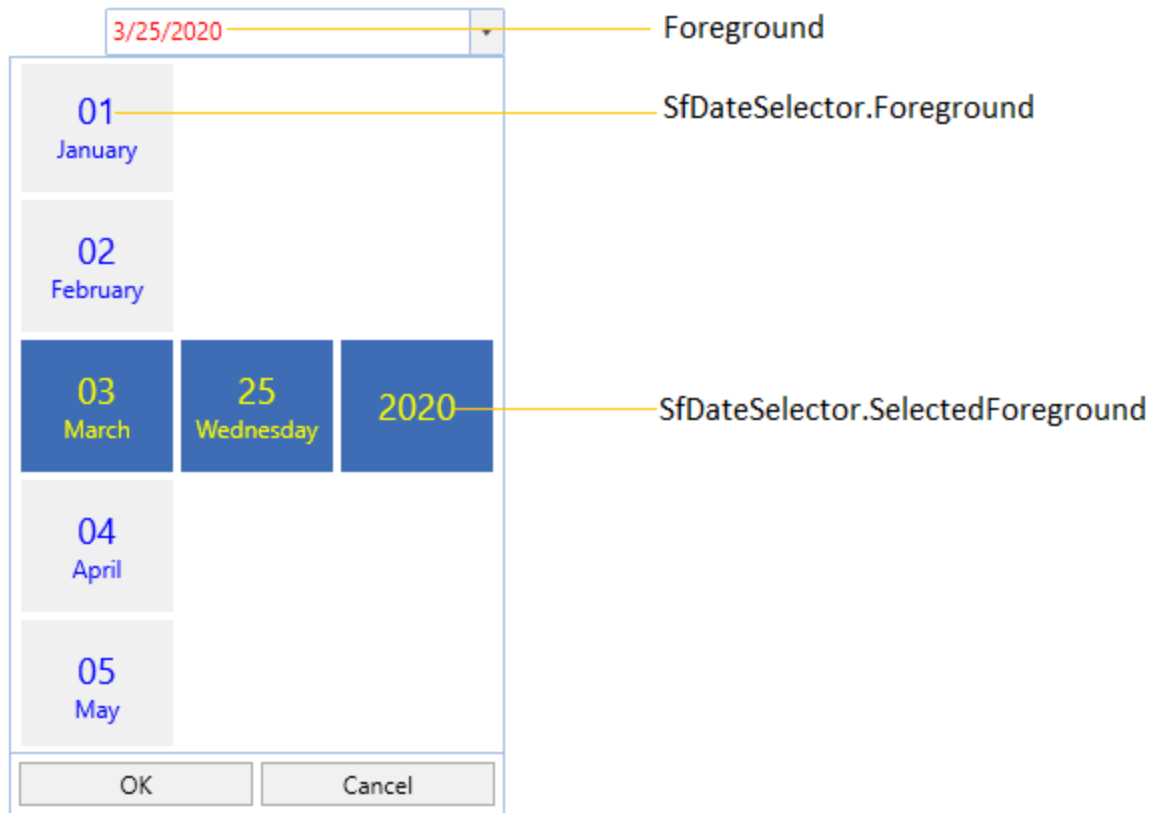
#### Setting the Foreground

We can change a foreground of the SfDatePicker by using the **Foreground** property and also we can change the [SfDateSelector](#) items and selected date item foreground by using the **Foreground** and [SfDateSelector.SelectedForeground](#) properties of SfDateSelector.

#### XML

```
<syncfusion:SfDatePicker Name="sfDatePicker"
    Foreground="Red"
    Width="200">
    <syncfusion:SfDatePicker.SelectorStyle>
    <Style TargetType="syncfusion:SfDateSelector">
    <Setter Property="Foreground" Value="Blue"/>
    <Setter Property="SelectedForeground" Value="Yellow"/>
    </Style>
    </syncfusion:SfDatePicker.SelectorStyle>
    </syncfusion:SfDatePicker>
```





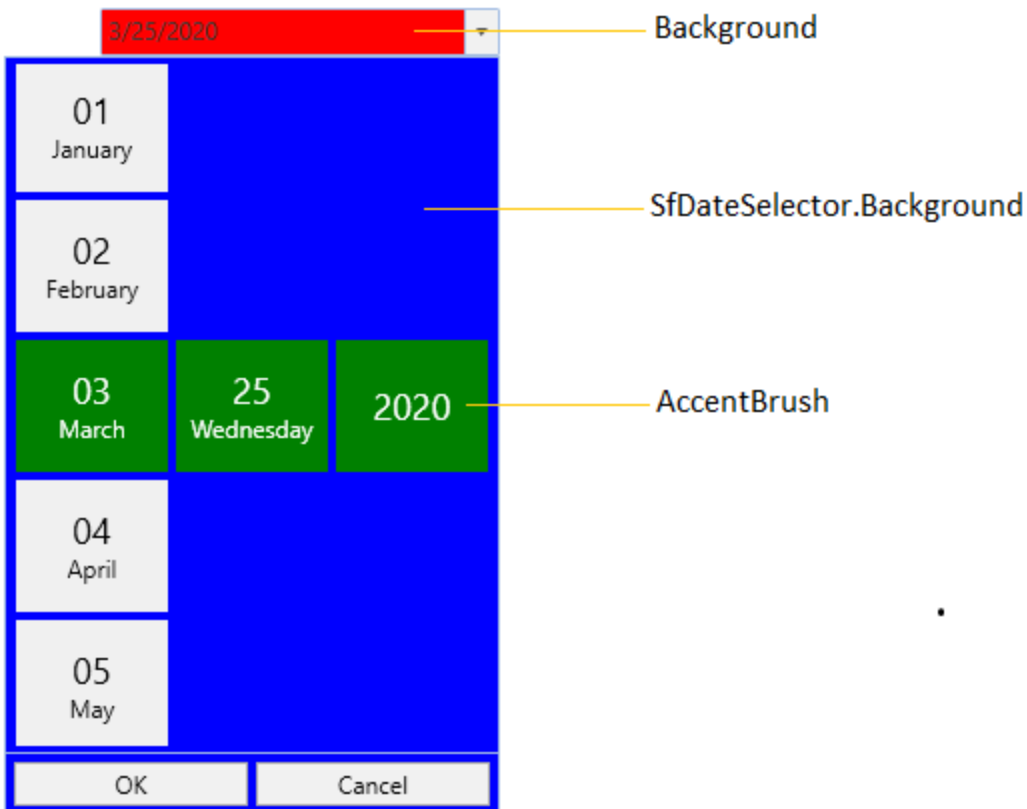
**Note:** [View Sample in GitHub](#)

### Setting the Background

We can change a background of the `SfDatePicker` by using the `background` property and also we can change the `SfDateSelector` items and selected date item background by using the `Background` and `SfDateSelector.AccentBrush` properties of `SfDateSelector`.

### XML

```
<syncfusion:SfDatePicker Name="sfDatePicker"
Background="Red"
AccentBrush="Green"
Width="200">
  <syncfusion:SfDatePicker.SelectorStyle>
    <Style TargetType="syncfusion:SfDateSelector">
      <Setter Property="Background" Value="Blue"/>
    </Style>
  </syncfusion:SfDatePicker.SelectorStyle>
</syncfusion:SfDatePicker>
```



**Note:** [View Sample in GitHub](#)

#### Change flow direction

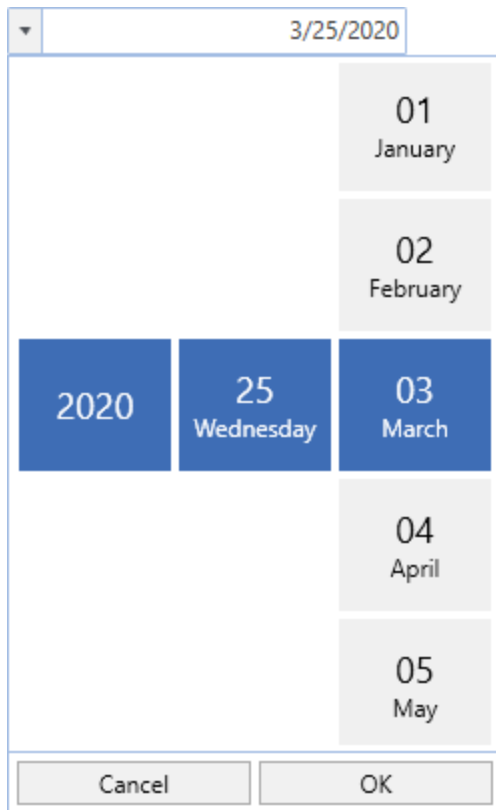
We can change the flow direction of the SfDatePicker control from right to left by setting the FlowDirection property value as RightToLeft. The Default value of FlowDirection property is LeftToRight.

#### XML

```
<syncfusion:SfDatePicker FlowDirection="RightToLeft" Name="sfDatePicker"/>
```

#### C#

```
SfDatePicker sfDatePicker= new SfDatePicker();  
sfDatePicker.FlowDirection = FlowDirection.RightToLeft;
```

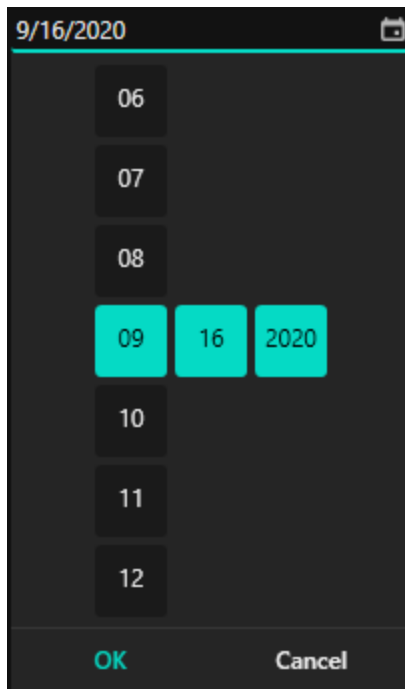


**Note:** [View Sample in GitHub](#)

### Theme

SfDatePicker supports various built-in themes. Refer to the below links to apply themes for the SfDatePicker,

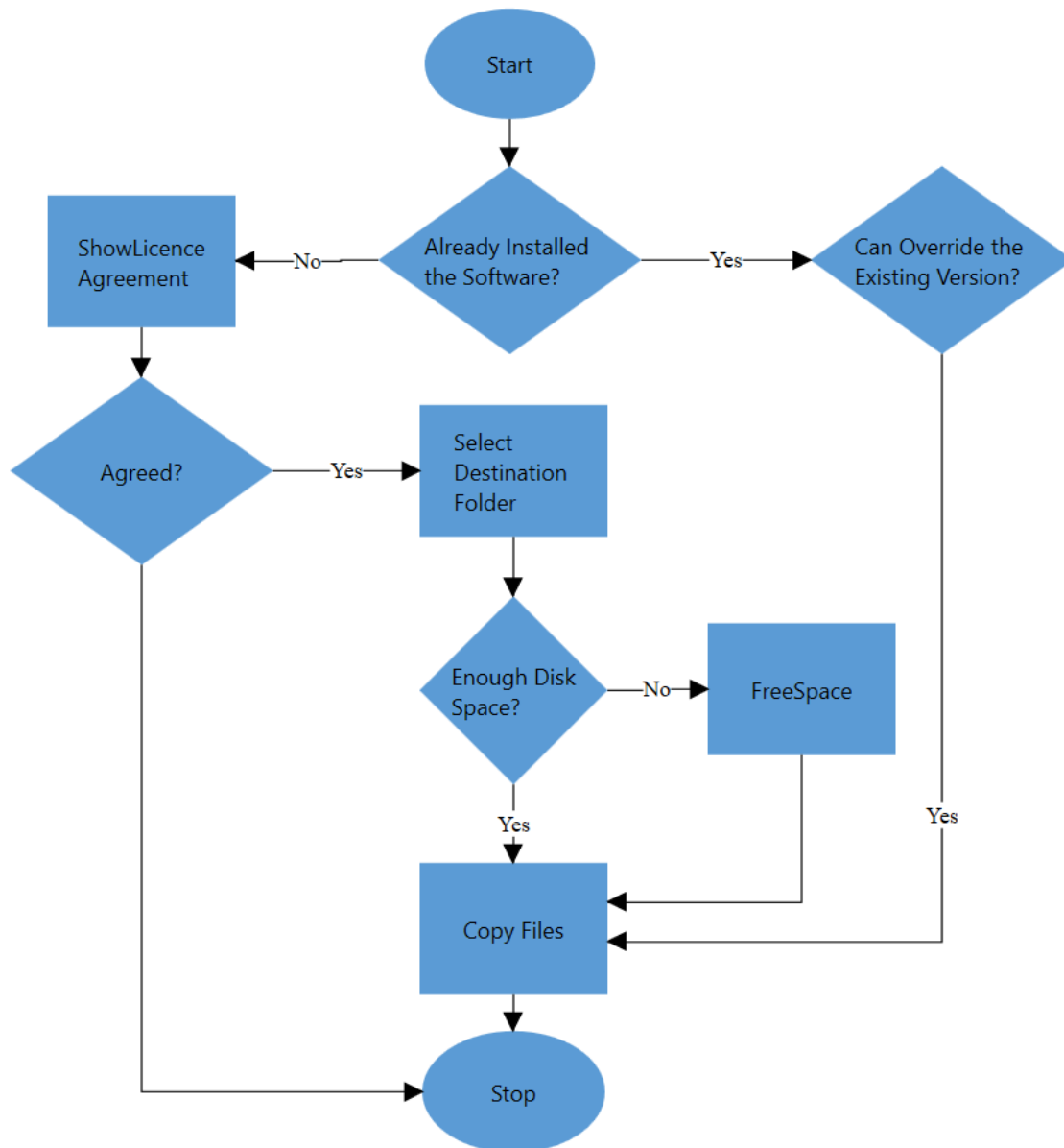
- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



## SfDiagram

### [WPF Diagram \(SfDiagram\) Overview](#)

**Essential Diagram WPF** is a powerful, extensible, and feature-rich library for visualizing, creating, and editing interactive diagrams. It supports creating flowcharts, organizational charts, mind maps, floor plans, UML diagrams, and BPMN charts either through code or a visual interface.



Key features of SfDiagram are as follows:

- **Data visualization:** Supports visualizing any graphical objects such as text, image, shapes, or any UIElement or template.
- **Built-in Shapes:** Provided 445+ standard built-in shapes as ResourceDictionary.
- **High Performance:** Supports loading large diagrams quickly using UI-virtualization techniques.
- **Interaction:** Diagram elements can be selected, rotated, resized, and moved.
- **Z-Order:** Diagram elements overlapping can be controlled by changing its Z-Order value.
- **Pan and Zoom:** Allows you to pan, zoom in or out of your current diagram.
- **Snapping:** Snaps the diagram elements towards the nearest element or gridlines.
- **Undo/Redo:** Allows you to keep track of the recent changes made in a Diagram to correct your mistakes.
- **Commands:** Built-in commands for various interactions that allows you to easily implement Diagram interaction logic in MVVM pattern.

- **Keyboard Shortcuts:** Allows you to trigger the diagramming features using keyboard shortcuts.
- **Clipboard Operations:** Ability to cut, copy, and paste or duplicate selected Diagram elements within and across diagrams.
- **Business objects:** Ability to populate diagram from datasource (business object) with the help of automatic-layouts.
- **Automatic Layouts:** Built-in automatic nodes arrangement support for organization chart layout, hierarchical tree layout, flowchart layout, and radial tree layout.
- **Serialization:** Saves current state of diagram as an XML file, and load them back when needed.
- **Printing and Exporting:** Supports printing and also exporting the diagram as image.
- **Stencil:** Gallery of reusable symbols and nodes, which can be dragged and dropped onto the diagram surface at any number of times.
- **Overview control:** Displays a small preview of the full diagram page, which allows you to improve the navigation.
- **Localization:** Localizes every static text in the control to any supported language.

## Getting Started with WPF Diagram (SfDiagram)

### Assembly deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

You can find more details about installing the NuGet package in a WPF application in the following link:

### [How to install nuget packages](#)

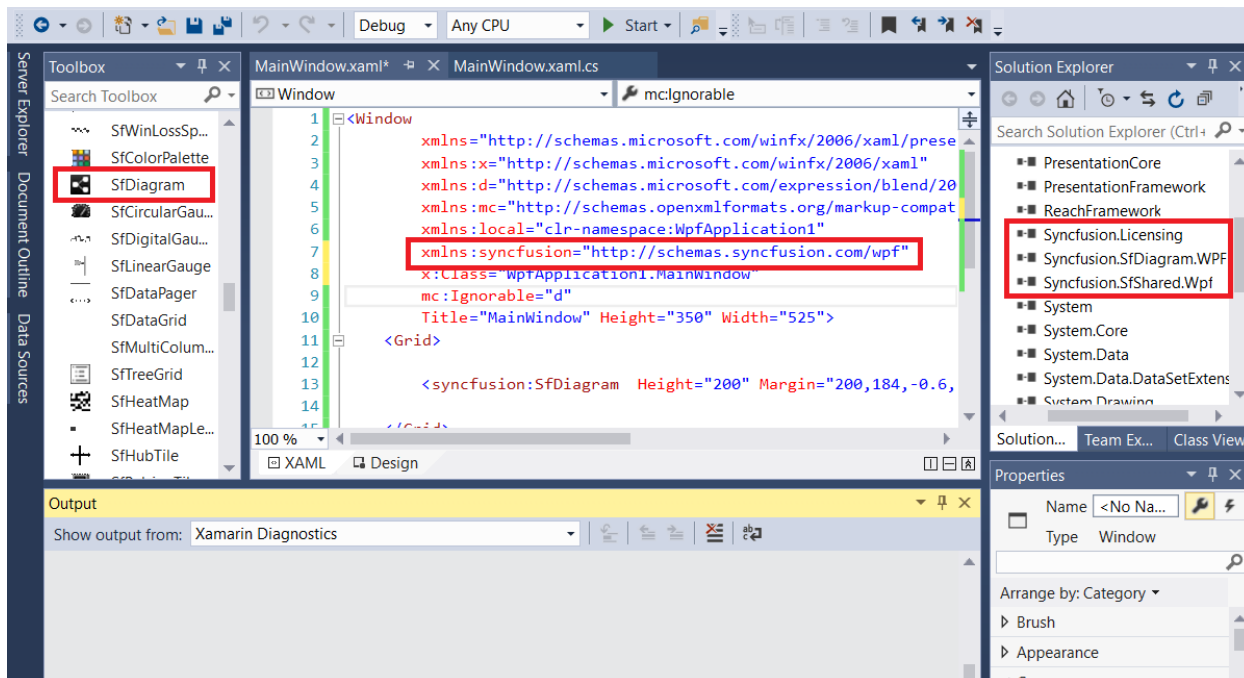
The following section helps you to build your application with SfDiagram.

### Creating the project

Create new WPF project using Visual Studio. For more [details](#).

### *Adding control via Designer*

[WPF Diagram](#) (SfDiagram) control can be added to the application by dragging it from Toolbox and dropping it in Designer view. The required assembly references will be added automatically.



### Adding control manually in XAML

To add control manually in XAML, do the following steps:

1. Add the following required assembly reference to the project, Syncfusion.SfDiagram.WPF .
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> or SfDiagram control namespace Syncfusion.UI.Xaml.Diagram in XAML page.
3. Declare SfDiagram control in XAML page.

**Note:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from trial setup or from the NuGet feed, you also have to include a license key in your projects. Refer to this [link](#) to learn about registering Syncfusion license key in your WPF application to use Syncfusion components.

### XML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:SfDiagram_WPF"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="SfDiagram_WPF.MainWindow"
mc:Ignorable="d" Title="MainWindow" Height="350" Width="525">
<Grid>
<!--Initializes the SfDiagram in XAML window-->
<syncfusion:SfDiagram x:Name="diagram"/>
</Grid>
</Window>
```

### Adding control manually in C#

To add control manually in C#, do the following steps:

1. Add the following required assembly references to the project, Syncfusion.SfDiagram.WPF.
2. Import SfDiagram namespace Syncfusion.UI.Xaml.Diagram.
3. Create SfDiagram control instance and add it to the Grid.

### C#

```
using Syncfusion.UI.Xaml.Diagram;
namespace SfDiagram_WPF
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            // Initializes the SfDiagram
            SfDiagram diagram=new SfDiagram();
            Root_Grid.Children.Add(diagram);
        }
    }
}
```

### Basic Diagram elements

- **Node:** Visualizes any graphical object using nodes, which can also be arranged and manipulated on a diagram page.
- **Connector:** Represents the relationship between two nodes. Four types of connectors provided as follows: 1) Orthogonal 2) Bezier 3) Straight 4) Cubic Bezier
- **Port:** Acts as the connection points of node or connector and allows you to create connections with only specific points.
- **Annotation:** Shows additional information by adding text or labels on nodes and connectors.

### Flow chart

Let us create a simple flow chart using SfDiagram.

#### Initialize the Diagram

The SfDiagram exists in the Syncfusion.UI.Xaml.Diagram namespace. Initialize SfDiagram in XAML as shown in the following code example.

### XML

```
<!--Initializes the SfDiagram in XAML window-->
<syncfusion:SfDiagram x:Name="diagram"/>
```

### C#

```
//Initializes the SfDiagram in C#
SfDiagram diagram =new SfDiagram();
```

#### Initialize nodes and connectors

To initialize the nodes and connectors properties of the SfDiagram, the Nodes property is assigned with the NodeCollection, that is, ObservableCollection of the Node.



The Connectors property is assigned with the ConnectorCollection, that is, ObservableCollection of the Connector.

### XML

```
<syncfusion:SfDiagram.Nodes>
<!--Observable Collection of NodeViewModel-->
<syncfusion:NodeCollection/>
</syncfusion:SfDiagram.Nodes>
<syncfusion:SfDiagram.Connectors>
<!--Observable Collection of ConnectorViewModel-->
<syncfusion:ConnectorCollection/>
</syncfusion:SfDiagram.Connectors>
```

### C#

```
//Initialize Nodes with Observable Collection of NodeViewModel.
diagram.Nodes = new NodeCollection();
//Initialize Connectors with Observable Collection of ConnectorViewModel
diagram.Connectors = new ConnectorCollection();
```

### Add nodes

Let us create and add a NodeViewModel with height, width, shape, shape style, specific position, size, and annotation.

### Creating a node

Creating NodeViewModel with specified height and width at a specific position.

### XML

```
<syncfusion:NodeViewModel ID="Begin" UnitHeight="40" UnitWidth="120"
OffsetX="300" OffsetY="60"/>
```

### C#

```
NodeViewModel Begin = new NodeViewModel()
{
    ID = "Begin",
    UnitWidth = 120,
    UnitHeight = 40,
    OffsetX = 300,
    OffsetY = 60,
};
```

### Adding shape and style to node

We have provided set of basic shapes for Diagram as ResourceDictionary. To use the built-in shapes, Shapes dictionary should be merged in the application.

Please refer to [Shapes](#) to know about built-in Shapes.

### XML

```
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<!--Initialize Shapes-->
```

```

<ResourceDictionary
Source="/Syncfusion.SfDiagram.Wpf;component/Resources/BasicShapes.xaml" />
</ResourceDictionary.MergedDictionaries>
<!--Style for Shape of the Node-->
<Style TargetType="Path" x:Key="ShapeStyle">
<Setter Property="Fill" Value="#FF5B9BD5"/>
<Setter Property="Stretch" Value="Fill"/>
<Setter Property="Stroke" Value="#FFEDF1F6"/>
</Style>
<!--To apply Style for NodeViewModel-->
<Style TargetType="syncfusion:Node">
<Setter Property="ShapeStyle" Value="{StaticResource ShapeStyle}"></Setter>
</Style>
</ResourceDictionary>
<syncfusion:NodeViewModel ID="Begin" OffsetX="300" OffsetY="60"
Shape="{StaticResource Ellipse}"
UnitHeight="40" UnitWidth="120"/>

```

### C#

```

NodeViewModel Begin = new NodeViewModel()
{
ID = "Begin", UnitWidth = 120, UnitHeight = 40, OffsetX = 300, OffsetY = 60,
//Specify shape to the Node from built-in Shape Dictionary
Shape = this.Resources["Ellipse"],
//Apply style to Shape
ShapeStyle = this.Resources["ShapeStyle"] as Style,
};

```

Now, the node will be as follows.




---

**Note:** ID sets for each node to identify nodes easily while setting connectors.

---

### Adding annotation to node

To initialize the Annotation property of the node and connector, it is assigned with the annotation collection, that is, ObservableCollection of the IAnnotation.

Now add the Annotation content to Node.

### XML

```

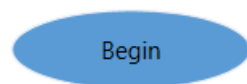
<syncfusion:NodeViewModel ID="Begin" OffsetX="300" OffsetY="60"
Shape="{StaticResource Ellipse}"
UnitHeight="40" UnitWidth="120">
<syncfusion:NodeViewModel.Annotations>
<!--Observable Collection of AnnotationEditorViewModel-->
<syncfusion:AnnotationCollection>
<syncfusion:AnnotationEditorViewModel Content="Begin"/>
</syncfusion:AnnotationCollection>
</syncfusion:NodeViewModel.Annotations>
</syncfusion:NodeViewModel>

```

**C#**

```
// Creating the NodeViewModel
NodeViewModel Begin = new NodeViewModel()
{
    ID = "Begin",
    UnitWidth = 120,
    UnitHeight = 40,
    OffsetX = 300,
    OffsetY = 60,
    //Specify shape to the Node from built-in Shape Dictionary
    Shape = this.Resources["Ellipse"],
    //Apply style to Shape
    ShapeStyle = this.Resources["ShapeStyle"] as Style,
    Annotations = new AnnotationCollection()
    {
        new AnnotationEditorViewModel()
        {
            Content="Begin",
        }
    }
};
//Add Node to Nodes property of the Diagram
(diagram.Nodes as NodeCollection).Add(Begin);
```

Now, the node will be as follows.



**Note:** `Annotations` property is a collection, which indicates that more than one Annotation can be added to a Node and Connector. By default, `Annotations` property of Node and Connector is null.

*Nodes for flow diagram*

You can add multiple nodes with different shapes into diagram as NodeViewModel.

**XML**

```
<syncfusion:NodeCollection>
  <!--Begin-->
  <syncfusion:NodeViewModel ID="Begin" OffsetX="300" OffsetY="60"
    Shape="{StaticResource Ellipse}"
    UnitHeight="40" UnitWidth="120">
    <syncfusion:NodeViewModel.Annotations>
      <!--Observable Collection of AnnotationEditorViewModel-->
      <syncfusion:AnnotationCollection>
        <syncfusion:AnnotationEditorViewModel Content="Begin"/>
      </syncfusion:AnnotationCollection>
    </syncfusion:NodeViewModel.Annotations>
  </syncfusion:NodeViewModel>
  <!--Process-->
  <syncfusion:NodeViewModel ID="Process" UnitHeight="60" UnitWidth="120"
    OffsetX="300" OffsetY="140"
    Shape="{StaticResource PredefinedProcess}">
    <syncfusion:NodeViewModel.Annotations>
```

```

<syncfusion:AnnotationCollection>
<syncfusion:AnnotationEditorViewModel Content="Process"/>
</syncfusion:AnnotationCollection>
</syncfusion:NodeViewModel.Annotations>
</syncfusion:NodeViewModel>
<!--End-->
<syncfusion:NodeViewModel ID="End" UnitHeight="40" UnitWidth="40"
OffsetX="300" OffsetY="225"
Shape="{StaticResource Ellipse}">
<syncfusion:NodeViewModel.Annotations>
<syncfusion:AnnotationCollection>
<syncfusion:AnnotationEditorViewModel Content="End"/>
</syncfusion:AnnotationCollection>
</syncfusion:NodeViewModel.Annotations>
</syncfusion:NodeViewModel>
</syncfusion:NodeCollection>

```

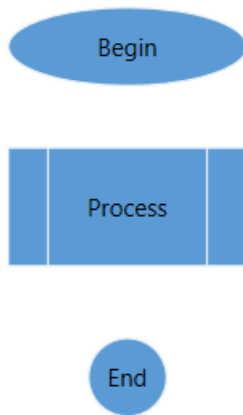
**C#**

```

//Create Begin node
NodeViewModel Begin = AddNode(300, 60, 120, 40, "Begin", "Ellipse");
//Create Process node
NodeViewModel Process = AddNode(300, 140, 120, 60, "Process",
"PredefinedProcess");
//Creae End node
NodeViewModel End = AddNode(300, 225, 40, 40, "End", "Ellipse");
//Add Node to Nodes property of the Diagram
(diagram.Nodes as NodeCollection).Add(Begin);
(diagram.Nodes as NodeCollection).Add(Process);
(diagram.Nodes as NodeCollection).Add(End);
//Creating NodeViewModel
public NodeViewModel AddNode(double offsetX, double offsetY, double width,
double height, string text, string shape)
{
NodeViewModel node = new NodeViewModel();
node.ID = text;
node.OffsetX = offsetX;
node.OffsetY = offsetY;
node.UnitHeight = height;
node.UnitWidth = width;
//Specify shape to the Node from built-in Shape Dictionary
node.Shape = this.Resources[shape];
//Apply style to Shape
node.ShapeStyle = this.Resources["ShapeStyle"] as Style;
node.Annotations = new AnnotationCollection()
{
new AnnotationEditorViewModel()
{
Content=text,
},
};
return node;
}

```

Finally, all the nodes are added to diagram and they will be as follows.



### Add connectors

Connector is to make connection or link between two nodes, ports, and points.

### Create connector With source node and target node

Here, the `SourceNodeID` and `TargetNodeID` properties of the Connector are used. These properties will be assigned with the `ID` property of the node.

### XML

```
<syncfusion:ConnectorViewModel SourceNodeID="Begin" TargetNodeID="Process"/>
```

### C#

```
ConnectorViewModel connector1 = new ConnectorViewModel()
{
    SourceNodeID = "Begin",
    TargetNodeID = "Process",
};
```

### Adding connector geometry style

Here, the `ConnectorGeometryStyle` property of the Connector are used to customize the appearance of the line. And, `SourceDecoratorStyle` and `TargetDecoratorStyle` properties are used to customize the appearance of the decorators.

### XML

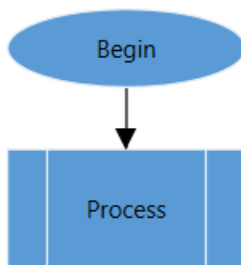
```
<!--Style for TargetDecorator of the Connector-->
<Style TargetType="Path" x:Key="TargetDecoratorStyle">
  <Setter Property="Stroke" Value="Black"/>
  <Setter Property="Stretch" Value="Fill"/>
  <Setter Property="Fill" Value="Black"/>
  <Setter Property="Height" Value="10"/>
  <Setter Property="Width" Value="10"/>
</Style>
<!--Style for Geometry of the Connector-->
<Style TargetType="Path" x:Key="ConnectorGeometryStyle">
  <Setter Property="Stroke" Value="Black" />
  <Setter Property="StrokeThickness" Value="1" />
</Style>
<!--To apply Style for ConnectorViewModel-->
```

```
<Style TargetType="syncfusion:Connector" >
<Setter Property="TargetDecoratorStyle" Value="{StaticResource
TargetDecoratorStyle}"/>
<Setter Property="ConnectorGeometryStyle" Value="{StaticResource
ConnectorGeometryStyle}"/>
</Style>
```

**C#**

```
ConnectorViewModel connector1 = new ConnectorViewModel()
{
    SourceNodeID = "Begin",
    TargetNodeID = "Process",
    //Apply Style to TargetDecorator
    TargetDecoratorStyle = this.Resources["TargetDecoratorStyle"] as Style,
    //Apply Style to Geometry of the Connector.
    ConnectorGeometryStyle = this.Resources["ConnectorGeometryStyle"] as Style
};
//Add Connector to Connectors property of the Diagram
(diagram.Connectors as ConnectorCollection).Add(connector1);
```

Now, the output will be as follows.

*Connectors for flow diagram*

Now, you can connect all nodes using ConnectorViewModel.

**XML**

```
<syncfusion:ConnectorCollection>
<syncfusion:ConnectorViewModel SourceNodeID="Begin" TargetNodeID="Process"/>
<syncfusion:ConnectorViewModel SourceNodeID="Process" TargetNodeID="End"/>
</syncfusion:ConnectorCollection>
```

**C#**

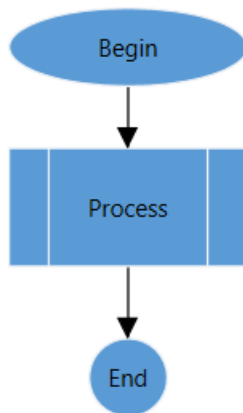
```
//Creating ConnectorViewModel
ConnectorViewModel connector1 = CreateConnector("Begin", "Process");
ConnectorViewModel connector2 = CreateConnector("Process", "End");
//Add Connector to Connectors property of the Diagram
(diagram.Connectors as ConnectorCollection).Add(connector1);
(diagram.Connectors as ConnectorCollection).Add(connector2);
//Creating ConnectorViewModel
private ConnectorViewModel CreateConnector(string source, string target)
{
    ConnectorViewModel connector = new ConnectorViewModel()
```

```

{
    SourceNodeID = source,
    TargetNodeID = target,
    //Apply Style to TargetDecorator
    TargetDecoratorStyle = this.Resources["TargetDecoratorStyle"] as Style,
    //Apply Style to Geometry of the Connector.
    ConnectorGeometryStyle = this.Resources["ConnectorGeometryStyle"] as Style
};
return connector;
}

```

Now, the output will be as follows.



[View Sample in GitHub](#)

*Flowchart creation using stencil*

Let us create a simple flowchart diagram using stencil.

**Define stencil**

**Stencil** is a gallery of reusable symbols and nodes, which can be dragged and dropped onto the diagram surface at any number of times.

**XML**

```

<!--Namespace for stencil-->
xmlns:stencil="clr-
namespace:Syncfusion.UI.Xaml.Diagram.Stencil;assembly=Syncfusion.SfDiagram.W
PF"
<!--Define the Stencil-->
<stencil:Stencil x:Name="stencil" ExpandMode="All"
BorderBrush="Black" BorderThickness="0,0,2,0"/>

```

**Define SymbolSource**

**SymbolSource** is the property of stencil, which is a collection of objects like symbol, node, connector, and more. Based on the SymbolSource, the Stencil will populate the Symbols. And the **SymbolGroupProvider** groups the symbols into SymbolGroup based on the **MappingName** property.

**XML**

```

<!--Define the Stencil-->

```

```

<stencil:Stencil x:Name="stencil" ExpandMode="All"
BorderBrush="Black" BorderThickness="0,0,2,0">
<!--Initialize the SymbolSource-->
<stencil:Stencil.SymbolSource>
<!--Initialize the SymbolCollection-->
<local:SymbolCollection>
<!--Define the DiagramElement-Node-->
<Syncfusion:NodeViewModel UnitHeight="40" UnitWidth="120"
Shape="{StaticResource Ellipse}" Key="Nodes">
<Syncfusion:NodeViewModel.Annotations>
<!--Observable Collection of AnnotationEditorViewModel-->
<Syncfusion:AnnotationCollection>
<Syncfusion:AnnotationEditorViewModel Content="Begin"/>
</Syncfusion:AnnotationCollection>
</Syncfusion:NodeViewModel.Annotations>
</Syncfusion:NodeViewModel>
<!--Define the DiagramElement-Connector-->
<Syncfusion:ConnectorViewModel Key="Connectors"
SourcePoint="100,100"
TargetPoint="200,200" />
</local:SymbolCollection>
</stencil:Stencil.SymbolSource>
<!--Initialize the SymbolGroup-->
<stencil:Stencil.SymbolGroups>
<stencil:SymbolGroups>
<!--Map Symbols Using MappingName-->
<stencil:SymbolGroupProvider MappingName="Key"/>
</stencil:SymbolGroups>
</stencil:Stencil.SymbolGroups>
</stencil:Stencil>
<!--Define DiagramControl to drag and drop elements into the diagram-->
<Syncfusion:SfDiagram x:Name="sfdiagram">
<Syncfusion:SfDiagram.Nodes>
<Syncfusion:NodeCollection/>
</Syncfusion:SfDiagram.Nodes>
<Syncfusion:SfDiagram.Connectors>
<Syncfusion:ConnectorCollection/>
</Syncfusion:SfDiagram.Connectors>
</Syncfusion:SfDiagram>

```

## C#

```

//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize NodeCollection to SfDiagram
diagram.Nodes = new NodeCollection();
//Initialize ConnectorCollection to SfDiagram
diagram.Connectors = new ConnectorCollection();
//Initialize Stencil and its properties
Stencil stencil = new Stencil();
stencil.ExpandMode = ExpandMode.All;
stencil.BorderBrush = new SolidColorBrush(Colors.Black);
stencil.BorderThickness = new Thickness(0,0,2,0);
NodeViewModel node1 = new NodeViewModel()
{
UnitHeight = 40,

```



```

UnitWidth = 120,
Shape = this.Resources["Ellipse"],
Key = "Nodes",
Annotations = new AnnotationCollection()
{
    new AnnotationEditorViewModel()
    {
        Content = "Begin"
    }
};
ConnectorViewModel connector = new ConnectorViewModel()
{
    SourcePoint = new Point(100, 100),
    TargetPoint = new Point(200, 200),
    Key = "Connectors",
    Segments = new ConnectorSegments()
    {
        new StraightSegment()
    }
};
//Initialize collection to hold the symbols
SymbolCollection symbolcollection = new SymbolCollection();
//Add the symbols to the collection
symbolcollection.Add(node1);
symbolcollection.Add(connector);
//Initialize the SymbolSource
stencil.SymbolSource = symbolcollection;
//Map Symbols Using MappingName
SymbolGroupProvider sgp = new SymbolGroupProvider()
{
    MappingName = "Key"
};
//Initialize SymbolGroup
stencil.SymbolGroups = new SymbolGroups();
//Add the SymbolGroupProvider to SymbolGroup
stencil.SymbolGroups.Add(sgp);
//Add the Diagram and Stencil to Mainwindow grid
RootGrid.Children.Add(stencil);
RootGrid.Children.Add(diagram);
//Position the stencil and Diagram in the Main Grid
stencil.SetValue(Grid.RowProperty, 1);
stencil.SetValue(Grid.ColumnProperty, 0);
diagram.SetValue(Grid.RowProperty, 1);
diagram.SetValue(Grid.ColumnProperty, 1);
public class SymbolCollection:ObservableCollection<object>
{
}

```

#### Visualization of stencil elements

Declare the style for node, connector, symbol, and symbol group to visualize the elements in the stencil.

#### XML

```

<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>

```

```

<ResourceDictionary
Source="/Syncfusion.SfDiagram.Wpf;component/Resources/BasicShapes.xaml"/>
</ResourceDictionary.MergedDictionaries>
<!--Style for Node-->
<Style TargetType="Syncfusion:Node">
<Setter Property="ShapeStyle">
<Setter.Value>
<Style TargetType="Path">
<Setter Property="Stretch" Value="Fill"></Setter>
<Setter Property="Fill" Value="#FF5B9BD5"></Setter>
</Style>
</Setter.Value>
</Setter>
</Style>
<!--Style for Connector-->
<Style TargetType="Syncfusion:Connector">
<Setter Property="TargetDecoratorStyle">
<Setter.Value>
<Style TargetType="Path">
<Setter Property="Stretch" Value="Fill"/>
<Setter Property="Fill" Value="Black"/>
<Setter Property="Stroke" Value="Black"/>
<Setter Property="StrokeThickness" Value="1"/>
</Style>
</Setter.Value>
</Setter>
</Style>
<!--Style for Symbol-->
<Style TargetType="stencil:Symbol">
<Setter Property="Width" Value="50"/>
<Setter Property="Height" Value="50"/>
<Setter Property="Padding" Value="3" />
<Setter Property="BorderThickness" Value="1" />
<Setter Property="Background" Value="Transparent" />
<Setter Property="BorderBrush" Value="Transparent" />
<Setter Property="Margin" Value="4"></Setter>
</Style>
<!--Style for Symbol Group-->
<Style TargetType="stencil:SymbolGroup">
<Setter Property="FontFamily" Value="Regular"/>
<Setter Property="Background" Value="#ffffff"/>
<Setter Property="Foreground" Value="#222222"/>
<Setter Property="FontSize" Value="14"/>
<Setter Property="HeaderTemplate">
<Setter.Value>
<DataTemplate>
<stencil:Header>
<stencil:Header.Template>
<ControlTemplate TargetType="stencil:Header">
<Grid>
<Border x:Name="header" Background="#f5f5f5"
BorderBrush="Black" BorderThickness="1">
<ContentPresenter Margin="10" Content="{Binding}"/>
</Border>
</Grid>
</ControlTemplate>
</stencil:Header.Template>

```

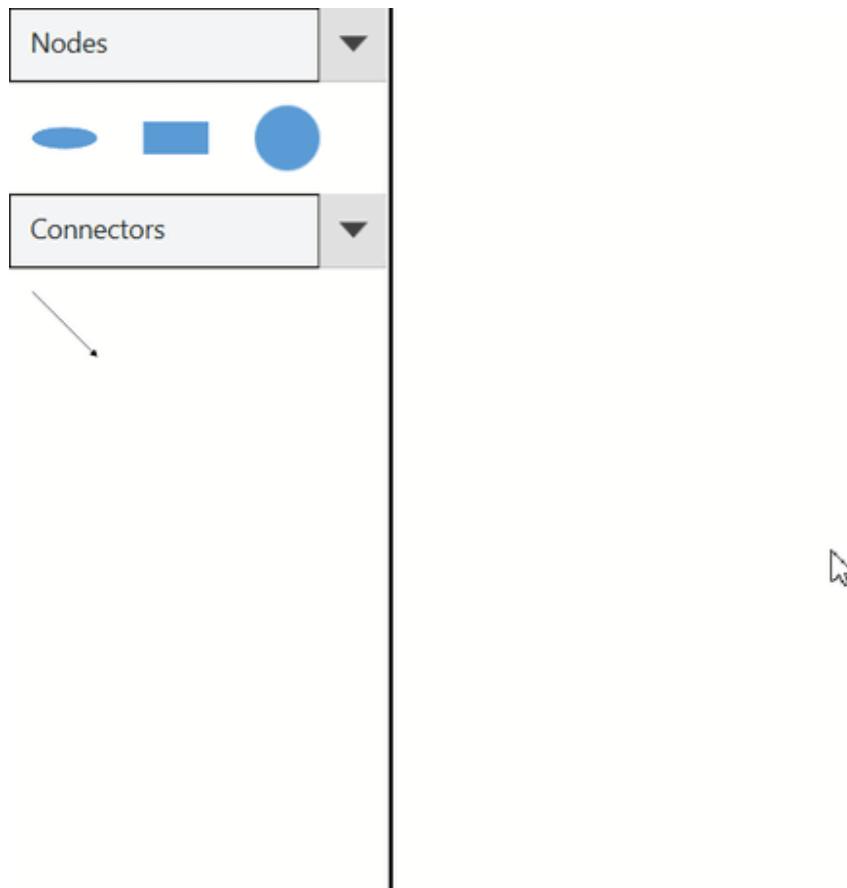
```
</stencil:Header>  
</DataTemplate>  
</Setter.Value>  
</Setter>  
</Style>  
</ResourceDictionary>
```

The output will be as follows.



#### [Interaction with stencil](#)

We have represented the steps to interact with stencil such as drag and drop elements from stencil to diagram in the following gif.



[View Sample in GitHub](#)

For more information about stencil, please [refer](#).

### Organization layout

WPF Diagram (SfDiagram) provides support to auto-arrange the nodes in the diagram area that is referred as Layout.

### Business object (employee information)

Define employee information as collection of data. The following code example shows the employee information whose, EmpId is used as an unique identifier and ParentId is used to identify the person to whom an employee report to, in the organization.

### XML

```
<!--Initializes the DataSource -->
<local:Employees x:Key="employees">
  <local:Employee EmpId="1" ParentId="" Designation="CEO"/>
  <local:Employee EmpId="2" ParentId="1" Designation="Project Manager1"/>
  <local:Employee EmpId="3" ParentId="1" Designation="Project Manager2"/>
  <local:Employee EmpId="4" ParentId="2" Designation="Engineer1"/>
  <local:Employee EmpId="5" ParentId="3" Designation="Engineer2"/>
</local:Employees>
```

### C#

```
private Employees GetData()
```

```

{
    Employees data = new Employees();
    data.Add(new Employee() { EmpId = "1", ParentId = "", Designation = "CEO"
    });
    data.Add(new Employee() { EmpId = "2", ParentId = "1", Designation =
    "Project Manager1" });
    data.Add(new Employee() { EmpId = "3", ParentId = "1", Designation =
    "Project Manager2" });
    data.Add(new Employee() { EmpId = "4", ParentId = "2", Designation =
    "Engineer1" });
    data.Add(new Employee() { EmpId = "5", ParentId = "3", Designation =
    "Engineer2" });
    return data;
}
public class Employee
{
    public Employee()
    {
    }
    public string EmpId { get; set; }
    public string ParentId { get; set; }
    public string Designation { get; set; }
}
public class Employees:ObservableCollection<Employee>
{
}

```

#### Map DataSource with Diagram

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

#### XML

```

<!--Initializes the DataSourceSettings -->
<Syncfusion:DataSourceSettings x:Key="DataSourcesettings"
DataSource="{StaticResource employees}"
ParentId="ParentId" Id="EmpId" />
<!--Map the DataSourceSettings class with Diagram-->
<Syncfusion:SfDiagram x:Name="sfdiagram"
DataSourceSettings="{StaticResource DataSourcesettings}"/>
</Syncfusion:SfDiagram>

```

#### C#

```

//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize NodeCollection to SfDiagram
diagram.Nodes = new NodeCollection();
//Initialize ConnectorCollection to SfDiagram
diagram.Connectors = new ConnectorCollection();
//Initialize DataSourceSettings to SfDiagram
diagram.DataSourceSettings = new DataSourceSettings()
{
}

```

```

DataSource =GetData(),
ParentId= "ParentId",
Id= "EmpId"
};

```

### Rendering layout with DataSource

To create an organizational chart, `TreeLayout` type should be set to `LayoutType.Organization`. The following code example shows how `LayoutManager` is used to generate `TreeLayout` based on the `DataSourceSettings` of the Diagram.

### XML

```

<!--Style for Connector-->
<Style TargetType="Syncfusion:Connector">
<Setter Property="ConnectorGeometryStyle">
<Setter.Value>
<Style TargetType="Path">
<Setter Property="Stroke" Value="Black" />
<Setter Property="StrokeThickness" Value="1" />
</Style>
</Setter.Value>
</Setter>
<Setter Property="TargetDecoratorStyle">
<Setter.Value>
<Style TargetType="Path">
<Setter Property="Stroke" Value="#4f4f4f" />
<Setter Property="Stretch" Value="Fill" />
<Setter Property="Fill" Value="#4f4f4f" />
<Setter Property="StrokeThickness" Value="1" />
</Style>
</Setter.Value>
</Setter>
</Style>
<!--Style for Node-->
<Style TargetType="Syncfusion:Node">
<Setter Property="ContentTemplate">
<Setter.Value>
<DataTemplate>
<Border Background="#FF5B9BD5" BorderBrush="Blue"
Width="120" Height="40"
VerticalAlignment="Center" HorizontalAlignment="Center">
<TextBlock Margin="5" TextWrapping="Wrap" FontSize="12"
Foreground="#ffffff" Text="{Binding Path=Designation}"
FontFamily="Segoe UI" VerticalAlignment="Center"
FontWeight="Bold" HorizontalAlignment="Center"/>
</Border>
</DataTemplate>
</Setter.Value>
</Setter>
</Style>
<!--Initializes the Layout-->
<Syncfusion:DirectedTreeLayout x:Key="treeLayout" Orientation="TopToBottom"
HorizontalSpacing="50" Type="Organization"/>
<!--Initializes the LayoutManager-->
<Syncfusion:LayoutManager x:Key="LayoutManager" Layout="{StaticResource
treeLayout}" />

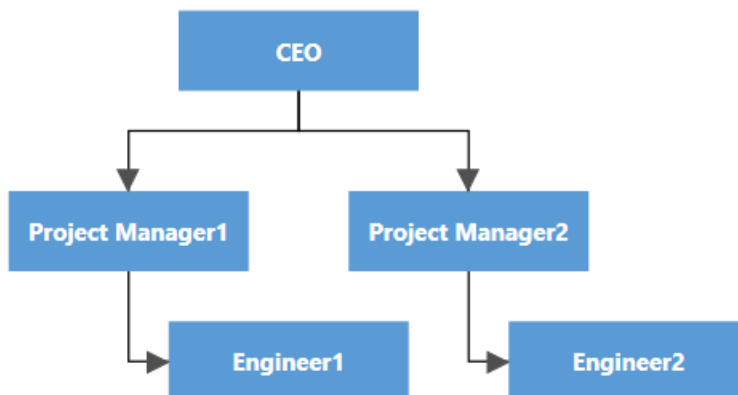
```

```
<!--Map the DataSourceSettings and LayoutManager class with Diagram-->  
<Syncfusion:SfDiagram x:Name="sfdiagram"  
DataSourceSettings="{StaticResource DataSourceSettings}"  
LayoutManager="{StaticResource LayoutManager}">  
  <Syncfusion:SfDiagram.Nodes>  
  <Syncfusion:NodeCollection/>  
  </Syncfusion:SfDiagram.Nodes>  
  <Syncfusion:SfDiagram.Connectors>  
  <Syncfusion:ConnectorCollection/>  
  </Syncfusion:SfDiagram.Connectors>  
</Syncfusion:SfDiagram>
```

## C#

```
//Initialize LayoutManager to SfDiagram  
diagram.LayoutManager = new LayoutManager()  
{  
  //Initialize Layout for LayoutManager  
  Layout=new DirectedTreeLayout()  
  {  
    Type=LayoutType.Organization,  
    Orientation=TreeOrientation.TopToBottom,  
    HorizontalSpacing=50  
  }  
};  
//RootGrid is the instance of the MainWindow Grid  
RootGrid.Children.Add(diagram);
```

The output will be as follows.



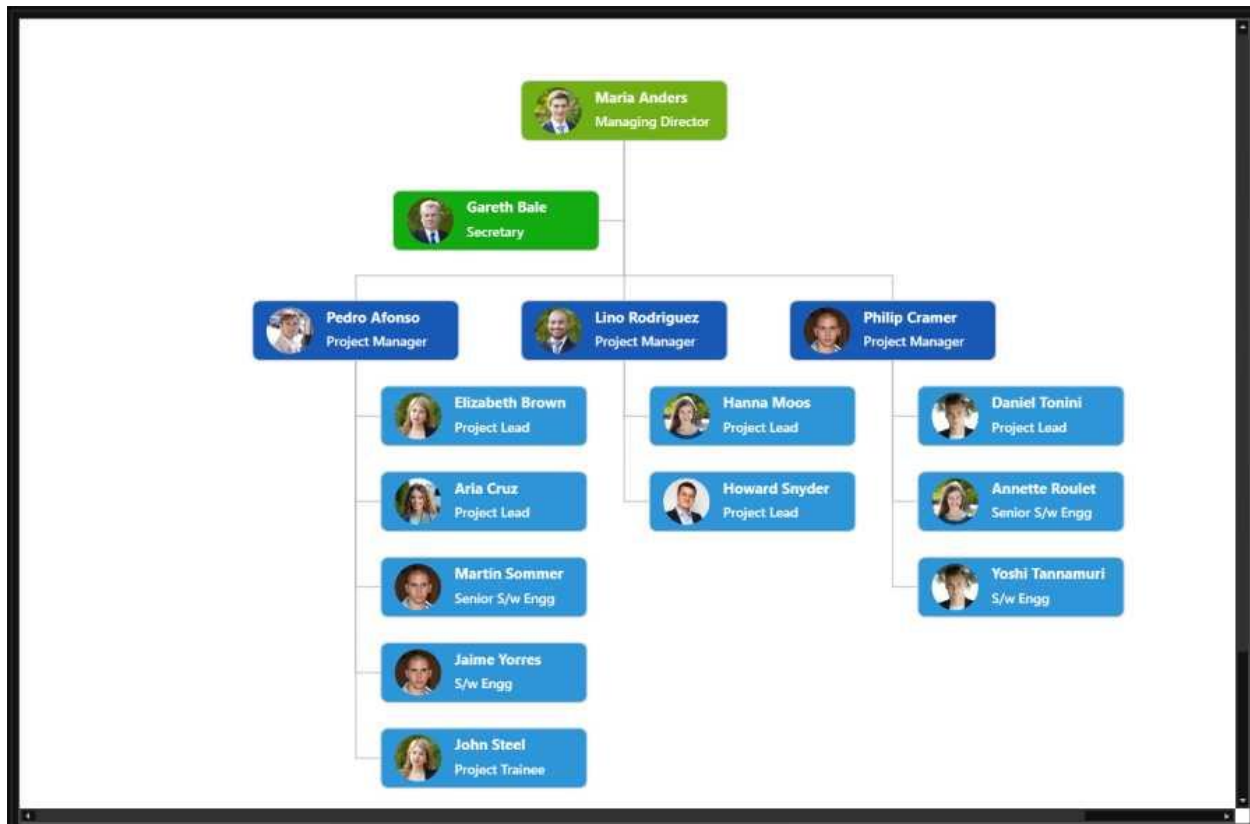
[View Sample in GitHub](#)

For more information about Layout, [refer](#)

## Theme

SfDiagram supports various built-in themes. Refer to the below links to apply themes for the SfDiagram,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



**Note:** [View Sample in GitHub](#)

## Node in WPF Diagram (SfDiagram)

The nodes are graphical objects used to visually represent the geometrical information, process flow, internal business procedure, or any other kind of data, and it represents the functions of a complete system in regards of 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 on the Diagram area from bottom to top in the order they are added.

### Add Node through Nodes collection

To create a node, you have to define the node object and add that to nodes collection of the Diagram.

### XML

```
<!--Resource Dictionary which contains predefined shapes for Node-->
<ResourceDictionary.MergedDictionaries>
  <ResourceDictionary
    Source="/Syncfusion.SfDiagram.Wpf;component/Resources/BasicShapes.xaml"/>
  </ResourceDictionary.MergedDictionaries>
<!--Shape style for Node-->
<Style x:Key="ShapeStyle" TargetType="Path">
  <Setter Property="Fill" Value="#FF5B9BD5"/>
  <Setter Property="Stretch" Value="Fill"/>
</Style>
```



```

<Setter Property="Stroke" Value="#FFEDF1F6"/>
</Style>
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <syncfusion:SfDiagram.Nodes>
    <!--Initialize the NodeCollection-->
    <syncfusion:NodeCollection>
      <!--Initialize the Node-->
      <syncfusion:NodeViewModel ID="Begin" OffsetX="300" OffsetY="60"
        Shape="{StaticResource Ellipse}"
        ShapeStyle="{StaticResource ShapeStyle}"
        UnitHeight="40" UnitWidth="120"/>
    </syncfusion:NodeCollection>
  </syncfusion:SfDiagram.Nodes>
</syncfusion:SfDiagram>

```

## C#

```

//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize NodeCollection to SfDiagram
diagram.Nodes = new NodeCollection();
//Initialize ConnectorCollection to SfDiagram
diagram.Connectors = new ConnectorCollection();
//Creating the NodeViewModel
NodeViewModel Begin = new NodeViewModel()
{
    ID = "Begin",
    UnitWidth = 120,
    UnitHeight = 40,
    OffsetX = 300,
    OffsetY = 60,
    //Specify shape to the Node from built-in Shape Dictionary
    Shape = App.Current.Resources["Ellipse"],
    //Apply style to Shape
    ShapeStyle = App.Current.Resources["ShapeStyle"] as Style,
};
//Add Node to Nodes property of the Diagram
(diagram.Nodes as NodeCollection).Add(Begin);

```

Now, node will be as follows.



[View sample in GitHub](#)

*Add node from stencil*

Nodes can be predefined and added to the stencil and can be dropped into the Diagram when needed. For more information about adding Nodes from Stencil, refer to the [Stencil](#).

*Create node through data source*

Nodes can be generated automatically with the information provided through data source. For more information about data source, refer to the [Data Source](#).

### Draw nodes

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

### Visualize a node

You can use text, image, controls, panels, or any UIElement or template to visualize a node as follows:

1)Content template 2)Content 3)Geometry 4)Custom shapes 5)Built-in resource

### Using content template

Node is a ContentControl, so you can use data template to display the content of the node using the ContentTemplate property. Refer to the following code example to define node's shape through ContentTemplate.

#### XML

```
<DataTemplate x:Key="NodeTemplate">
  <Border BorderThickness="2" BorderBrush="Black">
    <TextBlock Text="NodeTemplate" Width="100" Height="100" Background="White"/>
  </Border>
</DataTemplate>
<!--Initialize the Node-->
<syncfusion:NodeViewModel UnitHeight="100" UnitWidth="100"
  OffsetX="100" OffsetY="100"
  ContentTemplate="{StaticResource NodeTemplate}"/>
```

#### C#

```
//Define the Node
NodeViewModel node = new NodeViewModel()
{
  //sets the size
  UnitHeight = 100,
  UnitWidth = 100,
  //sets the position
  OffsetX = 100,
  OffsetY = 100,
  ContentTemplate = App.Current.Resources["NodeTemplate"] as DataTemplate
};
//Adding Node to Collection
(diagram.Nodes as NodeCollection).Add(node);
```

### Using content

Node is a ContentControl, so you can set text, image, or any UIElement as node content using the Content property. Refer to the following code example to define image as content of the node.

#### XML

```
<!--Initialize the Node-->
<syncfusion:NodeViewModel UnitHeight="100" UnitWidth="100"
  OffsetX="100" OffsetY="100">
  <!--Assigning user image as Node's Content-->
  <syncfusion:NodeViewModel.Content>
    <Image Source="/Image/user_image.png" Height="100" Width="100"/>
  </syncfusion:NodeViewModel.Content>
</syncfusion:NodeViewModel>
```

**C#**

```

Image img = new Image();
BitmapImage bmp = new BitmapImage();
bmp.BeginInit();
bmp.UriSource = new Uri("Image/user_image.png", UriKind.Relative);
bmp.EndInit();
img.Stretch = Stretch.Fill;
img.Source = bmp;
img.Height = 100;
img.Width = 100;
//Define the Node
NodeViewModel node = new NodeViewModel()
{
    //sets the size
    UnitHeight = 100,
    UnitWidth = 100,
    //sets the position
    OffsetX = 100,
    OffsetY = 100,
    //set image as content
    Content = img
};
//Adding Node to Collection
(diagram.Nodes as NodeCollection).Add(node);

```

*Using geometry*

The Shape property of the node class allows to visualize any geometry path as node's content. Refer to the following code example to define geometry for node's shape.

**XML**

```

<Style TargetType="syncfusion:Node">
  <Setter Property="ShapeStyle">
    <Setter.Value>
      <Style TargetType="Path">
        <Setter Property="Fill" Value="#FF5B9BD5"/>
        <Setter Property="Stretch" Value="Fill"/>
        <Setter Property="Stroke" Value="#FFEDF1F6 "/>
      </Style>
    </Setter.Value>
  </Setter>
</Style>
<!--Initialize the Node-->
<syncfusion:NodeViewModel UnitWidth="100" UnitHeight="100"
  OffsetX="100" OffsetY="100">
  <syncfusion:NodeViewModel.Shape>
    <RectangleGeometry Rect="100,100,100,100"/>
  </syncfusion:NodeViewModel.Shape>
</syncfusion:NodeViewModel>

```

**C#**

```

//Define the Node

```

```

NodeViewModel node = new NodeViewModel()
{
    //sets the size
    UnitHeight = 100,
    UnitWidth = 100,
    //sets the position
    OffsetX = 100,
    OffsetY = 100,
    Shape = new RectangleGeometry() { Rect = new Rect(100, 100, 100, 100) }
};
//Adding Node to Collection
(diagram.Nodes as NodeCollection).Add(node);

```

### Custom shapes

How to add custom shape using path data

Refer to the following code example to define custom path as node's shape.

### XML

```

<!--Namespace to define String in XAML-->
xmlns:sys="clr-namespace:System;assembly=mscorlib"
<sys:String x:Key="Rectangle">
M242,1078L231,1078L231,1067L242,1067z
</sys:String>
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
<syncfusion:SfDiagram.Nodes>
<!--Initialize the NodeCollection-->
<syncfusion:NodeCollection>
<!--Initialize the Node-->
<syncfusion:NodeViewModel UnitWidth="100" UnitHeight="100"
OffsetX="100" OffsetY="100"
Shape="{StaticResource Rectangle}"
ShapeStyle="{StaticResource ShapeStyle}"/>
</syncfusion:NodeCollection>
</syncfusion:SfDiagram.Nodes>
</syncfusion:SfDiagram>

```

### C#

```

//Define the Node
NodeViewModel node = new NodeViewModel()
{
    //sets the size
    UnitHeight = 100,
    UnitWidth = 100,
    //sets the position
    OffsetX = 100,
    OffsetY = 100,
    Shape= App.Current.Resources["Rectangle"] as Style,
    //Apply style to Shape
    ShapeStyle = App.Current.Resources["ShapeStyle"] as Style,
};
//Adding Node to Collection
(diagram.Nodes as NodeCollection).Add(node);

```



How to add custom shape using data template

Refer to the following code example to define data template for node's custom shape,

#### XML

```
<DataTemplate x:Key="Square">
<Path Stretch="Fill" Data="M242,1078L231,1078L231,1067L242,1067z"
Fill="#FF5B9BD5" Stroke="#FFC4C4C4" StrokeThickness="2"/>
</DataTemplate>
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
<syncfusion:SfDiagram.Nodes>
<!--Initialize the NodeCollection-->
<syncfusion:NodeCollection>
<!--Initialize the Node-->
<syncfusion:NodeViewModel UnitWidth="100" UnitHeight="100" OffsetX="100"
OffsetY="100" ContentTemplate="{StaticResource Square}" />
</syncfusion:NodeCollection>
</syncfusion:SfDiagram.Nodes>
</syncfusion:SfDiagram>
```

#### C#

```
//Define the Node
NodeViewModel node = new NodeViewModel()
{
    //sets the size
    UnitHeight = 100,
    UnitWidth = 100,
    //sets the position
    OffsetX = 100,
    OffsetY = 100,
    ContentTemplate = this.Resources["Square"] as DataTemplate,
};
//Adding Node to Collection
(diagram.Nodes as NodeCollection).Add(node);
```



[How to add image as node shape](#)

Refer to the following code example to add image as node shape,

#### XML

```
<DataTemplate x:Key="User">
  <Image Stretch="Uniform" HorizontalAlignment="Center"
    Source="Images\User.png"/>
</DataTemplate>
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <syncfusion:SfDiagram.Nodes>
    <!--Initialize the NodeCollection-->
    <syncfusion:NodeCollection>
      <!--Initialize the Node-->
      <syncfusion:NodeViewModel UnitWidth="100" UnitHeight="100" OffsetX="100"
        OffsetY="100" ContentTemplate="{StaticResource User}" />
    </syncfusion:NodeCollection>
  </syncfusion:SfDiagram.Nodes>
</syncfusion:SfDiagram>
```

#### C#

```
//Define the Node
NodeViewModel node = new NodeViewModel()
{
  //sets the size
  UnitHeight = 100,
  UnitWidth = 100,
  //sets the position
  OffsetX = 100,
  OffsetY = 100,
  ContentTemplate = this.Resources["User"] as DataTemplate,
};
//Adding Node to Collection
(diagram.Nodes as NodeCollection).Add(node);
```



[How to add framework element as node content](#)

Refer to the following code example to add framework elements as node content,

#### XML

```
<DataTemplate x:Key="UserProfile">
  <Border BorderThickness="1" Background="#2E95D8"
    BorderBrush="LightGray">
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="60" />
      </Grid.ColumnDefinitions>
    </Grid>
  </Border>
</DataTemplate>
```

```

<ColumnDefinition Width="100" />
</Grid.ColumnDefinitions>
<Grid Grid.Column="0">
<Border Grid.Column="0" VerticalAlignment="Stretch"
Background="Transparent"
BorderBrush="#FF5DC3B1"
Padding="5">
<Image Stretch="Uniform" HorizontalAlignment="Center"
Source="Images\User.png"/>
</Border>
</Grid>
<Grid Grid.Column="1">
<Grid.RowDefinitions>
<RowDefinition Height="25" />
<RowDefinition Height="25" />
</Grid.RowDefinitions>
<TextBlock x:Name="Name" Grid.Row="0"
HorizontalAlignment="Left"
VerticalAlignment="Center"
FontFamily="Segoe UI"
FontSize="12"
FontWeight="Bold"
Foreground="White"
Text="Daniel Tonini"
TextAlignment="Left" />
<TextBlock x:Name="Designation" Grid.Row="1"
HorizontalAlignment="Left"
VerticalAlignment="Top"
FontFamily="Segoe UI"
FontSize="11"
FontWeight="SemiBold"
Foreground="White"
Text="Project Lead"
TextAlignment="Left" />
</Grid>
</Grid>
</Border>
</DataTemplate>
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
<syncfusion:SfDiagram.Nodes>
<!--Initialize the NodeCollection-->
<syncfusion:NodeCollection>
<!--Initialize the Node-->
<syncfusion:NodeViewModel UnitWidth="150" UnitHeight="50" OffsetX="100"
OffsetY="100" ContentTemplate="{StaticResource UserProfile}" />
</syncfusion:NodeCollection>
</syncfusion:SfDiagram.Nodes>
</syncfusion:SfDiagram>

```

## C#

```

//Define the Node
NodeViewModel node = new NodeViewModel()
{
//sets the size

```

```

UnitHeight = 50,
UnitWidth = 150,
//sets the position
OffsetX = 100,
OffsetY = 100,
ContentTemplate = this.Resources["UserProfile"] as DataTemplate,
};
//Adding Node to Collection
(diagram.Nodes as NodeCollection).Add(node);

```



#### Built-in resource

Some basic built-in shapes are provided as ResourceDictionary. For more information, refer to the [Shapes](#).

[View Sample in GitHub](#)

#### Position

Position of a node is controlled by using its `OffsetX` and `OffsetY` properties. By default, these Offset properties represent the distance between 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 solve this problem. Default value of node's pivot point is (0.5, 0.5), that means center of node.

The following table explains how pivot relates Offset values with node boundaries:

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

#### XML

```

<!--Initialize the Node with Pivot-->
<syncfusion:NodeViewModel UnitHeight="65" UnitWidth="100"
OffsetX="100" OffsetY="100" Pivot="0,0"
Shape="{StaticResource Rectangle}"/>

```

#### C#

```

//Define the Node
NodeViewModel node = new NodeViewModel()
{
    //sets the size
    UnitHeight = 65,
    UnitWidth = 100,
    //sets the position

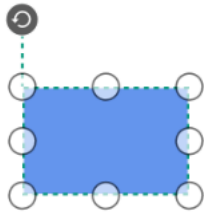
```



```

OffsetX = 100,
OffsetY = 100,
//sets the Pivot point
Pivot=new Point(0,0),
Shape = new RectangleGeometry() { Rect = new Rect(0, 0, 10, 10) },
};
//Adding Node to Collection
(diagram.Nodes as NodeCollection).Add(node);

```



### Flip

Diagram provides support to flip the node. **Flip** is performed to give the mirrored node of the original element.

The flip types are:

- Flip

**Flip** that involves both vertical and horizontal changes of the element.

- VerticalFlip

**VerticalFlip** that involves changes in the vertical direction of the element.

- HorizontalFlip

**HorizontalFlip** that involves changes in the horizontal direction of the element.

### XML

```

<!--Initialize Vertical Flip to the Node-->
<syncfusion:NodeViewModel Flip="VerticalFlip"
UnitHeight="100" UnitWidth="100"
OffsetX="200" OffsetY="100"
Shape="{StaticResource Triangle}"
ShapeStyle="{StaticResource ShapeStyle}"/>

```

### C#

```

//Define the Node
NodeViewModel node = new NodeViewModel()
{
UnitHeight = 100,
UnitWidth = 100,
OffsetX=200,
OffsetY=100,

```

```
//Initialize Vertical Flip to the Node
Flip = Flip.VerticalFlip,
Shape = App.Current.Resources["Triangle"],
ShapeStyle = App.Current.Resources["ShapeStyle"] as Style,
};
//Adding Node to Collection
(diagram.Nodes as NodeCollection).Add(node);
```



Before Flip



After Flip

### Padding

Padding is used to leave space between the connector's end point and the object to where it is connected. The `ConnectorPadding` property of node defines the space to be left between the node bounds and its edges.

### XML

```
<syncfusion:SfDiagram.Nodes>
<syncfusion:NodeCollection>
<!--Initialize connector padding to the Node-->
<syncfusion:NodeViewModel ID="node1" ConnectorPadding="5"
UnitHeight="65" UnitWidth="100"
OffsetX="200" OffsetY="200"
Shape="{StaticResource Rectangle}"
ShapeStyle="{StaticResource ShapeStyle}"/>
<syncfusion:NodeViewModel ID="node2" UnitHeight="65" UnitWidth="100"
OffsetX="400" OffsetY="200"
Shape="{StaticResource Rectangle}"
ShapeStyle="{StaticResource ShapeStyle}"/>
</syncfusion:NodeCollection>
</syncfusion:SfDiagram.Nodes>
<syncfusion:SfDiagram.Connectors>
<syncfusion:ConnectorCollection>
<!--Establish connection between the nodes-->
<syncfusion:ConnectorViewModel SourceNodeID="node1" TargetNodeID="node2"/>
</syncfusion:ConnectorCollection>
</syncfusion:SfDiagram.Connectors>
```

### C#

```
//Define NodeProperty
NodeViewModel node1 = AddNode(200, 200, 65, 100);
//Space between Connector and Node
node1.ConnectorPadding = 5;
//Adding Node to Collection
(diagram.Nodes as NodeCollection).Add(node1);
NodeViewModel node2 = AddNode(400, 200, 65, 100);
```

```

(diagram.Nodes as NodeCollection).Add(node2);
//Define ConnectorCollection
diagram.Connectors = new ConnectorCollection();
ConnectorViewModel conn1 = new ConnectorViewModel()
{
    SourceNode = node1,
    TargetNode=node2
};
//Adding Connector to Collection
(diagram.Connectors as ConnectorCollection).Add(conn1);
//Method for Creating Node
public NodeViewModel AddNode(double offsetX, double offsetY, double
height, double width)
{
    NodeViewModel node = new NodeViewModel();
    node.OffsetX = offsetX;
    node.OffsetY = offsetY;
    node.UnitHeight = height;
    node.UnitWidth = width;
    node.Shape = new RectangleGeometry { Rect = new Rect(0, 0, 10, 10) };
    return node;
}

```



### Appearance

You can customize the appearance of a node by changing its **ShapeStyle**. The following code explains how to customize the appearance of the **Shape**.

### XML

```

<Style TargetType="Path" x:key="shapestyle">
  <Setter Property="Fill" Value="#FF41719C"/>
  <Setter Property="Stretch" Value="Fill"/>
  <Setter Property="Stroke" Value="#FFEDF1F6"/>
  <Setter Property="StrokeDashArray" Value="4,5"/>
  <Setter Property="StrokeThickness" Value="2"/>
</Style>

```

### C#

```

Style style = new Style(typeof(Path));
style.Setters.Add(new Setter(Path.FillProperty, Brushes.SteelBlue));
style.Setters.Add(new Setter(Path.StrokeProperty, Brushes.WhiteSmoke));
style.Setters.Add(new Setter(Path.StrokeThicknessProperty, 2d));
style.Setters.Add(new Setter(Path.StrokeDashArrayProperty, new
DoubleCollection() { 5 }));
style.Setters.Add(new Setter(Path.StretchProperty, Stretch.Fill));
return style;

```



[View Sample in GitHub](#)

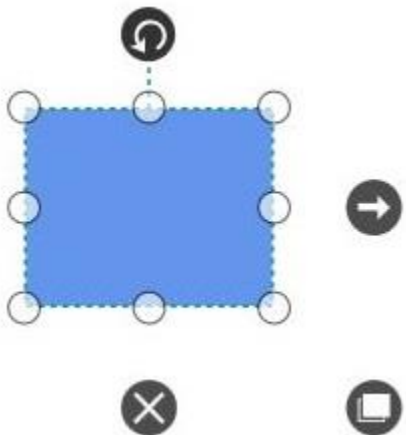
### Interaction

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

#### Select

Node can be selected by clicking (tap) it.

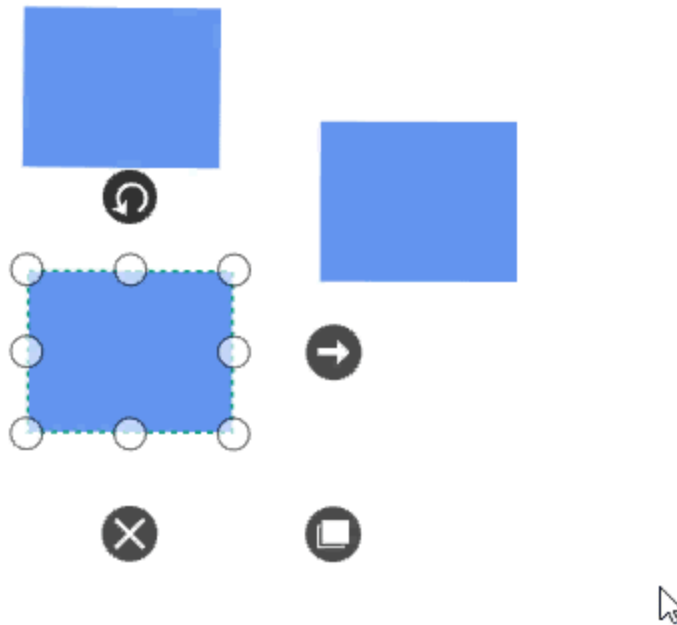
- The `IsSelected` Property is used to select or unselect the node at runtime.
- `ItemSelectingEvent` and `ItemSelectedEvent` for selecting an element, will notify you the item and its original source. To explore about arguments, refer to the [DiagramPreviewEventArgs](#) and [ItemSelectedEventArgs](#).
- `ItemUnselectingEvent` and `ItemUnselectedEvent` for unselecting an element, will notify you the item and its original source. To explore about arguments, refer to the [DiagramPreviewEventArgs](#) and [DiagramEventArgs](#).



To explore about selection and selection related events, refer to the [Selection](#).

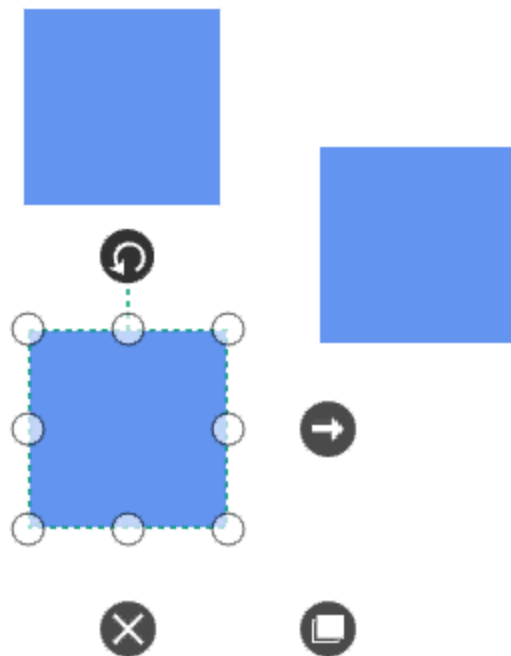
#### Drag

- Selected 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.
- Instead of dragging original object, preview of the node alone can be dragged. For preview dragging, refer to the [PreviewSettings](#).
- While dragging, the objects are snapped towards the nearest objects to make better alignments. For better alignments, refer to the [Snapping](#).
- The `NodeChangedEvent` will notify the `OffsetX` and `OffsetY` changes with their old and new values. Along with that, this event will give information about interaction state. To explore about arguments, refer to the [NodeChangedEventArgs](#).



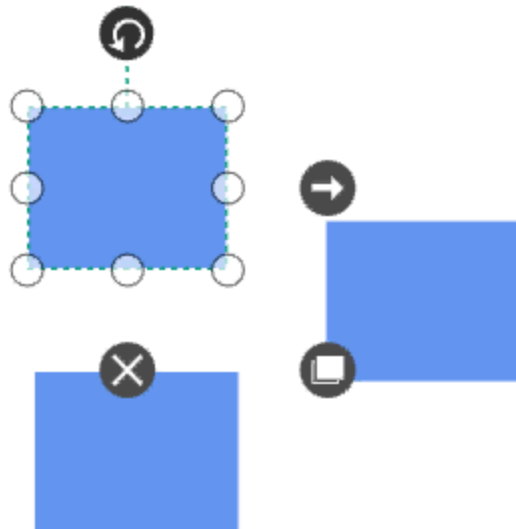
### Resize

- Selector is surrounded by eight thumbs. When dragging these thumbs, selected items can be resized smaller or larger.
- When one corner of the selector is dragged, opposite corner is in a static position.
- Enable [AspectRatio](#) NodeConstraints to maintain the aspect ratio of the node when its being resized.
- While resizing, the objects are snapped towards the nearest objects to make better alignments. For better alignments, refer to the [Snapping](#).
- The `NodeChangedEvent` will notify the `UnitHeight` and `UnitWidth` changes with their old and new values. Along with that, this event will give information about interaction state. To explore about arguments, refer to the [NodeChangedEventArgs](#).



### 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 when rotating the node to represent the static point. For more information about pivot, refer to [Position](#)
- The `NodeChangedEvent` will notify the `RotateAngle` changes with their old and new values. Along with that, this event will give information about interaction state. To explore about arguments, refer to the [NodeChangedEventArgs](#).



## Events

- The `ItemTappedEvent` is invoked on clicking the node. To explore about arguments, refer to the [ItemTappedEventArgs](#).
- The `ItemDoubleTappedEvent` is invoked on double-clicking the node. To explore about arguments, refer to the [ItemDoubleTappedEventArgs](#).
- The `MouseDown` and `MouseUp` are invoked as similar to framework element that is raised together with either `MouseLeftButtonUp` or `MouseRightButtonUp`. To explore about arguments, refer to the [MouseDownEventArgs](#) and

[MouseUpEventArgs](#).

## 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 add Annotations to the Node?](#)
- [How to add Port to the Node?](#)
- [How to add Nodes to the stencil?](#)
- [How to apply built-in theme for node and connector?](#)
- [How to customize the connection indicator style of node and port?](#)
- [How to host different UI elements as node content?](#)

- [How to restrict the child node dragging whereas allow group dragging?](#)
- [How to restrict the node dragging within boundaries?](#)
- [How to show assistants to the parent node of the organization layout?](#)
- [How to create port at runtime though SetTool?](#)
- [How to customize the context menu?](#)
- [How to drag node from one diagram to another diagram?](#)

## Shapes in WPF Diagram (SfDiagram)

We have provided some basic built-in shapes as ResourceDictionary.

The following code example illustrates how to merge shapes into diagram.

### XML

```
<ResourceDictionary.MergedDictionaries>
<!--Initialize Shapes-->
<ResourceDictionary
Source="/Syncfusion.SfDiagram.Wpf;component/Resources/BasicShapes.xaml" />
</ResourceDictionary.MergedDictionaries>
```

## Basic Shapes

The following code example illustrates how to assign Shape property of the Node.

### XML

```
<!--Style for Node-->
<Style TargetType="syncfusion:Node">
<Setter Property="ShapeStyle">
<Setter.Value>
<Style TargetType="Path">
<Setter Property="Fill" Value="CornflowerBlue"/>
<Setter Property="Stretch" Value="Fill"/>
<Setter Property="Stroke" Value="Black"/>
</Style>
</Setter.Value>
</Setter>
</Style>
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
<!--Initialize the NodeCollection-->
<syncfusion:SfDiagram.Nodes>
<syncfusion:NodeCollection>
<!--Add Node with basic shape-->
<syncfusion:NodeViewModel x:Name="Node" UnitHeight="100" UnitWidth="100"
OffsetX="100" OffsetY="100"
Shape="{StaticResource Rectangle}"/>
</syncfusion:NodeCollection>
</syncfusion:SfDiagram.Nodes>
</syncfusion:SfDiagram>
```

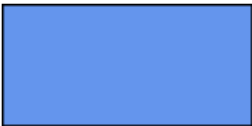
### C#

```
//Initialize the diagram
SfDiagram diagram = new SfDiagram();
//Initialize the NodeViewModel
```



```
NodeViewModel node = new NodeViewModel()  
{  
    UnitHeight = 100,  
    UnitWidth = 100,  
    OffsetX = 100,  
    OffsetY = 100,  
    Shape = new RectangleGeometry() { Rect = new Rect(0, 0, 10, 10) },  
};  
// Add the node into Nodes collection  
(diagram.Nodes as NodeCollection).Add(node);
```

Output Node will be,

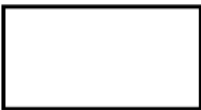


The list of shapes are available in the resource dictionary as follows

| Shape Category | Shape Name | Output Shape |

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

| Basic Shapes | Rectangle |



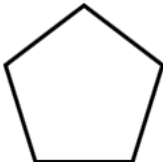
| | Triangle |

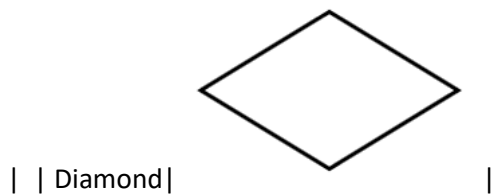
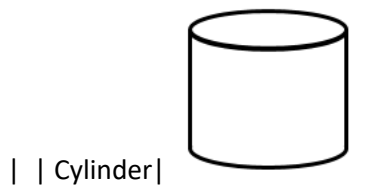
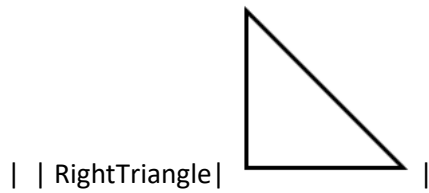
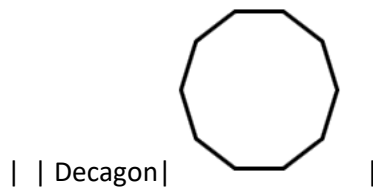
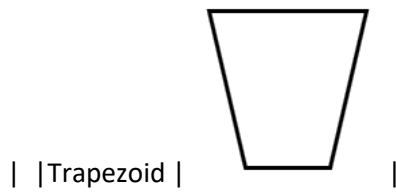
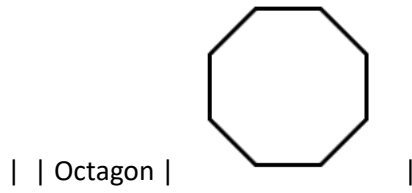
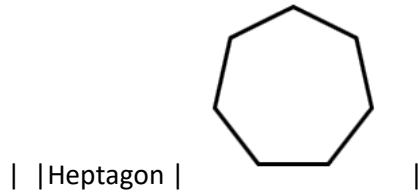
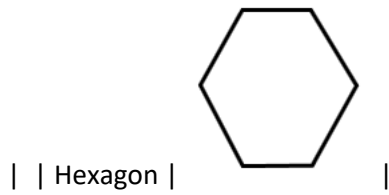


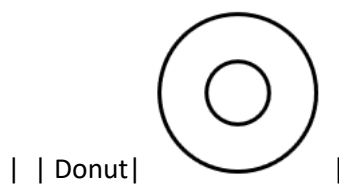
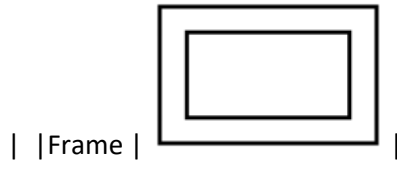
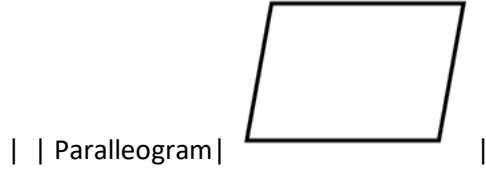
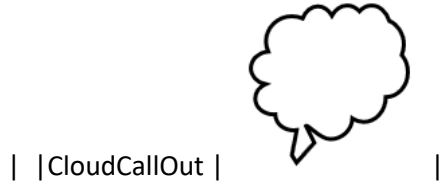
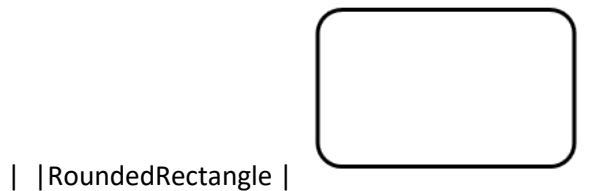
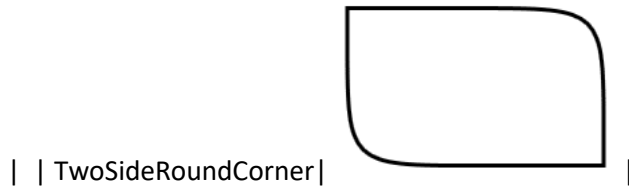
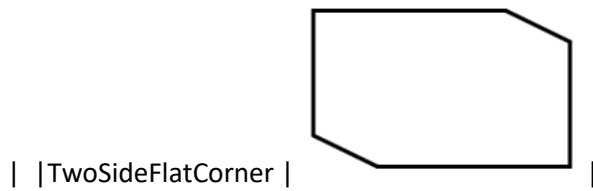
| | Plus |

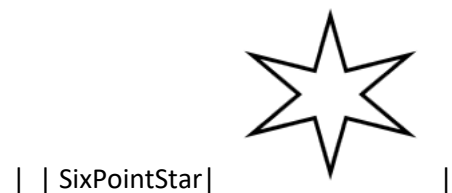
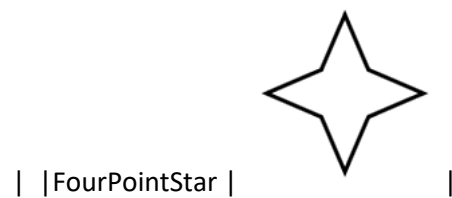
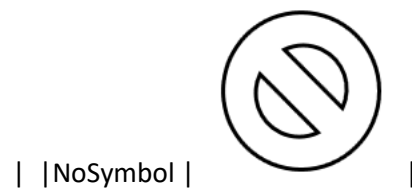
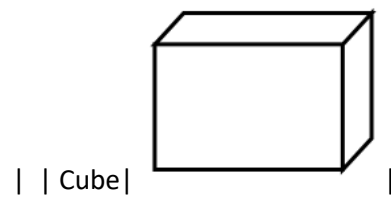
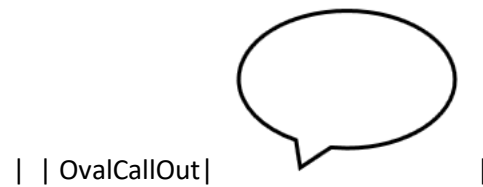
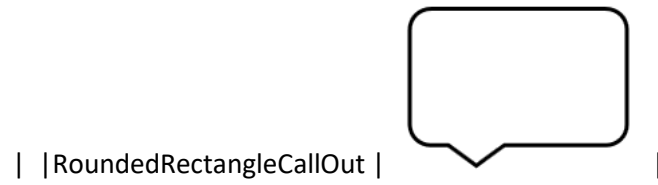
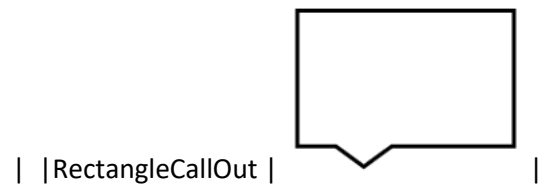


| | Pentagon |









| |SevenPointStar |



| |SixteenPointStar |



| |TwentyFourPointStar |



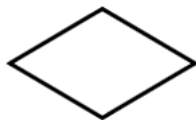
| |ThirtyTwoPointStar |



| Flow Shapes |Process |



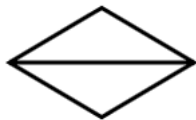
| |Decision|



| |Terminator |

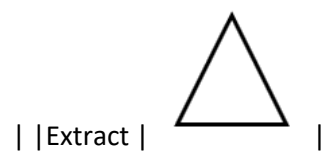
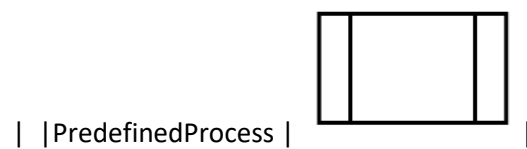
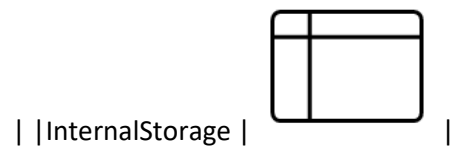
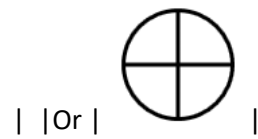
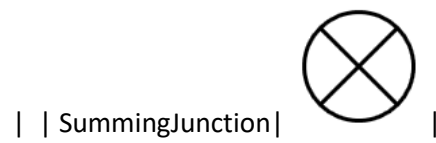
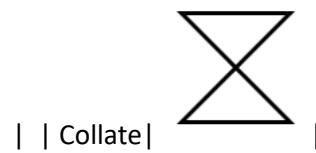
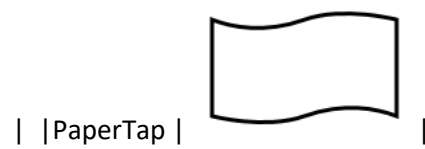
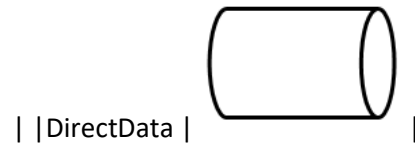
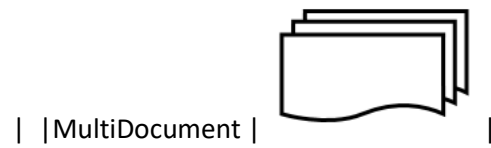


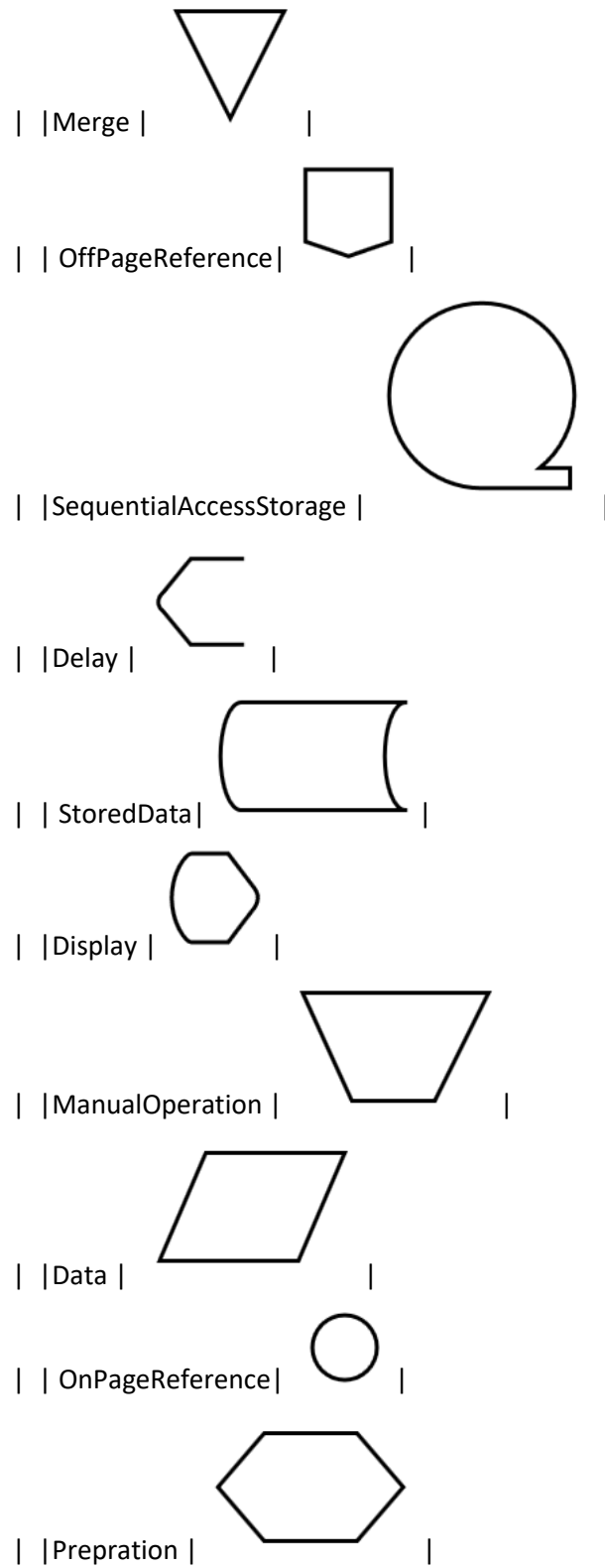
| |Sort |

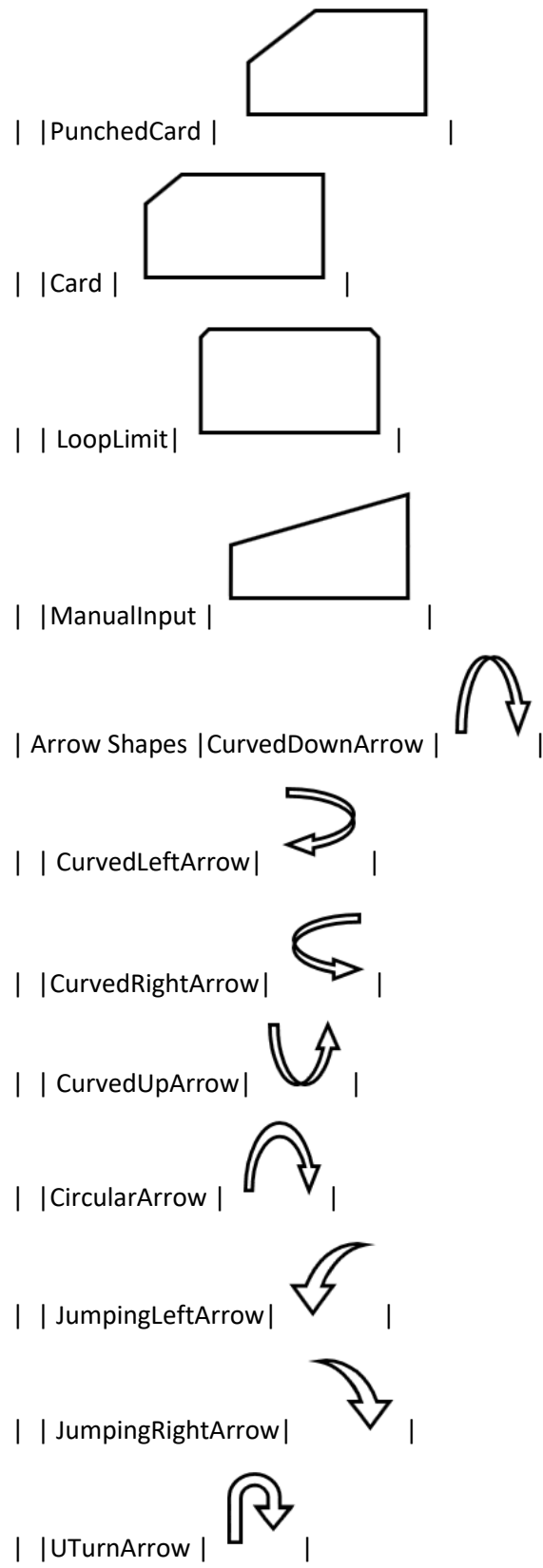


| |Document |

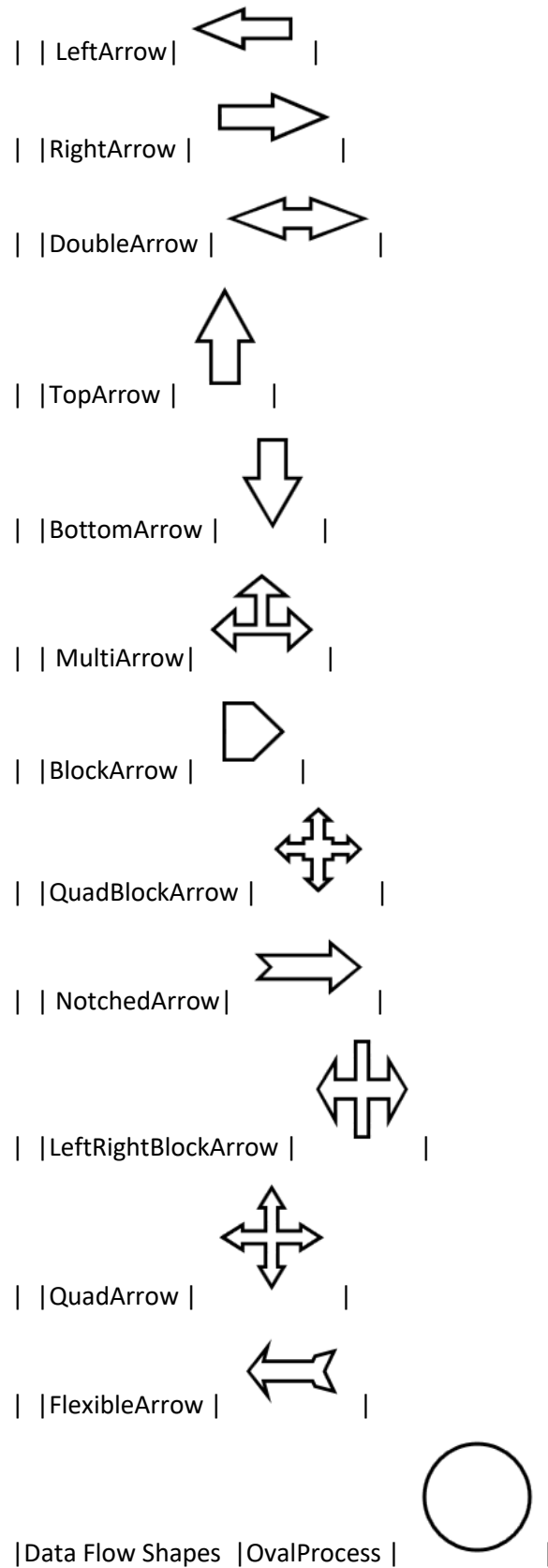






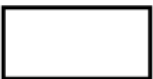






|


| ExternalIndicator |



|

|

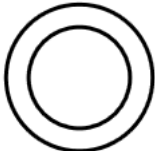
| Object |



|

|

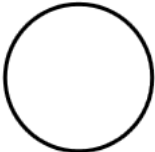
| MultipleProcess |



|

|

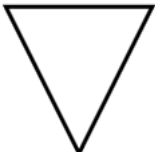
| State |



|

|


| Stop1 |



|

|

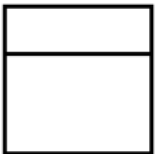
| Entity1 |



|

|


| Entity2 |



|

| Electrical Shapes |


| Resistor |



|

|


| Capacitor |



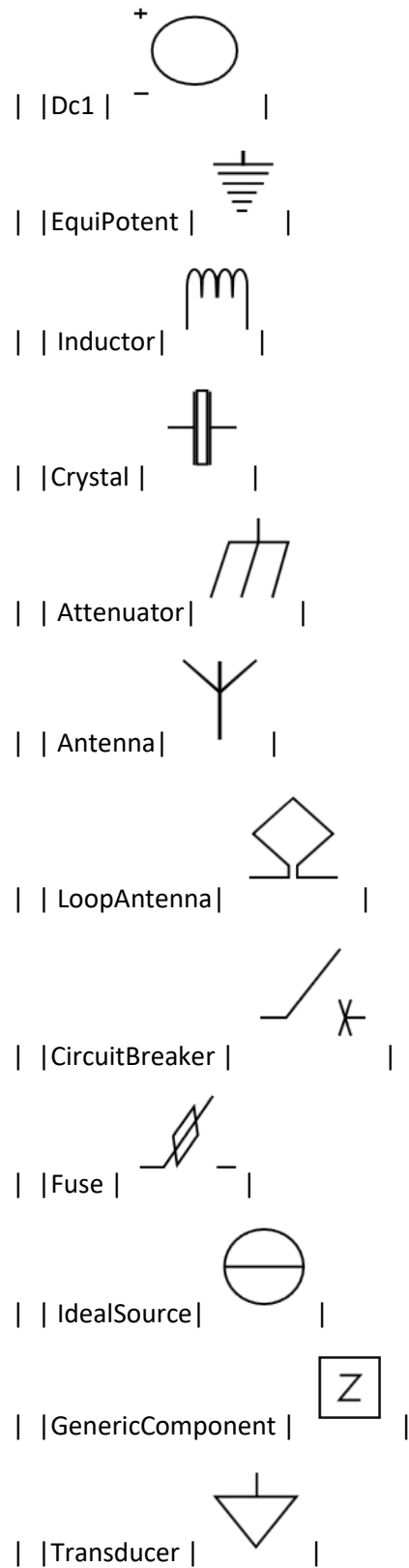
|

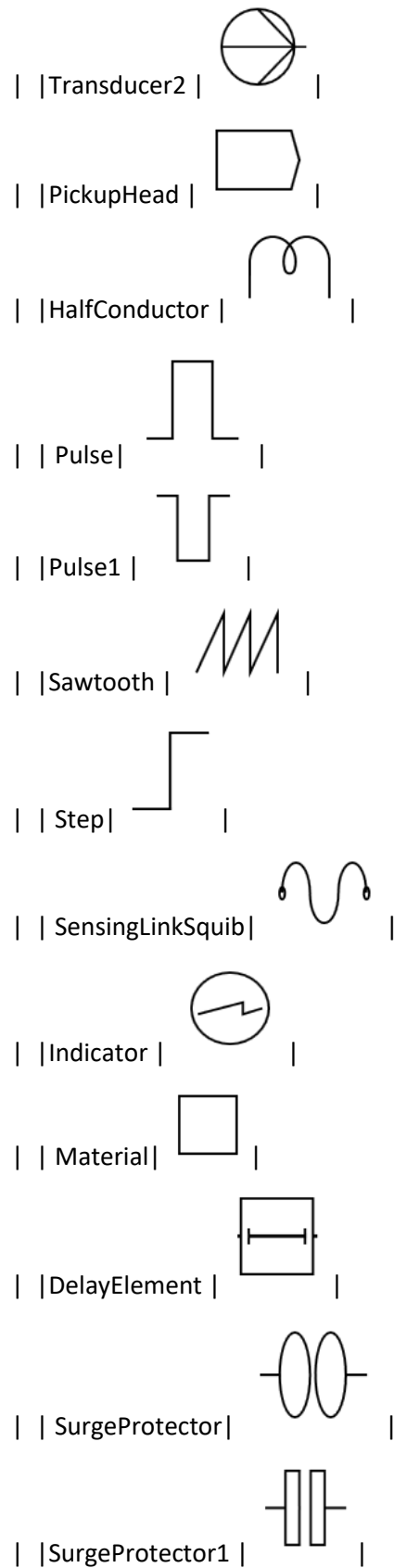
|

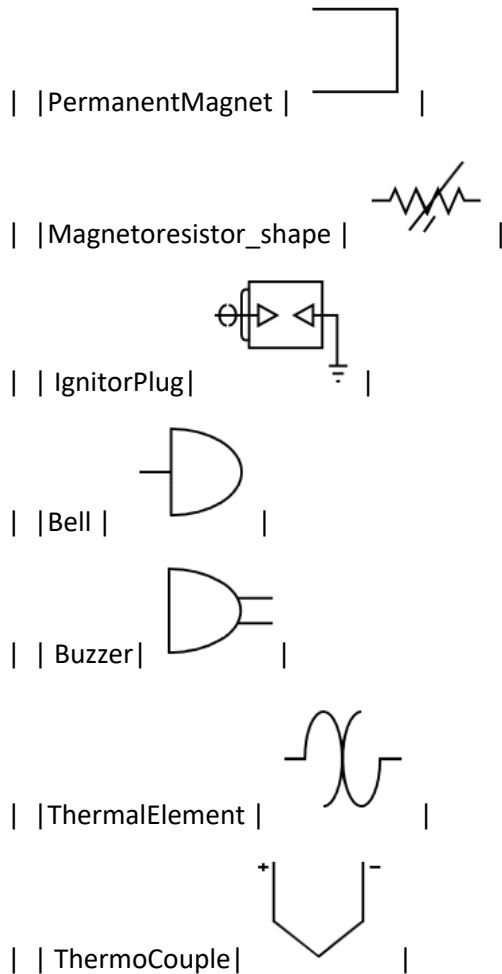
| AC |



|







Find the [Shapes sample](#) to depict the shapes.

### BPMN Shapes in WPF Diagram (SfDiagram)

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 BPMN shapes, you have to initialize [BpmnNodeViewModel](#) with the [Type](#) property. The Type property can be set to any one of the built-in bpmn shapes using the BpmnShapeType enumeration. The default value for the Type property of BpmnNodeViewModel is "Activity".

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

### XML

```
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Initialize the Node-->
  <syncfusion:SfDiagram.Nodes>
    <!--Initialize the Node Collection-->
    <syncfusion:NodeCollection>
      <!--Initialize the BpmnNodeViewModel-->
      <syncfusion:BpmnNodeViewModel UnitHeight="70" UnitWidth="100" OffsetX="100"
      OffsetY="100" Type="Activity">
    </syncfusion:BpmnNodeViewModel>
  </syncfusion:NodeCollection>
</syncfusion:SfDiagram>
```

```
</syncfusion:SfDiagram.Nodes>
</syncfusion:SfDiagram>
```


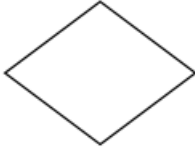


**C#**

```
//Initialize the diagram.
SfDiagram diagram = new SfDiagram();
//Initialize the BpmnNodeViewModel.
BpmnNodeViewModel node = new BpmnNodeViewModel()
{
    OffsetX = 100,
    OffsetY = 100,
    UnitHeight = 70,
    UnitWidth = 100,
    Type = BpmnShapeType.Activity,
};
// Add the node into the Node's collection.
(Diagram.Nodes as NodeCollection).Add(node);
```

```
{% endtabs %}
```



The list of supported BPMN shapes are as follows:

Shape	Symbol	Description
Event		Event shape represents something happens during a business process
Gateway		Gateway is used to control the flow of a process
Activity instance		Activities describe the kind of work being done in a particular process
Message		The message is just the content of the communication



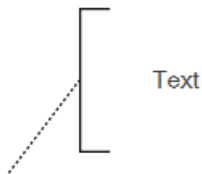
| DataStore |  
process |

| DataStore is used to store or access data associated with a business



| DataObject |

| A DataObject represents information flowing using 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 |



| TextAnnotation |

| A TextAnnotation points at or references the another BPMN shape, which we call as the TextAnnotationTarget of the TextAnnotation |



| Group |

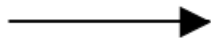
| Organize tasks or processes that have significance in the overall process. |

| Organize tasks or processes that have



| ExpandedSubprocess |  
the extended version of the Group |

| ExpandedSubProcess is



| Sequenceflow |  
objects. |

| Sequence flows represent the typical path between two flow



| DefaultSequenceflow |  
with a tic mark on the one end |

| Default sequence flows are represented by an arrow



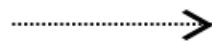
| ConditionalSequenceflow |  
flow of a process based on the certain conditions |

| Conditional sequence flows are used to control the



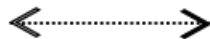
| Association |  
opened arrow. |

| An Association is represented as a dotted graphical line with an



| DirectionalAssociation |  
graphical line with one side arrow. |

| DirectionalAssociation is represented as a dotted



| BiDirectionalAssociation |  
graphical line with the double side arrow. |

| BiDirectionalAssociation is represented as a dotted



| MessageFlow |  
participants and is represented by line. |

| A MessageFlow flow shows the flow of messages between two



| InitiatingMessageflow |  
another pool |

| An activity or event in one pool can initiate a message to





| NonInitiatingMessageflow |  
message to another pool |

| An activity or event in one pool can't initiate a

Please find the BPMN Editor sample as follows.

[View BPMN Editor sample in GitHub](#)

### Connector in WPF Diagram (SfDiagram)

Connectors are objects used to create link between two points or nodes to indicate the flow of operation or relationships between them.



### Connector types

Diagram supports to create five types of connectors. They are:

- Line
- Orthogonal
- CubicBezier
- QuadraticBezier
- PolyLine

The [DefaultConnectorType](#) property allows you to change the connector type. By default, the diagram connector type is **Orthogonal**.

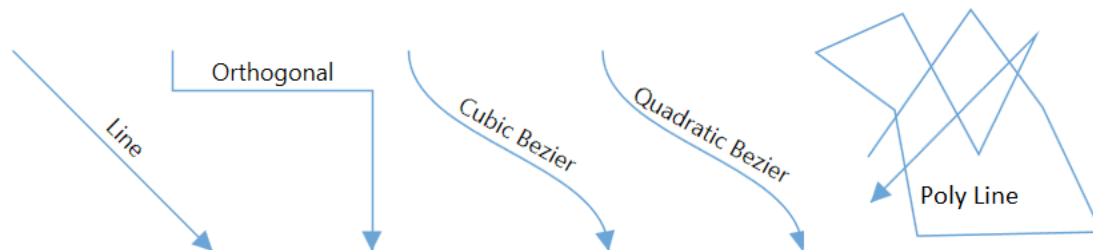
### XML

```
<!--Style for the Connector-->
<Style TargetType="syncfusion:Connector">
  <Setter Property="ConnectorGeometryStyle">
    <Setter.Value>
      <Style TargetType="Path">
        <Setter Property="Stroke" Value="#6BA5D7"/>
        <Setter Property="StrokeThickness" Value="1"/>
      </Style>
    </Setter.Value>
  </Setter>
  <Setter Property="TargetDecoratorStyle">
    <Setter.Value>
      <Style TargetType="Path">
        <Setter Property="Fill" Value="#6BA5D7"/>
        <Setter Property="StrokeThickness" Value="1"/>
      </Style>
    </Setter.Value>
  </Setter>
</Style>
<!--Initialize the SfDiagram-->
```

```
<syncfusion:SfDiagram x:Name="diagram" DefaultConnectorType="Line">
  <syncfusion:SfDiagram.Connectors>
    <!--Initialize the Connector Collection-->
    <syncfusion:ConnectorCollection>
      <!--create the simple connector with source point and target point values-->
      <syncfusion:ConnectorViewModel x:Name="simpleConnector"
        SourcePoint="100,100"
        TargetPoint="200,200" />
    </syncfusion:ConnectorCollection>
  </syncfusion:SfDiagram.Connectors>
</syncfusion:SfDiagram>
```

**C#**

```
//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Define the Connector Type as line
diagram.DefaultConnectorType = ConnectorType.Line;
//creating simple connector through collection using source and target
points.
ConnectorViewModel simpleConnector = new ConnectorViewModel()
{
  SourcePoint = new Point(100,100),
  TargetPoint = new Point(200,200),
};
//Adding the connector into Collection
(diagram.Connectors as ConnectorCollection).Add(simpleConnector);
```

**How to draw polyline**

Polyline is a continuous line of a segment or a continuous line composed of more line segments. When you click the diagram page, a line will be drawn, and then new segments will be kept on added for every click on page. Line drawing will be stopped when double-click the page. This polyline will be drawn using the [Tool](#), [DrawingTool](#), and [DefaultConnectorType](#) properties.

**XML**

```
<!--Initialize the Sfdiagram-->
<syncfusion:SfDiagram x:Name="diagram" DefaultConnectorType="PolyLine"
  Tool="ContinuesDraw" DrawingTool="Connector" />
```

**C#**

```
//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Define the Connector type as poly line
```

```
diagram.DefaultConnectorType = ConnectorType.PolyLine;  
//Define tool as continious draw and drawing tool as connector  
diagram.Tool = Tool.ContinuesDraw;  
diagram.DrawingTool = DrawingTool.Connector;
```



#### Free-hand drawing

Diagram has support for free-hand drawing to draw anything on the diagram page independently. Free-hand drawing will be enabled by using the **DrawingTool** property and setting its value to **FreeHand**.

#### XML

```
<!--Initialize the Sfdiagram-->  
<syncfusion:SfDiagram x:Name="diagram" Tool="ContinuesDraw"  
DrawingTool="FreeHand" />
```

#### C#

```
//Initialize the SfDiagram  
SfDiagram diagram = new SfDiagram();  
diagram.Tool = Tool.ContinuesDraw;  
//Define drawing tool as free hand  
diagram.DrawingTool = DrawingTool.FreeHand;
```



For more information about changing drawing tool of diagram, refer to [Tools](#).

Find the [Drawing tools sample](#) to depict the Tools.

### Create connector

Connector can be created by defining the start and end points. The path to be drawn can be defined with a collection of [Segments](#).

### Create connectors through connection points

The connector can be created by defining the source and target point of the connection. The [SourcePoint](#) and [TargetPoint](#) properties, which are `Point` type allows you to set or get the start and end points of a connection.

### XML

```
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram" DefaultConnectorType="Line">
  <syncfusion:SfDiagram.Connectors>
    <!--Initialize the Connector Collection-->
    <syncfusion:ConnectorCollection>
      <!--create the simple connector with source point and target point values-->
      <syncfusion:ConnectorViewModel x:Name="simpleConnector"
        SourcePoint="100,100" TargetPoint="200,200" />
    </syncfusion:ConnectorCollection>
  </syncfusion:SfDiagram.Connectors>
</syncfusion:SfDiagram>
```

### C#

```
//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Define the Connector Type
diagram.DefaultConnectorType = ConnectorType.Line;
//creating simple connector through collection using source and target
points.
ConnectorViewModel simpleConnector = new ConnectorViewModel()
{
  SourcePoint = new Point(100,100),
  TargetPoint = new Point(200,200),
};
//Adding the connector into Collection
(diagram.Connectors as ConnectorCollection).Add(simpleConnector);
```



### Create connection between nodes

The connector can be created between nodes to display the relationship between them. The [SourceNode/SourceNodeID](#) and [TargetNode/TargetNodeID](#) properties allows you to represent the nodes to be connected.

**XML**

```

<!--Initialize the Sfdiagram-->
<syncfusion:SfDiagram x:Name="diagram">
<syncfusion:SfDiagram.Nodes>
<!--Initialize the Nodes Collection-->
<syncfusion:NodeCollection>
<!--Creating source node-->
<syncfusion:NodeViewModel ID="sourceNode"
UnitWidth="100" UnitHeight="50"
OffsetX="300" OffsetY="200"
Shape="{StaticResource Rectangle}"/>
<!--Creating target node-->
<syncfusion:NodeViewModel ID="targetNode"
UnitWidth="100" UnitHeight="50"
OffsetX="500" OffsetY="200"
Shape="{StaticResource Rectangle}"/>
</syncfusion:NodeCollection>
</syncfusion:SfDiagram.Nodes>
<syncfusion:SfDiagram.Connectors>
<!--Initialize the ConnectorCollection-->
<syncfusion:ConnectorCollection>
<!--create the connector with source node and target node values-->
<syncfusion:ConnectorViewModel x:Name="NodeToNodeConnection"
SourceNodeID="sourceNode"
TargetNodeID="targetNode"/>
</syncfusion:ConnectorCollection>
</syncfusion:SfDiagram.Connectors>
</syncfusion:SfDiagram>

```

**C#**

```

//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//creating source node
NodeViewModel sourcenode = new NodeViewModel()
{
    UnitWidth = 100,
    UnitHeight = 50,
    OffsetX = 300,
    OffsetY = 200,
    Shape = this.Resources["Rectangle"],
};
//creating target node
NodeViewModel targetenode = new NodeViewModel()
{
    UnitWidth = 100,
    UnitHeight = 50,
    OffsetX = 500,
    OffsetY = 200,
    Shape = this.Resources["Rectangle"],
};
//Adding nodes into Collection
(diagram.Nodes as NodeCollection).Add(sourcenode);
(diagram.Nodes as NodeCollection).Add(targetenode);
//create the connector with source node and target node values

```

```
ConnectorViewModel nodeToNodeConnection = new ConnectorViewModel()
{
    SourceNode = sourcenode,
    TargetNode = targetenode,
};
//Adding connector into Collection
(diagram.Connectors as ConnectorCollection).Add(nodeToNodeConnection);
```



### Connections with ports

By default, connections are created at the intersecting point of segments and node bounds. The connection between any specific point of source and target nodes can be achieved with ports.

The [SourcePort/SourcePortID](#) and [TargetPort/TargetPortID](#) properties allows you to create connections between some specific points of Source/Target nodes.

### XML

```
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <syncfusion:SfDiagram.Nodes>
    <!--Initialize the Nodes Collection-->
    <syncfusion:NodeCollection>
      <!--Creating source node-->
      <syncfusion:NodeViewModel ID="sourceNode"
        UnitWidth="100" UnitHeight="50"
        OffsetX="300" OffsetY="200"
        Shape="{StaticResource Rectangle}">
        <syncfusion:NodeViewModel.Ports>
          <!--Initialize the Ports Collection-->
          <syncfusion:PortCollection>
            <!--create source node port-->
            <syncfusion:NodePortViewModel ID="SourcePort"
              NodeOffsetX="0"
              NodeOffsetY="0.5"/>
          </syncfusion:PortCollection>
        </syncfusion:NodeViewModel.Ports>
      </syncfusion:NodeViewModel>
      <!--Creating target node-->
      <syncfusion:NodeViewModel ID="targetNode"
        UnitWidth="100" UnitHeight="50"
        OffsetX="500" OffsetY="200"
        Shape="{StaticResource Rectangle}">
        <syncfusion:NodeViewModel.Ports>
          <!--Initialize the Ports Collection-->
          <syncfusion:PortCollection>
            <!--create source node port-->
            <syncfusion:NodePortViewModel ID="TargetPort"
              NodeOffsetX="1"
              NodeOffsetY="0.5"/>
          </syncfusion:PortCollection>
        </syncfusion:NodeViewModel.Ports>
      </syncfusion:NodeViewModel>
    </syncfusion:NodeCollection>
  </syncfusion:SfDiagram.Nodes>
</syncfusion:SfDiagram>
```

```

</syncfusion:NodeViewModel>
</syncfusion:NodeCollection>
</syncfusion:SfDiagram.Nodes>
<syncfusion:SfDiagram.Connectors>
<!--Initialize the ConnectorCollection-->
<syncfusion:ConnectorCollection>
<!--create the connector with source node and target node values-->
<syncfusion:ConnectorViewModel x:Name="PortToPortConnection"
SourcePortID="SourcePort"
TargetPortID="TargetPort"
SourceNodeID="sourceNode"
TargetNodeID="targetNode"/>
</syncfusion:ConnectorCollection>
</syncfusion:SfDiagram.Connectors>
</syncfusion:SfDiagram>

```

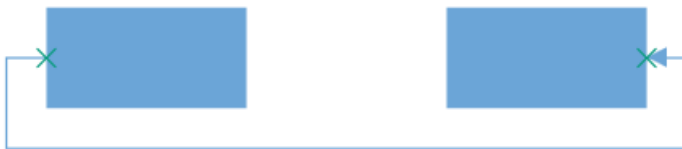
**C#**

```

//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Create source node port
NodePortViewModel sourcePort = new NodePortViewModel()
{
    NodeOffsetX = 0,
    NodeOffsetY = 0.5,
};
//Create target node port
NodePortViewModel targetPort = new NodePortViewModel()
{
    NodeOffsetX = 1,
    NodeOffsetY = 0.5,
};
//creating source node
NodeViewModel sourcenode = new NodeViewModel()
{
    UnitWidth = 100,
    UnitHeight = 50,
    OffsetX = 300,
    OffsetY = 200,
    Shape = this.Resources["Rectangle"],
    Ports = new PortCollection()
    {
        //Add source port to source node
        sourcePort,
    }
};
//creating target node
NodeViewModel targetenode = new NodeViewModel()
{
    UnitWidth = 100,
    UnitHeight = 50,
    OffsetX = 500,
    OffsetY = 200,
    Shape = this.Resources["Rectangle"],
    Ports = new PortCollection()
    {

```

```
//Add target port to target node
targetPort,
};
//Adding nodes into Collection
(diagram.Nodes as NodeCollection).Add(sourcenode);
(diagram.Nodes as NodeCollection).Add(targetenode);
//create the connector from port to port
ConnectorViewModel PortToPortConnection = new ConnectorViewModel()
{
    SourceNode = sourcenode,
    //Define the source port
    SourcePort = sourcePort,
    TargetNode = targetenode,
    //Define the target port
    TargetPort = targetPort,
};
//Adding port to port connector into connector Collection
(diagram.Connectors as ConnectorCollection).Add(PortToPortConnection);
```



For more details about Ports, refer to [Port](#).

Find the [Connector creation sample](#) to depict the connector creation.

#### Draw connectors

Connectors can be interactively drawn by clicking and dragging on the diagram surface by using the drawing tool. For more information about drawing connectors, refer to [Drawing Tools](#).

#### Connectors through data source

Connectors are automatically generated based on the relationships defined through the data source. For more information about data source, refer to [Data Source](#).

#### Add connectors from stencil

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

For more information about adding connectors from stencil, refer to [Stencil](#).

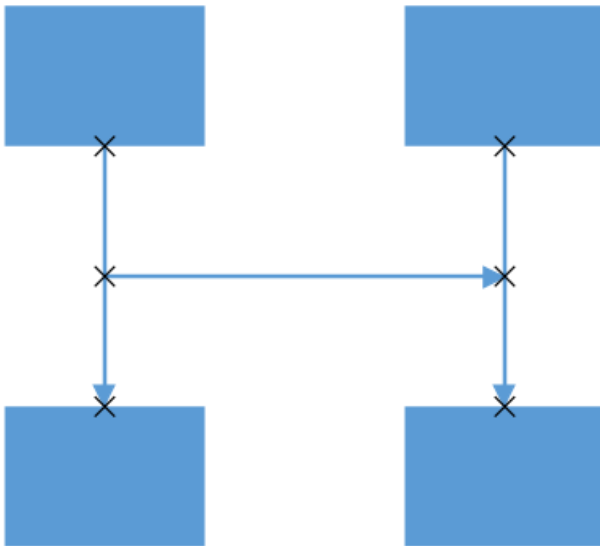
#### See Also

- [How to apply built-in theme for node and connector?](#)
- [How to decide whether to drag or draw a connection on port at runtime?](#)
- [How to customize the context menu?](#)
- [How to create port at runtime though SetTool?](#)



### Port in WPF Diagram (SfDiagram)

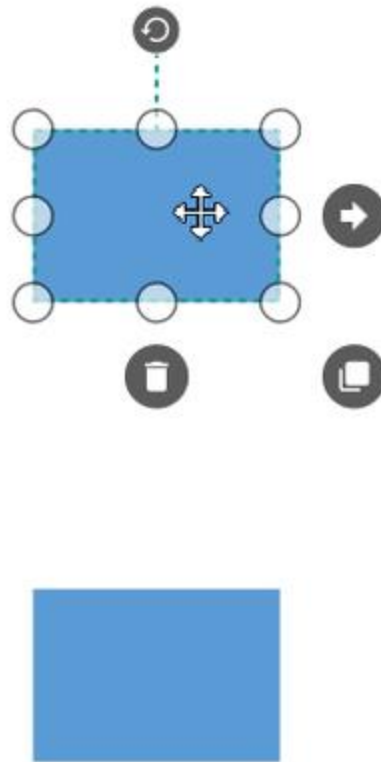
Port is a special connection point on a Node or Connector that you can glue the connectors. When you glue a connector to a node or port, they stay connected, even if one of the node is moved.



### 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).



A port connection is one where the connector is glued on the connection point of a node. When you move this node, the connector will always remain attached to the same connection point. You can create a port connection by connecting a specific port on one node to another port on another shape.



SfDiagram supports three types of ports: Node Port, Connector Port, and DockPort.

#### Node port

A port on a node can be created using the instance of `NodePort` object. The `NodeOffsetX` and `NodeOffsetY` properties of `NodePort` class is used to specify the position of the port on a node.

#### XML

```
<!--Style for Node-->
<Style TargetType="Syncfusion:Node">
  <Setter Property="ShapeStyle">
    <Setter.Value>
      <Style TargetType="Path">
        <Setter Property="Fill" Value="#FF5B9BD5"/>
        <Setter Property="Stretch" Value="Fill"/>
      </Style>
    </Setter.Value>
  </Setter>
</Style>
```

```

<Setter Property="Stroke" Value="#FFEDF1F6"/>
</Setter>
</Style>
<!--style for NodePort-->
<Style TargetType="Syncfusion:NodePort">
<Setter Property="Shape">
<Setter.Value>
<RectangleGeometry Rect="0,0,10,10"/>
</Setter.Value>
</Setter>
<Setter Property="ShapeStyle">
<Setter.Value>
<Style TargetType="Path">
<Setter Property="Stretch" Value="Fill"/>
<Setter Property="Fill" Value="#FF808081"/>
</Style>
</Setter.Value>
</Setter>
</Style>
<syncfusion:SfDiagram x:Name="diagram" PortVisibility="Visible">
<!--Initializes the NodeCollection-->
<syncfusion:SfDiagram.Nodes>
<syncfusion:NodeCollection>
<!--Initializes the Node-->
<syncfusion:NodeViewModel UnitHeight="100" UnitWidth="100"
OffsetX="100" OffsetY="100"
Shape="{StaticResource Rectangle}">
<!--Initialization of PortCollection-->
<syncfusion:NodeViewModel.Ports>
<syncfusion:PortCollection>
<!--Initialization of NodePort-->
<syncfusion:NodePortViewModel x:Name="port"/>
</syncfusion:PortCollection>
</syncfusion:NodeViewModel.Ports>
</syncfusion:NodeViewModel>
</syncfusion:NodeCollection>
</syncfusion:SfDiagram.Nodes>
</syncfusion:SfDiagram>

```

**C#**

```

//Define diagram
SfDiagram diagram = new SfDiagram();
//Initialize the visibility of the port as visible
diagram.PortVisibility = PortVisibility.Visible;
//Define Node's Collection
diagram.Nodes = new NodeCollection();
//Define Connector's Collection
diagram.Connectors = new ConnectorCollection();
//Create nodeviewmodel
NodeViewModel node = new NodeViewModel()
{
    UnitHeight = 100,
    UnitWidth = 100,

```

```

OffsetX = 300,
OffsetY = 300,
Shape = App.Current.Resources["Rectangle"],
};
//Define port to the PortCollection of the Node
(node.Ports as PortCollection).Add(new NodePortViewModel());
//Add node to the NodeCollection of the diagram
(diagram.Nodes as NodeCollection).Add(node);
//RootGrid is the instance of Mainwindow Grid
RootGrid.Children.Add(diagram);

```



N By default, port will be visible while dragging the connector thumb hover the diagram element where port presents within the diagram element. For more information , refer to the [PortVisibility](#)

#### NodeOffset

The **NodeOffsetX** and **NodeOffsetY** properties of port is used to position the port based on fractions. 0 represents Node's top/left corner, 1 represents Node's bottom/right corner, and 0.5 represents Node's center point. Default value is (0.5, 0.5).

| Offset values | Output |

|---|---|



| (0,0) |



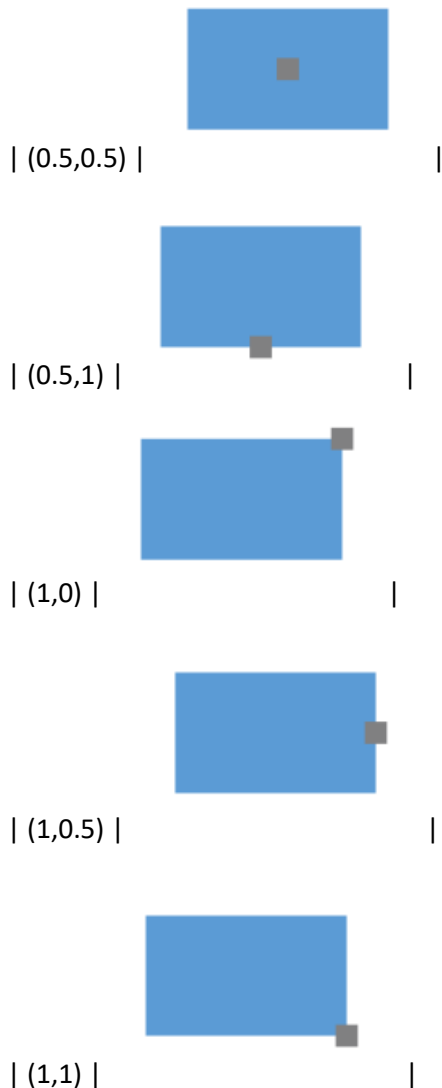
| (0,0.5) |



| (0,1) |



| (0.5,0) |



### Displacement

The **Displacement** property is used to dislocate the port by the value given. By default, port will be in the center of the node. When you assign value to the **Displacement** property, port will be displaced from its position by displacement value. Default value is 0d.

### XML

```
<!--Displace the port 5pixel away from left side-->
<Syncfusion:NodePortViewModel x:Name="port" Displacement="5,0,0,0"
UnitWidth="7" UnitHeight="7" NodeOffsetX="1" NodeOffsetY="0.5"/>
```

### C#

```
NodePortViewModel port = new NodePortViewModel()
{
    UnitHeight = 7,
    UnitWidth = 7,
    NodeOffsetX = 1,
```

```
NodeOffsetY = 0.5,
//Displace the port 5pixel away from left side
Displacement = new Thickness(5,0,0,0)
};
```



### Connector port

To specify and make connection with Connector at precise Length.

#### Define connector port

A port on a connector can be created using the instance of `ConnectorPort` object. The `Length` property of the `ConnectorPort` class is used to specify the position of the port on a connector path.

Find the common style for Connector and ConnectorPort.

### XML

```
<!--Style for the Connector-->
<Style TargetType="Syncfusion:Connector">
  <Setter Property="ConnectorGeometryStyle">
    <Setter.Value>
      <Style TargetType="Path">
        <Setter Property="Stroke" Value="#FF5B9BD5"></Setter>
        <Setter Property="StrokeThickness" Value="1"></Setter>
      </Style>
    </Setter.Value>
  </Setter>
  <Setter Property="TargetDecoratorStyle">
    <Setter.Value>
      <Style TargetType="Path">
        <Setter Property="Stroke" Value="#FF5B9BD5"></Setter>
        <Setter Property="StrokeThickness" Value="1"></Setter>
      </Style>
    </Setter.Value>
  </Setter>
</Style>
<!--Style For ConnectorPort-->
<Style TargetType="Syncfusion:ConnectorPort">
  <Setter Property="ShapeStyle">
    <Setter.Value>
      <Style TargetType="Path">
        <Setter Property="Fill" Value="#FF808081"/>
      </Style>
    </Setter.Value>
  </Setter>
  <Setter Property="Shape">
    <Setter.Value>
      <RectangleGeometry Rect="0,0,10,10"/>
    </Setter.Value>
  </Setter>
</Style>
```

```

</Setter.Value>
</Setter>
</Style>
<!--Initialize the sfdiagram-->
<syncfusion:SfDiagram x:Name="diagram" PortVisibility="Visible">
<!--Initialize the ConnectorCollection-->
<syncfusion:SfDiagram.Connectors>
<syncfusion:ConnectorCollection>
<!--Initialize the Connector-->
<syncfusion:ConnectorViewModel SourcePoint="100,100" TargetPoint="200,200">
<syncfusion:ConnectorViewModel.Ports>
<!--Initializes the PortCollection-->
<syncfusion:PortCollection>
<!--Initializes the ConnectorPort-->
<syncfusion:ConnectorPortViewModel x:Name="Port" Length="0.5"/>
</syncfusion:PortCollection>
</syncfusion:ConnectorViewModel.Ports>
</syncfusion:ConnectorViewModel>
</syncfusion:ConnectorCollection>
</syncfusion:SfDiagram.Connectors>
</syncfusion:SfDiagram>

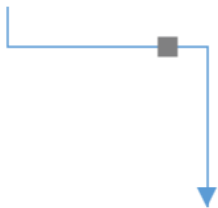
```

## C#

```

//Define diagram
SfDiagram diagram = new SfDiagram();
//Initialize the visibility of the port as visible
diagram.PortVisibility = PortVisibility.Visible;
//Define Connector Property
diagram.Connectors = new ConnectorCollection();
ConnectorViewModel connector = new ConnectorViewModel()
{
    SourcePoint = new Point(100, 100),
    TargetPoint = new Point(200, 200),
};
//Define port to the PortCollection of the connector
(connector.Ports as PortCollection).Add(new ConnectorPortViewModel());
//Adding Connector to Collection
(diagram.Connectors as ConnectorCollection).Add(connector);

```

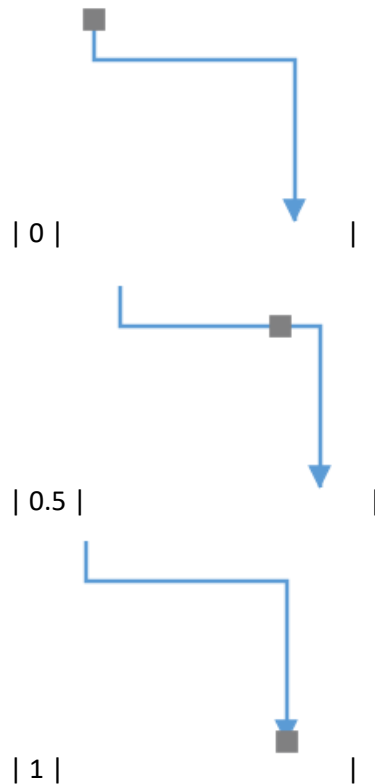


### Length

The **Length** property of port is used to align the port based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height. Default value is 0.5.

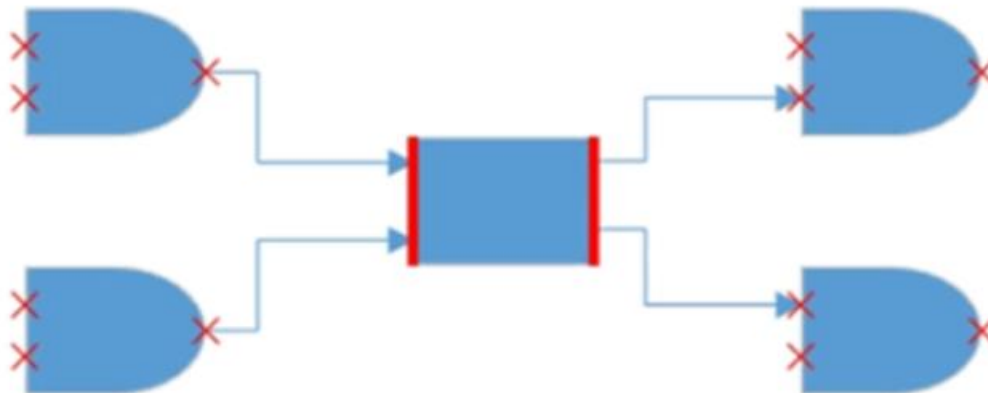
Length value	Output
---	---





### Dock port

DockPort is different from standard ports like `NodePort` and `ConnectorPort`. It is a placeholder that allows us to create connections at any point within it. Using `DockPort`, you can achieve lines of connecting points within the node boundary as shown in the following image.



A `DockPort` on a node can be created using the instance of `DockPort` object. The `SourcePoint` and `TargetPoint` properties of `DockPort` allows you to define its start and end points.

### XML

```
<!--Style for DockPort-->
<Style TargetType="Syncfusion:DockPort">
  <Setter Property="ConnectorGeometryStyle">
    <Setter.Value>
```

```

<Style TargetType="Path">
  <Setter Property="Stroke" Value="Black"></Setter>
  <Setter Property="StrokeThickness" Value="5"></Setter>
</Style>
</Setter.Value>
</Setter>
</Style>
<!--Initializes the Node-->
<syncfusion:NodeViewModel x:Name="node" OffsetX="100" OffsetY="100"
UnitHeight="100" UnitWidth="100"
Shape="{StaticResource Rectangle}">
  <!--Initializes the PortCollection-->
  <syncfusion:NodeViewModel.Ports>
    <syncfusion:PortCollection>
      <!--Initializes the DockPort-->
      <syncfusion:DockPortViewModel x:Name="port"
SourcePoint="0,1"
TargetPoint="1,1"/>
    </syncfusion:PortCollection>
  </syncfusion:NodeViewModel.Ports>
</syncfusion:NodeViewModel>

```

**C#**

```

//Create nodeviewmodel
NodeViewModel node = new NodeViewModel()
{
    UnitHeight = 100,
    UnitWidth = 100,
    OffsetX = 100,
    OffsetY = 100,
    Shape = App.Current.Resources["Rectangle"]
};
//Initialize dockportviewmodel to the nodeviewmodel
(node.Ports as PortCollection).Add
(
    new DockPortViewModel()
    {
        SourcePoint = new Point(0, 1),
        TargetPoint = new Point(1, 1)
    }
);

```



[View Sample in GitHub](#)

### Geometry Style

The appearance of **DockPort** such as stroke and stroke thickness can be customized using the **ConnectorGeometryStyle** property.

#### XML

```
<!--Style for DockPort-->
<Style TargetType="Syncfusion:DockPort">
  <Setter Property="ConnectorGeometryStyle">
    <Setter.Value>
      <Style TargetType="Path">
        <Setter Property="Stroke" Value="DarkOrange"></Setter>
        <Setter Property="StrokeThickness" Value="5"></Setter>
      </Style>
    </Setter.Value>
  </Setter>
</Style>
```



To visualize the DockPort, it must specify the **SourcePoint**, **TargetPoint**, and **ConnectorGeometryStyle** properties.

### Padding

Padding is used to leave space between the connector's end point and the object to where it is connected. The **ConnectorPadding** property of port defines the space to be left between the port bounds and its edges. Default value is 0d.

#### XML

```
<!--Declaring the ConnectorPadding value-->
<syncfusion:NodePortViewModel NodeOffsetX="0" NodeOffsetY="0.5"
  ConnectorPadding="10"/>
```

#### C#

```
NodePortViewModel nodePort = new NodePortViewModel()
{
  //Declaring the ConnectorPadding value
  ConnectorPadding = 10,
  NodeOffsetX = 0,
  NodeOffsetY = 0.5,
};
```

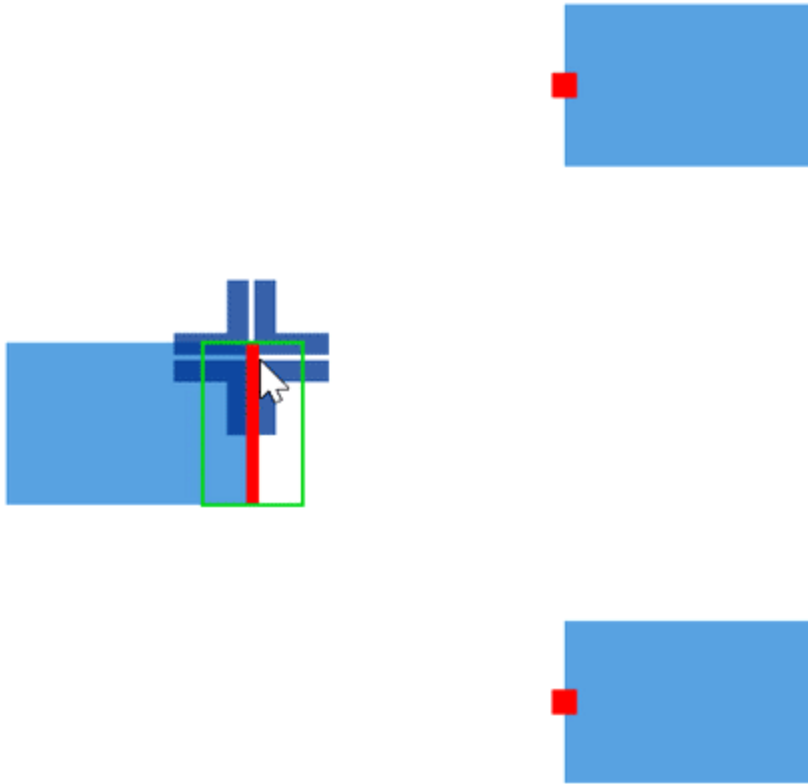


### HitPadding

Connection can be made from or to nodes, connectors, port or on an empty area in a diagram. Making a connection with ports is usually difficult as thickness is usually small. To make it easy to connect, it should be possible to connect when the mouse comes near its vicinity area. The **HitPadding** property allows us to customize the vicinity area when connecting. The connector can be created by clicking and dragging to any point of hit padding of ports and can be dropped at any point of hit padding region of ports. Default value is 0d.

### C#

```
DockPortViewModel dockPort = new DockPortViewModel()
{
    Constraints=PortConstraints.Default & ~PortConstraints.InheritHitPadding,
    //Declaring the value for HitPadding
    HitPadding = 40,
    SourcePoint = new Point(1, 0),
    TargetPoint = new Point(1, 1),
};
NodePortViewModel nodePort = new NodePortViewModel()
{
    Constraints = PortConstraints.Default & ~PortConstraints.InheritHitPadding,
    //Declaring the value for HitPadding
    HitPadding = 40,
    NodeOffsetX = 0,
    NodeOffsetY = 0.5,
};
```



### PortVisibility

The visibility of ports depends on the properties of `MouseOver`, `Default`, `Collapse`, `MouseOverOnConnect`, `ValidConnection`, and `Visible`.

| Property | Definition |

|---|---|

| `MouseOver` | Port is visible when mouseover the `DiagramElement`. |

| `Default` | Port is visible while dragging the connector thumb hover the `DiagramElement` where port presents within the `DiagramElement`. |

| `Collapse` | Port is not visible for the `DiagramElement`. |

| `MouseOverOnConnect` | Port is visible while dragging the connector thumb hover the `DiagramElement` where port presents within the `DiagramElement`. |

| `ValidConnection` | 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`. |

To learn more about `PortVisibility`, refer to [PortVisibility](#).

### Connection direction

The `ConnectionDirection` is a port's property, which allows users to specify the direction in which the connector's connection to be established to a port. This property will be active only if the port constraints contains `PortConstraints.ConnectionDirection`. Default value is `Auto`.

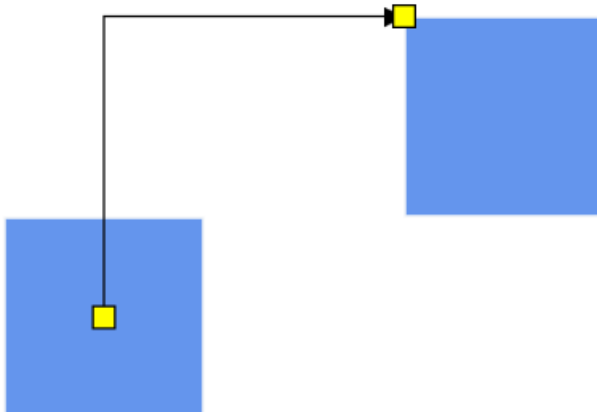
### XML

```
<syncfusion:NodePortViewModel NodeOffsetX="0.5" NodeOffsetY="0.5"
Constraints="Connectable,ConnectionDirection" ConnectionDirection="Right"/>
```

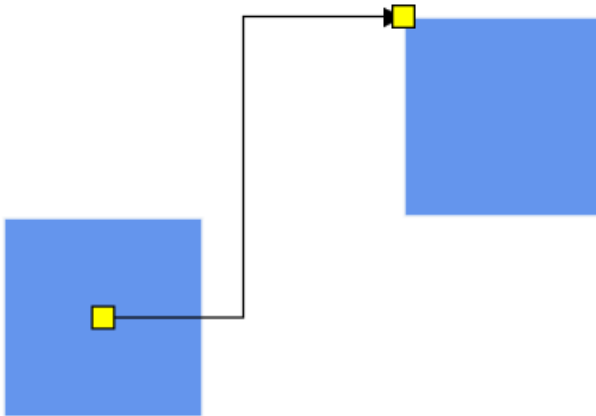
### C#

```
NodePortViewModel port = new NodePortViewModel()
{
    NodeOffsetX = 0.5,
    NodeOffsetY = 0.5
};
port.Constraints = PortConstraints.Connectable |
PortConstraints.ConnectionDirection;
//Specifying the direction in which connector need to connected to the port
port.ConnectionDirection = ConnectionDirection.Right;
```

### Before ConnectionDirection



After ConnectionDirection



For more information , refer to [ConnectionDirection](#)

Appearance

- The shape of the port can be changed by using its `shape` property. The shape can be any [Built-In Shapes](#) or any custom geometric path.
- The appearance of ports can be customized by using the `ShapeStyle` property of the port.
- Customize the port size by using the `UnitWidth` and `UnitHeight` properties of port.
- The `PortVisibility` property allows you to define, when the port should be visible.

---

N For DockPort customization, refer to [Geometry Style](#)

---

### XML

```
<!--style for NodePort-->
<Style TargetType="Syncfusion:NodePort" >
  <Setter Property="Shape">
    <Setter.Value>
      <EllipseGeometry RadiusX="5" RadiusY="5"/>
    </Setter.Value>
  </Setter>
  <Setter Property="ShapeStyle">
    <Setter.Value>
      <Style TargetType="Path">
        <Setter Property="Fill" Value="Yellow"/>
      </Style>
    </Setter.Value>
  </Setter>
</Style>
<!--Initializes the NodePort-->
<syncfusion:NodePortViewModel x:Name="port" UnitWidth="7" UnitHeight="7"
  NodeOffsetX="1" NodeOffsetY="0.5"/>
```

### C#

```
NodePortViewModel port = new NodePortViewModel()
{
```

```
UnitHeight = 7,  
UnitWidth = 7,  
NodeOffsetX=1,  
NodeOffsetY=0.5  
};  
(node.Ports as PortCollection).Add(port);
```



[View Sample in GitHub](#)

### Events

The `PortChangedEvent` will get invoked when you drag the port.

- `NodePort`: The `PortChangedEvent` will notify the `OffsetX` and `OffsetY` changes with their `OldValue` and `NewValue`.
- `ConnectorPort`: The `PortChangedEvent` will notify the `Length` changes with their `OldValue` and `NewValue`.
- `DockPort`: The `PortChangedEvent` will notify the `SourcePoint` and `TargetPoint` changes with their `OldValue` and `NewValue`.

To explore about arguments, refer to [PortChangedEvent](#)

### Constraints

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

See Also

- [How to control the visibility of Ports?](#)
- [How to connect only with port not with node?](#)
- [How to validate the connection and port visibility \(inport and outport\) in the diagram?](#)
- [How to customize the connection indicator style of node and port?](#)
- [How to add multiple ports for the node?](#)
- [How to decide whether to drag or draw a connection on port at runtime?](#)

### Annotation in WPF Diagram (SfDiagram)

Annotation is a block of text that can be displayed over a node or connector. Annotation is used to textually represent an object with a string that can be edited at run time. Multiple annotations can be added to a node or connector.

#### Define annotation

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 `AnnotationEditorViewModel` class defines the text to be displayed. The following code explains how to create an annotation.



**XML**

```

<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Initialize the Node-->
  <syncfusion:SfDiagram.Nodes>
    <!--Initialize the Node Collection-->
    <syncfusion:NodeCollection>
      <!--Initialize the node view model-->
      <syncfusion:NodeViewModel UnitWidth="100" UnitHeight="100"
        OffsetX="100" OffsetY="100"
        Shape="{StaticResource Rectangle}" >
        <!--Initialize the annotations-->
        <syncfusion:NodeViewModel.Annotations>
          <!--Initialize the AnnotationCollection-->
          <syncfusion:AnnotationCollection>
            <!--Initialize the Annotation editor view model-->
            <syncfusion:AnnotationEditorViewModel Content="Annotation"/>
          </syncfusion:AnnotationCollection>
        </syncfusion:NodeViewModel.Annotations>
      </syncfusion:NodeViewModel>
    </syncfusion:NodeCollection>
  </syncfusion:SfDiagram.Nodes>
  <!--Initialize the Connector-->
  <syncfusion:SfDiagram.Connectors>
    <!--Initialize the Connector Collection-->
    <syncfusion:ConnectorCollection>
      <!--Initialize the Connector view model-->
      <syncfusion:ConnectorViewModel SourcePoint="200,50" TargetPoint="300,150">
        <syncfusion:ConnectorViewModel.Annotations>
          <!--Initialize the AnnotationCollection-->
          <syncfusion:AnnotationCollection>
            <!--Initialize the Annotation editor view model-->
            <syncfusion:AnnotationEditorViewModel Content="Annotation"/>
          </syncfusion:AnnotationCollection>
        </syncfusion:ConnectorViewModel.Annotations>
      </syncfusion:ConnectorViewModel>
    </syncfusion:ConnectorCollection>
  </syncfusion:SfDiagram.Connectors>
</syncfusion:SfDiagram>

```

**C#**

```

//Initialize the diagram
SfDiagram diagram = new SfDiagram();
//Initialize the Node View Model
NodeViewModel node = new NodeViewModel()
{
  UnitWidth = 100,
  UnitHeight = 100,
  OffsetX = 100,
  OffsetY = 100,
  Shape = new RectangleGeometry() { Rect = new Rect(0, 0, 10, 10) },
  //Initialize the AnnotationCollection
  Annotations = new ObservableCollection<IAnnotation>()
{

```

```

//Initialize the Annotation with content
new AnnotationEditorViewModel ()
{
    Content="Annotation"
}
};
//Initialize the Connector View Model
ConnectorViewModel connector = new ConnectorViewModel ()
{
    SourcePoint = new Point (200, 50),
    TargetPoint = new Point (300, 150),
    //Initialize the AnnotationCollection
    Annotations = new ObservableCollection<IAnnotation>()
    {
        //Initialize the Annotation with content
        new AnnotationEditorViewModel ()
        {
            Content="Annotation"
        }
    }
};
// Add the node into Node's collection
(diagram.Nodes as NodeCollection).Add(node);
// Add the Connector into connector's collection
(diagram.Connectors as ConnectorCollection).Add(connector);

```

{% endtabs %}

![Create Annotation](Annotationimages/CreateAnnotation.jpg)

### Multiple Annotations

You can add any number of annotations to a node or connector.

### XML

```

<!--Initialize the Annotation Collection-->
<syncfusion:AnnotationCollection>
<!--Initialize the multiple annotation-->
<syncfusion:AnnotationEditorViewModel Content="Annotation"/>
<syncfusion:AnnotationEditorViewModel Content="Annotation"/>
<syncfusion:AnnotationEditorViewModel Content="Annotation"/>
</syncfusion:AnnotationCollection>

```

### C#

```

//Initialize the Annotation Collection
Annotations = new ObservableCollection<IAnnotation>()
{
    //Initialize the multiple annotation
    new AnnotationEditorViewModel ()
    {
        Content = "Annotation",
    },
    new AnnotationEditorViewModel ()
    {

```

```
Content = "Annotation",
},
new AnnotationEditorViewModel()
{
Content = "Annotation",
},
}
```

![Multiple Annotations](Annotationimages/annotationimg21.png)  
&#8194;&#8194;&#8194;&#8194;&#8194;&#8194; ![Multiple  
Annotations](Annotation\_images/MultipleAnnotationConnector.png)

[View sample in GitHub.](#)

## How to add annotations to nodes/connectors and its customization?

## Group in WPF Diagram (SfDiagram)

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

The following code illustrates how to create a Group Node.

## C#

```
ObservableCollection<NodeViewModel> nodes = new
ObservableCollection<NodeViewModel>>();
NodeViewModel node = new NodeViewModel()
{
    UnitWidth = 100,
    UnitHeight = 100,
    OffsetX = 100,
    OffsetY = 100,
    Shape = new RectangleGeometry() { Rect = new Rect(0, 0, 10, 10) },
    ShapeStyle = App.Current.Resources["shapestyle"] as Style
};
NodeViewModel node1 = new NodeViewModel()
{
    UnitWidth = 100,
    UnitHeight = 100,
    OffsetX = 200,
    OffsetY = 200,
    Shape = new RectangleGeometry() { Rect = new Rect(0, 0, 10, 10) },
    ShapeStyle = App.Current.Resources["shapestyle"] as Style
};
ObservableCollection<GroupViewModel> groups = new
ObservableCollection<GroupViewModel>>();
GroupViewModel group = new GroupViewModel()
{
    Nodes = new ObservableCollection<NodeViewModel>()
```

```
{  
    node,  
    node1  
},  
};  
groups.Add(group);  
diagram.Groups = groups;
```

#### *Group from Stencil*

Group Nodes can be predefined and added to stencil. You can drop those Groups into Diagram, when required.

To explore how to add Groups from stencil, refer to [Stencil](#).

#### *Interaction*

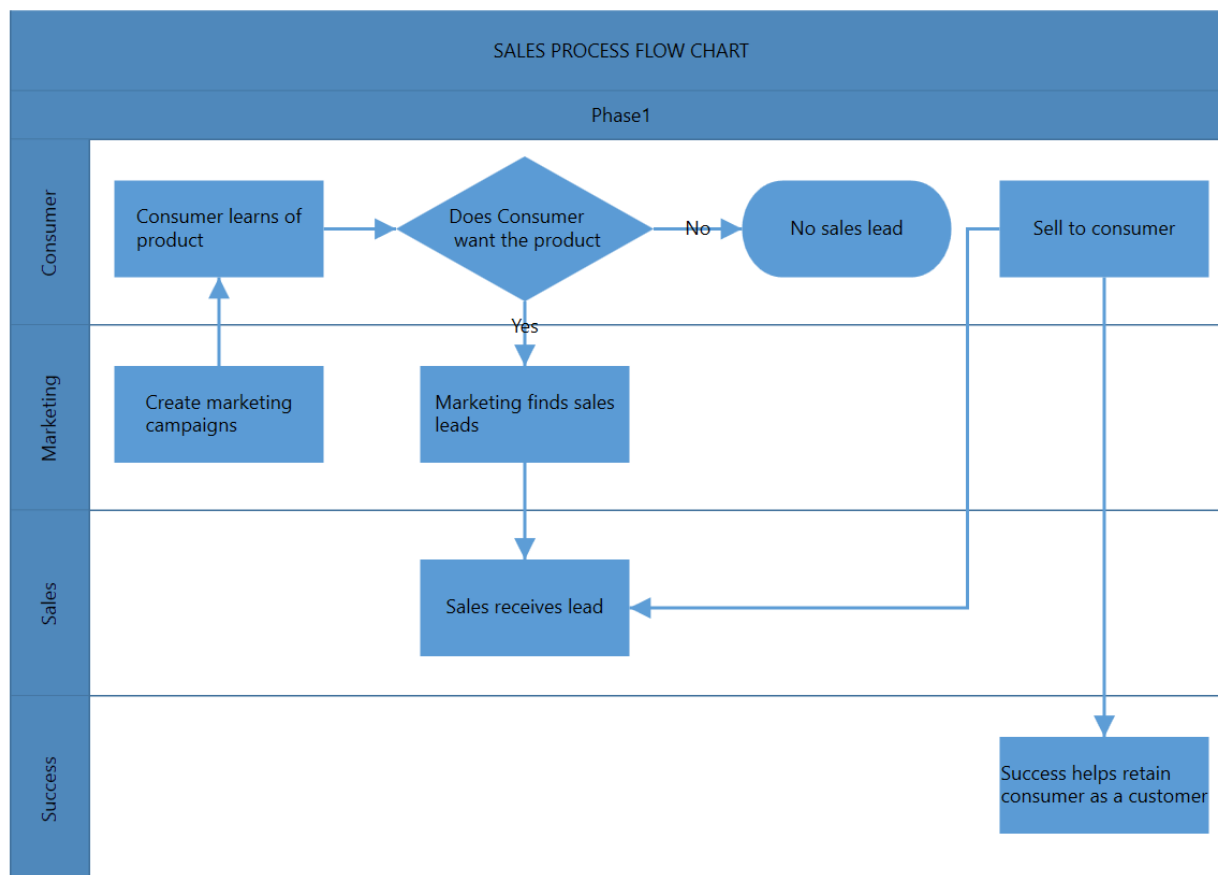
You can edit the Group and its children at runtime. For more information about how to interact with a Group, refer to [Interaction](#).

#### *See Also*

- [How to restrict the child node dragging whereas allow group dragging?](#)

#### *Swimlane in WPF Diagram (SfDiagram)*

A [Swimlane](#) is a type of diagram nodes, which is typically used to visualize the relationship between a business process and the department responsible for it by focusing on the logical relationships between activities.



### Create a swimlane

A swimlane can be created and added to the diagram, either programmatically or interactively.

#### Add Swimlane through the Swimlanes collection

To create a swimlane, you have to define the swimlane object and add that to the [Swimlanes](#) collection of the diagram.

**Note:** By default, if you create a swimlane, one lane and phase will be added.

### XML

```

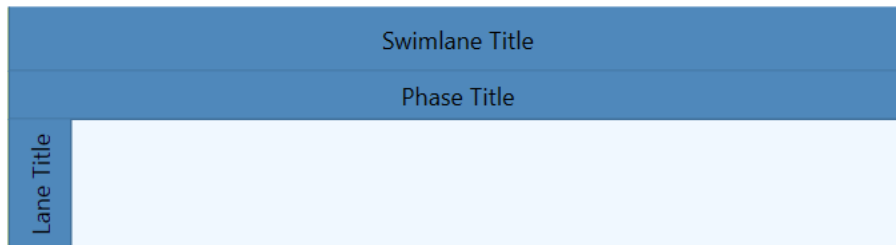
<!-- Initialize the SfDiagram -->
<syncfusion:SfDiagram x:Name="diagram">
  <syncfusion:SfDiagram.Swimlanes>
    <!-- Initialize the SwimlaneCollection -->
    <syncfusion:SwimlaneCollection>
      <!--Initialize the Swimlane-->
      <syncfusion:SwimlaneViewModel OffsetX="300" OffsetY="150"
UnitHeight="120" UnitWidth="450"/>
    </syncfusion:SwimlaneCollection>
  </syncfusion:SfDiagram.Swimlanes>
</syncfusion:SfDiagram>

```

### C#

```
//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize the SwimlaneCollection to SfDiagram
diagram.Swimlanes = new SwimlaneCollection();
//Creating the SwimlaneViewModel
SwimlaneViewModel swimlane = new SwimlaneViewModel()
{
    UnitWidth = 450,
    UnitHeight = 120,
    OffsetX = 300,
    OffsetY = 150,
};
//Add Swimlane to the Swimlanes property of the Diagram
(diagram.Swimlanes as SwimlaneCollection).Add(swimlane);
```

Now, the swimlane will be as follows.



[View sample in GitHub](#)

#### Swimlane Header

The Swimlane Header was the primary element for swimlanes. The [Header](#) property of swimlane allows you to define its textual description and to customize its appearance.

**Note:** By using this header, the swimlane interaction will be performed, like selection, dragging, and more.

The following code example explains how to define the swimlane header.

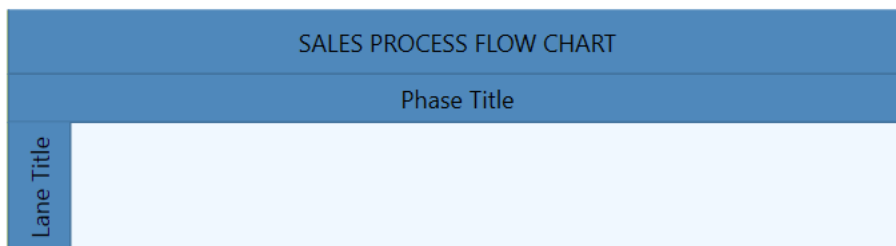
#### XML

```
<!-- Initialize the SfDiagram -->
<syncfusion:SfDiagram x:Name="diagram">
  <syncfusion:SfDiagram.Swimlanes>
    <!-- Initialize the SwimlaneCollection -->
    <syncfusion:SwimlaneCollection>
      <!--Initialize the Swimlane-->
      <syncfusion:SwimlaneViewModel OffsetX="300" OffsetY="150"
        UnitHeight="120" UnitWidth="450">
        <!--Initialize the Swimlane Header-->
        <syncfusion:SwimlaneViewModel.Header>
          <syncfusion:SwimlaneHeader UnitHeight="32" >
            <syncfusion:SwimlaneHeader.Annotation>
              <syncfusion:AnnotationEditorViewModel Content="SALES PROCESS FLOW
                CHART"/></syncfusion:AnnotationEditorViewModel>
            </syncfusion:SwimlaneHeader.Annotation>
          </syncfusion:SwimlaneHeader>
        </syncfusion:SwimlaneViewModel.Header>
      </syncfusion:SwimlaneViewModel>
    </syncfusion:SwimlaneCollection>
```

```
</syncfusion:SfDiagram.Swimlanes>
</syncfusion:SfDiagram>
```

**C#**

```
//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize the SwimlaneCollection to SfDiagram
diagram.Swimlanes = new SwimlaneCollection();
//Creating the SwimlaneViewModel
SwimlaneViewModel swimlane = new SwimlaneViewModel()
{
    UnitWidth = 450,
    UnitHeight = 120,
    OffsetX = 300,
    OffsetY = 150,
};
//Creating header for SwimlaneViewModel
swimlane.Header = new SwimlaneHeader()
{
    UnitHeight = 32,
    Annotation = new AnnotationEditorViewModel()
    {
        Content = "SALES PROCESS FLOW CHART"
    }
};
//Add Swimlane to the Swimlanes property of the diagram
(diagram.Swimlanes as SwimlaneCollection).Add(swimlane);
```



[View sample in GitHub](#)

*Customization of headers*

The height and width of swimlane header can be customized with the [UnitWidth](#) and [UnitHeight](#) properties of swimlane header. Set fill color of header by using the [ShapeStyle](#) property.

The following code example explains how to customize the swimlane header.

**XML**

```
<Style x:Key="SwimlaneHeaderStyle" TargetType="Path">
    <Setter Property="Fill" Value="CadetBlue"/>
    <Setter Property="Stretch" Value="Fill"/>
    <Setter Property="Stroke" Value="#41719C"/>
    <Setter Property="StrokeThickness" Value="1"/>
</Style>
<!--Template overriding for view template-->
<DataTemplate x:Key="viewTemplate">
```

```

<TextBlock Text="{Binding Path=Content, Mode=TwoWay}"
FontStyle="Italic" FontSize="12"
FontFamily="TimesNewRomen"
TextDecorations="Underline"
FontWeight="Bold"
Foreground="AliceBlue"/>
</DataTemplate>
<!-- Initialize the SfDiagram -->
<syncfusion:SfDiagram x:Name="diagram">
<syncfusion:SfDiagram.Swimlanes>
<!-- Initialize the SwimlaneCollection -->
<syncfusion:SwimlaneCollection>
<!--Initialize the Swimlane-->
<syncfusion:SwimlaneViewModel OffsetX="300" OffsetY="150"
Orientation="Horizontal"
UnitHeight="120" UnitWidth="450">
<syncfusion:SwimlaneViewModel.Header>
<!--Initialize the Swimlane Header-->
<syncfusion:SwimlaneHeader UnitHeight="32" ShapeStyle="{StaticResource
SwimlaneHeaderStyle}">
<syncfusion:SwimlaneHeader.Annotation>
<syncfusion:AnnotationEditorViewModel Content="SALES PROCESS FLOW CHART"
ViewTemplate="{StaticResource viewTemplate}">
</syncfusion:AnnotationEditorViewModel>
</syncfusion:SwimlaneHeader.Annotation>
</syncfusion:SwimlaneHeader>
</syncfusion:SwimlaneViewModel.Header>
</syncfusion:SwimlaneViewModel>
</syncfusion:SwimlaneCollection>
</syncfusion:SfDiagram.Swimlanes>
</syncfusion:SfDiagram>

```

**C#**

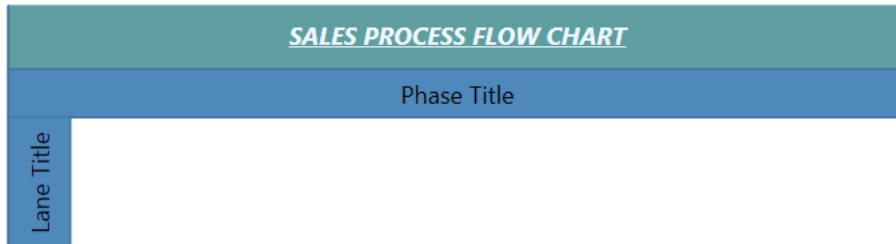
```

//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize the SwimlaneCollection to SfDiagram
diagram.Swimlanes = new SwimlaneCollection();
//Creating the SwimlaneViewModel
SwimlaneViewModel swimlane = new SwimlaneViewModel()
{
    UnitWidth = 450,
    UnitHeight = 120,
    OffsetX = 300,
    OffsetY = 150,
    Orientation=Orientation.Horizontal,
};
//Creating header for the SwimlaneViewModel
swimlane.Header = new SwimlaneHeader()
{
    UnitHeight = 32,
    Annotation = new AnnotationEditorViewModel()
    {
        Content = "SALES PROCESS FLOW CHART",
        ViewTemplate=this.Resources["viewTemplate"] as DataTemplate
    },

```



```
ShapeStyle=this.Resources["SwimlaneHeaderStyle"] as Style,
};
//Add Swimlane to the Swimlanes property of the diagram
(diagram.Swimlanes as SwimlaneCollection).Add(swimlane);
```

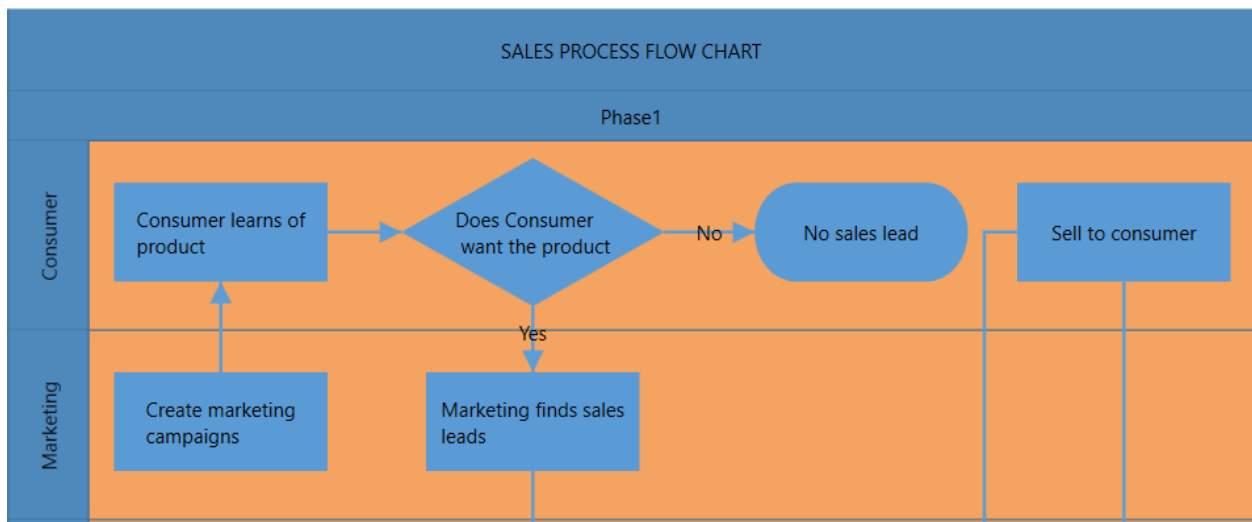


#### How to customize the Swimlane Background

You can customize the Swimlane background by changing its **ShapeStyle** property. The following code explains how to customize the Swimlane background.

#### XML

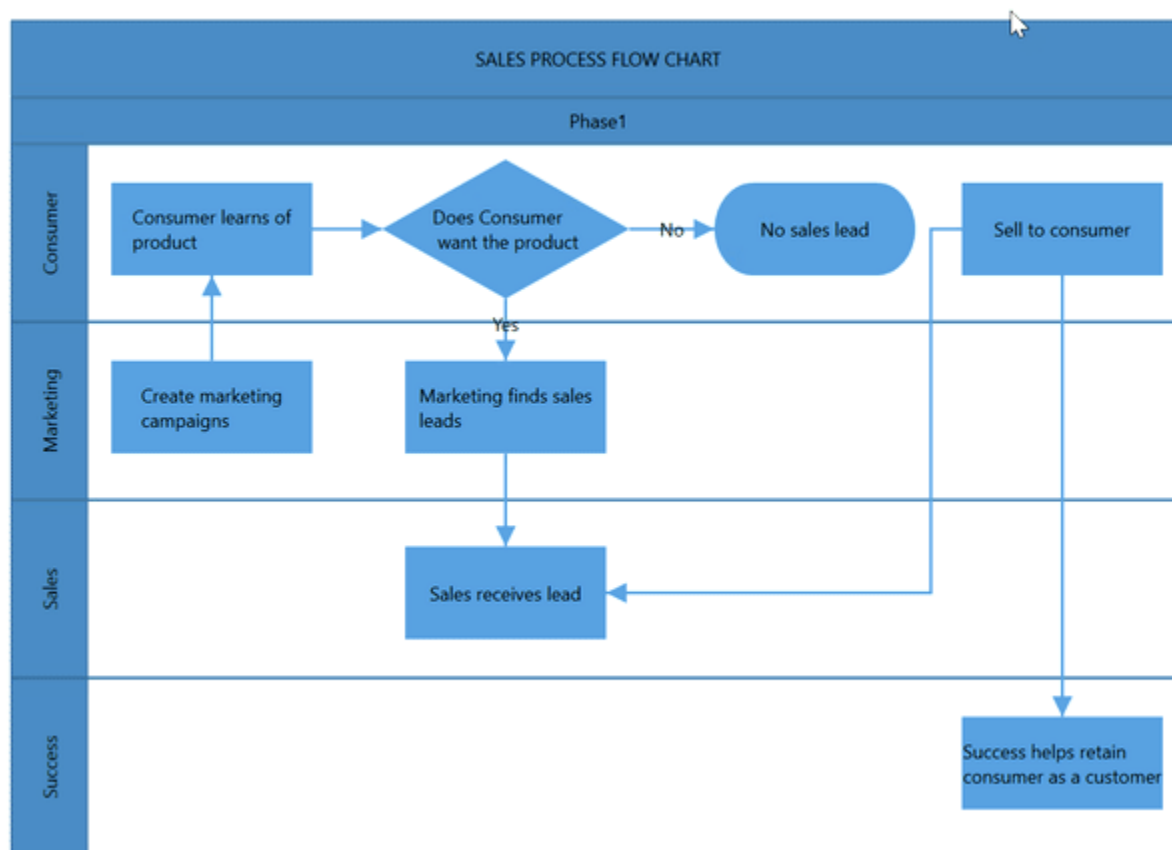
```
<Style TargetType="syncfusion:Swimlane">
  <Setter Property="ShapeStyle">
    <Setter.Value>
      <Style TargetType="Path">
        <Setter Property="Fill" Value="SandyBrown"/>
        <Setter Property="Stretch" Value="Fill"/>
        <Setter Property="Stroke" Value="#487ca9"/>
        <Setter Property="StrokeThickness" Value="1"/>
      </Style>
    </Setter.Value>
  </Setter>
</Style>
```



#### Header editing

The diagram provides the support to edit swimlane headers at runtime. You can achieve the header editing by double clicking on it. Double clicking the header label will enable the editing mode.

The following image shows how to edit the swimlane header.



### Orientation

The orientation of swimlane can be customized with the [Orientation](#) property of the header.

Note: By default the swimlane orientation has Horizontal.

The following code example explains how to set the orientation of the swimlane.

### XML

```

<!-- Initialize the SfDiagram -->
<syncfusion:SfDiagram x:Name="diagram">
  <syncfusion:SfDiagram.Swimlanes>
    <!-- Initialize the SwimlaneCollection -->
    <syncfusion:SwimlaneCollection>
      <!--Initialize the Swimlane-->
      <syncfusion:SwimlaneViewModel OffsetX="300" OffsetY="150"
        Orientation="Horizontal"
        UnitHeight="120" UnitWidth="450"/>
    </syncfusion:SwimlaneCollection>
  </syncfusion:SfDiagram.Swimlanes>
</syncfusion:SfDiagram>

```

### C#

```

//Initialize the SfDiagram

```

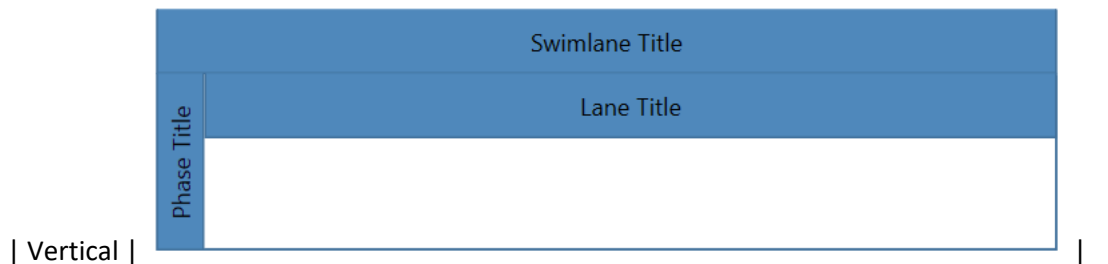
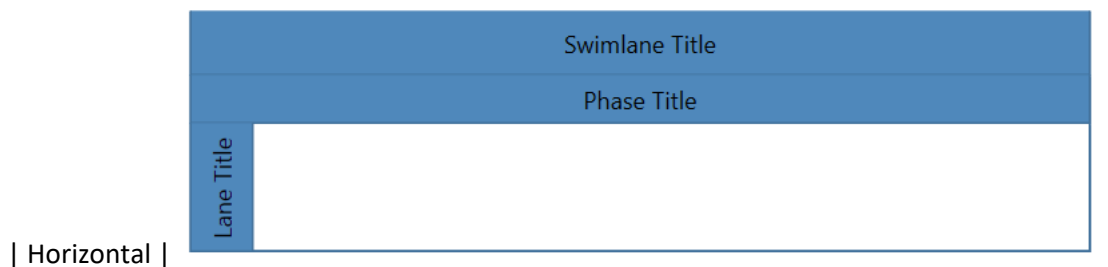
```

SfDiagram diagram = new SfDiagram();
//Initialize the SwimlaneCollection to SfDiagram
diagram.Swimlanes = new SwimlaneCollection();
//Creating the SwimlaneViewModel
SwimlaneViewModel swimlane = new SwimlaneViewModel()
{
    UnitWidth = 450,
    UnitHeight = 120,
    OffsetX = 300,
    OffsetY = 150,
    Orientation=Orientation.Horizontal,
};
//Add Swimlane to the Swimlanes property of the Diagram
(diagram.Swimlanes as SwimlaneCollection).Add(swimlane);

```

| Orientation | Output |

---|---



## Interaction

### Select

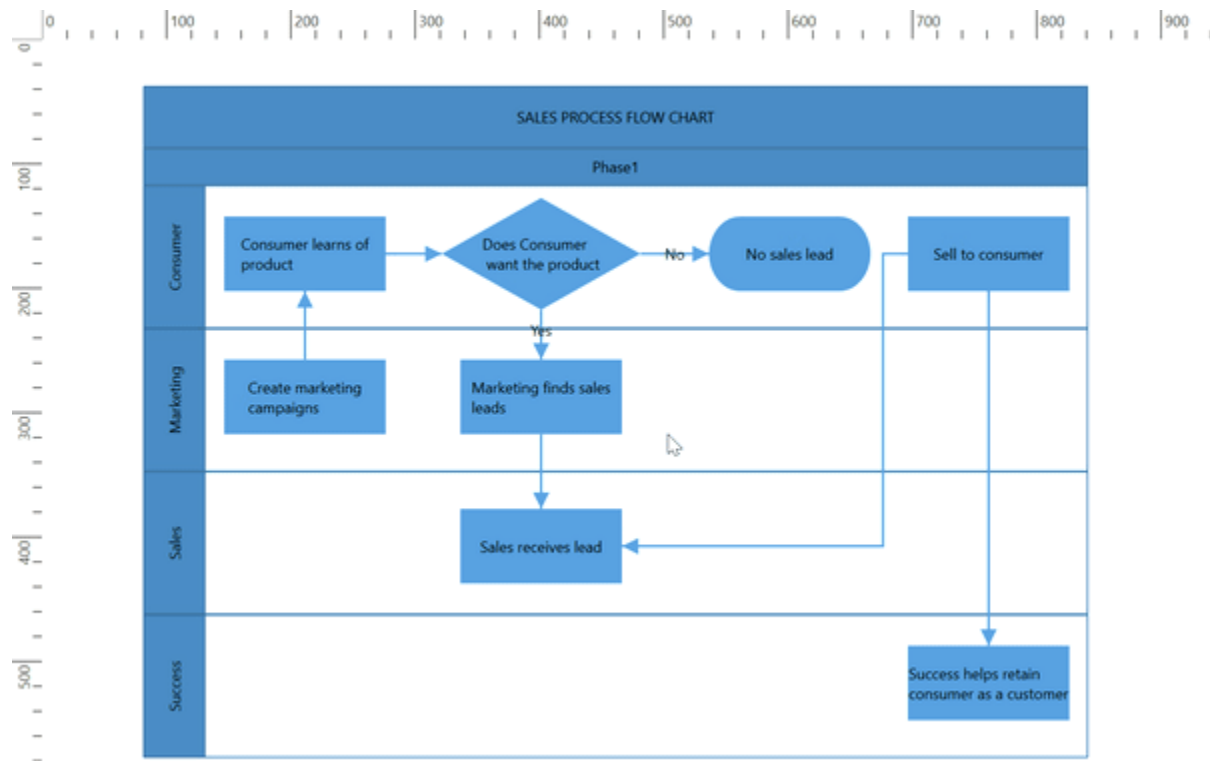
Swimlane can be selected by clicking (tap) the header of the swimlane.

- The `IsSelected` Property is used to select or unselect the swimlane at runtime.
- The `ItemSelectingEvent` and `ItemSelectedEvent` for selecting an element, will notify you the item and its original source. To explore about arguments, refer to the [DiagramPreviewEventArgs](#) and [ItemSelectedEventArgs](#).
- The `ItemUnselectingEvent` and `ItemUnselectedEvent` for unselecting an element, will notify you the item and its original source. To explore about arguments, refer to the [DiagramPreviewEventArgs](#) and [DiagramEventArgs](#).

### Drag

- Selected object can be dragged by clicking and dragging the header of the swimlane.

- Instead of dragging original object, preview of the node alone can be dragged. For preview dragging, refer to the [PreviewSettings](#).
- The `NodeChangedEvent` will notify the `OffsetX` and `OffsetY` changes with their old and new values. Along with that, this event will give information about interaction state. To explore about arguments, refer to the [NodeChangedEventArgs](#).



Please find the swimlane sample as follows.

[View Swimlane sample in GitHub](#)

### Gridlines in WPF Diagram (SfDiagram)

**Gridlines** are crisscross lines drawn in diagram page like the lines on traditional graph paper. It helps to position the diagram elements on the diagram page.

The `SnapConstraints` property of `SnapSettings` class allows you to control the visibility of the gridlines.

#### XML

```

<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
<!--Initialize SnapSettings constraints to show Gridlines-->
<syncfusion:SfDiagram.SnapSettings>
<syncfusion:SnapSettings SnapConstraints="ShowLines"/>
</syncfusion:SfDiagram.SnapSettings>
</syncfusion:SfDiagram>

```

#### C#

```

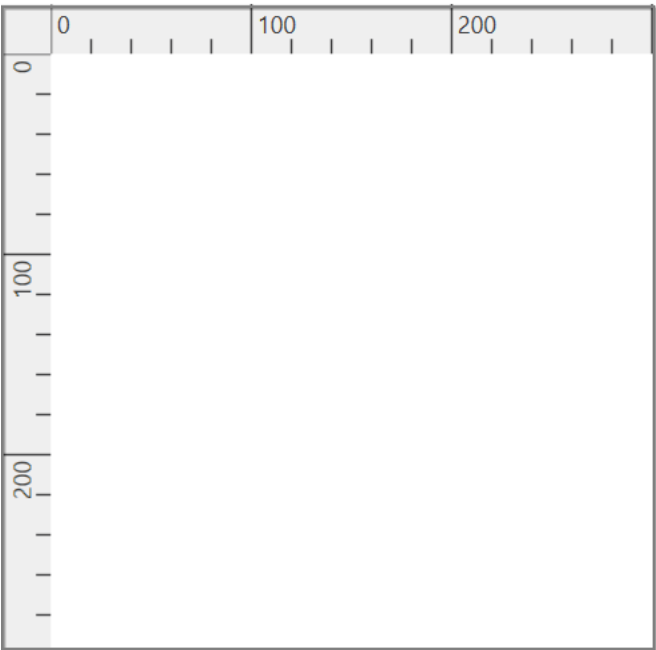
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();

```

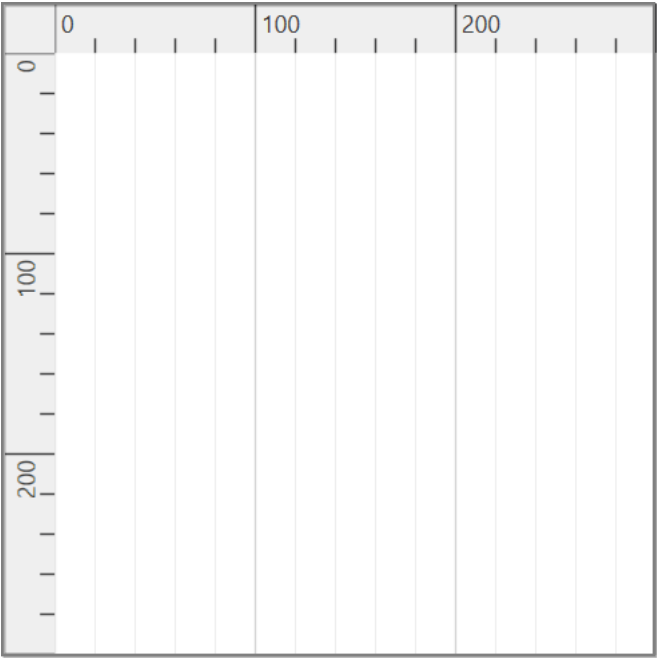
```
//Initialize SnapSettings constraints to show Gridlines
diagram.SnapSettings = new SnapSettings()
{
    SnapConstraints = SnapConstraints.ShowLines,
};
```

| Enum | Value | Output |

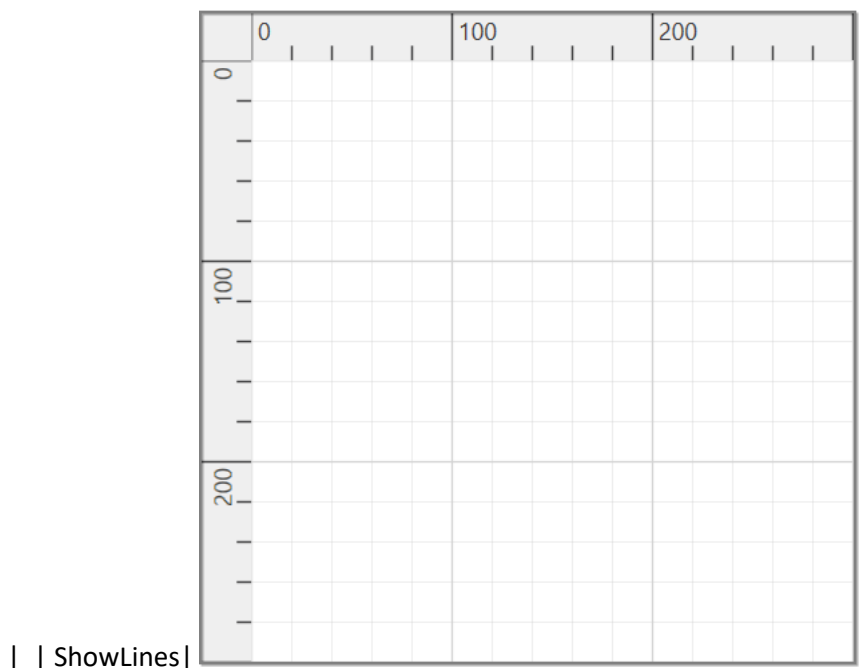
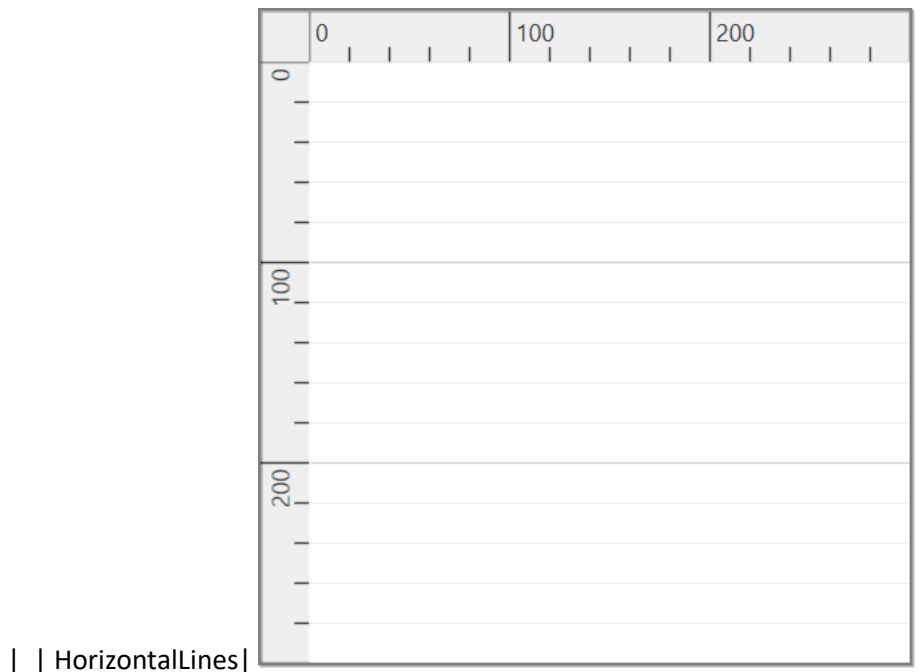
| --- | --- | --- |



| SnapConstraints | None |



| VerticalLines |



### Change grid lines appearance

The `HorizontalGridLines` and `VerticalGridLines` properties of `SnapSettings` class allows you to customize the appearance of the gridlines.

### XML

```
<!--Style for Gridlines-->
<local:Gridlinestyle x:Key="gridlinestyle">
<Style TargetType="Path">
<Setter Property="Stroke" Value="Blue"/>
<Setter Property="StrokeDashArray" Value="3"/>
```

```

</Style>
</local:Gridlinestyle>
<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram" >
<!--Initialize Snap Settings-->
<syncfusion:SfDiagram.SnapSettings>
<syncfusion:SnapSettings SnapConstraints="ShowLines">
<!--Initialize Horizontal Gridlines-->
<syncfusion:SnapSettings.HorizontalGridlines>
<syncfusion:Gridlines Strokes="{StaticResource gridlinestyle}"/>
</syncfusion:SnapSettings.HorizontalGridlines>
<!--Initialize Vertical Gridlines-->
<syncfusion:SnapSettings.VerticalGridlines>
<syncfusion:Gridlines Strokes="{StaticResource gridlinestyle}"/>
</syncfusion:SnapSettings.VerticalGridlines>
</syncfusion:SnapSettings>
</syncfusion:SfDiagram.SnapSettings>
</syncfusion:SfDiagram>

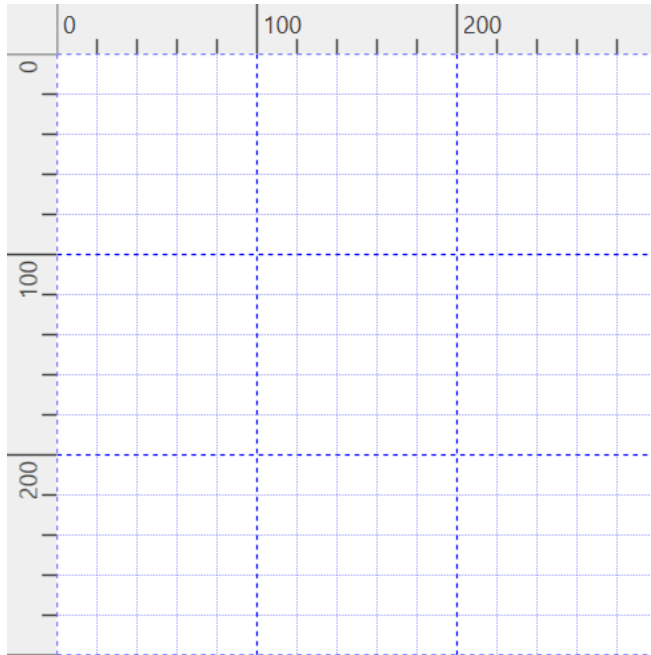
```

**C#**

```

//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Style for Gridlines
Style pathStyle = new Style(typeof(Path));
pathStyle.Setters.Add(new Setter(Shape.StrokeProperty, new
SolidColorBrush(Colors.Blue)));
pathStyle.Setters.Add(new Setter(Shape.StrokeDashArrayProperty, new
DoubleCollection() { 3, 3 }));
//Initialize SnapSettings constraints with HorizontalGridlines and
VerticalGridlines values
diagram.SnapSettings = new SnapSettings()
{
    SnapConstraints = SnapConstraints.ShowLines,
    HorizontalGridlines = new Gridlines()
    {
        Strokes = new Gridlinestyle { pathStyle }
    },
    VerticalGridlines = new Gridlines()
    {
        Strokes = new Gridlinestyle { pathStyle }
    },
};
//Creates collection for the style.
public class Gridlinestyle : List<Style>
{
}

```



### Change grid spacing

The thickness and space between the gridlines can be customized by using `LinesInterval` property of `Gridlines` class. The `LinesInterval` is a type of `List<double>` collection, where the values at the odd indexes are referred as thickness of the lines and the values at the even indexes are referred as space between the gridlines.

### XML

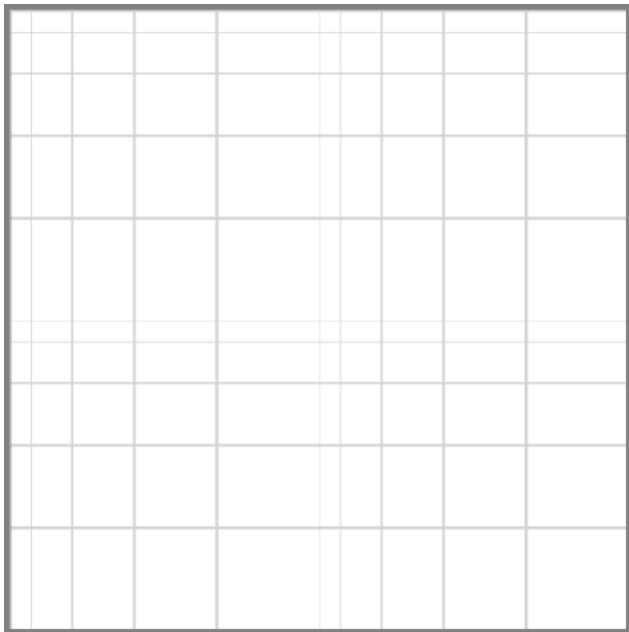
```
<!--Initializes the double collection-->
<local:Intervals x:Key="Intervals">
  <!--Thickness of the grid line-->
  <sys:Double>0.25</sys:Double>
  <!--Space between each gridlines-->
  <sys:Double>10</sys:Double>
  <sys:Double>0.5</sys:Double>
  <sys:Double>20</sys:Double>
  <sys:Double>1</sys:Double>
  <sys:Double>30</sys:Double>
  <sys:Double>1.25</sys:Double>
  <sys:Double>40</sys:Double>
  <sys:Double>1.5</sys:Double>
  <sys:Double>50</sys:Double>
</local:Intervals>
<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram" >
  <syncfusion:SfDiagram.SnapSettings>
    <syncfusion:SnapSettings SnapConstraints="ShowLines">
      <!--Initialize Horizontal Gridlines-->
      <syncfusion:SnapSettings.HorizontalGridlines>
        <syncfusion:Gridlines LinesInterval="{StaticResource Intervals}" />
      </syncfusion:SnapSettings.HorizontalGridlines>
      <!--Initialize Vertical Gridlines-->
      <syncfusion:SnapSettings.VerticalGridlines>
        <syncfusion:Gridlines LinesInterval="{StaticResource Intervals}" />
      </syncfusion:SnapSettings.VerticalGridlines>
    </syncfusion:SnapSettings>
  </syncfusion:SfDiagram>
```



```
</syncfusion:SnapSettings.VerticalGridlines>  
</syncfusion:SnapSettings>  
</syncfusion:SfDiagram.SnapSettings>  
</syncfusion:SfDiagram>
```

## C#

```
//Initialize SfDiagram  
SfDiagram diagram = new SfDiagram();  
//Initialize the double collection  
Intervals intervals = new Intervals { 0.25, 10, 0.5, 20, 1, 30, 1.25, 40,  
1.5, 50 };  
//Initialize Snap Setting constraints with HorizontalGridlines and  
VerticalGridlines values  
diagram.SnapSettings = new SnapSettings()  
{  
    SnapConstraints = SnapConstraints.ShowLines,  
    HorizontalGridlines = new Gridlines()  
    {  
        //Define lines interval value  
        LinesInterval = intervals,  
    },  
    VerticalGridlines = new Gridlines()  
    {  
        //Define lines interval value  
        LinesInterval = intervals,  
    },  
};  
//Creates collection for the double values.  
public class Intervals : List<double>  
{  
}
```



Find the [Gridlines sample](#) to depict the Gridlines.

## [How to snap the objects on gridlines](#)

### Snapping in WPF Diagram (SfDiagram)

When you draw, resize, or move a diagramming element in the page, you can set it, so that it will align or snap to the nearest intersection in the page even when the grid is visible or not. However, you can control the alignment and snap-to capabilities of elements by using the snapping option of SfDiagram.

The `SnapSettings` class and its properties allows you to snap the shapes to the ruler subdivisions more easily.

Refer to the [SnapSettings](#) properties.

#### Snap-to-objects

The snap-to-object provides visual cues to assist with aligning and spacing diagram. A node can be snapped with its neighboring objects based on certain alignments (same size and same position). Such alignments are visually represented as smart guide lines, which are in cyan shade color and its color code is #83F6F0.

Refer to the members of [SnapToObject](#).

Snapping to objects can be enabled by assigning values to the `SnapToObject` property of `SnapSettings` class. Default value is None.

#### XML

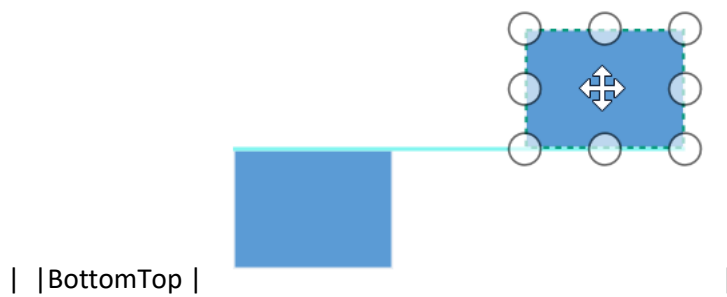
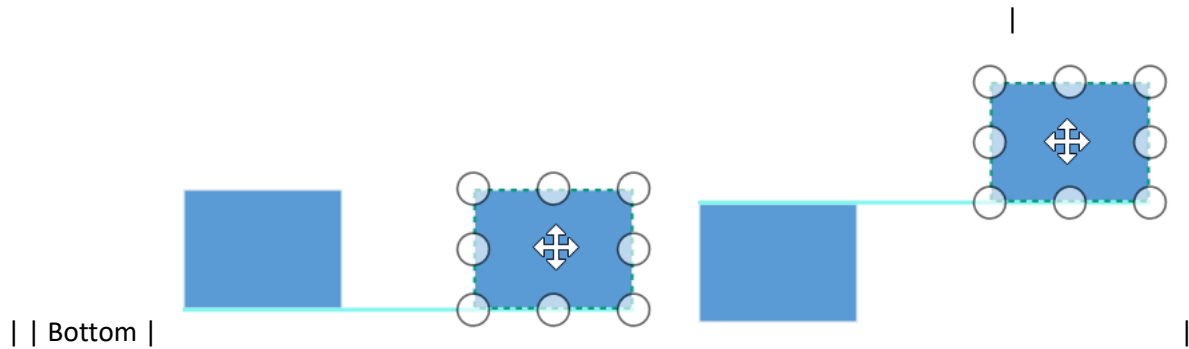
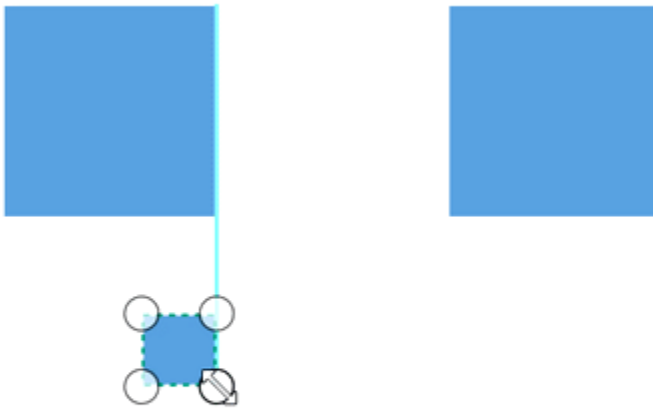
```
<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Enables the SnapToObject value as All -->
  <syncfusion:SfDiagram.SnapSettings>
    <syncfusion:SnapSettings SnapToObject="All"/>
  </syncfusion:SfDiagram.SnapSettings>
</syncfusion:SfDiagram>
```

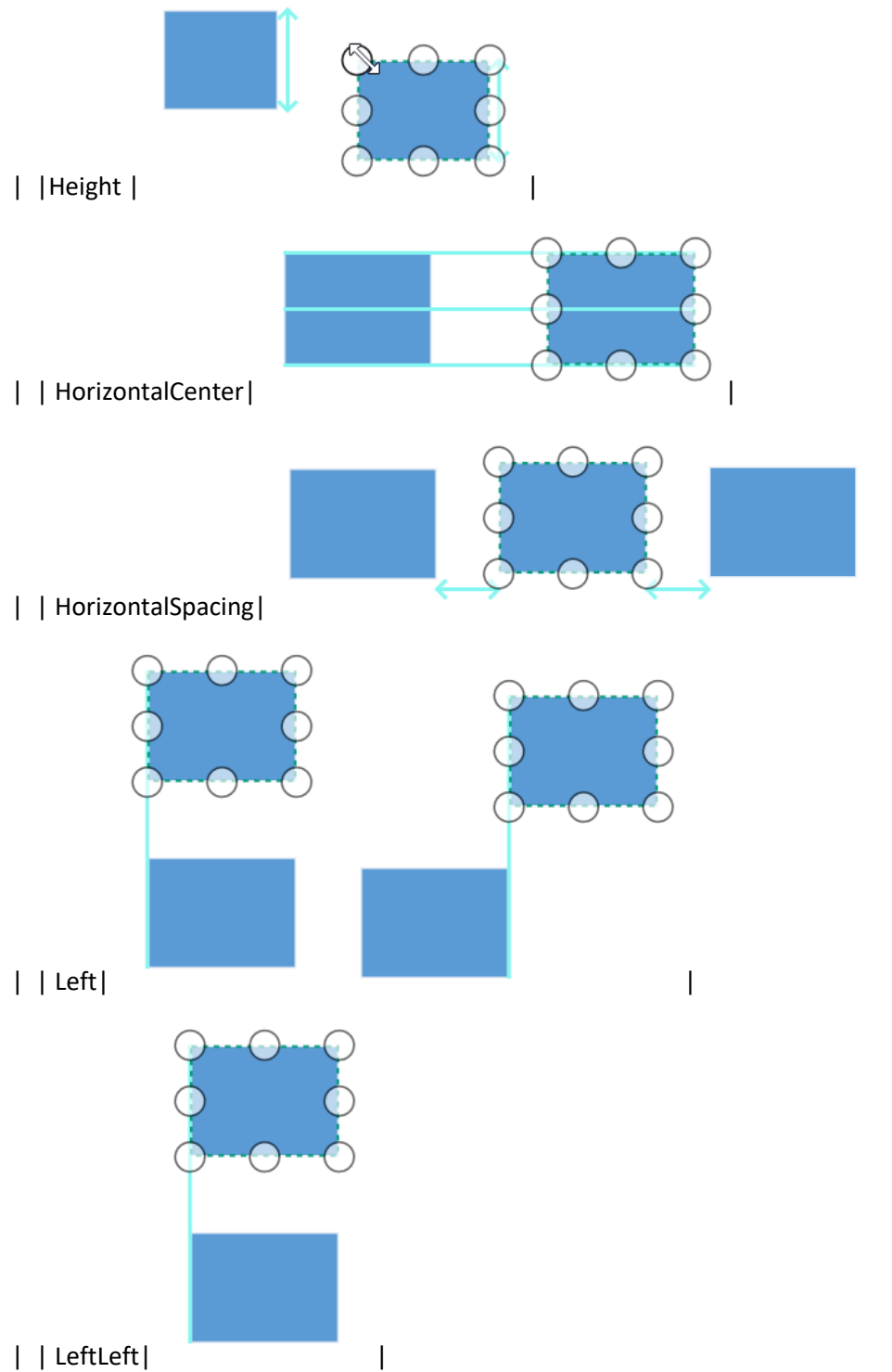
#### C#

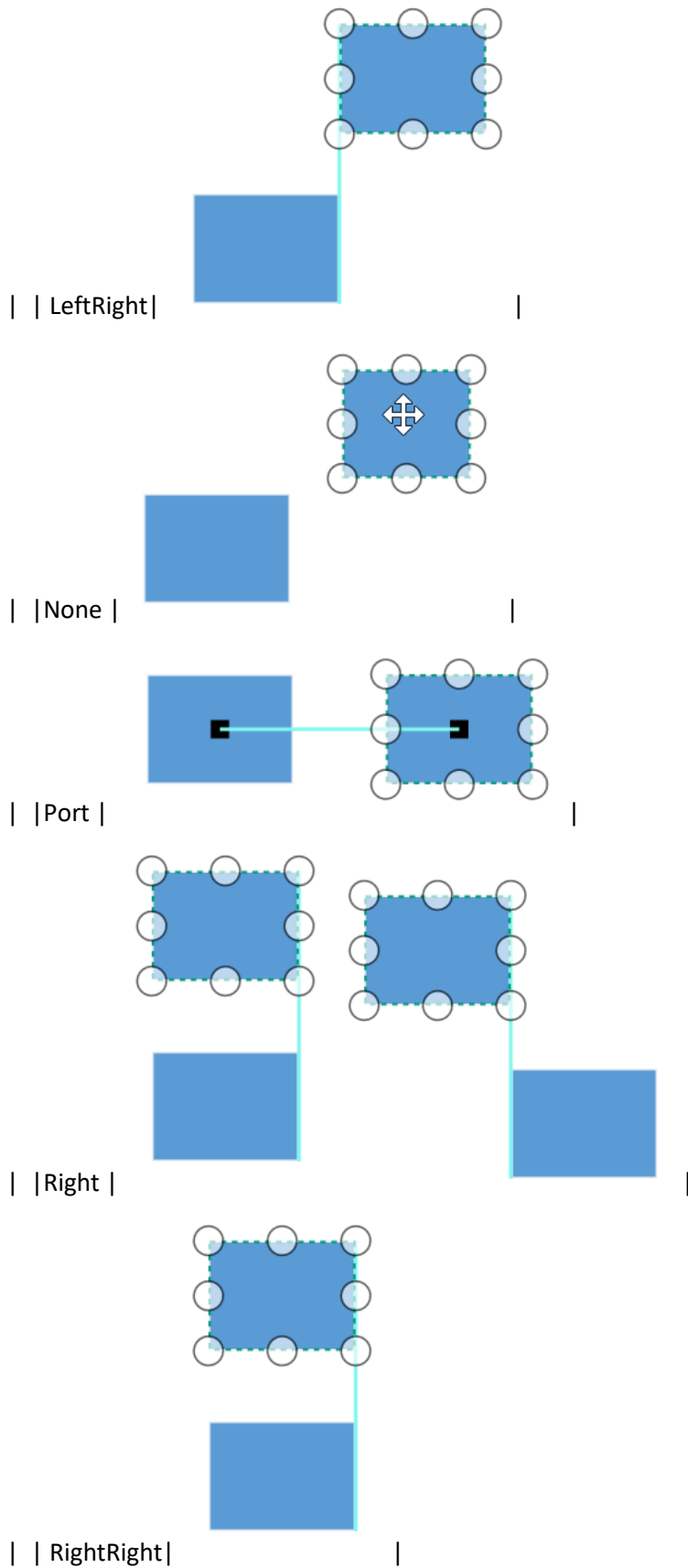
```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Enables the SnapToObject value as All
diagram.SnapSettings.SnapToObject = SnapToObject.All;
```

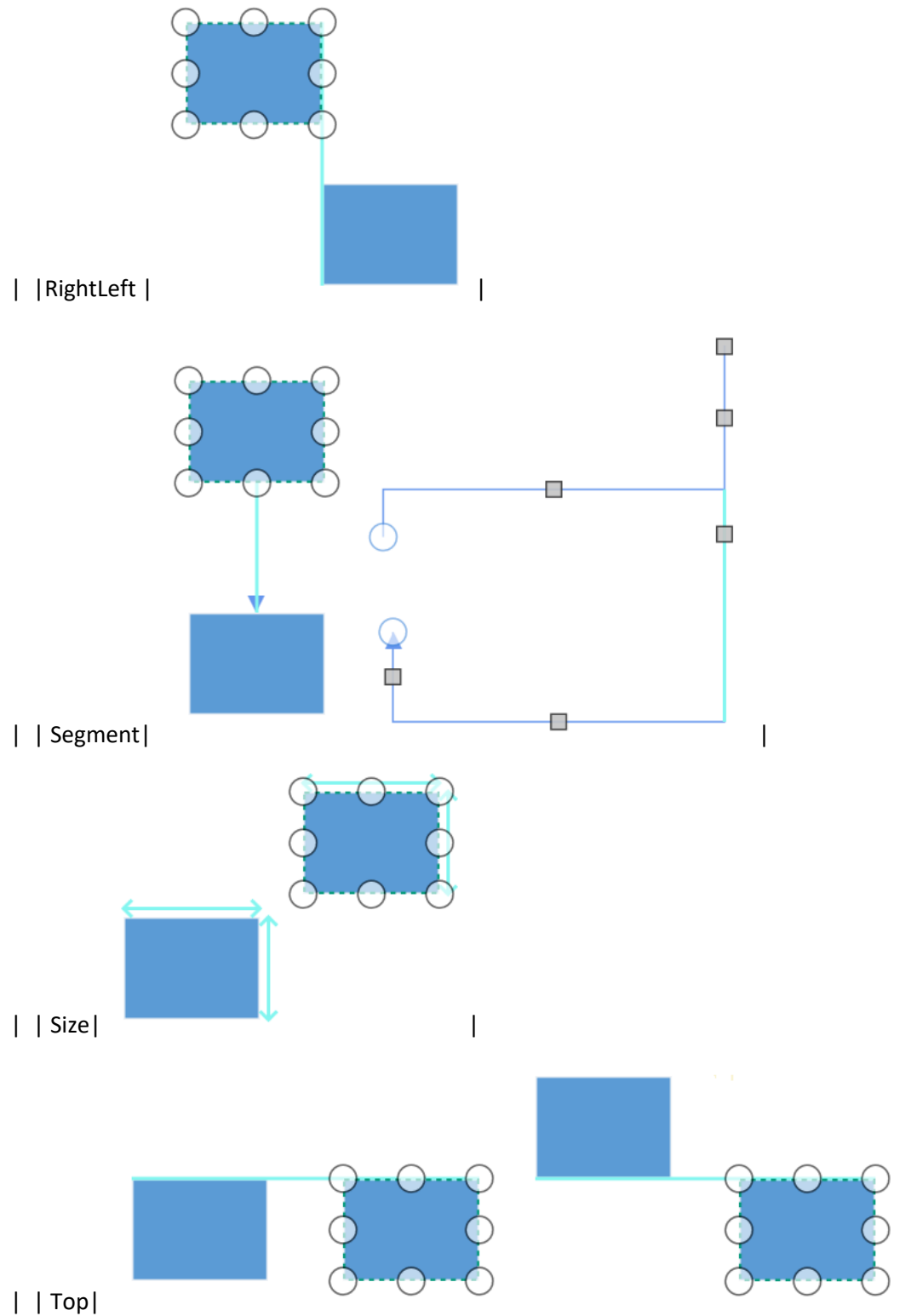
Enum	Value	Output
---	---	---

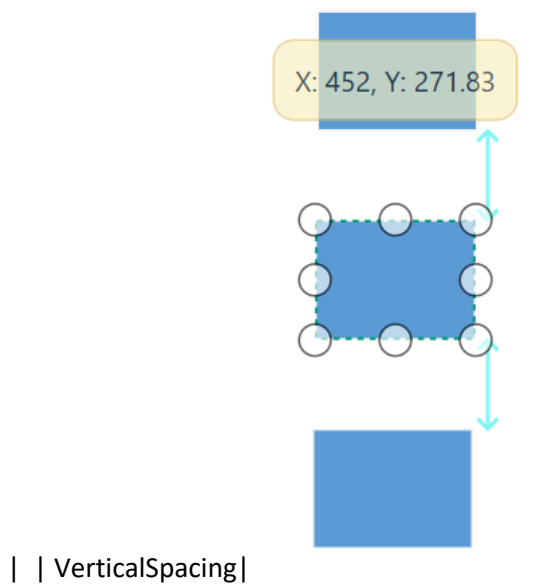
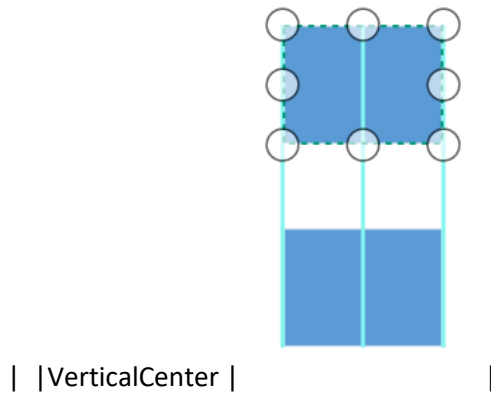
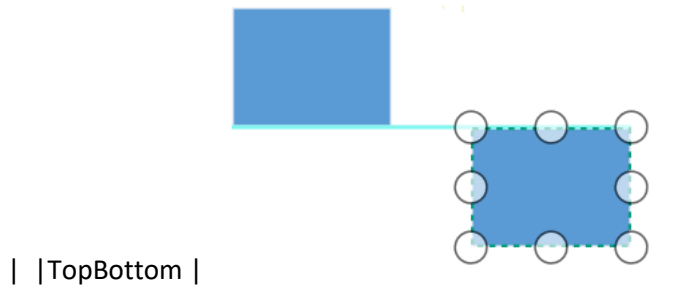
| SnapToObject|All |













### How to change the snap indication style

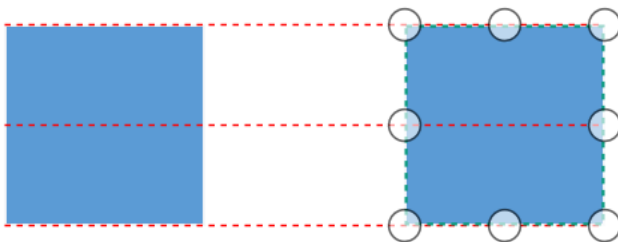
When snapping, cyan shade color indication lines will be shown. These indication lines styles can be customized by using the `SnapIndicatorStyle` property of `SnapSettings` class.

### XML

```
<!--Custom style for snap indicator-->
<Style TargetType="Shape" x:Key="snapIndicatorStyle">
  <Setter Property="Stroke" Value="Red"/>
  <Setter Property="StrokeDashArray" Value="3"/>
</Style>
<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Specify the snap indicator style -->
  <syncfusion:SfDiagram.SnapSettings>
    <syncfusion:SnapSettings SnapToObject="All"
      SnapIndicatorStyle="{StaticResource snapIndicatorStyle}" />
  </syncfusion:SfDiagram.SnapSettings>
</syncfusion:SfDiagram>
```

### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Custom style for Snap indicator
Style pathStyle = new Style(typeof(Shape));
pathStyle.Setters.Add(new Setter(Shape.StrokeProperty, new
SolidColorBrush(Colors.Red)));
pathStyle.Setters.Add(new Setter(Shape.StrokeDashArrayProperty, new
DoubleCollection() { 3, 3 }));
diagram.SnapSettings = new SnapSettings()
{
  //SnapConstraints = SnapConstraints.SnapToLines ,
  SnapToObject = SnapToObject.All,
  //Specifies custom snap indicator style
  SnapIndicatorStyle = pathStyle as Style,
};
```





### 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.

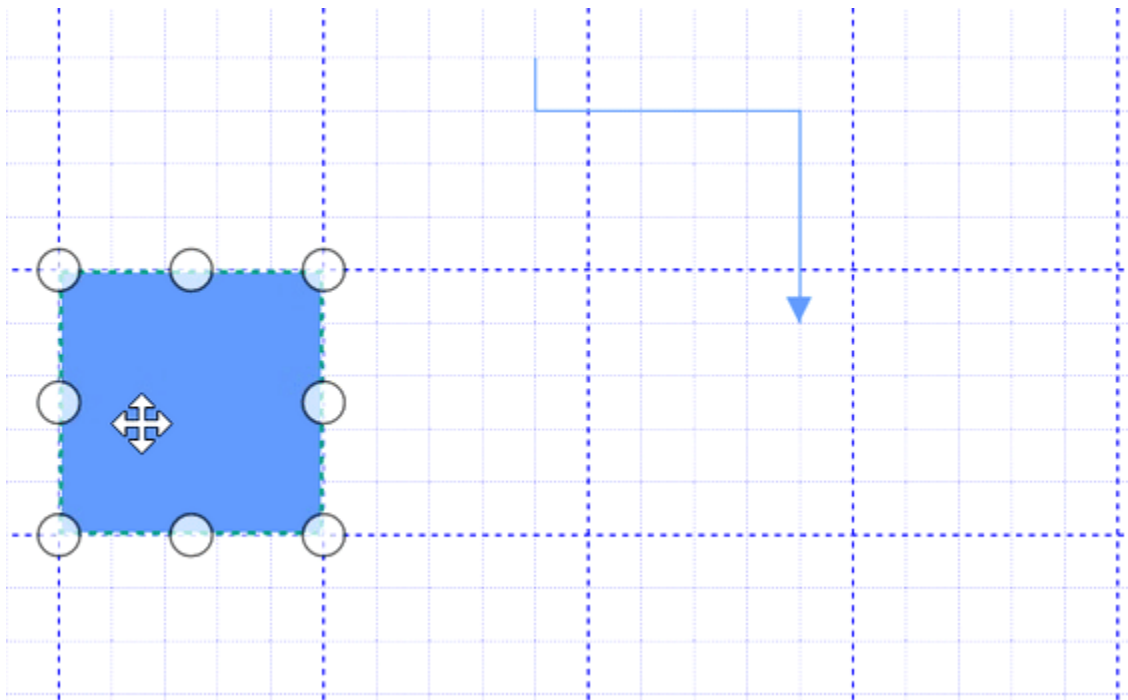
Snapping to gridlines can be enabled or disabled with the [SnapConstraints](#) property of SnapSettings. Default value is None.

### XML

```
<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Enables the SnapConstraints value as SnapToLines -->
  <syncfusion:SfDiagram.SnapSettings>
    <syncfusion:SnapSettings SnapConstraints="SnapToLines,ShowLines"/>
  </syncfusion:SfDiagram.SnapSettings>
</syncfusion:SfDiagram>
```

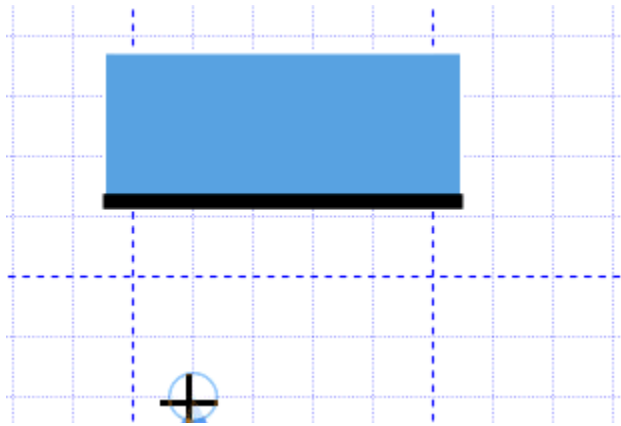
### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Enables the SnapConstraints value as SnapToLines
diagram.SnapSettings.SnapConstraints = SnapConstraints.ShowLines |
SnapConstraints.SnapToLines;
```



### Snapping on dock ports

Diagram allows snapping on dock ports while making new connection on it and dragging connection over dock ports.



Find the [Snapping sample](#) to depict the Snapping.

### Page Settings in WPF Diagram (SfDiagram)

By default, Diagram's page size is decided based on the position of diagramming elements. The size and appearance of the diagram pages can be customized using the [PageSettings](#) property of SfDiagram.

- The `PageWidth` and `PageHeight` properties of `PageSettings` define the size of the page.
- The `PageOrientation` property of `PageSettings` used to change the page orientation to portrait or landscape.
- Page breaks are used as a visual guide to see how the pages split into multiple pages. The `ShowPageBreaks` property decides the Visibility of Page breaks.

### XML

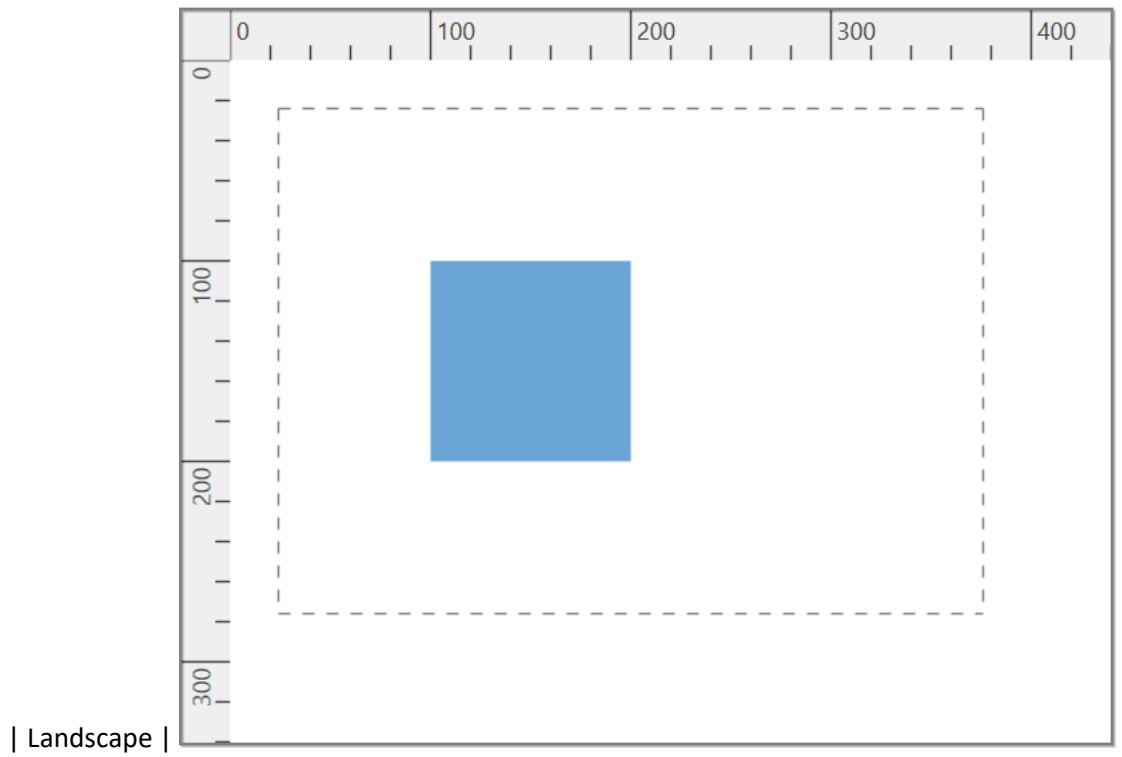
```
<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Initialize the page settings and page orientation property-->
  <syncfusion:SfDiagram.PageSettings>
    <syncfusion:PageSettings PageOrientation="Portrait"
    PageWidth="300" PageHeight="400"
    ShowPageBreaks="True" />
  </syncfusion:SfDiagram.PageSettings>
</syncfusion:SfDiagram>
```

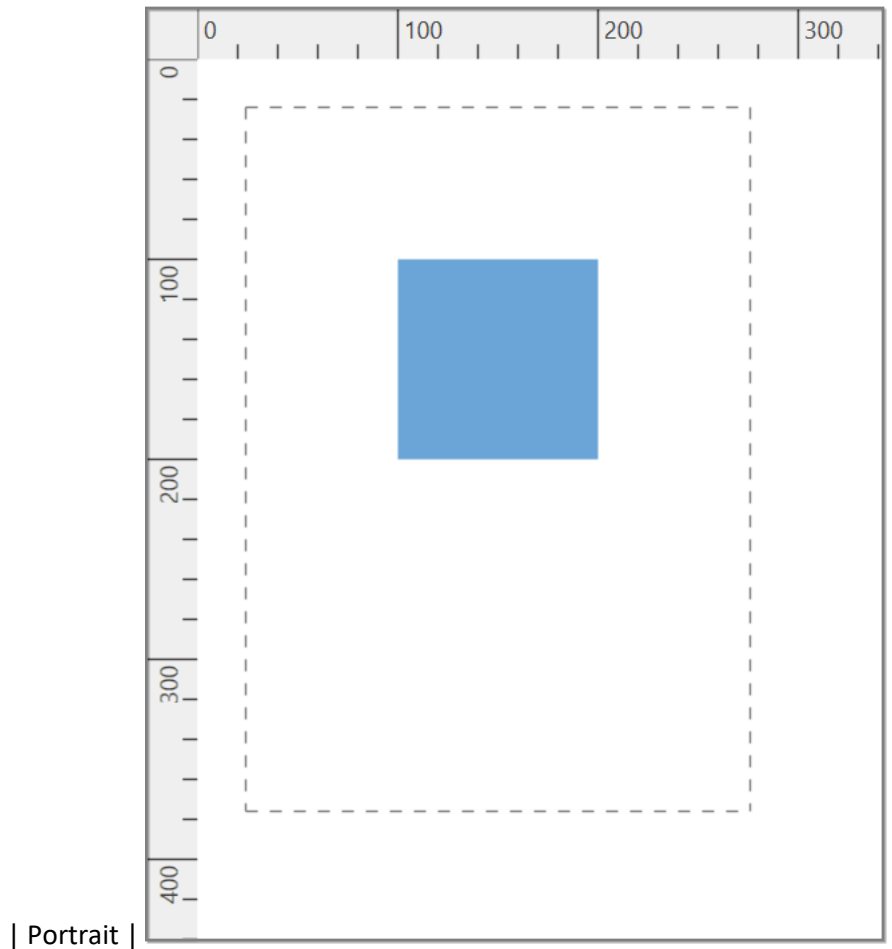
### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize the page settings and page orientation property
diagram.PageSettings = new PageSettings()
{
    PageOrientation = PageOrientation.Portrait,
    PageWidth = 300,
    PageHeight = 400,
    ShowPageBreaks = true,
};
```

| Page Orientation | Output |

|---|---|





### How to enable the multiple page

Based on the diagramming element position, the size of the page dynamically increases or decreases in multiples of page width and height using the **MultiplePage** property of PageSettings.

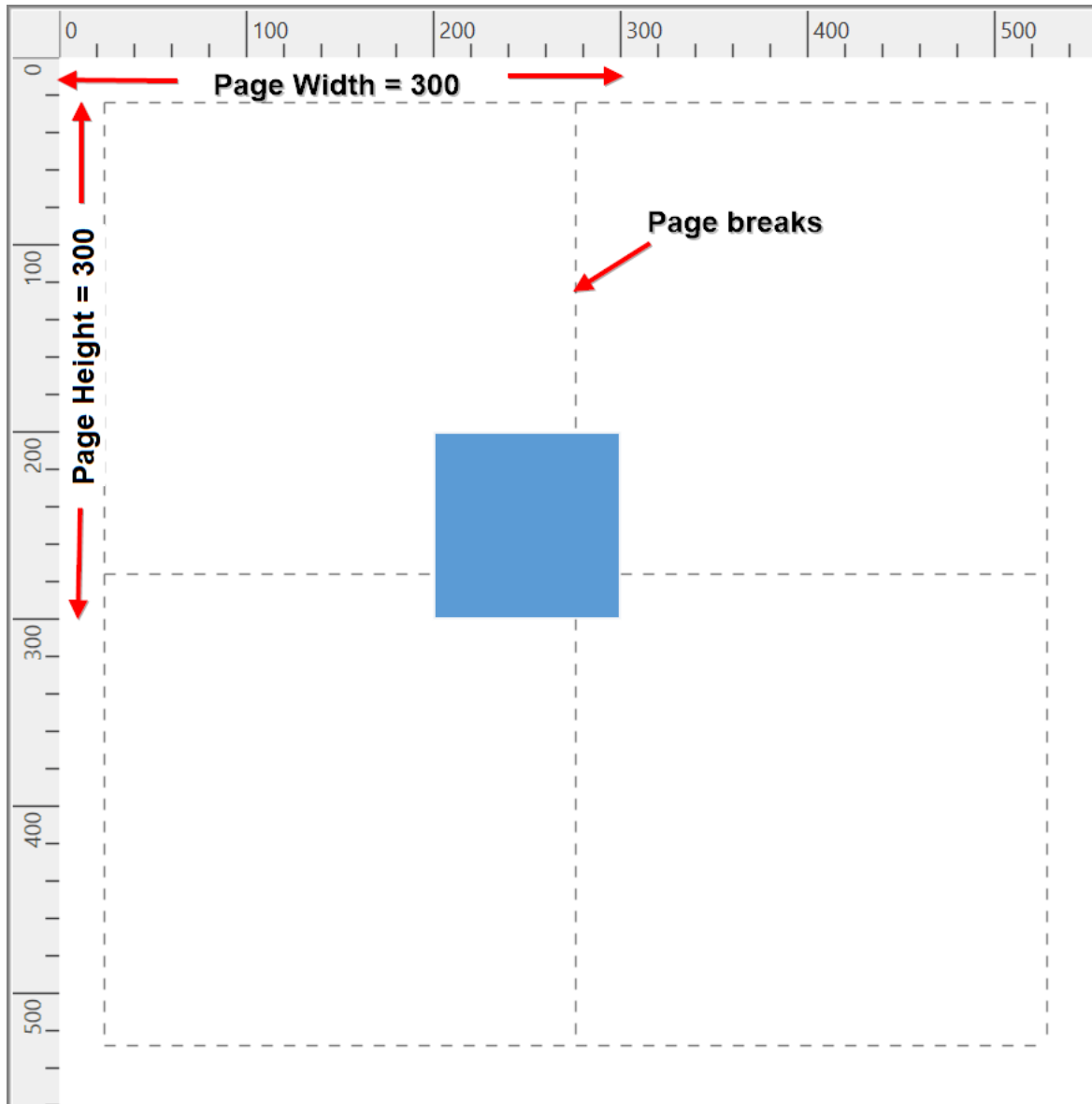
### XML

```
<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
<!--Initialize the page settings and its properties-->
<syncfusion:SfDiagram.PageSettings>
<syncfusion:PageSettings PageWidth="500"
PageHeight="500"
MultiplePage="True"
ShowPageBreaks="True" />
</syncfusion:SfDiagram.PageSettings>
</syncfusion:SfDiagram>
```

### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize the page settings and its properties
diagram.PageSettings = new PageSettings()
{
```

```
PageHeight = 300,  
PageWidth = 300,  
MultiplePage = true,  
ShowPageBreaks = true,  
};
```



#### How to change the page appearance

The appearance of the pages can be customized by using the following properties of `PageSettings` class:

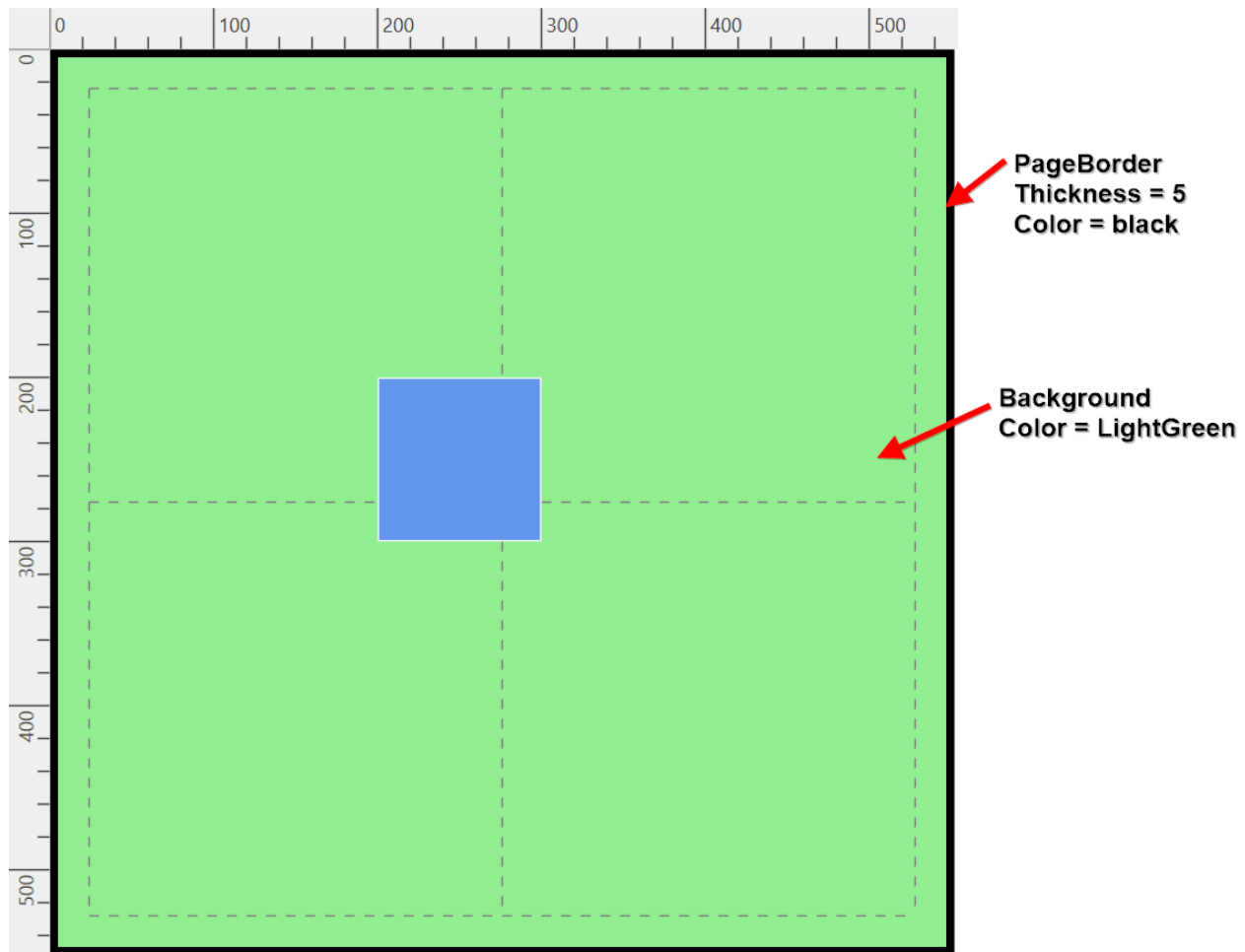
- `PageBorderThickness`: Defines the thickness of the border around the entire page.
- `PageBackground`: Defines the background colors of the page.
- `PageBorderBrush`: Defines the color of the page border.

#### XML

```
<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Initialize the page settings and its properties-->
  <syncfusion:SfDiagram.PageSettings>
    <syncfusion:PageSettings PageBorderThickness="5"
    PageBackground="LightGreen"
    PageBorderBrush="Black"
    PageWidth="500" PageHeight="500"
    MultiplePage="True"
    ShowPageBreaks="True" />
  </syncfusion:SfDiagram.PageSettings>
</syncfusion:SfDiagram>
```

## C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize the page settings and its properties
diagram.PageSettings = new PageSettings()
{
    PageBorderThickness = new Thickness(5),
    PageBackground = new SolidColorBrush(Colors.LightGreen),
    PageBorderBrush = new SolidColorBrush(Colors.Black),
    PageWidth = 300,
    PageHeight = 300,
    MultiplePage = true,
    ShowPageBreaks = true,
};
```



### How to change the margin around the pages

The area between the main content of a page and the page edges can be changed by using the `PrintMargin` property. Default value is 24.

### XML

```
<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Initialize the page settings and print margin property-->
  <syncfusion:SfDiagram.PageSettings>
    <syncfusion:PageSettings PrintMargin="40"
      PageWidth="300" PageHeight="300"
      ShowPageBreaks="True" />
  </syncfusion:SfDiagram.PageSettings>
</syncfusion:SfDiagram>
```

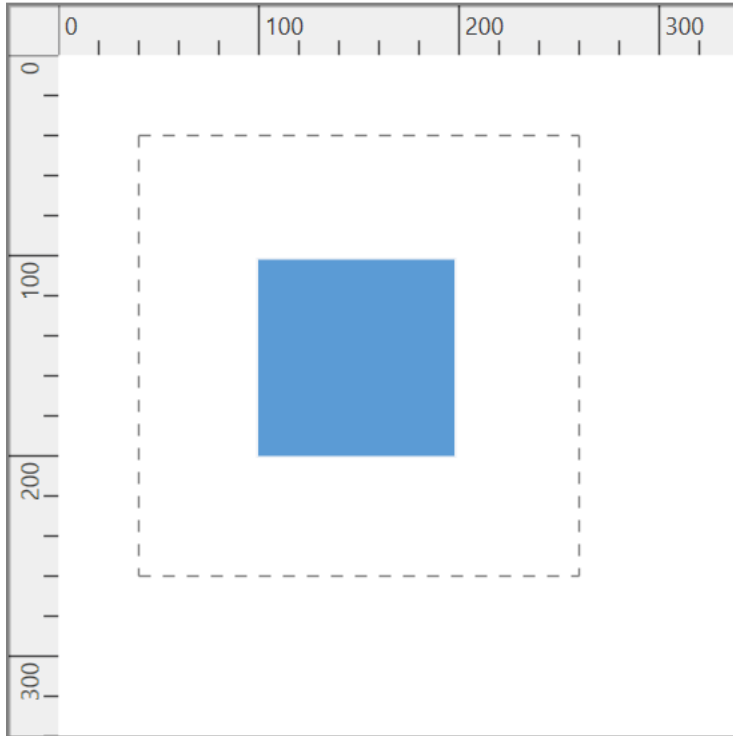
### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize the page settings and print margin property
diagram.PageSettings = new PageSettings()
{
    PrintMargin = new Thickness(40),
```

```

PageWidth = 300,
PageHeight = 300,
ShowPageBreaks = true,
};

```



### How to change the unit of the page

The measurement units of the page can be changed by using the **Unit** property of **PageSettings** class. Default unit value is Pixels.

#### XML

```

<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Initialize the page settings and unit property-->
  <syncfusion:SfDiagram.PageSettings>
    <syncfusion:PageSettings>
      <syncfusion:PageSettings.Unit>
        <syncfusion:LengthUnit Unit="Inches"/>
      </syncfusion:PageSettings.Unit>
    </syncfusion:PageSettings>
  </syncfusion:SfDiagram.PageSettings>
</syncfusion:SfDiagram>

```

#### C#

```

//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize the page settings and unit property.
diagram.PageSettings = new PageSettings()
{

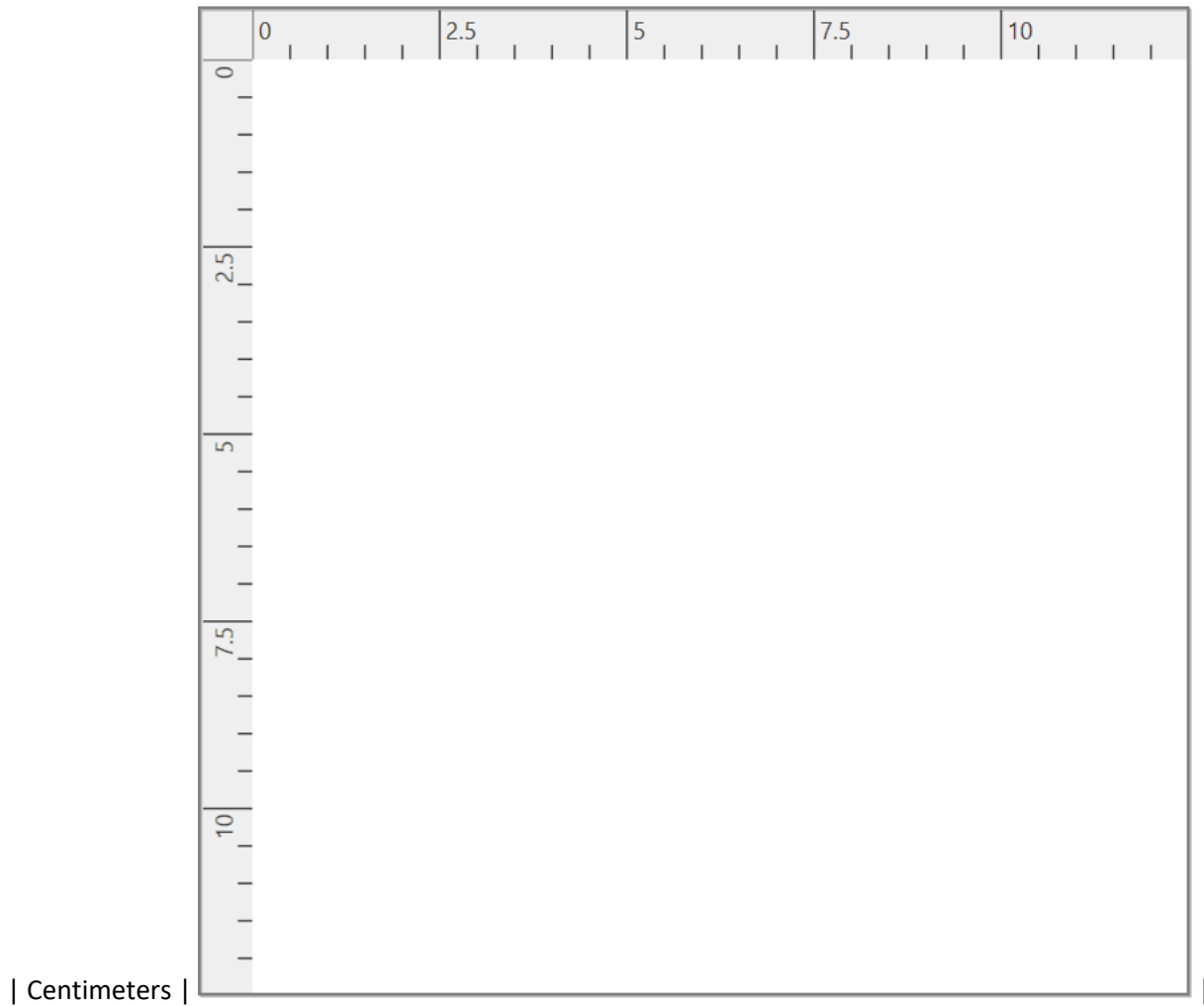
```

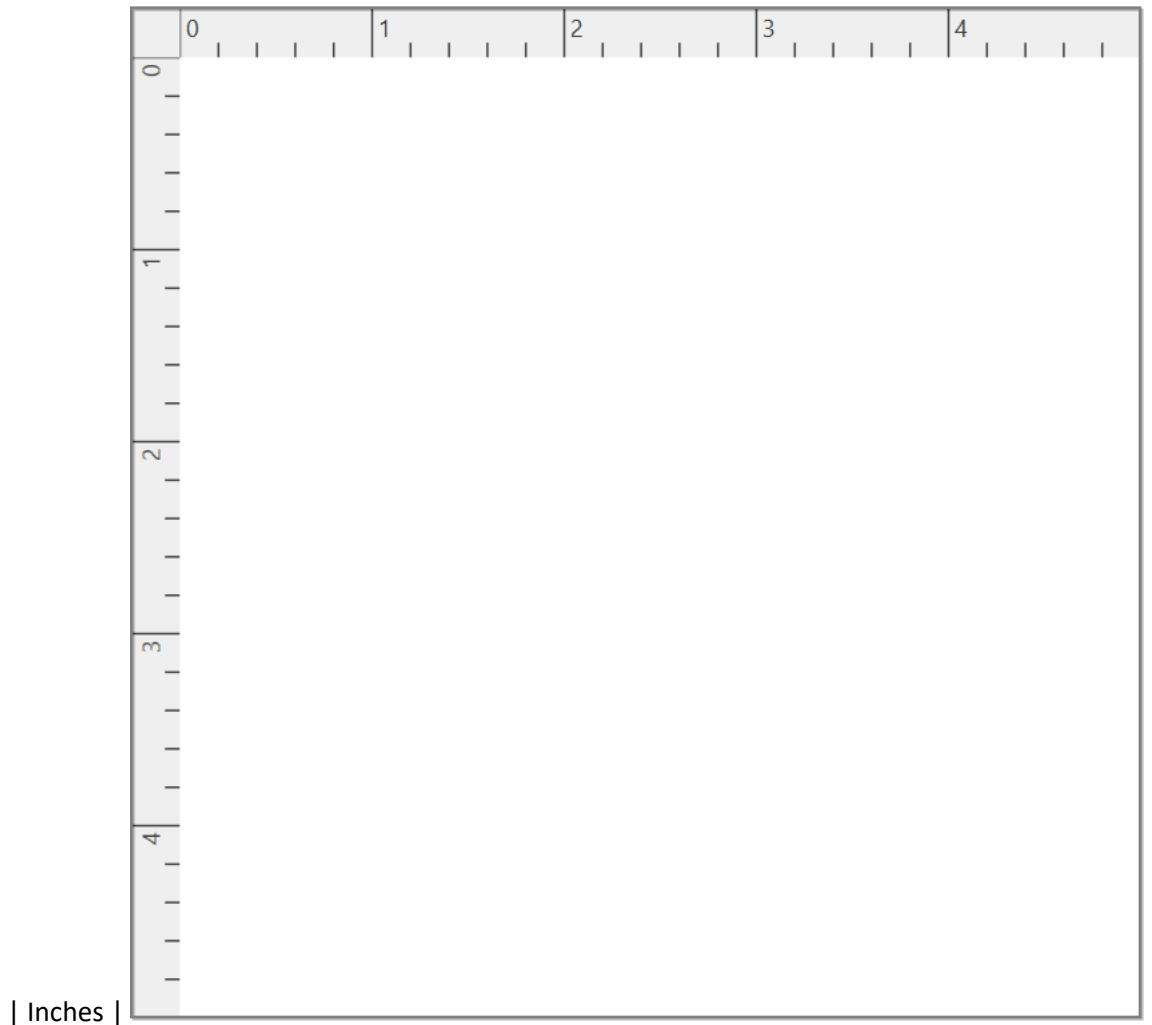


```
Unit = new LengthUnit() { Unit = LengthUnits.Inches },  
};
```

| Unit values | Output |

|---|---|





### How to change the scaling of Page

The `PrintScale` property of `PageSettings` class allows you to reduce or enlarge the diagram page to fit into specific area when you print it. The default scaling of the page will be 1 (i.e 100%).

### XML

```
<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Initialize the page settings -->
  <syncfusion:SfDiagram.PageSettings>
    <syncfusion:PageSettings PrintMargin="40"
      PageWidth="300" PageHeight="300"
      ShowPageBreaks="True"
      MultiplePage="True"
      PrintScale="3" />
  </syncfusion:SfDiagram.PageSettings>
</syncfusion:SfDiagram>
```

### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
```

```
//Initialize the page settings and print scale value.
diagram.PageSettings = new PageSettings()
{
    PrintMargin = new Thickness(40),
    PageWidth = 300,
    PageHeight = 300,
    ShowPageBreaks = true,
    MultiplePage = true,
    //Specify the print scale value.
    PrintScale = 3,
};
```

You can restrict the diagram page scaling to minimum and maximum values by using the `MinimumPrintScale` and `MaximumPrintScale` properties.

### XML

```
<!--Initialize SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Initialize the page settings -->
  <syncfusion:SfDiagram.PageSettings>
    <syncfusion:PageSettings PrintMargin="40"
    PageWidth="300" PageHeight="300"
    ShowPageBreaks="True"
    MultiplePage="True"
    MinimumPrintScale="2"
    MaximumPrintScale="4"/>
  </syncfusion:SfDiagram.PageSettings>
</syncfusion:SfDiagram>
```

### C#

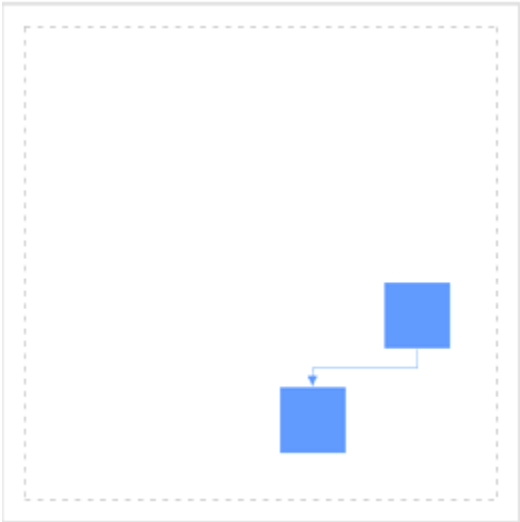
```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize the page settings and print scale value.
diagram.PageSettings = new PageSettings()
{
    PrintMargin = new Thickness(40),
    PageWidth = 300,
    PageHeight = 300,
    ShowPageBreaks = true,
    MultiplePage = true,
    //Specify the minimum and maximum print scale values.
    MinimumPrintScale = 2,
    MaximumPrintScale = 4,
};
```

### How to customize the page origin

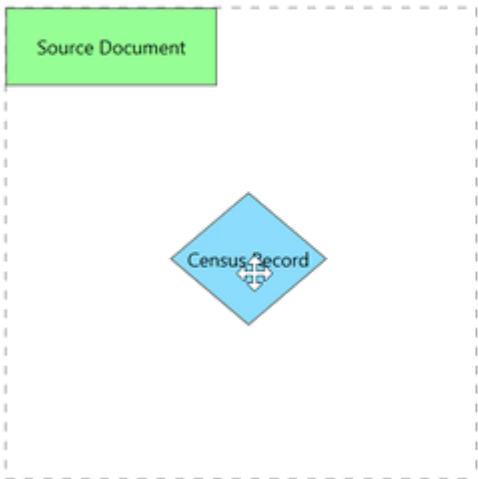
By default, origin of the diagram page will be at (0,0). A virtual method `AdjustPageOrigin` of `PageSettings` class is used to customize the origin of the page based on the page size (`PageWidth` and `PageHeight`) and diagram elements position, which in turn helps to reduce the number of pages being printed.

The `AdjustPageOrigin` method contains the `Info` property in its argument, and also have the following properties and methods:

Properties/Methods	Description	Output
<code>Truncate</code>	Customizes the origin based on the page size.	

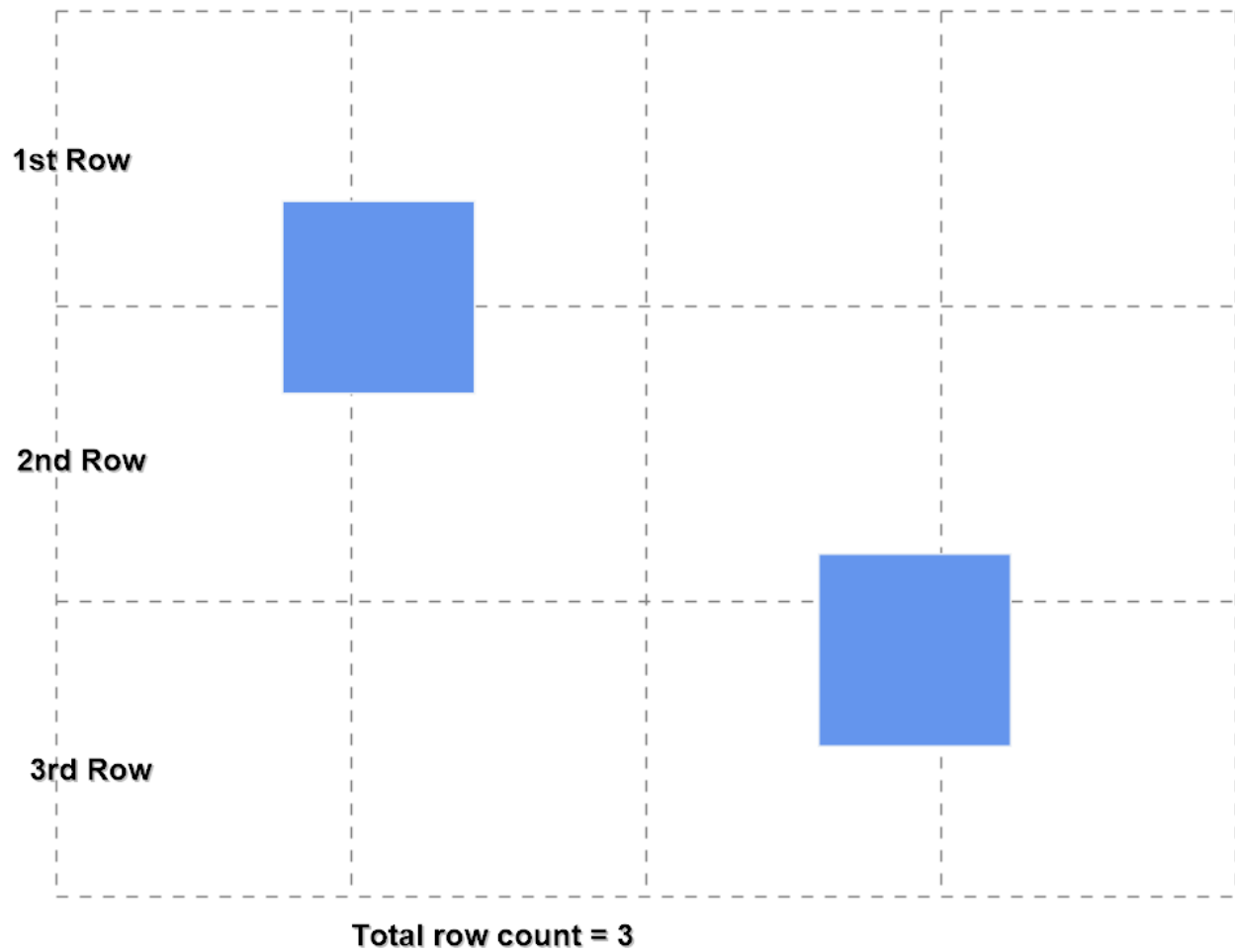


`Trim` | Customizes the origin based on the pixel. Page origin will be position of the most top left



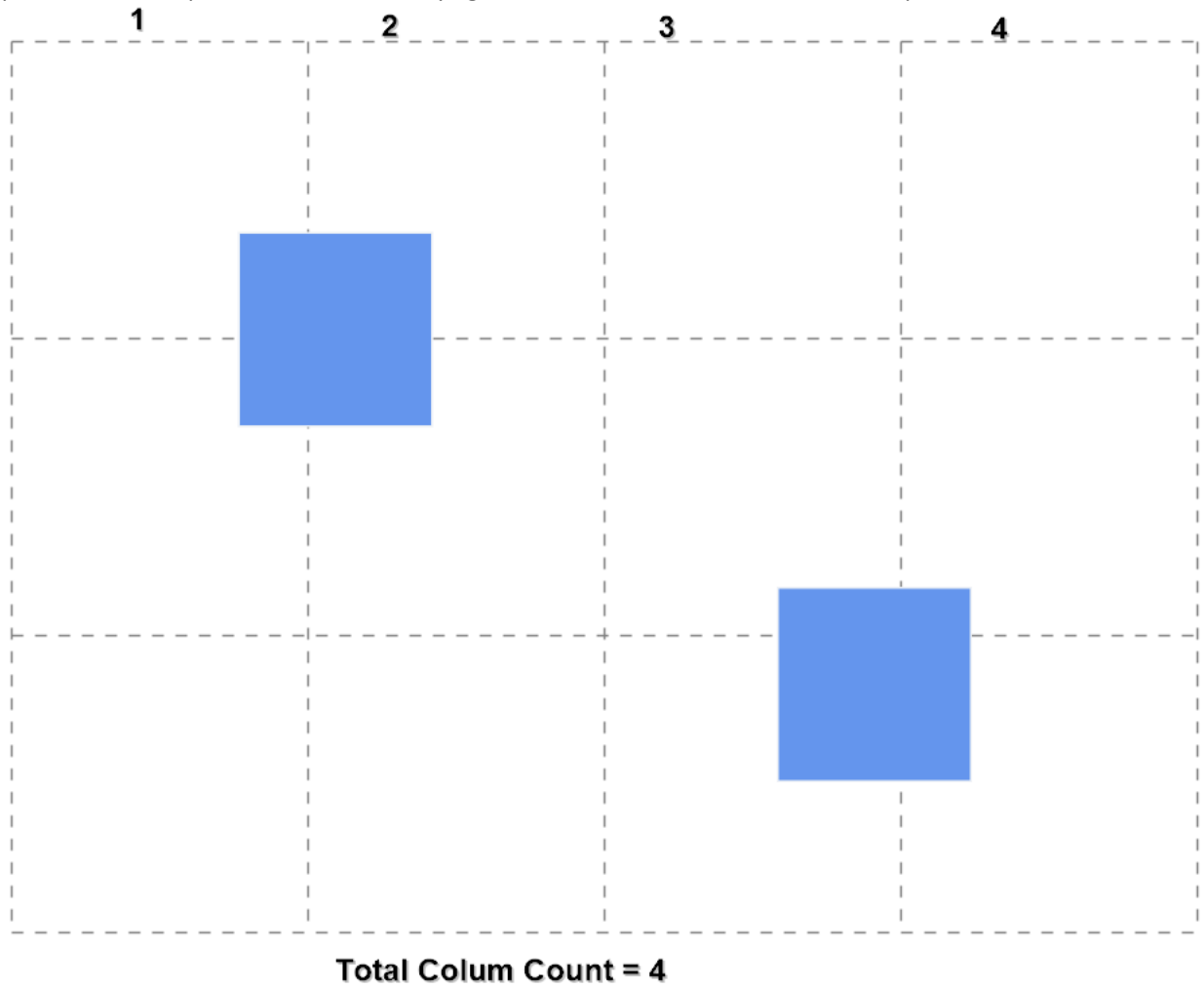
element.

| RowCount | Gets the number of pages available in row-wise manner. |



|

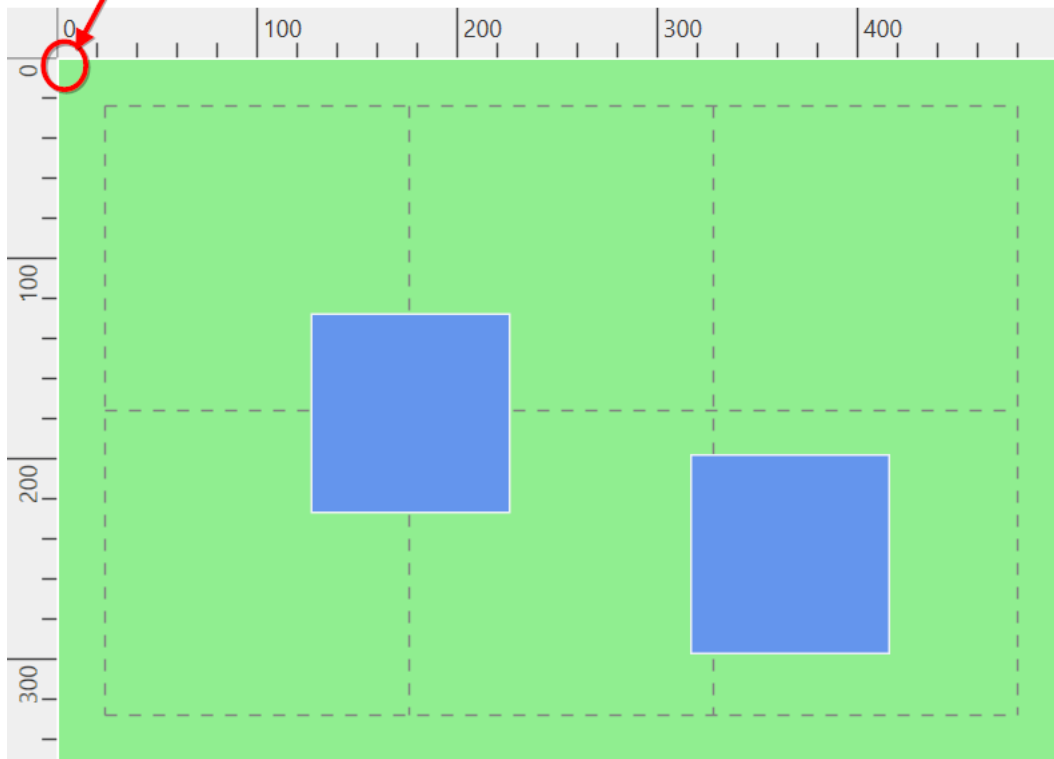
| ColumnCount | Gets the number of pages available in column-wise manner. |



|

| PageOrigin | Gets the origin of the page. |

**Page Origin = (0,0)**



### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Create instance for PageCustomClass using PageSettings
diagram.PageSettings = new CustomPageSettings();
//Create a custom class which is derived from page setting class
public class CustomPageSettings : PageSettings
{
    public override void AdjustPageOrigin()
    {
        //Specifies the row count
        double rowCount = Info.RowCount;
        //Specifies the column count
        double columnCount = Info.ColumnCount;
        //Specifies the page origin
        Point pageOrigin = Info.PageOrigin;
        Info.Truncate();
    }
}
```

**Note:** This customization plays a role in reducing the number of pages in printing.

Find the [Page Settings sample](#) to depict the Page Settings.

[How to add or remove grid lines for diagram page](#)

[How to define scroll setting for diagram page](#)

[How to add rulers for diagram page](#)

[How to print the diagram pages](#)

[How to export the diagram page](#)

[How to interact with diagram page](#)

[How to add multiple diagram pages?](#)

[How to use the property grid for diagram?](#)

## Scroll-Settings in WPF Diagram (SfDiagram)

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 scroll settings allows you to read the current scroll status, view port size, current zoom, and zoom factor values.

### Get current scroll status

Diagram allows to get the scroll settings related values using the **ScrollInfo** property of **ScrollSettings** class.

- **CurrentZoom**: Specifies the zooming level of the diagram page.
- **Viewport**: Specifies the position and dimensions of diagram's visible area.
- **ViewportHeight**: Specifies the height of the view port area of diagram control.
- **ViewportWidth**: Specifies the width of the view port area of diagram control.
- **HorizontalOffset**: Specifies the horizontal origin or left side origin of the view port of the diagram page.
- **VerticalOffset**: Specifies the vertical origin or top side of the view port of the diagram page.

### C#

```
//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Get type values of ScrollInfo property in scroll settings class
double currentZoom = diagram.ScrollSettings.ScrollInfo.CurrentZoom;
double horizontalOffset =
diagram.ScrollSettings.ScrollInfo.HorizontalOffset;
double verticalOffset = diagram.ScrollSettings.ScrollInfo.VerticalOffset;
Rect viewport = diagram.ScrollSettings.ScrollInfo.Viewport;
double viewportHeight = diagram.ScrollSettings.ScrollInfo.ViewportHeight;
double viewportWidth = diagram.ScrollSettings.ScrollInfo.ViewportWidth;
```





### How to update the scroll status

You can programmatically change scrolling amount at runtime by using the `ZoomPan()` method.

The `ScrollDelta` property of `ZoomPositionParameter` class allows you to specify how much the page should be scrolling. The `ZoomCommand` property is used to specify whether it is vertical scroll bar or horizontal scroll bar.

### C#

```
//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize the scroll settings with zoom to value and zoom command
diagram.ScrollSettings.ScrollInfo.ZoomPan(new ZoomPositionParameter
{
    ZoomCommand = ZoomCommand.VerticalScroll,
    ScrollDelta = 50,
});
```

| ScrollDelta value | ZoomCommand | Output |

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

| 50 | `ZoomCommand.VerticalScroll` |



| 50 | `ZoomCommand.HorizontalScroll` |



### 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

### Autoscroll border

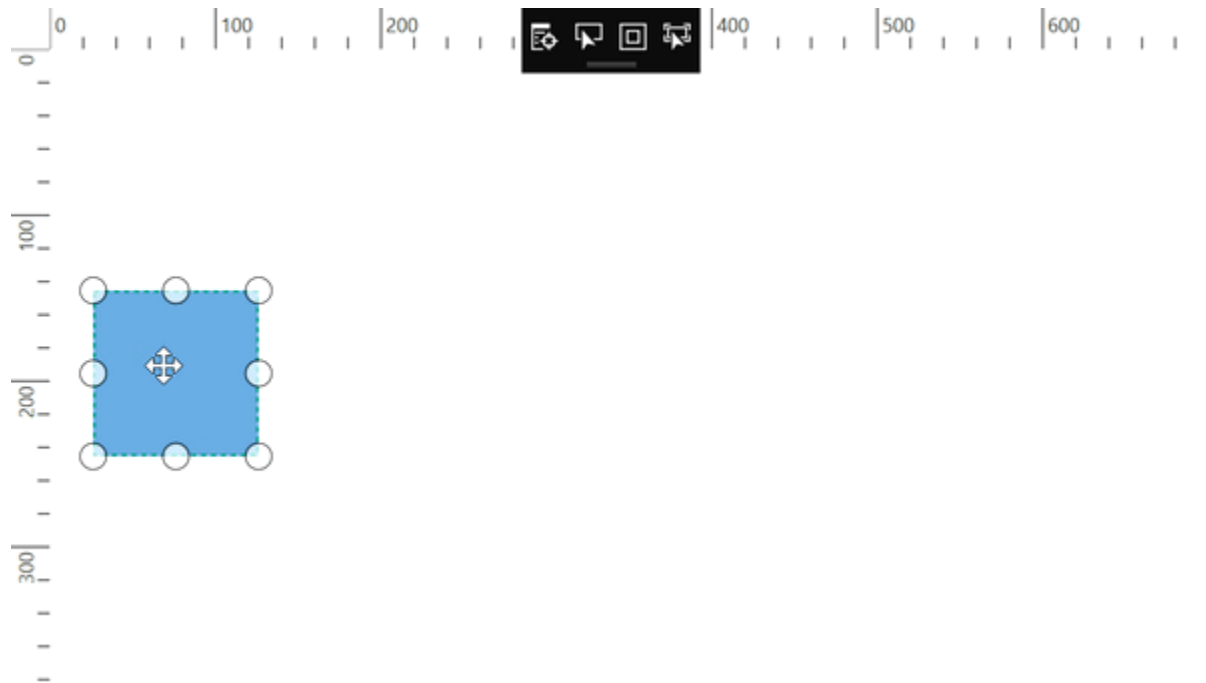
The Autoscroll border is used to specify the maximum distance between the object and Diagram edge to trigger Autoscroll. The default value is set as 20 for all sides (left, right, top, and bottom) and it can be changed by using the `AutoScrollBorder` property of `ScrollSettings`.

### XML

```
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Initialize the scroll setting class with auto scroll border-->
  <syncfusion:SfDiagram.ScrollSettings>
    <syncfusion:ScrollSettings AutoScrollBorder="40"/>
  </syncfusion:SfDiagram.ScrollSettings>
</syncfusion:SfDiagram>
```

### C#

```
//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Initialize the scroll settings with auto scroll border
diagram.ScrollSettings = new ScrollSettings()
{
    AutoScrollBorder = new Thickness(40),
};
```



Find the [Scroll Settings sample](#) to depict the Scroll Settings.

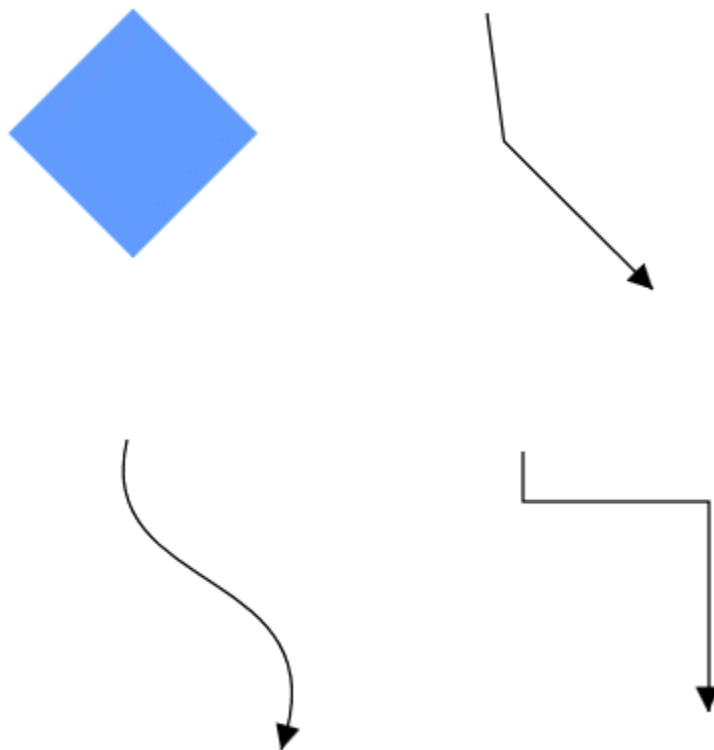
## Interaction

### Selection in WPF Diagram (SfDiagram)

[Selector](#) provides a visual representation of selected elements. It behaves like a container and enables you to update the size, position, and rotation angle of the selected elements through interaction and programmatically. 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.



#### 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.

1. Shift+Click

You can select the group of elements that are contiguous (i.e. next to each other) by clicking one element, and then holding Shift and clicking the last element. All the element in the specified region are then selected.

## 2. 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.

## 3. 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.

Multiple selected elements are visually represented as shown.



- [SelectorChangedEvent](#) will notify you the OffsetX, OffsetY, Height, Width, Rotate Angle and interaction state with their old and new values. To explore about arguments, please refer to [SelectorChangedEventArgs](#) .

Selection mode

[SingleSelectionMode](#) and [MultipleSelectionMode](#) properties of SfDiagram allows us to decide which kind of selection need to be handle .To explore about modes, please refer to [SingleSelectionMode](#) and [MultipleSelectionMode](#).

|SingleSelectionMode|Description|

|--|--|

|Select| Enables or disables single selection mode as Select. It is used to stop Unselection again click the same node which means the node remains always selected. |

|ToggleSelection| Enables or disables single selection mode as ToggleSelection.It is used to perform selection or unselection again click the same node. |

### XML

```
<Syncfusion:SfDiagram x:Name="Diagram"
SingleSelectionMode="Select">
```

### C#

```
SfDiagram Diagram = new SfDiagram();
Diagram.SingleSelectionMode = SingleSelectionMode.Select;
```

|MultipleSelectionMode|Description|

|--|--|

|Default| Enables all behaviors of the control. |

|HoldKeyAndTap| Enables or disables elements can be selected by holding a key and tapping. |

|JustTap| Enables or disables elements can be selected by tapping. |

|None| Disables all behaviors. |

|RubberBandCompleteIntersect| Enables or disables elements that are completely positioned in the selection rectangle will be selected. |

|RubberBandPartialIntersect| Elements that intersect with the selection rectangle will be selected. |

[View sample in GitHub](#)

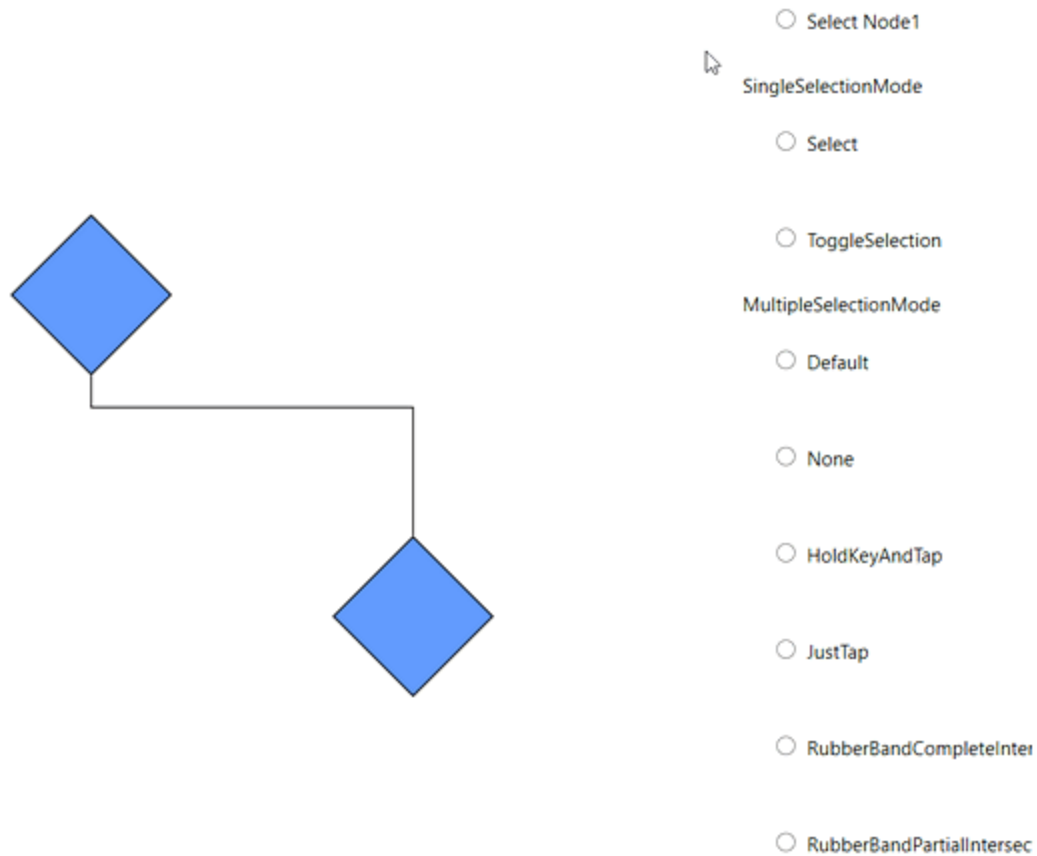
[Select/Unselect the elements programmatically](#)

The `IsSelected` Property is used to select/unselect the elements at runtime.

The following code example illustrates how to select/unselect an item programmatically.

### C#

```
// Selects an elements
node.IsSelected = true;
// Unselect an element
node.IsSelected = false;
```



[View sample in GitHub](#)

#### Selection Indicator Style

Multiple Selection will show the preview for the selected items. We have provided customization option for the appearance of the Preview.

| [Style](#) | [Behavior](#) |

| --|--|

| [NodeSelectionIndicatorStyle](#) | Defines the customization option for Selection Preview for the Node. |

| [ConnectorSelectionIndicatorStyle](#) | Defines the customization option for Selection Preview for the Connector. |

| [FirstSelectionIndicatorStyle](#) | Defines the customization option for selection preview of first selected item. |

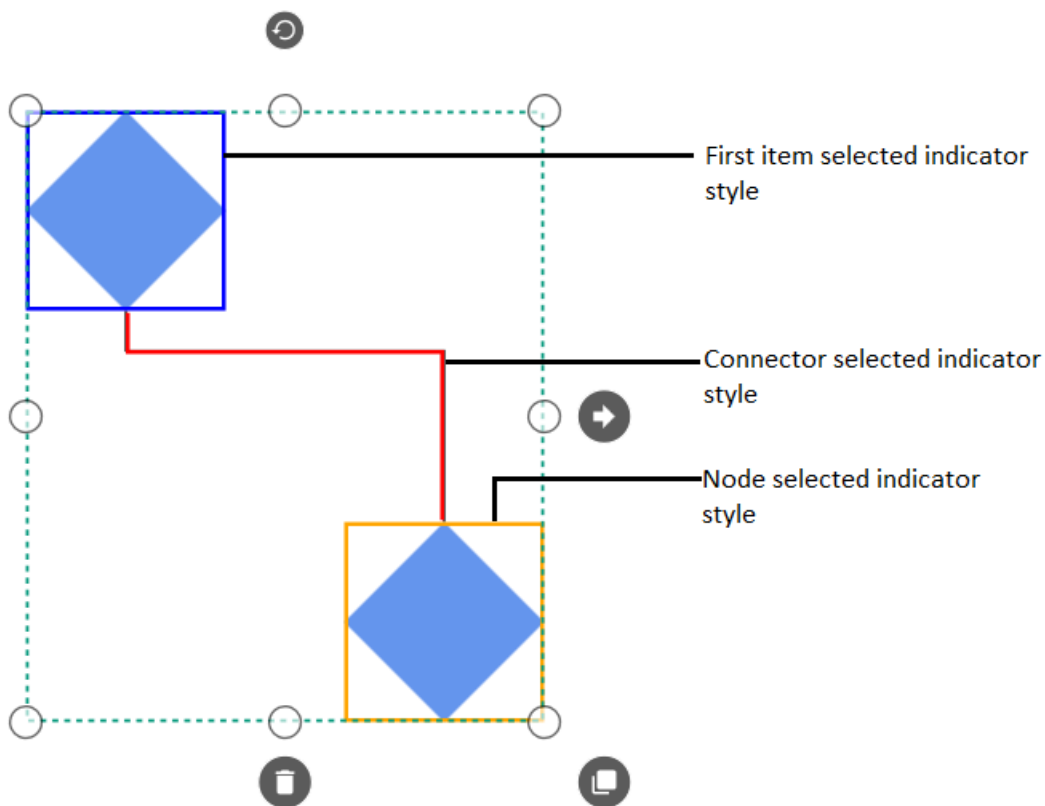
#### XML

```
<Style TargetType="Shape" x:Key="FirstSelectionindicatorstyle">
  <Setter Property="StrokeThickness" Value="2"/>
  <Setter Property="Stroke" Value="Orange"/>
</Style>
<Style TargetType="Shape" x:Key="NodeSelectionindicatorstyle">
  <Setter Property="StrokeThickness" Value="2"/>
  <Setter Property="Stroke" Value="Blue"/>
</Style>
```

```
<Style TargetType="Shape" x:Key="connectorselectionindicatorstyle">
  <Setter Property="StrokeThickness" Value="2"/>
  <Setter Property="Stroke" Value="Red"/>
</Style>
<Syncfusion:SfDiagram x:Name="Diagram"
  FirstSelectionIndicatorStyle="{StaticResource FirstSelectionindicatorstyle}"
  NodeSelectionIndicatorStyle="{StaticResource NodeSelectionindicatorstyle}"
  ConnectorSelectionIndicatorStyle="{StaticResource
  connectorselectionindicatorstyle}">
```

**C#**

```
SfDiagram Diagram = new SfDiagram();
Diagram.NodeSelectionIndicatorStyle =
this.Resources["NodeSelectionindicatorstyle"] as Style;
Diagram.FirstSelectionIndicatorStyle =
this.Resources["FirstSelectionindicatorstyle"] as Style;
Diagram.ConnectorSelectionIndicatorStyle =
this.Resources["connectorselectionindicatorstyle"] as Style;
```



[View Sample in GitHub](#)

#### Selector handle display mode

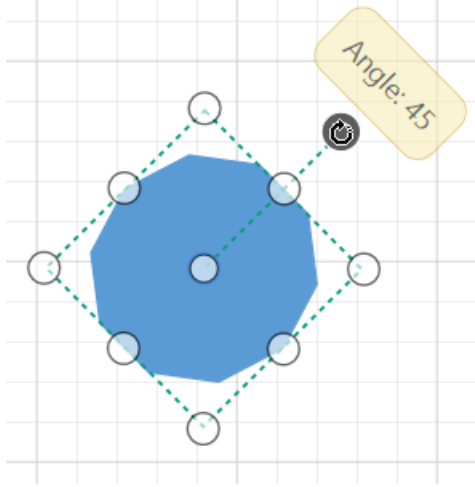
Diagram control provides support to change the selection handle display mode of the Node, Connector, and Group by using the `SelectorHandleDisplayMode` property.



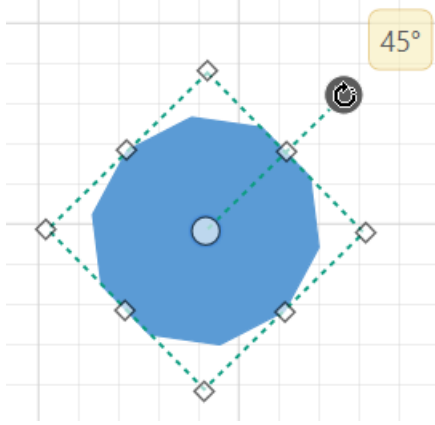
|SelectorHandleDisplayMode|Description| Output|

|--|--|--|

|Default| It is used to display selection handle display mode as larger size bubbles. |



|CompactSelector| It is used to display selection handle display mode as compact size rectangle. |



### Events

- [ItemSelectingEvent](#) and [ItemSelectedEvent](#) for selecting an element, will notify you the item and its original source. To explore about arguments ,please refer to [DiagramPreviewEventArgs](#) and [ItemSelectedEventArgs](#) .
- [ItemUnselectingEvent](#) and [ItemUnselectedEvent](#) for unselecting an element, will notify you the item and its original source.To explore about arguments ,please refer to [DiagramPreviewEventArgs](#) and [DiagramEventArgs](#) .

### See Also

- [How to customize the selection behavior of nodes and connectors?](#)
- [How to disable the selection in Diagram?](#)

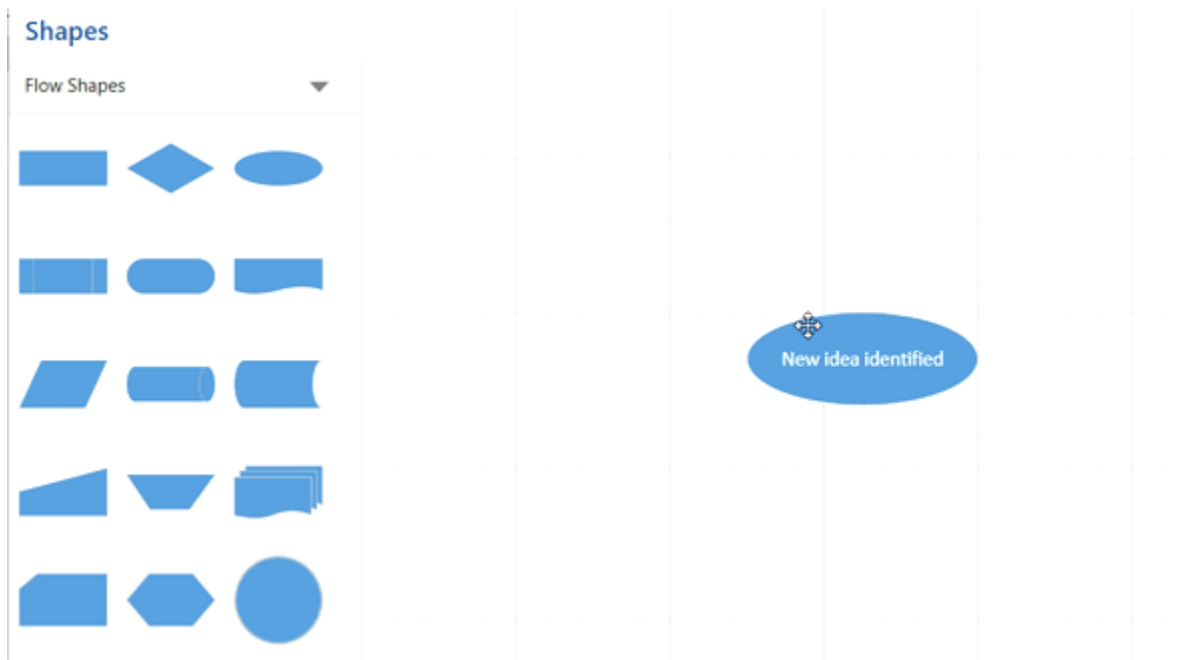
### Deletion in WPF Diagram (SfDiagram)

Selected objects can be deleted by using the Delete key or In-built Delete command through QuickCommand.

- [ItemDeletedEvent](#) will notify you with the deleted item in argument. To explore about arguments , please refer to [ItemDeletedEventArgs](#).
- [ItemDeletingEvent](#) will notify you with the item , option to cancel the deleting operation of item and option to decide on deleting dependent Connector when its Source/Target gets deleting. To explore about arguments , please refer to [ItemDeletingEventArgs](#).

### C#

```
(diagram.Info as IGraphInfo).ItemDeletingEvent +=
MainWindow_ItemDeletingEvent;
/// <summary>
/// DiagramPreviewEventArgs is the Base class for EventArgs.
/// Casting the args will help us to get ItemDeletingEventArgs.
/// </summary>
/// <param name="sender"></param>
/// <param name="args"></param>
private void MainWindow_ItemDeletingEvent(object sender,
DiagramPreviewEventArgs args)
{
    //For Deleting Node Without its Dependent Connector
    (args as ItemDeletingEventArgs).DeleteDependentConnector = false;
}
```

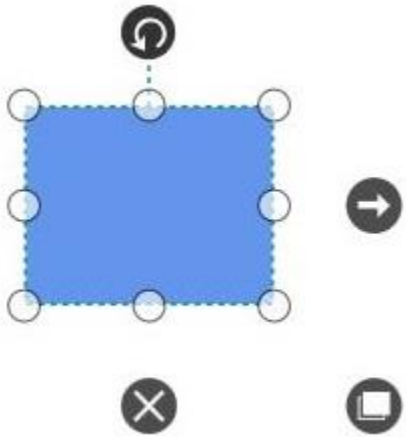


### Quick Command in WPF Diagram (SfDiagram)

#### Define QuickCommand

Quick Commands are used to execute the commonly or frequently used commands around the [Nodes](#), [Connectors](#) and [Groups](#). There are 3 default Quick Commands for Nodes and Groups to execute Draw,

Delete and Duplicate commands. For example, if you select the node then the quick command of the node will get visible.



#### Define Custom QuickCommand

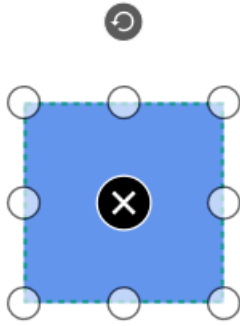
SfDiagram provides support to define custom quick command around the Nodes, Connectors and Groups.

#### XML

```
<Style TargetType="Path" x:Key="QuickCommandstyle">
  <Setter Property="Stretch" Value="Fill"/>
  <Setter Property="Fill" Value="Black"/>
  <Setter Property="Stroke" Value="White"/>
</Style>
```

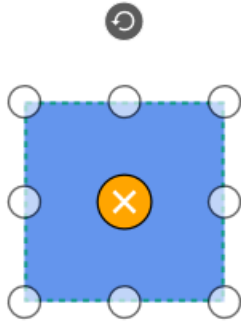
#### C#

```
// Element to represent the frequently used commands
QuickCommandViewModel quick = new QuickCommandViewModel()
{
    // Outer part of quick command.
    Shape = this.Resources["Ellipse"],
    // appearance of shape.
    ShapeStyle = this.Resources["QuickCommandstyle"] as Style,
    // Inner part of quick command and it allows to host any UI elements
    Content =
        "M3.7399902,0L16,12.258972 28.26001,0 32,3.7399902 19.73999,16 32,28.258972 "
        "28.26001,32 16,19.73999 3.7399902,32 0,28.258972 12.26001,16 0,3.7399902z",
    Command = (Diagram.Info as IGraphInfo).Commands.Cut
};
// Adding new QuickCommand object in Commands collection
(Diagram.SelectedItems as SelectorViewModel).Commands = new
QuickCommandCollection()
{
    quick
};
```



#### Customize quick command appearance

Appearance of the [QuickCommand](#) can be customized by using [Shape](#), [ShapeStyle](#), [Content](#) and [ContentTemplate](#) properties.



---

**Note:** By default QuickCommand will host on Node. [VisibilityMode](#) property is to define the host of the QuickCommand on either Node or Connector or both.

---

#### Alignment

QuickCommand can be aligned relative to boundaries of the Node or segments of the Connector.

- [OffsetX](#) and [OffsetY](#) property of QuickCommand is used to align the QuickCommand based on fractions. The default value is 0.5.
- [HorizontalAlignment](#) and [VerticalAlignment](#) properties are used to align the quick commands for horizontal and vertical positions.
- [Margin](#) is an absolute value used to add some blank space in any one of its four sides.

The Alignment of QuickCommand is similar to [Annotation Alignment](#).

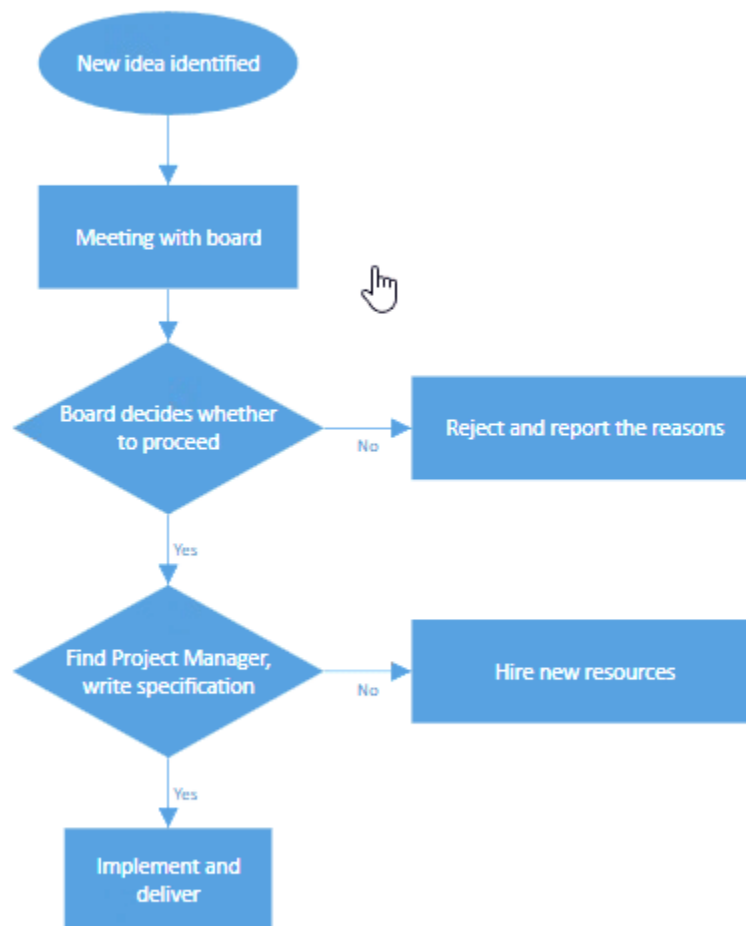
[View Sample in GitHub](#)

#### See Also

- [How to create a quick command in diagram?](#)
- [How to enable or disable QuickCommands?](#)

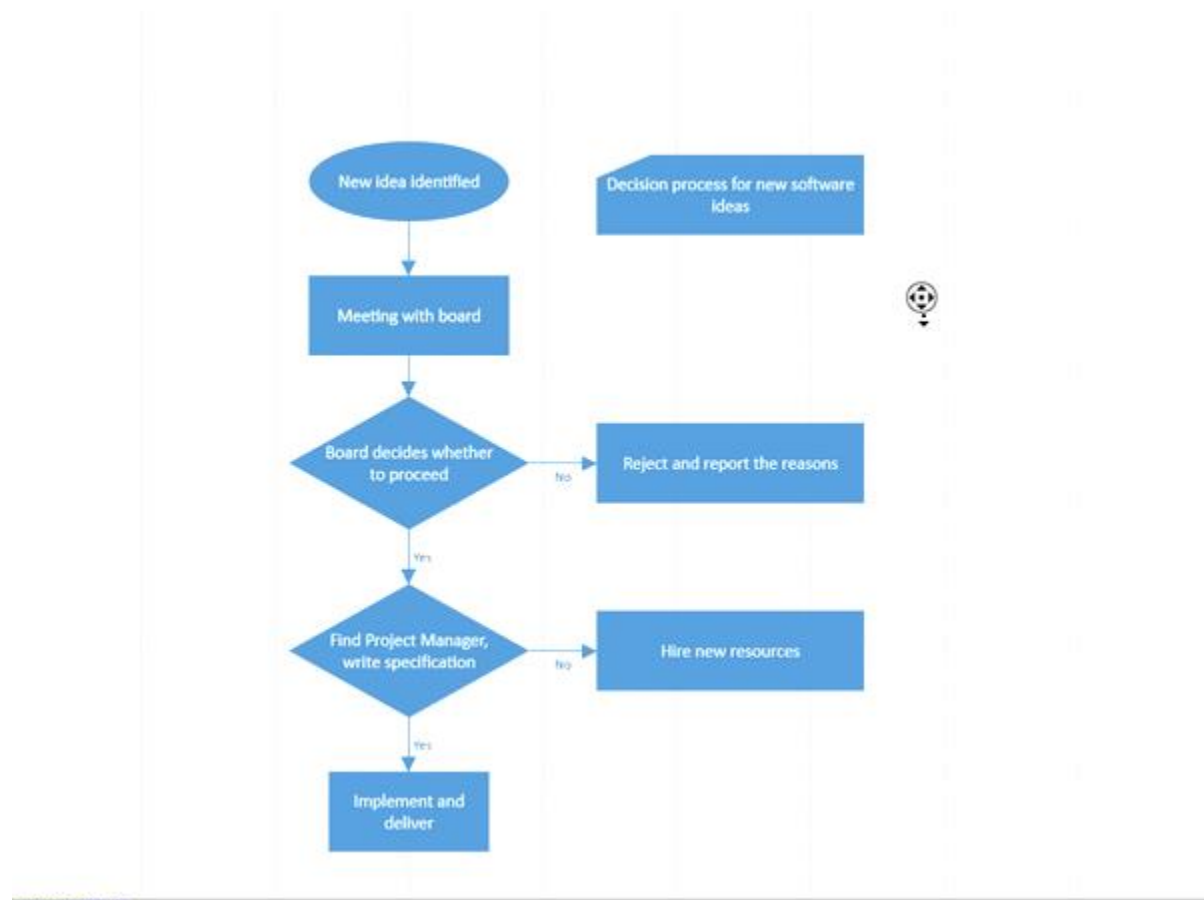
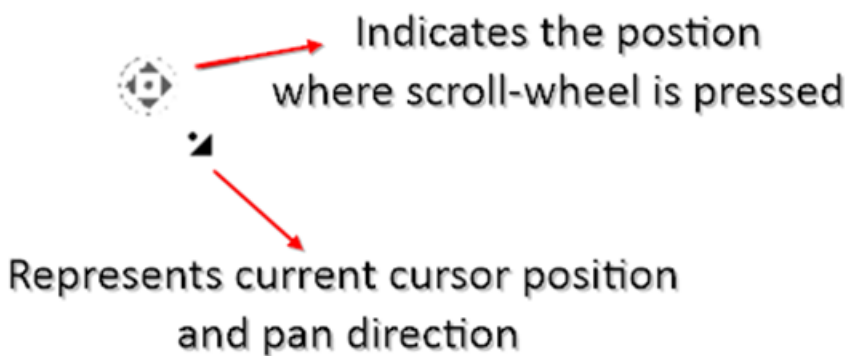
### Zoom Pan

- We can Zoom in or Zoom out the diagram view. Diagram can be zoomed in or out by using Ctrl + mouse wheel.
- 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.



- **IntelliMouse Pan** : We can pan the diagram using scroll wheel. To pan, user must hold down the Scroll Wheel anywhere in the diagram and move the cursor towards any direction (for example: On moving cursor towards left direction, diagram will be panned towards left). And releasing Scroll Wheel will stop the panning. Speed of the Panning is proportional to the

distance between the Scroll Wheel pressed position and current mouse position, i.e. panning will be faster if the distance is large and slower if the distance is small.



See Also

- [How to get notified when zooming and panning the diagram](#)

#### Keyboard support in WPF Diagram

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 list of Commands with key Gesture.

Shortcut Key	Command	Description
--- --- ---		
Ctrl + A	Select all	Select all the Nodes/Connectors in diagram.
Ctrl + C	Copy	Copy the selected elements in the diagram.
Ctrl + X	Cut	Cut the selected elements in the diagram.
Ctrl + V	Paste	Paste the cut or copy the elements in the diagram.
Ctrl + Z	Undo	Undo(Reverse the last editing action performed on diagram).
Ctrl + Y	Redo	Redo(Restores the last editing action when no other actions have occurred since the last undo on diagram).
Ctrl + D	Duplicate	Copies the selected element from the diagram and pastes the copied element into the diagram.
Delete	Delete	Delete the selected elements in the diagram.
LeftArrow	MoveLeft	MoveLeft (move the selected elements towards left by one pixel).
RightArrow	MoveRight	MoveRight (move the selected elements towards right by one pixel).
UpArrow	MoveUp	MoveUp (move the selected elements towards up by one pixel).
DownArrow	MoveDown	MoveDown (move the selected elements towards up by one pixel).
Ctrl + MouseScroll	Zoom	Zoom(Zoom in/Zoom out the diagram).
Ctrl + G	Group	Grouping the element in the diagram.
Ctrl + Shift + U	UnGroup	UnGrouping the element in the diagram.
Ctrl + Shift + B	SendToBack	Moves the selected element behind all the other overlapped elements.
Ctrl + [	SendBackward	Moves the selected element behind the underlying element.
Ctrl + Shift + F	BringToFront	Brings the selected element to front over all the other overlapped elements.
Ctrl + ]	BringForward	Moves the selected element over the nearest overlapping element.
Ctrl + +	Zoom	Zoom in the diagram.
Ctrl -	Zoom	Zoom out the diagram.
Ctrl + 0 (number 0)	Reset	To reset horizontal Offset, vertical Offset, and zoom level of the Diagram.
Ctrl + Shift + W	FitToPage	Fit the diagram content into the view with respect to either width, height, or at the whole.
F2	EditAnnotation	Enable annotation editing mode of the first selected diagram element.
Ctrl + 1	Pointer Tool	To select, move or resize the elements in the diagram.
Ctrl + 2	Text Tool	To draw a text node.
Ctrl + 3	Connector Tool	To draw a orthogonal connector.

| Ctrl + 5 | Freehand Tool | To draw a free hand connections. |

| Ctrl + 6 | Line Tool | To draw a straight line connector. |

| Ctrl + 8 | Rectangle Tool | To draw a rectangle shaped node. |

| Ctrl + 9 | Ellipse Tool | To draw an ellipse shaped node. |

| Esc | Cancel | Stops the annotation editing or clears the selection of a diagram element. |

To add custom commands, configure or modify key or mouse gesture through [Command Manager](#)  
[View sample in GitHub](#)

*See Also*

- [How to map the custom commands to existing gestures \(keyboard shortcuts and mouse\)?](#)

Dragging based on DragLimit in WPF Diagram (SfDiagram)

Diagram provides support to drag the elements within the given limitations using [EditableArea](#), [ScrollLimit.Limited](#) property and based on [SelectorChangedEvent](#) enabling/disabling of dragging within the limits occur.

In [SelectorChangedEvent](#) based on the arguments the process occurs,

- [Block](#)-If this boolean expression is set true, then the dragging occurs within the given rectangular area. If dragging exceeds than the limit, then it hit back to previous position.

Based on the [BlockPosition](#) the dragging of Block occurs.

- Block Position

[SourcePosition](#)- the element moves to the previous position if it exceeds the limitation during dragging.

[CurrentPosition](#)- the element present at the limited area position, it does not hit back to previous position during dragging.

- [Abort](#)- If this boolean is set to true, then dragging is occurs within the limit.
- [Cancel](#)- If this boolean is set to true, then the dragging of element does not occurs.

[Refer](#) for Scroll-Limit.

*Drag and Drop Nodes over other elements*

Diagram provides support to drop a [node](#) over another node or [connector](#). Drop event is raised to notify that an element is dropped over another one and it is disabled by default. It can enabled with the [AllowDrop](#) constraints property for both node and connector.

**C#**

```
//Enable AllowDrop Constraints for Node  
Node.Constraints |= NodeConstraints.AllowDrop;
```

Similarly, you can enable [AllowDrop](#) constraints for connector to drop a node over connector.



[View Sample in GitHub](#)
*Customize the appearance of Drop Indicator*

Drag and drop a node over another node or connector will show the preview for the target node or connector. We have provided customization option for the appearance of the Preview.

| [Style](#) | [Behavior](#) |

| --|--|

| [NodeDropIndicatorStyle](#) | Defines the customization option for preview for the Node. |

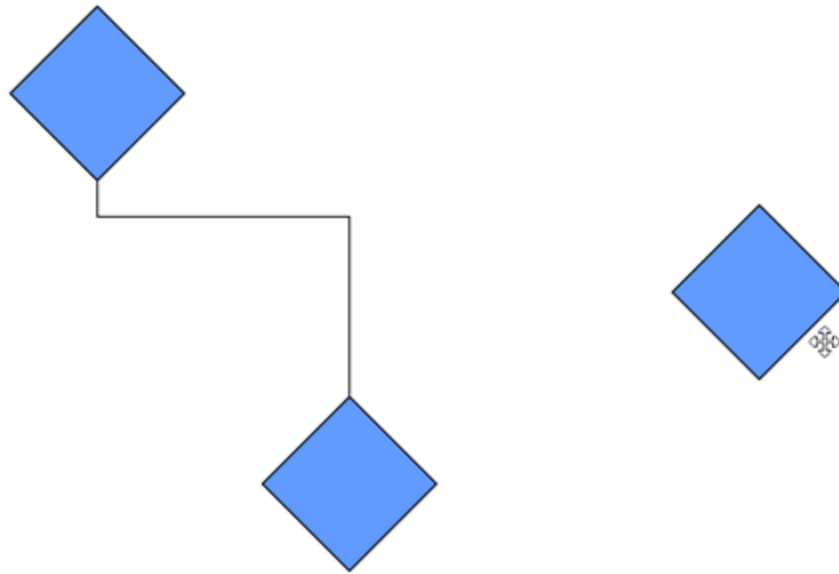
| [ConnectorDropIndicatorStyle](#) | Defines the customization option for preview for the Connector. |

**XML**

```
<<Style TargetType="Shape" x:Key="Nodedropindicator">
<Setter Property="StrokeThickness" Value="4"/>
<Setter Property="Stroke" Value="Orange"/>
</Style>
<Style TargetType="Shape" x:Key="connectordropindicator">
<Setter Property="StrokeThickness" Value="2"/>
<Setter Property="Stroke" Value="Blue"/>
</Style>
<Syncfusion:SfDiagram x:Name="Diagram"
NodeDropIndicatorStyle="{StaticResource Nodedropindicator}"
ConnectorDropIndicatorStyle="{StaticResource connectordropindicator}">
```

**C#**

```
SfDiagram Diagram = new SfDiagram();
Diagram.NodeDropIndicatorStyle = this.Resources["Nodedropindicator"] as
Style;
Diagram.ConnectorDropIndicatorStyle =
this.Resources["connectordropindicator"] as Style;
```



[View Sample in GitHub](#)

#### Events

- [ItemDropEvent](#), [DragEnter](#), [DragOver](#) and [DragLeave](#) events will notify you the Source and elements that are interacted with the dropped element(target).To explore about arguments, please refer to [ItemDropEventArgs](#).

#### See Also

- [How to drag and drop elements between diagrams](#)
- [How to restrict the diagram objects dragging in the positive side?](#)
- [How to drag and drop elements from treeview?](#)
- [How to create parent and child relationship by drag and drop nodes?](#)
- [How to restrict the child node dragging whereas allow group dragging?](#)

#### Serialization in WPF Diagram (SfDiagram)

Serialization is the process of converting the state of SfDiagram's objects into a stream of bytes to recreate them when needed. Such streams can be stored in a database,as a file or memory. The reverse process is called deserialization.

### Save

In SfDiagram, DataContractSerializer is used for serialization. The functionalities in DataContractSerializer are applicable to the SfDiagram serialization. It supports saving the SfDiagram into stream. The SfDiagram gets saved with all its properties.

### C#

```
//To Save as stream in file
SaveFileDialog dialog = new SaveFileDialog();
dialog.Title = "Save XAML";
dialog.Filter = "XAML File (*.xaml) | *.xaml";
if (dialog.ShowDialog() == true)
{
    using (Stream str = File.Open(dialog.FileName, FileMode.CreateNew))
    {
        sfDiagram.Save(str);
    }
}
//To Save as memory stream
MemoryStream str = new MemoryStream();
sfDiagram.Save(str);
```

### Load

On deserialization, the saved stream is used to load the SfDiagram's nodes and connectors in current view. With this, you can continue working on the earlier saved SfDiagram by loading the appropriate stream.

### C#

```
//Load from saved XAML file
OpenFileDialog dialog = new OpenFileDialog();
if (dialog.ShowDialog() == true)
{
    using (Stream myStream = dialog.OpenFile())
    {
        sfDiagram.Load(myStream);
    }
}
//Load from saved memory stream
str.Position = 0;
sfDiagram.Load(str);
```

### Has the diagram modified?

[HasChanges](#) property of diagram control is used to notify that the diagram has any unsaved changes. This property track all changes that are made through interaction and through the public APIs.

### XML

```
<!--Initialize the diagram-->
<Syncfusion:SfDiagram x:Name="diagram">
</Syncfusion:SfDiagram>
<!--Initialize the button to save the diagram-->
<Button x:Name="SaveButton" Content="Save" Click="SaveButton_Click">
</Button>
```

**C#**

```
//Method to promote the save dialouge box when diagram has any unsaved changes.
private void SaveButton_Click(object sender, RoutedEventArgs e)
{
    if (diagram != null && diagram.HasChanges)
    {
        MessageBoxResult messageBoxResult = MessageBox.Show(
            "Do you want to save the Diagram?",
            "SfDiagram",
            MessageBoxButton.YesNo);
        if (messageBoxResult == MessageBoxResult.Yes)
        {
            SaveDiagram();
        }
    }
}

//Method to save the diagram.
private void SaveDiagram()
{
    SaveFileDialog dialog = new SaveFileDialog();
    dialog.Title = "Save XAML";
    dialog.Filter = "XAML File (*.xaml) | *.xaml";
    if (dialog.ShowDialog() == true)
    {
        File.Delete(dialog.FileName);
        using (Stream s = File.Open(dialog.FileName, FileMode.OpenOrCreate))
        {
            diagram.Save(s);
        }
    }
}
```

**How to serialize custom properties**

In SfDiagram, you cannot serialize Content, ContentTemplate, Shape, and ShapeStyle of each and every diagramming objects. If you want to preserve the ShapeStyle and ContentTemplate of diagramming objects, keep them in resources and apply once the diagramming objects are added to the Diagram page.

The custom properties in custom class derived from any of our SfDiagram's interface or from any of the view model classes are serialized with the help of DataMember attribute.

**C#**

```
public class NodeContent : INode
{
    [DataMember]
    public string NodeType
    {
        get;
        set;
    }
}
```

---

**Note:** SfDiagram's interface and view model classes are created without DataContract attribute. So there is no need to add DataContract attribute for a class, which is derived from them.

---

#### How to serialize a custom class

You can serialize a business class with the help of DataContract attribute and SfDiagram's KnownTypes property. You have to add DataContract attribute to serialize the whole class, which is not derived from a base class without DataContract attribute.

#### C#

```
[DataContract]
public class NodeContent
{
    [DataMember]
    public string NodeType
    {
        get;
        set;
    }
}
Diagram.KnownTypes = () => new List<Type>()
{
    typeof(NodeContent)
};
```

#### [View sample in GitHub](#)

#### How to load SfDiagram's old version in new version

You can load any of the old version SfDiagram's stream in new version with the help of upgrade method. Refer to the following code example.

#### C#

```
using (Stream myStream = dialog.OpenFile())
{
    sfDiagram.Upgrade(myStream);
    sfDiagram.Load(myStream);
}
```

#### Virtualization in WPF Diagram (SfDiagram)

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 ScrollViewer are loaded and remaining objects will be loaded only when they come into view.

This feature gives optimized performance and low memory consumption while loading and dragging items to the SfDiagram that consists of large diagram objects.

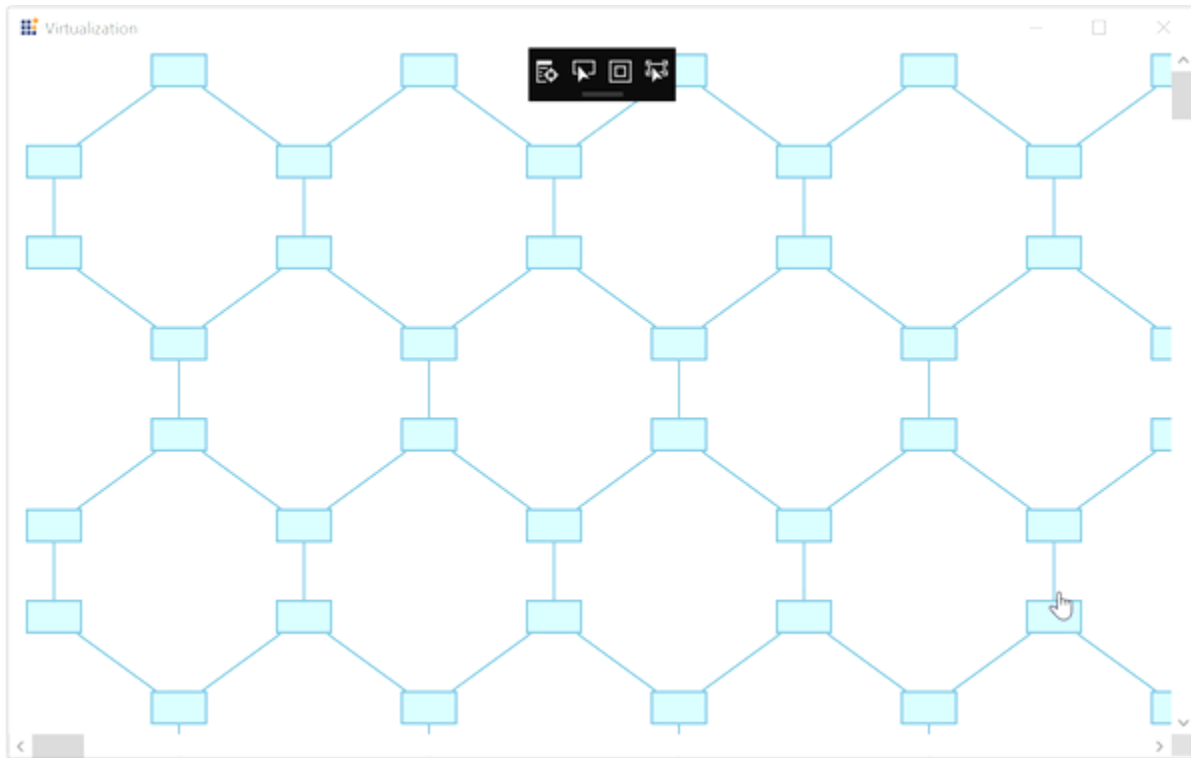
#### XML

```
<!--Initialize the SfDiagram and enable the virtualize constraint-->
<syncfusion:SfDiagram x:Name="diagram" Constraints="Default,Virtualize"/>
```

#### C#

```
//Initialize the SfDiagram
```

```
SfDiagram diagram = new SfDiagram();  
//Enable the Virtualize constraint  
diagram.Constraints = diagram.Constraints | GraphConstraints.Virtualize;
```



### Deferred Scrolling

To improve scrolling performance, the outline of a diagram element will be displayed until the UI element is loaded, regardless of the weight of the element.

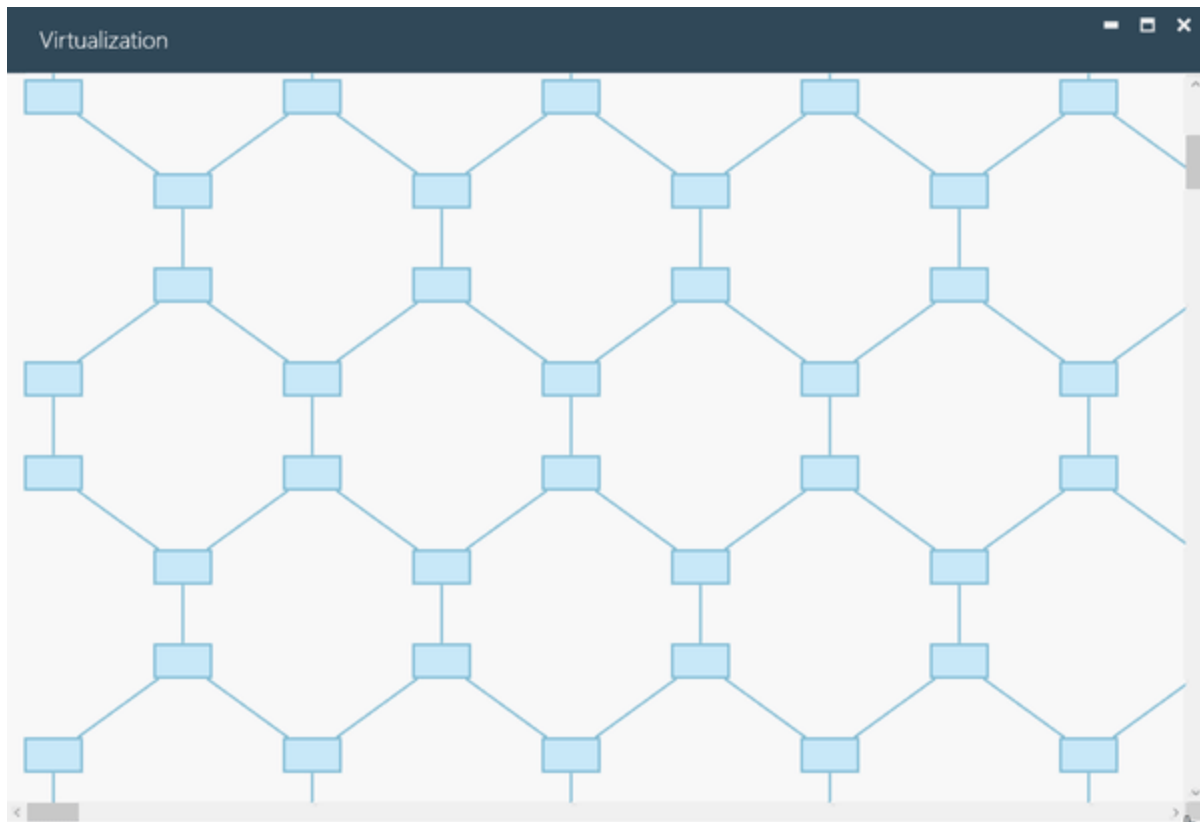
### XML

```
<!--Initialize the SfDiagram and enable the virtualize and outline  
constraint-->  
<syncfusion:SfDiagram x:Name="diagram"  
Constraints="Default,Virtualize,Outline"/>
```

### C#

```
//Initialize the SfDiagram  
SfDiagram diagram = new SfDiagram();  
//Enable the Virtualize and outline constraints  
diagram.Constraints |= GraphConstraints.Virtualize |  
GraphConstraints.Outline;
```

**Note:** In SfDiagram, Deferred Scrolling support is named as **Outline**. Outline is only applicable when virtualization is enabled.



### Outline customization

Options are provided to override the appearance, style, and interval time of outline by using the [OutlineSettings](#) class of diagram.

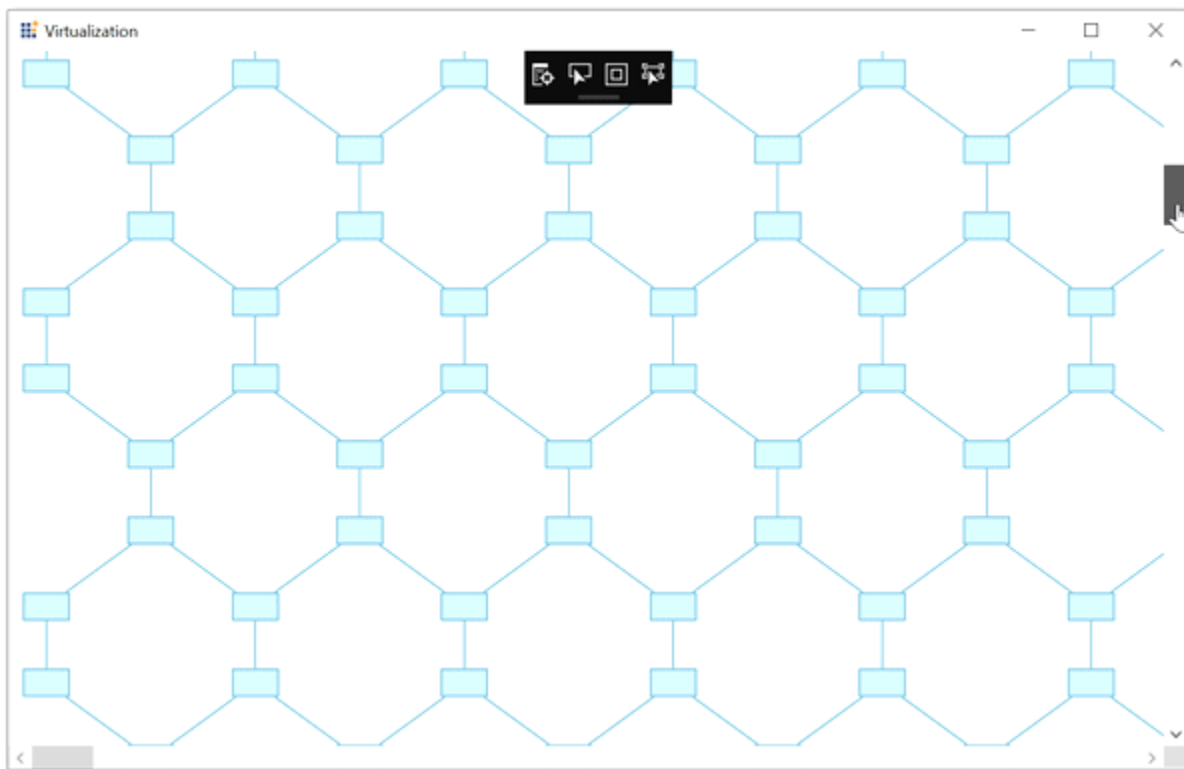
- **OutlineStyle**: Specifies the style for the outline of the diagram elements.
- **RenderInterval**: Specifies the time interval to render the diagram elements into view. Default time interval is 200ms.

### XML

```
<!--Custom style for outline of overview-->
<Style TargetType="Path" x:Key="outlineStyle">
  <Setter Property="Stroke" Value="Red"/>
  <Setter Property="StrokeThickness" Value="2"/>
</Style>
<!--Initialize outline setting with outline style and outline interval-->
<syncfusion:SfDiagram x:Name="diagram"
  Constraints="Default,Virtualize,Outline" >
  <syncfusion:SfDiagram.OutlineSettings>
    <syncfusion:OutlineSettings OutlineStyle="{StaticResource outlineStyle}">
    <syncfusion:OutlineSettings.RenderInterval>
      <sys:TimeSpan>0:0:0:20</sys:TimeSpan>
    </syncfusion:OutlineSettings.RenderInterval>
    </syncfusion:OutlineSettings>
  </syncfusion:SfDiagram.OutlineSettings>
</syncfusion:SfDiagram>
```

**C#**

```
//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Enable the outline and virtualize constraints
diagram.Constraints |= GraphConstraints.Virtualize |
GraphConstraints.Outline;
//Style for custom outline of overview
Style style = new Style(typeof(Path));
style.Setters.Add(new Setter(Shape.StrokeProperty, new
SolidColorBrush(Colors.Red)));
style.Setters.Add(new Setter(Shape.StrokeThicknessProperty, 2d));
//Initiaize the outline setting
diagram.OutlineSettings = new OutlineSettings()
{
    //Specifies the outline style
    OutlineStyle = style,
    //Specifies the outline rendering interval
    RenderInterval = new TimeSpan(0, 0, 0, 20),
};
```



Find the [Virtualization sample](#) to depict the Virtualization.

[How to serialize the diagram control](#)

[How to localize the diagram control](#)

[How to have overview for diagram control](#)



## Rulers in WPF Diagram (SfDiagram)

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. You can set the unit of measure, such as centimeters or inches. The default [Unit](#) of measure is pixels.

Please refer to the sample from Dashboard->Desktop->WPF->Diagram->GettingStarted->Rulers and Units.

### Define Rulers

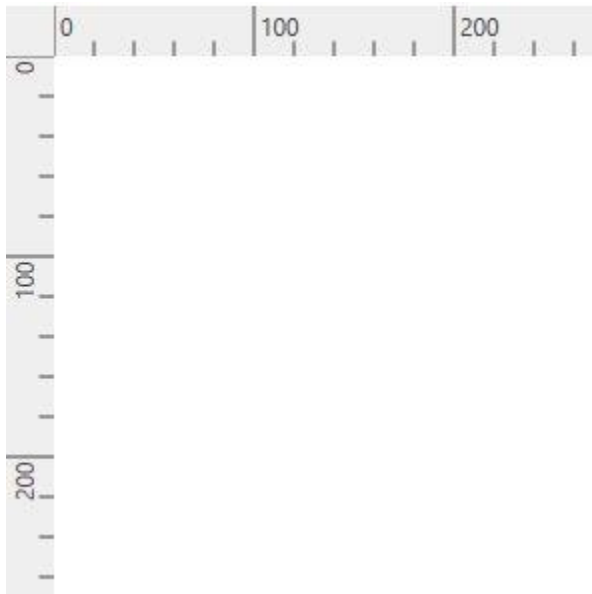
#### XML

```
xmlns:Syncfusion="clr-namespace:Syncfusion.UI.Xaml.Diagram;assembly=Syncfusion.SfDiagram.WPF"
xmlns:SyncControls="clr-namespace:Syncfusion.UI.Xaml.Diagram.Controls;assembly=Syncfusion.SfDiagram.WPF"

<syncfusion:SfDiagram x:Name="diagram">
  <syncfusion:SfDiagram.HorizontalRuler>
  <syncfusion:Ruler/>
</syncfusion:SfDiagram.HorizontalRuler>
  <syncfusion:SfDiagram.VerticalRuler>
  <syncfusion:Ruler Orientation="Vertical"/>
</syncfusion:SfDiagram.VerticalRuler>
</syncfusion:SfDiagram>
```

#### C#

```
diagram.HorizontalRuler = new Ruler();
diagram.VerticalRuler = new Ruler() { Orientation = Orientation.Vertical };
```



### Customizing the Ruler

By default, ruler segments are arranged based on the `MeasurementUnit`. See the available [LengthUnits](#) for Ruler.

Segment width, the textual description of the ruler segment, and the appearance of the ruler ticks can be customized.



Please refer to the sample to [Customize the Ruler](#).

### Commands

#### Alignment Commands in WPF Diagram (SfDiagram)

Alignment commands are used to align the selected objects such as Nodes and Connectors on a page with respect to a reference object(first item in the selection list).

##### *AlignLeft command*

The [AlignLeft](#) command is used to align all the selected objects along the left corner of the reference object.

##### **XML**

```
<Button Height="50" Content="AlignLeft" Name="AlignLeft"
Command="Syncfusion:DiagramCommands.AlignLeft"></Button>
```

##### **C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
// sets direction to Left
graphinfo.Commands.AlignLeft.Execute(null);
```

##### *AlignRight command*

The [AlignRight](#) command is used to align all the selected objects along the right corner of the reference object.

##### **XML**

```
<Button Height="50" Content="AlignRight" Name="AlignRight"
Command="Syncfusion:DiagramCommands.AlignRight"></Button>
```

**C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
// sets direction to Right  
graphinfo.Commands.AlignRight.Execute(null);
```

*AlignCenter command*

The [AlignCenter](#) command is used to center all selected objects vertically. It aligns selected objects to the center with respect to the horizontal axis by changing the x-coordinate of the object.

**XML**

```
<Button Height="50" Content="AlignCenter" Name="AlignCenter"  
Command="Syncfusion:DiagramCommands.AlignCenter"></Button>
```

**C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
// sets direction to Center vertically  
graphinfo.Commands.AlignCenter.Execute(null);
```

*AlignTop command*

The [AlignTop](#) command is used to align all the selected objects along the top corner of the reference object.

### XML

```
<Button Height="50" Content="AlignTop" Name="AlignTop"  
Command="Syncfusion:DiagramCommands.AlignTop"></Button>
```

### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
// sets direction to Top  
graphinfo.Commands.AlignTop.Execute(null);
```

#### *AlignBottom command*

The [AlignBottom](#) command is used to align all the selected objects along the bottom corner of the reference object.

### XML

```
<Button Height="50" Content="AlignBottom" Name="AlignBottom"  
Command="Syncfusion:DiagramCommands.AlignBottom"></Button>
```

### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
// sets direction as Bottom  
graphinfo.Commands.AlignBottom.Execute(null);
```

#### *AlignMiddle command*

The [AlignMiddle](#) command is used to center all selected objects horizontally. It aligns selected objects to the center with respect to the vertical axis by changing the y-coordinate of the object.

### XML

```
<Button Height="50" Content="AlignMiddle" Name="AlignMiddle"  
Command="Syncfusion:DiagramCommands.AlignMiddle"></Button>
```

### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
// sets direction to Center horizontally  
graphinfo.Commands.AlignCenter.Execute(null);
```



[View sample in GitHub](#)

### Spacing Commands in WPF Diagram (SfDiagram)

Spacing commands are used to place selected objects on the page at equal intervals from each other. The objects are spaced within the bounds of the first and last objects in the selection.

#### *SpaceAcross command*

The [SpaceAcross](#) command is used to place selected objects on the page at equal intervals from each other horizontally.

#### **XML**

```
<Button Height="50" Content="SpaceAcross" Name="SpaceAcross"  
Command="Syncfusion:DiagramCommands.SpaceAcross"></Button>
```

#### **C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
// Equally spaces the selected nodes horizontally  
graphinfo.Commands.SpaceAcross.Execute(null);
```



#### *SpaceDown command*

The [SpaceDown](#) command is used to place selected objects on the page at equal intervals from each other vertically.

#### **XML**

```
<Button Height="50" Content="SpaceDown" Name="SpaceDown"  
Command="Syncfusion:DiagramCommands.SpaceDown"></Button>
```

#### **C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
// Equally spaces the selected nodes vertically  
graphinfo.Commands.SpaceDown.Execute(null);
```



[View sample in GitHub](#)

### Sizing Commands in WPF Diagram (SfDiagram)

Sizing commands are used to resize all selected object based on width, height, and size of the reference object (FirstSelectedItem).

#### *SameSize command*

The [SameSize](#) command is used to resize all the selected object based on the size of the first item in the selection list.

#### **XML**

```
<Button Height="50" Content="SameSize" Name="SameSize"  
Command="Syncfusion:DiagramCommands.SameSize"></Button>
```

#### **C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
//Scales the selected items to the size of first selected object  
graphinfo.Commands.SameSize.Execute(null);
```

#### *SameHeight command*

The [SameHeight](#) command is used to resize all the selected object based on the height of the first item in the selection list.

#### **XML**

```
<Button Height="50" Content="SameHeight" Name="SameHeight"  
Command="Syncfusion:DiagramCommands.SameHeight"></Button>
```

#### **C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
//Vertically scales the selected items to the height of first selected  
object  
graphinfo.Commands.SameHeight.Execute(null);
```

#### *SameWidth command*

The [SameWidth](#) command is used to resize all the selected object based on the width of the first item in the selection list.

#### **XML**

```
<Button Height="50" Content="SameWidth" Name="SameWidth"  
Command="Syncfusion:DiagramCommands.SameWidth"></Button>
```

#### **C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
//Horizontally scales the selected items to the width of first selected  
object  
graphinfo.Commands.SameWidth.Execute(null);
```



[View sample in GitHub](#)

#### Clipboard Commands in WPF Diagram (SfDiagram)

Clipboard commands are used to cut or copy the selected diagram objects to the clipboard and paste the valid clipboard content into the diagram page.



### Cut command

The [Cut](#) command is used to cut the selected diagram objects to the clipboard. Cut command can be executed by the keyboard shortcut CTRL + X.

#### XML

```
<Button Height="50" Content="Cut" Name="Cut"  
Command="Syncfusion:DiagramCommands.Cut"></Button>
```

#### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
//Cuts the selected elements from the diagram to the diagram's clipboard  
graphinfo.Commands.Cut.Execute (null);
```

### Copy command

The [Copy](#) command is used to copy the selected diagram objects to the clipboard. Copy command can be executed by the keyboard shortcut CTRL + C.

#### XML

```
<Button Height="50" Content="Copy" Name="Copy"  
Command="Syncfusion:DiagramCommands.Copy"></Button>
```

#### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
//Copies the selected elements from the diagram to the diagram's clipboard.  
graphinfo.Commands.Copy.Execute (null);
```

### Paste command

The [Paste](#) command is used to paste the clipboard content to the diagram page. Paste command can be executed by the keyboard shortcut CTRL + V.

#### XML

```
<Button Height="50" Content="Paste" Name="Paste"  
Command="Syncfusion:DiagramCommands.Paste"></Button>
```

#### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
//Pastes the diagram's clipboard data (nodes or connectors) into the  
Diagram.  
graphinfo.Commands.Paste.Execute (null);
```

### Duplicate command

The [Duplicate](#) command is used to copy the selected diagram objects to the clipboard and paste the clipboard content to the diagram page. Duplicate command can be executed by the keyboard shortcut CTRL + D.

#### XML

```
<Button Height="50" Content="Duplicate" Name="Duplicate"  
Command="Syncfusion:DiagramCommands.Duplicate"></Button>
```

### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
//Copy the diagram objects and Paste the clipboard's data (nodes or  
connectors) into the Diagram.  
graphinfo.Commands.Duplicate.Execute(null);
```



[View sample in GitHub](#)

### Grouping Commands in WPF Diagram (SfDiagram)

The Grouping commands are used to group or ungroup the selected diagramming objects such as node and connector in the diagram.

#### Group command

The [Group](#) command is used to group the selected diagramming objects in the diagram page. The Group command can be executed by the keyboard shortcut CTRL + G.

### XML

```
<Button Height="50" Content="Cancel" Name="Cancel"  
Command="Syncfusion:DiagramCommands.Group"></Button>
```

### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
//Groups the selected elements.  
graphinfo.Commands.Group.Execute(null);
```

### *UnGroup command*

The [UnGroup](#) command is used to ungroup the selected group diagramming objects in the diagram page. UnGroup command can be executed by the keyboard shortcut CTRL + G.

#### **XML**

```
<Button Height="50" Content="Cancel" Name="Cancel"
Command="Syncfusion:DiagramCommands.UnGroup"></Button>
```

#### **C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
//UnGroups the selected group elements.
graphinfo.Commands.UnGroup.Execute(null);
```



### [View sample in GitHub](#)

### Flip Command in WPF Diagram (SfDiagram)

The [Flip](#) command is used to mirror the selected object's content and port in the diagram page in both horizontal and vertical direction.

#### **XML**

```
<Button Height="50" Content="Flip" Name="Flip"
Command="Syncfusion:DiagramCommands.Flip"></Button>
```

#### **C#**

```
IGraphInfo graphinfo = Diagram.Info as IGraphInfo;
// Apply flip to selected objects.
graphinfo.Commands.Flip.Execute(null);
```

### *Flip parameter*

The [Flip parameter](#) is used to customize the flip mode and flip direction. If the parameter is null, then the object will be flipped both horizontally and vertically.

### Flip mode

The [FlipMode](#) is used to control the behaviour of the flip object.

FlipMode	Description
---	---
Content	It is used to enable or disables the flip for object's content.
Port	It is used to enable or disables the flip for object's port.
FlipMode	It is used to enable or disables the flip for both object's content and port.
None	It is used to disables all the flipmode behaviour.

### Flip

The [Flip](#) is used to specify the flip direction in flip command.

Flip	Description
---	---
Flip	It is used to flip the node or port is mirrored across the both horizontal and vertical axis.
HorizontalFlip	It is used to flip the node or port is mirrored across the horizontal axis.
VerticalFlip	It is used to flip the node or port is mirrored across the vertical axis.
None	It is used to disables all the flip behaviour.



### [View sample in GitHub](#)

### Z-Order Commands in WPF Diagram (SfDiagram)

Z – Order commands are used to visually arrange the selected objects such as Nodes and Connectors on the diagram page with its Z-order values.

### *BringToFront*

The [BringToFront](#) command is used to visually brings the selected element to the front over all other overlapped elements.

### **XML**

```
<Button Height="50" Content="BringToFront" Name="BringToFront"
Command="Syncfusion:DiagramCommands.BringToFront"></Button>
```

**C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
//Brings to front
graphinfo.Commands.BringToFront.Execute(null);
```

*SendToBack*

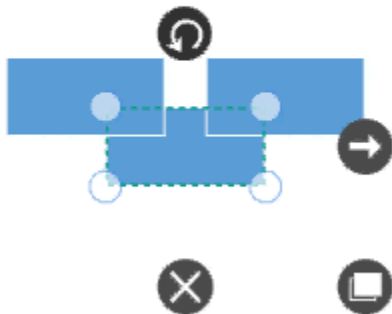
The [SendToBack](#) command visually moves the selected elements behind all the other overlapped elements.

**XML**

```
<Button Height="50" Content="SendToBack" Name="SendToBack"
Command="Syncfusion:DiagramCommands.SendToBack"></Button>
```

**C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
// Send To Back
graphinfo.Commands.SendToBack.Execute(null);
```

*SendBackward*

The [SendBackward](#) command visually moves the selected elements behind the underlying element.

**XML**

```
<Button Height="50" Content="SendBackward" Name="SendBackward"
Command="Syncfusion:DiagramCommands.SendBackward"></Button>
```

**C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
// Send To Backward
```

```
graphinfo.Commands.SendBackward.Execute(null);
```

### BringForward

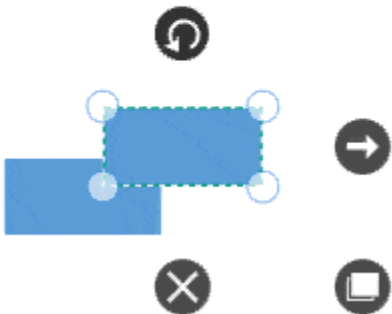
The [BringForward](#) command visually moves the selected element over the nearest overlapping element.

### XML

```
<Button Height="50" Content="BringForward" Name="BringForward"  
Command="Syncfusion:DiagramCommands.BringForward"></Button>
```

### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
// Brings To Forward  
graphinfo.Commands.BringForward.Execute(null);
```



[View sample in GitHub](#)

### Zoom Command in WPF Diagram (SfDiagram)

The [Zoom](#) commands are used to do zoom-in and zoom-out operations on the Diagram view. This command is also used to do scroll and pan operations with its parameter.

To execute zoom commands, IZoomParameter type [IZoomPositionParameter](#) have to be passed.

#### Zoom position parameter

The [ZoomPositionParameter](#) is used to represent the position parameters to for executing zoom command. Please find its properties and their description as follows.

Property name	Description
---	---
<a href="#">FocusPoint</a>	It is used to set the point of focus when zooming. Usually used to specify a particular point in the diagram view.
<a href="#">PanDelta</a>	It is used to set the finite increament in the pan value.
<a href="#">ScrollDelta</a>	It is used to set the finite increament in the scroll value.

| [ZoomCommand](#) | It is used to set the operation to be performed. |

| [ZoomFactor](#) | It is used to set the percentage of scale value for each ZoomIn or ZoomOut functionalities. |

| [ZoomTo](#) | It is used to set the zoom to a particular value. |

For ZoomIn operation

#### XML

```
<Syncfusion:ZoomPositionParameter ZoomCommand="ZoomIn" ZoomFactor="0.2"
x:Key="ZoomInParameter"/>
<Button Height="50" Content="ZoomIn" Name="ZoomIn"
Command="Syncfusion:DiagramCommands.Zoom" CommandParameter="{StaticResource
ZoomInParameter}"></Button>
```

#### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
// For ZoomIn function
graphinfo.Commands.Zoom.Execute(new ZoomPositionParameter()
{
    ZoomCommand = ZoomCommand.ZoomIn,
    ZoomFactor = 0.2,
});
```

For ZoomOut operation

#### XML

```
<Syncfusion:ZoomPositionParameter ZoomCommand="ZoomOut" ZoomFactor="0.2"
x:Key="ZoomOutParameter"/>
<Button Height="50" Content="ZoomOut" Name="ZoomOut"
Command="Syncfusion:DiagramCommands.Zoom" CommandParameter="{StaticResource
ZoomOutParameter}"></Button>
```

#### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
// For ZoomOut function
graphinfo.Commands.Zoom.Execute(new ZoomPositionParameter()
{
    ZoomCommand = ZoomCommand.ZoomOut,
    ZoomFactor = 0.2,
});
```



For ZoomTo specific value

#### XML

```
<Syncfusion:ZoomPositionParameter ZoomCommand="Zoom" ZoomTo="2"
x:Key="ZoomToParameter"/>
<Button Height="50" Content="ZoomTo" Name="ZoomTo"
Command="Syncfusion:DiagramCommands.Zoom" CommandParameter="{StaticResource
ZoomToParameter}"></Button>
```

#### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
// For ZoomTo specific value
graphinfo.Commands.Zoom.Execute(new ZoomPositionParameter()
{
    ZoomCommand = ZoomCommand.Zoom,
    ZoomTo = 2,
});
```





For Panning

### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
// For horizontal scroll  
graphinfo.Commands.Zoom.Execute(new ZoomPositionParameter()  
{  
    ZoomCommand = ZoomCommand.HorizontalScroll,  
    ScrollDelta = 50,  
});
```

For HorizontalScroll

### XML

```
<Syncfusion:ZoomPositionParameter ZoomCommand="HorizontalScroll"  
    ScrollDelta="5" x:Key="HorizontalScrollParameter"/>  
<Button Height="50" Content="HorizontalScroll" Name="HorizontalScroll"  
    Command="Syncfusion:DiagramCommands.Zoom" CommandParameter="{StaticResource  
    HorizontalScrollParameter}"></Button>
```

### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
// For horizontal scroll  
graphinfo.Commands.Zoom.Execute(new ZoomPositionParameter()  
{  
    ZoomCommand = ZoomCommand.HorizontalScroll,
```

```
ScrollDelta = 50,  
});
```

For VerticalScroll

#### XML

```
<Syncfusion:ZoomPositionParameter ZoomCommand="VerticalScroll"  
ScrollDelta="50" x:Key="ZoomPositionCommandParameter"/>  
<Button Height="50" Content="VerticalScroll" Name="VerticalScroll"  
Command="Syncfusion:DiagramCommands.Zoom" CommandParameter="{StaticResource  
ZoomPositionCommandParameter}"></Button>
```

#### C#

```
IGraphInfo graphinfo = Diagram.Info as IGraphInfo;  
// For vertical scroll  
graphinfo.Commands.Zoom.Execute(new ZoomPositionParameter()  
{  
ZoomCommand = ZoomCommand.VerticalScroll,  
ScrollDelta = 50,  
});
```



#### Reset

The [Reset](#) command is used to reset horizontal Offset, vertical Offset, and zoom level of the Diagram. If you want to customize the Reset command, you can use the [IReset](#) as parameter.

### ResetParameter

The [Reset](#) parameter is used to define the behavior of the Reset Command.

Reset Enum Values	Description
---	---
None	It is used to disables all the behaviors of the Reset Command.
Pan	It is used to specifies to reset the panned diagram.
Zoom	It is used to specifies to reset the zoomed diagram.
ZoomPan	It is used to Specifies to reset the zoomed and panned diagram.

### XML

```
<Syncfusion:ResetParameter Reset="ZoomPan"
x:Key="ResetParameterCommandParameter"/>
<Button Height="50" Content="Reset" Name="Reset"
Command="Syncfusion:DiagramCommands.Reset" CommandParameter="{StaticResource
ResetParameterCommandParameter}"></Button>
```

### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
// Reset the Zoom level of the Diagram
graphinfo.Commands.Reset.Execute(new ResetParameter() { Reset =
Reset.ZoomPan });
```

### [View sample in GitHub](#)

### Undo Redo Commands in WPF Diagram (SfDiagram)

The [Undo](#) command reverses the last editing action performed. For example, some of the basic operations performed on diagram objects such as translation, rotation, resizing, grouping, ungrouping, changing z-order, addition, deletion, and so on, can be reversed. The [Redo](#) command restores the last editing action if no other actions have occurred since the last undo.

Undo and Redo actions are disabled by default, to enable this you can use the **Constraints** property of the SfDiagram. Please refer to the [GraphConstraints](#)

### C#

### XML

```
<Syncfusion:SfDiagram x:Name="diagramcontrol"
Constraints="Default,Undoable"/>
<Button Height="50" Content="Undo" Name="Undo"
Command="Syncfusion:DiagramCommands.Undo"></Button>
<Button Height="50" Content="Redo" Name="Redo"
Command="Syncfusion:DiagramCommands.Redo"></Button>
```

{% endhighlight %}

### C#

```
// To enable the Undo and Redo action
diagramcontrol.Constraints |= GraphConstraints.Undoable;
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
// To perform the Undo action in Diagram
graphinfo.Commands.Undo.Execute(null);
// To Perform the Redo action in Diagram
graphinfo.Commands.Redo.Execute(null);
```



[View sample in GitHub](#)

### Nudge Commands in WPF Diagram (SfDiagram)

Nudge commands are used to move the selected elements towards up, down, left, or right by 1 pixel. The [IMoveParameter](#) is provided to customize the movement of the selected objects. The Nudge Commands as follows.

Commands	Description
NudgeUp	The <a href="#">NudgeUp</a> command moves the selected object towards the top by 1 pixel.
NudgeDown	The <a href="#">NudgeDown</a> command moves the selected object towards the bottom by 1 pixel.
NudgeLeft	The <a href="#">NudgeLeft</a> command moves the selected object towards the left by 1 pixel.
NudgeRight	The <a href="#">NudgeRight</a> command moves the selected object towards the right by 1 pixel.

### XML

```
<Syncfusion:MoveParameter MoveDelta="5" x:Key="MoveupCommandParameter"/>
<Button Height="50" Content="MoveUp" Name="MoveUp"
Command="Syncfusion:DiagramCommands.MoveUp"
CommandParameter="{StaticResource MoveupCommandParameter}"></Button>
```

### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
// Nudge up the selected objects
graphinfo.Commands.MoveUp.Execute(new MoveParameter() { MoveDelta = 5 });
```



[View sample in GitHub](#)

FitToPage Command in WPF Diagram (SfDiagram)

The [FitToPage](#) commands are used to bring the entire Diagram into the view. The [IFitToPage parameter](#) is used to customize the FitToPage command behavior.

If the parameter is null, entire diagram is fit into the view.

#### XML

```
<Button Height="50" Content="FitToPage" Name="FitToPage"
Command="Syncfusion:DiagramCommands.FitToPage"></Button>
```

#### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
// To fit the Diagram into the view
graphinfo.Commands.FitToPage.Execute(null);
```

*FitToPageParameter*

The [IFitToPage parameter](#) is used to customize the FitToPage command behavior.

*CanZoomIn*

The [CanZoomIn](#) is used to set whether small diagram gets zoom in to whole view or not.



FitToPage

The [FitToPage](#) is used to enable or disable the fit to page behavior with respect to height or width.

Values	Description
---	---
None	It is used disable the fit to page behavior.
FitToHeight	It is used to enable the fit to page behavior only with respect to height.
FitToWidth	It is used to enable the fit to page behavior only with respect to width.
FitToPage	It is used to enable the fit to page behavior with respect to both height and width of the diagram.



Region

The [Region](#) is used to set the region where to perform fittopage in diagram.

| Values | Description |

| --- | --- |

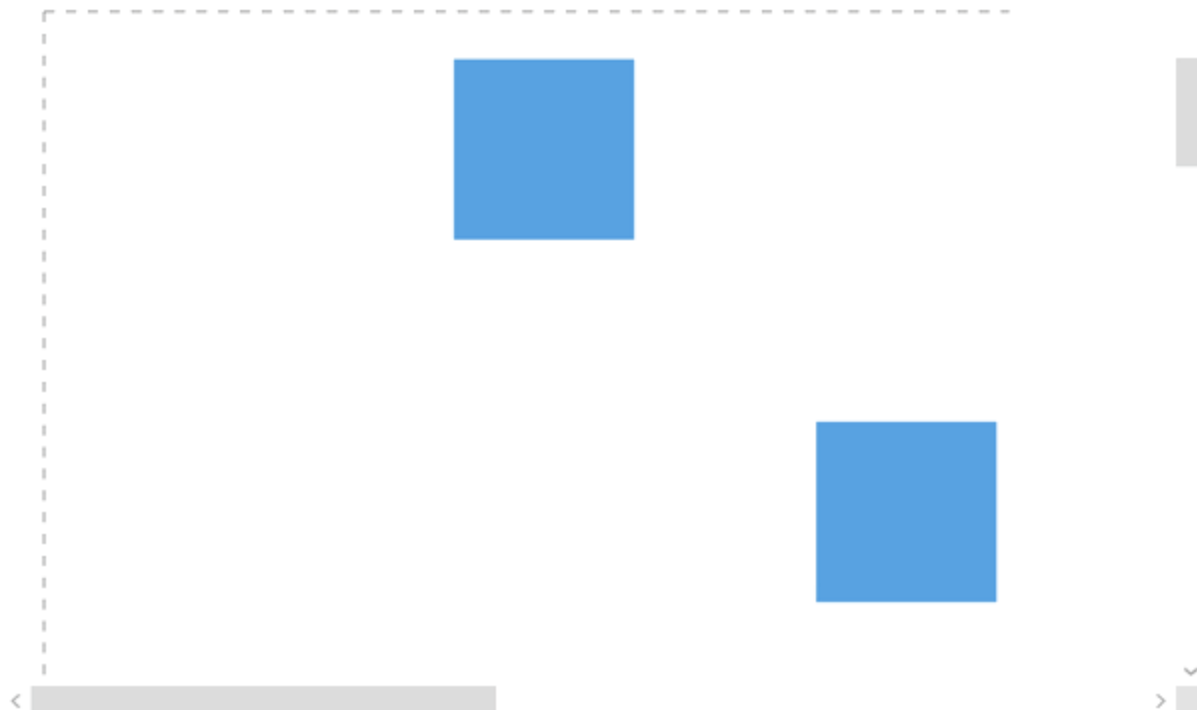
| Content | It is used to perform fit to page for the content area only. |

| PageSettings | It is used to perform fit to page based on the page width and page height. |

| Custom | It is used to perform fit to page for custom region. |

FocusArea

The [FocusArea](#) is used to set the focus area to execute the `FitToPage` command in custom region.



[View sample in GitHub](#)

#### Expand Collapse Command in WPF Diagram (SfDiagram)

The [ExpandCollapse](#) command is used to show or hide children and view only the relevant nodes in the diagram. The ExpandCollapse command will be executed with the [ExpandCollapseParameter](#) where the parameter contains the information about node that need to be expanded or collapsed. The IsExpanded property of node is used to expand or collapse the children nodes.

#### [ExpandCollapseParameter](#)

The [ExpandCollapseParameter](#) is used to compress a view of a hierarchy so that only the roots of each elements are visible. The opposite of collapse is expand that makes the entire elements visible.

#### [Node](#)

The [Node](#) is used to set the node that is to be act as root element.

#### [IsUpdateLayout](#)

The [IsUpdateLayout](#) is used to set whether the layout to be updated or not after the command execution.

#### **C#**

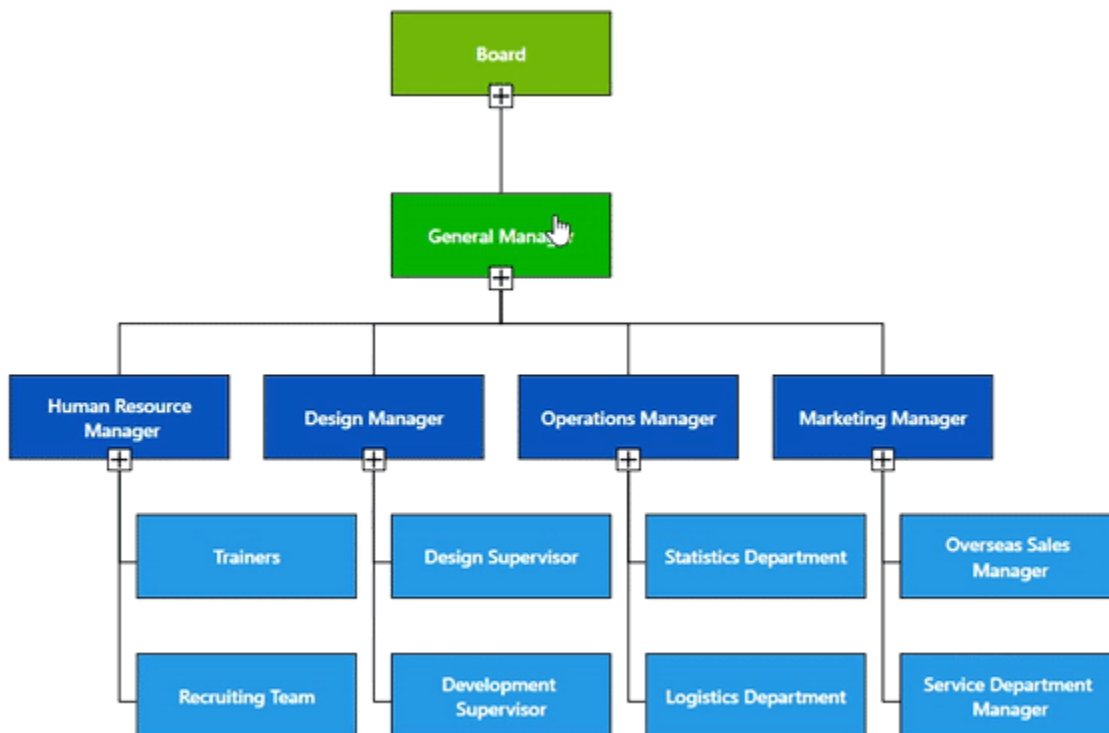
```
if (args.Item is NodeViewModel && (args.Item as NodeViewModel).IsExpanded)
{
    ExpandCollapseParameter param = new ExpandCollapseParameter()
    {
        IsUpdateLayout = true,
        // Set the node that is going to be show or hide its children
        node = args.Item as NodeViewModel,
    };
    IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
    //Show or hide the children of node.
```



```

graphinfo.Commands.ExpandCollapse.Execute(param);
(args.Item as NodeViewModel).IsExpanded = false;
}
else
{
ExpandCollapseParameter param = new ExpandCollapseParameter()
{
IsUpdateLayout = true,
// Set the node that is going to be show or hide its children
node = args.Item as NodeViewModel,
};
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
//Show or hide the children of node.
graphinfo.Commands.ExpandCollapse.Execute(param);
(args.Item as NodeViewModel).IsExpanded = true;
}

```



[View sample in GitHub](#)

Rotate Command in WPF Diagram (SfDiagram)

The [Rotate](#) Command is used to rotate the elements in diagram. The [RotateParameter](#) is used to represent [RotationDirection](#) and [Angle](#) to rotate the element in diagram.

#### XML

```

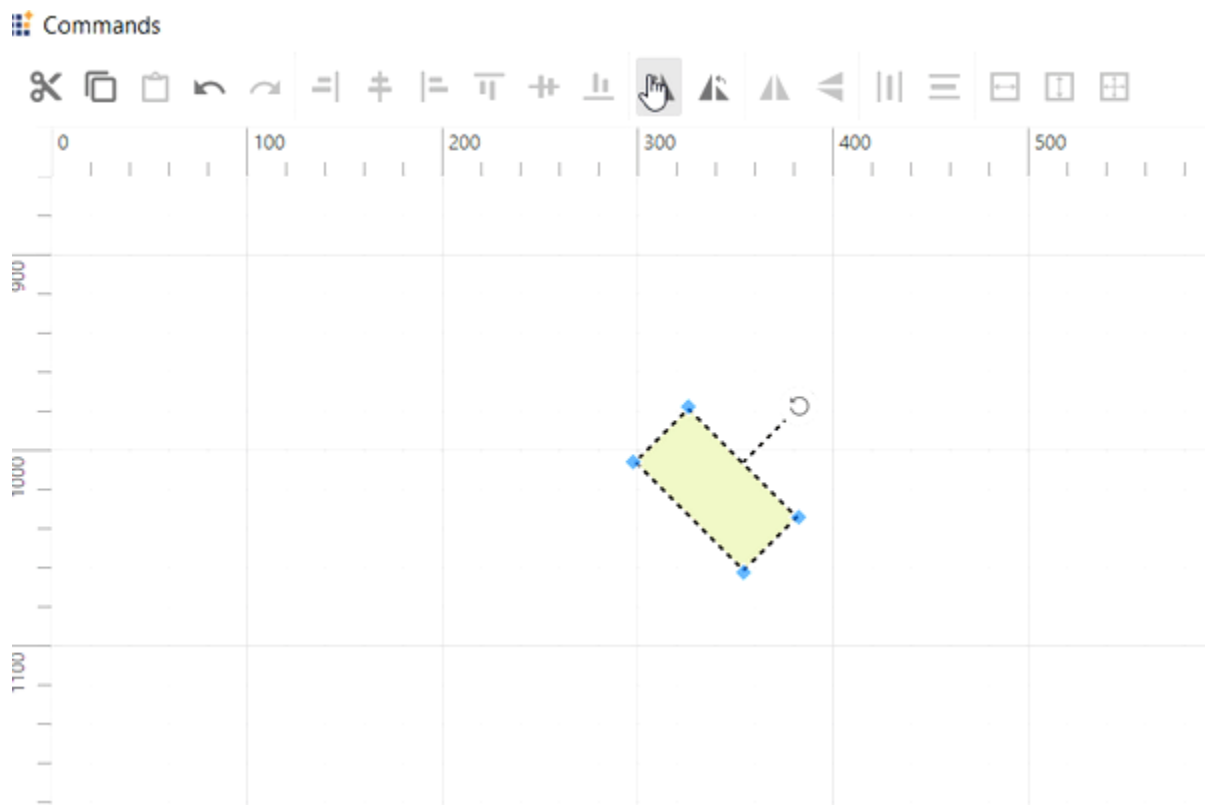
<Syncfusion:RotateParameter RotationDirection="Clockwise" Angle="45"
x:Key="RotateCommandParameter"/>

```

```
<Button Height="50" Content="Rotate" Name="RotateCommand"
Command="Syncfusion:DiagramCommands.Rotate"
CommandParameter="{StaticResource RotateCommandParameter}"></Button>
```

**C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
graphinfo.Commands.Rotate.Execute(new RotateParameter()
{
    RotationDirection=RotationDirection.Clockwise,Angle=45
});
```



[View sample in GitHub](#)

SelectByType Command in WPF Diagram (SfDiagram)

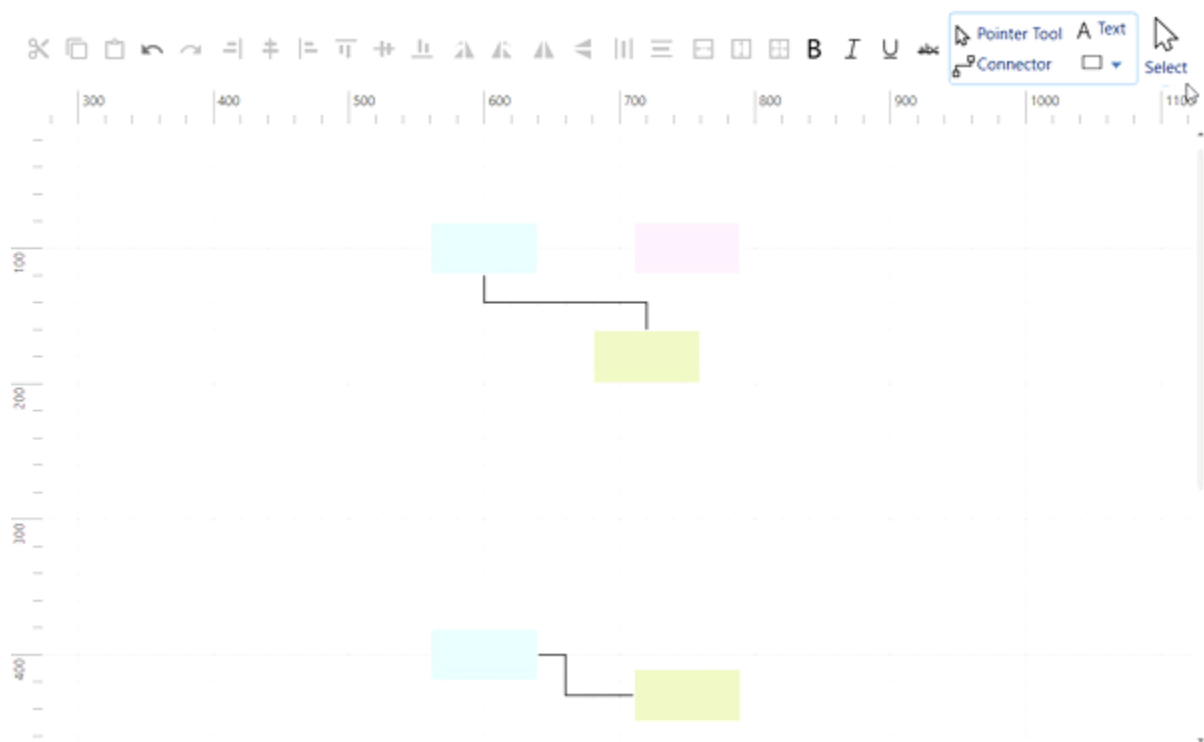
The [SelectByType](#) command is used to select the specified type (NodeViewModel,BpmnNodeViewModel,...) of elements in the diagram.

**XML**

```
<Button Height="50" Content="SelectByType" Name="SelectByType"
Command="Syncfusion:DiagramCommands.SelectByType" CommandParameter="{x:Type
Syncfusion:NodeViewModel}"></Button>
```

**C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
graphinfo.Commands.SelectByType.Execute(typeof(NodeViewModel));
```



[View sample in GitHub](#)

### SelectTool Command in WPF Diagram (SfDiagram)

The [SelectTool](#) command of the diagram helps to change the [DrawingTool](#) and [Tool](#) properties to a specified value. The [SelectToolCommandParameter](#) of [SelectTool](#) command is used to represent a specific tool.

Property	Description
---	---
Tool	It is used to customize the tools of the diagram.
DrawingTool	It is used to specify the drawing tool, which is valid only if the <b>Tool</b> is set as either <b>ContinuesDraw</b> or <b>DrawOnce</b> .
ConnectorType	It is used to specify the type (such as Orthogonal, Straight and Cubic-Curve etc.) of the connector to be drawn.

For details , refer [Tools and DrawingTools](#)

### XML

```
<!-- To draw an ellipse node-->
<Syncfusion:SelectToolCommandParameter DrawingTool="Ellipse"
Tool="ContinuesDraw" x:Key="SelectToolEllipseCommandParameter"/>
<!-- To draw a straight line connector-->
<Syncfusion:SelectToolCommandParameter DrawingTool="Connector"
ConnectorType="Line" Tool="ContinuesDraw"
x:Key="SelectToolConnectorCommandParameter"/>
<!-- To draw a text node-->
```

```

<Syncfusion:SelectToolCommandParameter DrawingTool="TextNode"
Tool="ContinuesDraw" x:Key="SelectToolTextCommandParameter"/>
<Button Height="50" Content="ConnectorTool" Name="Connector"
Command="Syncfusion:DiagramCommands.SelectTool"
CommandParameter="{StaticResource
SelectToolConnectorCommandParameter}"></Button>
<Button Height="50" Content="EllipseTool" Name="Ellipse"
Command="Syncfusion:DiagramCommands.SelectTool"
CommandParameter="{StaticResource
SelectToolEllipseCommandParameter}"></Button>
<Button Height="50" Content="TextTool" Name="TextNode"
Command="Syncfusion:DiagramCommands.SelectTool"
CommandParameter="{StaticResource SelectToolTextCommandParameter}"></Button>

```

**C#**

```

IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
// To draw a straight line connector
graphinfo.Commands.SelectTool.Execute(new SelectToolCommandParameter()
{
    Tool = Tool.ContinuesDraw, DrawingTool = DrawingTool.Connector,
    ConnectorType = ConnectorType.Line
});
// To draw an ellipse node
graphinfo.Commands.SelectTool.Execute(new SelectToolCommandParameter()
{
    Tool = Tool.ContinuesDraw, DrawingTool = DrawingTool.Ellipse
});
// To draw a text node
graphinfo.Commands.SelectTool.Execute(new SelectToolCommandParameter()
{
    Tool = Tool.ContinuesDraw, DrawingTool = DrawingTool.TextNode
});

```



[View sample in GitHub](#)

SetShapeStyle Commands in WPF Diagram (SfDiagram)

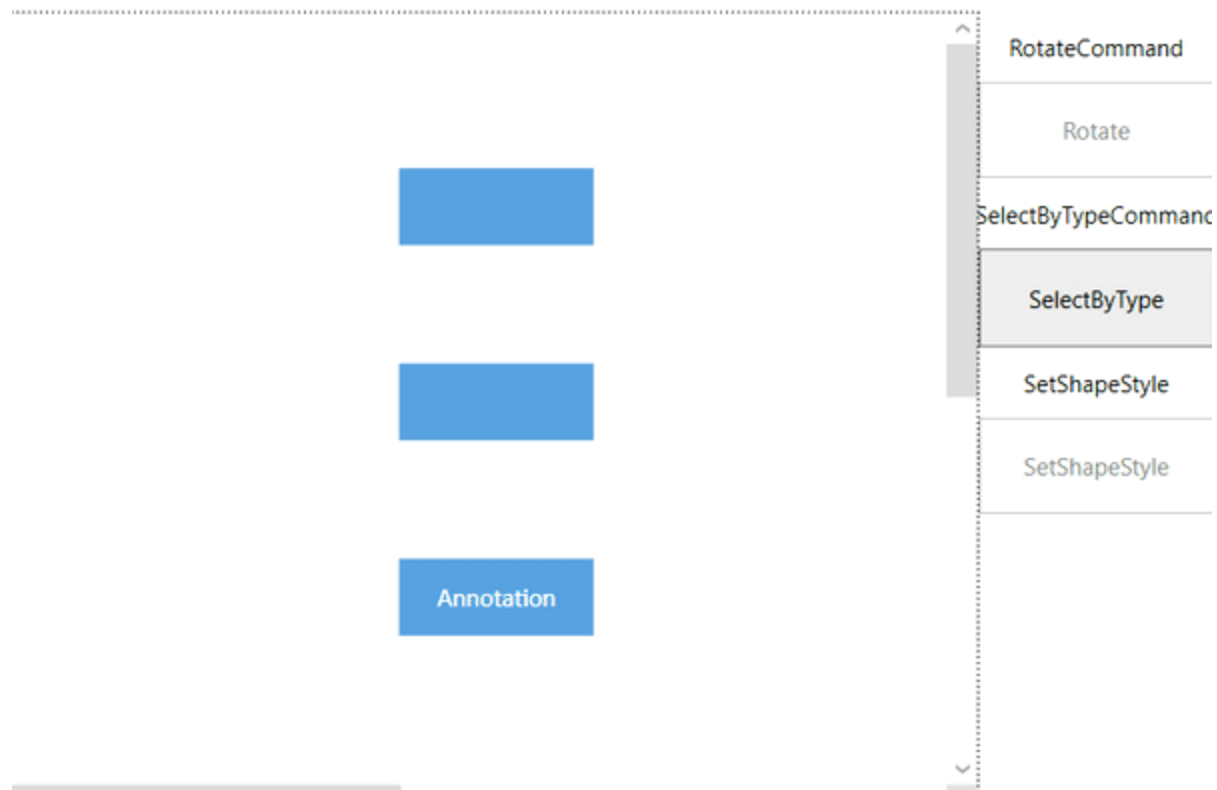
The [SetShapeStyle](#) command is used to set a specific style for the selected elements in the diagram.

#### XML

```
<Style TargetType="Path" x:Key="SetShapeStyleParameter">
  <Setter Property="Stretch" Value="Fill"></Setter>
  <Setter Property="Fill" Value="Green"></Setter>
</Style>
<Button Height="50" Content="SetShapeStyle" Name="SetShapeStyle"
  Command="Syncfusion:DiagramCommands.SetShapeStyle"
  CommandParameter="{StaticResource SetShapeStyleParameter}"></Button>
```

#### C#

```
Style style = new Style();
style.Setters.Add(new Setter() { Property =
System.Windows.Shapes.Path.FillProperty, Value = new
SolidColorBrush(Colors.Green) });
style.Setters.Add(new Setter() { Property =
System.Windows.Shapes.Path.StretchProperty, Value = Stretch.Fill });
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
graphinfo.Commands.SetShapeStyle.Execute(style);
```



[View sample in GitHub](#)

Edit and Format Text Commands in WPF Diagram (SfDiagram)

#### *EditAnnotation*

The [EditAnnotation](#) command is used to enable editing mode for the annotation of the selected element. And in the case of multiple selection, editing will be enabled for the first selected element.

#### **XML**

```
<Button Height="50" Content="EditAnnotation" Name="EditAnnotation"
Command="Syncfusion:DiagramCommands.EditAnnotation"></Button>
```

#### **C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
graphinfo.Commands.EditAnnotation.Execute(null);
```

#### *ToggleBold Command*

The [ToggleBold](#) command is used to toggle the bold style for the annotation of the selected element in the diagram.

#### **XML**

```
<Button Height="50" Content="ToggleBold" Name="ToggleBold"
Command="Syncfusion:DiagramCommands.ToggleBold"></Button>
```

#### **C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
graphinfo.Commands.ToggleBold.Execute(null);
```

#### *ToggleItalic Command*

The [ToggleItalic](#) command is used to toggle the italic style for the annotation of the selected element in the diagram.

#### XML

```
<Button Height="50" Content="ToggleItalic" Name="ToggleItalic"  
Command="Syncfusion:DiagramCommands.ToggleItalic"></Button>
```

#### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
graphinfo.Commands.ToggleItalic.Execute(null);
```

#### *ToggleUnderline Command*

The [ToggleUnderline](#) command is used to toggle the underline for the annotation of the selected elements in the diagram.

#### XML

```
<Button Height="50" Content="ToggleUnderline" Name="ToggleUnderline"  
Command="Syncfusion:DiagramCommands.ToggleUnderline"></Button>
```

#### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
graphinfo.Commands.ToggleUnderline.Execute(null);
```

#### *ToggleStrikeThrough Command*

The [ToggleStrikeThrough](#) command is used to toggle the strikethrough style for the annotation of the selected elements in the diagram.

#### XML

```
<Button Height="50" Content="ToggleStrikeThrough" Name="ToggleStrikeThrough"  
Command="Syncfusion:DiagramCommands.ToggleStrikeThrough"></Button>
```

#### C#

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;  
graphinfo.Commands.ToggleStrikeThrough.Execute(null);
```

#### *Cancel command*

The [Cancel](#) command is used to perform either one of the below action with higher priority.

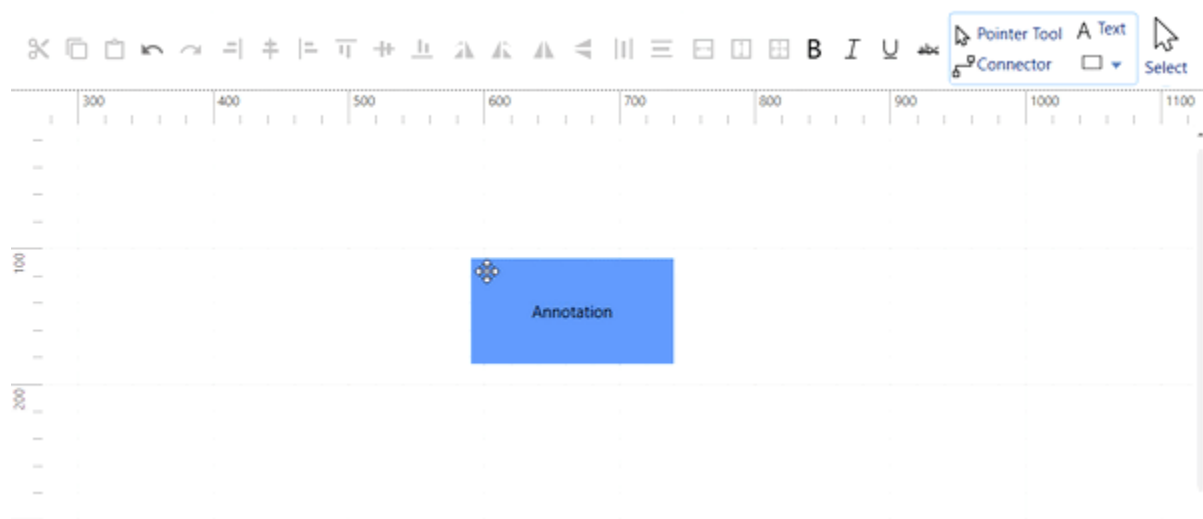
- Stops the annotation editing and accepts the current value.
- Clears the selection of or keyboard focus on the diagram elements.

**XML**

```
<Button Height="50" Content="Cancel" Name="Cancel"
Command="Syncfusion:DiagramCommands.Cancel"></Button>
```

**C#**

```
IGraphInfo graphinfo = diagramcontrol.Info as IGraphInfo;
graphinfo.Commands.Cancel.Execute(null);
```



[View sample in GitHub](#)

### Command Manager in WPF Diagram (SfDiagram)

The [CommandManager](#) is used to map the user gestures (keyboard, mouse) with SfDiagram commands and helps to include new gesture commands in SfDiagram. Refer to the following table for built-in commands with Key gesture and Mouse gesture.

List of Commands and Key Gesture:

Command	Key	Key Modifiers
---	---	---
Copy	C	Control
Cut	X	Control
Paste	V	Control
Duplicate	D	Control
Undo	Z	Control
Redo	Y	Control
MoveLeft	Left	
MoveRight	Right	
MoveUp	Up	



MoveDown	Down	
Delete	Delete	
SelectAll	A	Control
Group	G	Control
Ungroup	G	Control
SendToBack	[	Control+Shift
SendBackward	[	Control
BringToFront	]	Control+Shift
BringForward	]	Control
Group	G	Control
Ungroup	G	Control

#### List of Commands and Key Gestures with Parameter

Command	Key	KeyModifier	Parameter
Zoom	-	Control	new ZoomPositionParameter { ZoomCommand=ZoomCommand.ZoomOut }
Zoom	+	Control	new ZoomPositionParameter { ZoomCommand = ZoomCommand.ZoomIn }
Reset	0	Control	new ResetParameter { Reset = Diagram.Reset.ZoomPan }
FitToPage	0	Control+ Menu	new FitToPageParameter { FitToPage = Diagram.FitToPage.FitToPage, Margin = new Thickness(20) }

#### List of Commands and Mouse Gesture with Parameter

Command	Scroll State	Parameter
Vertical Scroll using 'Zoom' command	Scroll	new ZoomPointerParameter { ZoomCommand = ZoomCommand.VerticalScroll }

#### List of Commands and Key and Mouse Gesture with Parameter

Command	KeyModifier	Scroll State	Parameter	
Horizontal Scroll using 'Zoom' command	Shift	Scroll	new ZoomPointerParameter { ZoomCommand = ZoomCommand.HorizontalScroll }	
Zoom	Control	Scroll	new ZoomPointerParameter { ZoomCommand = ZoomCommand.ZoomIn	ZoomCommand.ZoomOut }

---

**Note:** When different commands are registered for the same key / mouse gestures, you need to handle the command while execution.

---

#### Custom command

CommandManager provides support to define custom commands. The custom commands are executed when the specified key gesture is recognized.

The [GestureCommand](#) and [Gesture](#) help you to define a custom command.

The following code example represents how to define custom command to Save Command (Control + S).

#### C#

```
// To define the mouse and keyboard gesture for the commands
GestureCommand saveGesture = new GestureCommand()
{
    // Define the command with custom command
    Command = Save,
    // Define gesture for custom Command
    Gesture = new Gesture
    {
        KeyModifiers = ModifierKeys.Control,
        KeyState = KeyStates.Down,
        Key = Key.S
    },
    // Parameter for command - file name for save command
    Parameter = "diagram"
};
// Add the custom command to the existing command collection.
diagram.CommandManager.Commands.Add(saveGesture);
```

[View sample in GitHub](#)

### Undo and Redo in WPF Diagram (SfDiagram)

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 and Redo actions

Diagram tracks the history of actions that are performed after initializing the diagram and provides support to reverse and restore those changes.

The redo function restores any actions that have been previously undone using an undo

Undo is a function performed to reverse the action of an earlier action.

Undo/redo actions can be executed through shortcut keys. Shortcut key for undo is Ctrl+z and shortcut key for redo is Ctrl+y.

Undo/Redo for diagram can be enabled/disabled with the [Constraints](#) property of SfDiagram class.

#### XML

```
<!--Initialize SfDiagram with undo/redo constraint-->
<syncfusion:SfDiagram x:Name="diagram" Constraints="Default,Undoable">
</syncfusion:SfDiagram>
```

#### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Enable the undo and redo actions
diagram.Constraints = GraphConstraints.Default | GraphConstraints.Undoable;
```

Undo/redo actions can be executed using commands of diagram control instead of using short cut keys.

### XML

```
<!--Initialize SfDiagram with undo constraint-->
<syncfusion:SfDiagram x:Name="diagram" Constraints="Default,Undoable">
</syncfusion:SfDiagram>
```

### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Enable the undo and redo actions
diagram.Constraints = GraphConstraints.Default | GraphConstraints.Undoable;
//To perform the Undo action in Diagram
(diagram.Info as IGraphInfo).Commands.Undo.Execute(null);
//To Perform the Redo action in Diagram
(diagram.Info as IGraphInfo).Commands.Redo.Execute(null);
```

### History actions

The **Action** property of the **HistoryEntry** class is used to store the history of actions performed after initializing the diagram. Changing the size of a node, dragging a connection, deleting a node, clicking and dragging the mouse, and so on are some of the actions. These types of actions are saved as **Action** properties that can be performed. This property has the following actions:

|HistoryAction|Description|

|--|--|

|None|Specifies the default value for the action type|

|PositionChanged|Specifies the element's position changed action|

|RotationChanged|Specifies the rotation value changed action for Node, Group element|

|SizeChanged|Specifies the element's size changed action|

|CollectionChanged|Specifies the diagram element's collection changed action that new element added or deleted|

|ConnectorSourceChanged|Specifies the connector element's source value changed action|

|ConnectorTargetChanged|Specifies the connector element's target value changed action|

|CustomAction|Specifies the custom element changed action|

### Entry mode

**Mode** property of **HistoryEntry** class allows to know where the changed action comes from. The **Mode** property has following type of sources,

The **Mode** property of the **HistoryEntry** class allows you to know whether the source of action entry is internal or external. Internal source action is accomplished by using the **SfDiagram** control and its elements, while external source action is accomplished by the external application users. This property has following type of sources:

|EntryMode|Description|

|--|--|

|Internal|Specifies source of action entry is added internally by SfDiagram to the undo-redo stack |

|External|Specifies source of action is added externally by the user to the undo-redo stack|

### Stack limit

The **StackLimit** property of **HistoryManager** class allows you to specify a stack limit value, which is used to limit the maximum number of history entries that can be stored in the Undo and Redo stacks.

### XML

```
<!--Initialize SfDiagram with undo and redo constraint-->
<syncfusion:SfDiagram x:Name="diagram" Constraints="Default,Undoable">
  <Syncfusion:SfDiagram.HistoryManager>
    <Syncfusion:HistoryManager StackLimit="3"/>
  </Syncfusion:SfDiagram.HistoryManager>
</syncfusion:SfDiagram>
```

### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Enable the undo and redo actions
diagram.Constraints = GraphConstraints.Default | GraphConstraints.Undoable;
diagram.HistoryManager = new HistoryManager()
{
    StackLimit = 3,
};
```

### Start group action

The **BeginComposite()** method of **HistoryManager** class allows you to log multiple actions at a time in the history manager stack. It is easier to undo or revert the changes made in the diagram in a single undo/redo process instead of reverting every actions one by one.

### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Enable the undo and redo actions
diagram.Constraints = GraphConstraints.Default | GraphConstraints.Undoable;
//Initialize the history manager class
diagram.HistoryManager = new HistoryManager();
//method to initiate the group action
diagram.HistoryManager.BeginComposite();
```

### End group action

The **EndComposite()** method of the **HistoryManager** class allows you to end the group actions that are stored in the stack history.

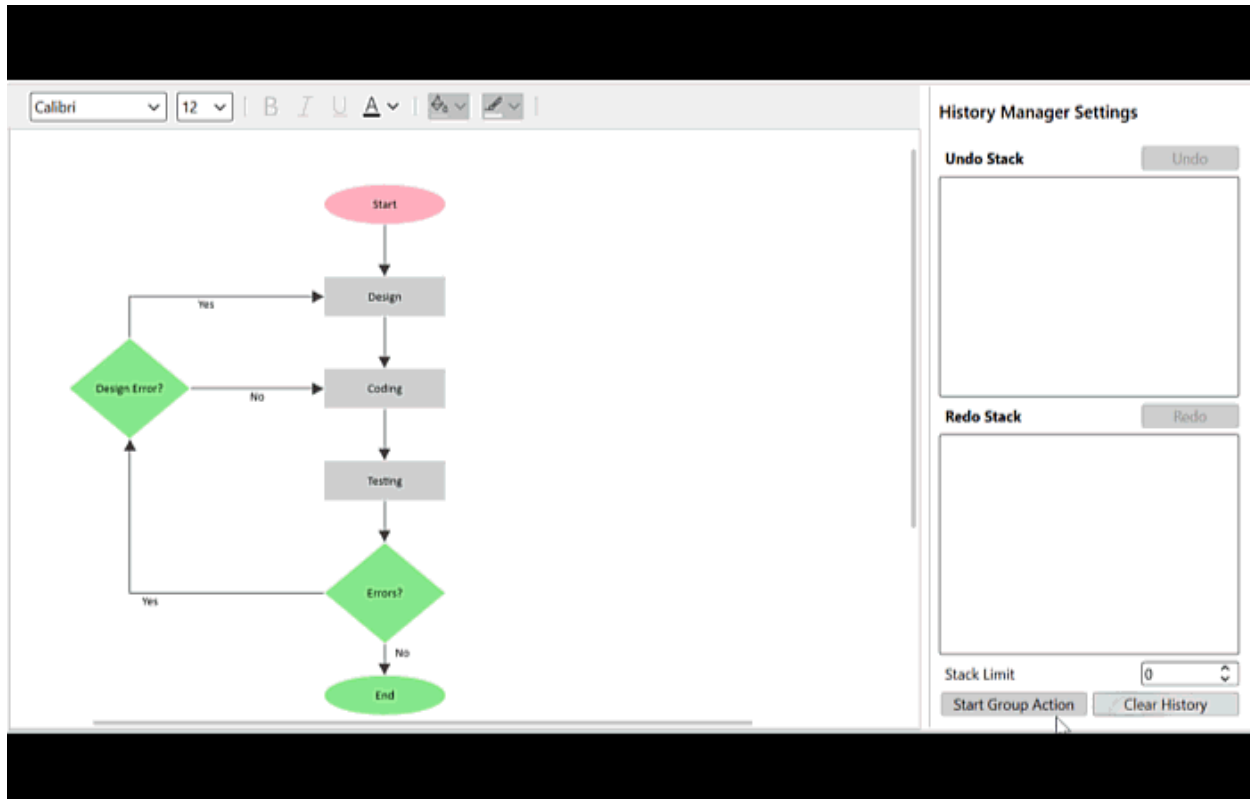
### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Enable the undo and redo actions
```

```

diagram.Constraints = GraphConstraints.Default | GraphConstraints.Undoable;
//Initialize the history manager class
diagram.HistoryManager = new HistoryManager();
//method to stop the group action
diagram.HistoryManager.EndComposite();

```



### How to view Undo/Redo stack

History manager class of SfDiagram control allows you to view the undo and redo stack values where you can get what the actions, values, and elements are stored in the history manager stack by using the `HistoryChangedEventArgs` argument value of `HistoryChangedEvent` event.

### C#

```

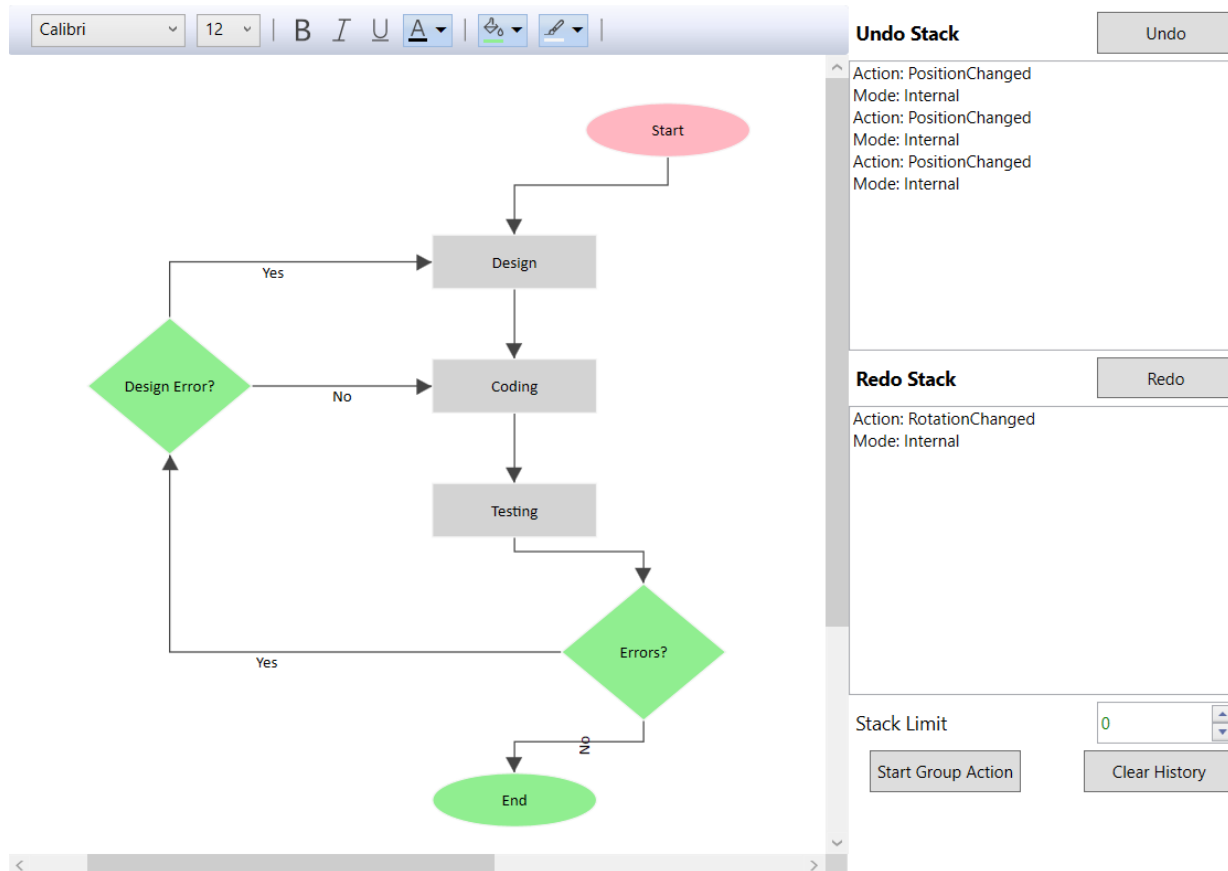
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Hook the history changed event.
(diagram.Info as IGraphInfo).HistoryChangedEvent += HistoryChangedEvent;
private void HistoryChangedEvent(object sender, HistoryChangedEventArgs
args)
{
    UndoText = string.Empty;
    foreach (HistoryEntry entry in HistoryManager.UndoStack as
Stack<HistoryEntry>)
    {
        if (UndoText == string.Empty)
        {
            UndoText = "Action: " + entry.Action.ToString();
            UndoText = UndoText + "\n" + "Mode: " + entry.Mode.ToString();
        }
    }
}

```

```

else
{
    UndoText = UndoText+ "\n" + "Action: " + entry.Action.ToString();
    UndoText = UndoText + "\n" + "Mode: " + entry.Mode.ToString();
}
}
RedoText = string.Empty;
foreach (HistoryEntry entry in HistoryManager.RedoStack as
Stack<HistoryEntry>)
{
    if (RedoText == string.Empty)
    {
        RedoText = "Action: " + entry.Action.ToString();
        RedoText = RedoText + "\n" + "Mode: " + entry.Mode.ToString();
    }
    else
    {
        RedoText = RedoText + "\n" + "Action: " + entry.Action.ToString();
        RedoText = RedoText + "\n" + "Mode: " + entry.Mode.ToString();
    }
}
}
}

```



[View Sample in GitHub](#)

### How to log custom actions in undo/redo stack

History list allows to revert or restore single and multiple changes through a single undo/redo command. The purpose of custom undo redo process is to store actions which are not done through default undo redo history list. Appearance level changes and its history informations did not stored in the history list. For example, revert/restore the fill color change of multiple elements at a time. To store multiple actions at a time, actions should be logged using [CompositeTransactions](#) class.

To achieve this you need to customize the [HistoryManager](#) class of diagram control and need to override the Undo Redo methods.

### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Assigning the custom history manager class
diagram.HistoryManager = new HistoryManager();
//To change the nodes fill color into blue and logging the action into
history manager class.
diagram.HistoryManager.BeginComposite();
foreach (NodeVm node in ((IEnumerable<object>) (diagram.SelectedItems as
SelectorViewModel).Nodes))
{
    if (node is NodeVm)
    {
        (node as NodeVm).Fill = new SolidColorBrush(Colors.CornflowerBlue);
    }
}
//To stop the group action stored in the undo/red stack
diagram.HistoryManager.EndComposite(diagram.HistoryManager, end);
//Creating custom node view model to update the fill property to nodes
public class NodeVm : NodeViewModel, IUndoRedo
{
    public NodeState _mCollectionData;
    public NodeVm()
    {
        _mCollectionData = new NodeState( Fill);
    }
    private Brush _mFill = new SolidColorBrush(Colors.Black);
    //Creating fill property to node to get or set the fill color for node
    public Brush Fill
    {
        get
        {
            return _mFill;
        }
        set
        {
            if (_mFill != value)
            {
                _mFill = value;
                OnPropertyChanged(NodeConstants.Fill);
            }
        }
    }
    //Allows to store the fill color data
    private bool AllowToLogData(string name)
```

```

{
    if (name == "Fill")
    {
        return true;
    }
    return false;
}

//To store the actions performed
public void LogData(object data)
{
    var info = this.Info as INodeInfo;
    if (info != null && info.Graph != null )
    {
        info.Graph.HistoryManager.LogData(this, data);
    }
}

//To apply the fill color to the node when undo/redo
private void OnFillChanged()
{
    Style s = new Style(typeof(Path));
    if (Fill != null)
    {
        s.Setters.Add(new Setter(Path.FillProperty, Fill));
        s.Setters.Add(new Setter(Path.StretchProperty, Stretch.Fill));
    }
    ShapeStyle = s;
}

//To update the fill property when it is changed
protected override void OnPropertyChanged(string name)
{
    var info = this.Info as INodeInfo;
    if (info != null && info.Graph != null && info.Graph.HistoryManager != null
        && AllowToLogData(name))
    {
        if (info.Graph.HistoryManager.CanLogData(info.Graph.HistoryManager,
            _mCollectionData))
        {
            LogData(_mCollectionData);
        }
    }
    base.OnPropertyChanged(name);
    switch (name)
    {
        case NodeConstants.Fill:
            OnFillChanged();
            _mCollectionData.Fill = this.Fill;
            break;
    }
}

public UndoRedoState State { get; set; }
public bool CanRedo(object data)
{
    if (State == UndoRedoState.Idle)
    {
        return true;
    }
    return false;
}

```

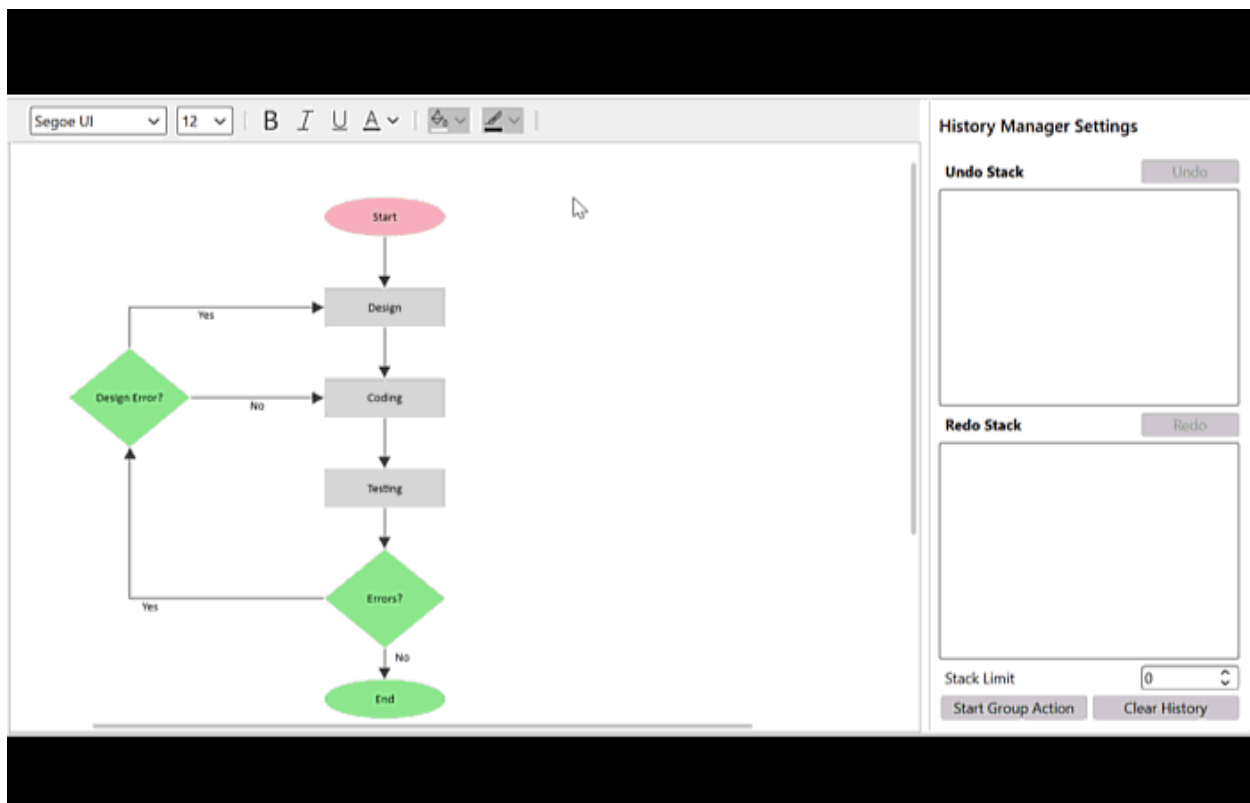


```
}
public bool CanUndo(object data)
{
    if (State == UndoRedoState.Idle)
    {
        return true;
    }
    return false;
}
public object Undo(object data)
{
    if (data is NodeState)
    {
        return RevertTo(data);
    }
    else
    return data;
}
public object Redo(object data)
{
    if (data is NodeState)
    {
        return RevertTo(data);
    }
    else
    return data;
}
public object RevertTo(object data)
{
    if (data is NodeState)
    {
        var current = GetData();
        NodeState toState = (NodeState)data;
        if (toState.Fill != this.Fill)
        {
            this.Fill = toState.Fill;
        }
        return current;
    }
    return data;
}
public object GetData()
{
    return _mCollectionData;
}
}
//Create class for node constants value
internal static class NodeConstants
{
    public const string Fill = "Fill";
}
//To specify the fill state for nodes.
public struct NodeState
{
    private Brush _mFill;
    public Brush Fill
{
```

```

get
{
    return _mFill;
}
set
{
    if (_mFill != value)
    {
        _mFill = value;
    }
}
}
public NodeState(Brush fill)
{
    _mFill = fill;
}
}

```



### How to restrict Undo/Redo

Undo, Redo process can be avoided for particular element by using [CanLogHistoryEntry](#) virtual method of diagram control.

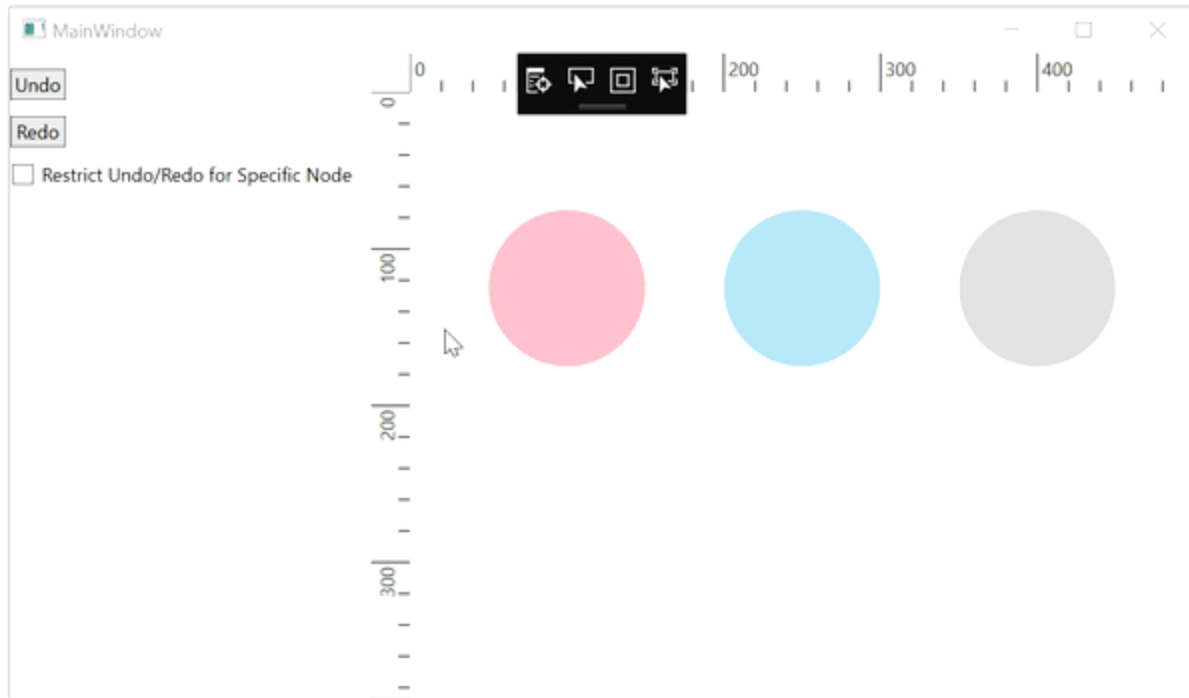
### C#

```

//Initialize the custom diagram class
CustomDiagram diagram = new CustomDiagram();
//Create custom node class to add can log property
public class NodeVm : NodeViewModel, IUndoRedo
{

```

```
private bool canlog = true;
//Specifies undo/redo process is enabled or disabled for a node
public bool CanLogmultipleselect
{
    get
    {
        return canlog;
    }
    set
    {
        if (value != canlog)
        {
            canlog = value;
            OnPropertyChanged("CanLogmultipleselect");
        }
    }
}
//Create the custom SfDiagram class
public class CustomDiagram: SfDiagram
{
    //Overriding the method to avoid actions stored
    protected override bool CanLogHistoryEntry(LogDataArgs item)
    {
        if (item.Item is NodeVm)
        {
            if (!(item.Item as NodeVm).CanLogmultipleselect)
            {
                return false;
            }
            return base.CanLogHistoryEntry(item);
        }
        else
            return base.CanLogHistoryEntry(item);
    }
}
```



[View Sample in GitHub](#)

History changed event

History manager class of SfDiagram control provides HistoryChangedEvent and HistoryChangedCommand to notify while there is a change in undo/redo stack values.

### XML

```
<!--Initialize SfDiagram with undo and redo constraint-->
<syncfusion:SfDiagram x:Name="diagram" Constraints="Default,Undoable"
HistoryChangedCommand="{Binding HistoryChangedCommand}">
</syncfusion:SfDiagram>
```

### C#

```
//Initialize SfDiagram
SfDiagram diagram = new SfDiagram();
//Enable the undo and redo actions
diagram.Constraints = GraphConstraints.Default | GraphConstraints.Undoable;
//Hook history changed event
(diagram.Info as IGraphInfo).HistoryChangedEvent += HistoryChangedEvent;
private void HistoryChangedEvent(object sender, HistoryChangedEventArgs
args)
{
}
```

Events for Undo Redo Process

Diagram allows to notify undo/redo action for the below events,

- [NodeChangedEvent](#)

- [ItemDeletedEvent](#)
- [ItemAddedEvent](#)
- [PortChangedEvent](#)
- [AnnotationChangedEvent](#)
- [ConnectorSourceChangedEvent](#)
- [ConnectorTargetChangedEvent](#)
- [ViewPortChanged](#)

## C#

```
//Initialize the SfDiagram
SfDiagram diagram = new SfDiagram();
//Register the node changed event
(diagram.Info as IGraphInfo).NodeChangedEvent += Diagram_NodeChangedEvent;
private void Diagram_NodeChangedEvent(object sender, ChangeEventArgs<object,
NodeChangedEventArgs> args)
{
    MessageBox.Show(args.Cause.ToString());
}
```

[View Sample in GitHub](#)

See Also

- [How to add enable undo and redo using commands?](#)

## DataSource in WPF Diagram (SfDiagram)

Diagram can be populated with the nodes and connectors based on the information provided from an external data source.

- The DataSourceSettings DataSource property is used to define the data source as a collection of objects, which needs to be populated as diagram.
- The DataSourceSettings Id property is used to define the unique field of each data.
- The DataSourceSettings ParentId property is used to define the parent field, which builds the relationship between ID and parent field.
- The DataSourceSettings Root property is used to define the root node for the diagram populated from the data source.

To explore those properties, refer to [DataSourceSettings](#)

### Defining DataSource

[DataSource](#) can be a collection of any business objects or collection of nodes. If you use collection of business objects as datasource, then nodes has been created automatically to populate a layout. As a collection of objects, datasource has the functionalities of add, remove, reset, and move. The following code example explains the defining of DataSource using business objects.

## XML

```
<!-- Initializes the employee collection-->
<local:Employees x:Key="employees">
<local:Employee Name="Steve" EmpId="1" ParentId="" Designation="CEO"/>
```

```

<local:Employee Name="Kevin" EmpId="2" ParentId="1" Designation="Manager"/>
<local:Employee Name="John" EmpId="3" ParentId="1" Designation="Manager"/>
<local:Employee Name="Raj" EmpId="4" ParentId="2" Designation="Team Lead"/>
<local:Employee Name="Will" EmpId="5" ParentId="2" Designation="S/w
Developer"/>
<local:Employee Name="Sarah" EmpId="6" ParentId="3" Designation="TeamLead"/>
<local:Employee Name="Mike" EmpId="7" ParentId="3" Designation="Testing
Engineer"/>
</local:Employees>
<!--Initializes the DataSourceSettings-->
<syncfusion:DataSourceSettings x:Key="DataSourceSettings"
Id="EmpId" ParentId="ParentId"
DataSource="{StaticResource employees}" />
<!--Initializes the SfDiagram-->
<syncfusion:SfDiagram x:Name="Diagram"
DataSourceSettings="{StaticResource DataSourceSettings}"/>

```

**C#**

```

/// <summary>
/// Business object class for creating datasource
/// </summary>
public class Employee
{
    public string Name { get; set; }
    public string EmpId { get; set; }
    public string ParentId { get; set; }
    public string Designation { get; set; }
}

public class Employees : ObservableCollection<Employee>
{
}

// Initialize DataSourceSettings for SfDiagram
Diagram.DataSourceSettings = new DataSourceSettings()
{
    Id = "EmpId",
    ParentId = "ParentId",
    Root = "1",
    DataSource = GetData(),
};

// Method to initialize the value for DataSource
private Employees GetData()
{
    Employees employees = new Employees();
    employees.Add(new Employee()
    {
        Name = "Steve", EmpId = "1", ParentId = "", Designation = "CEO"
    });
    employees.Add(new Employee()
    {
        Name = "Kevin", EmpId = "2", ParentId = "1", Designation = "Manager"
    });
    employees.Add(new Employee()
    {
        Name = "John", EmpId = "3", ParentId = "1", Designation = "Manager"
    });
}

```

```

employees.Add(new Employee()
{
    Name = "Raj", EmpId = "4", ParentId = "2", Designation = "Team Lead"
});
employees.Add(new Employee()
{
    Name = "Will", EmpId = "5", ParentId = "2", Designation = "S/w Developer"
});
employees.Add(new Employee()
{
    Name = "Sarah", EmpId = "6", ParentId = "3", Designation = "TeamLead"
});
employees.Add(new Employee()
{
    Name = "Mike", EmpId = "7", ParentId = "3", Designation = "Testing Engineer"
});
return employees;
}

```

---

**Note:** ParentId and Id must be in same type to populate a layout.

---

### Defining layout

By default, populated nodes are positioned at (0,0). You can use built-in automatic layout algorithm to define the position of the each populated nodes. The following code explains how to define the tree layout.

### XML

```

<!--Initializes the Layout-->
<syncfusion:DirectedTreeLayout x:Key="treeLayout"
    Type="Hierarchical"
    Orientation="TopToBottom"
    HorizontalSpacing="80"
    VerticalSpacing="50" />
<!--Initialize the layout manager-->
<syncfusion:LayoutManager x:Key="layoutManager" Layout="{StaticResource
    treeLayout}" />
<!--Initializes the SfDiagram-->
<syncfusion:SfDiagram x:Name="Diagram"
    DataSourceSettings="{StaticResource DataSourceSettings}"
    LayoutManager="{StaticResource layoutManager}" />

```

### C#

```

//Initialize LayoutManager and Layout for SfDiagram
Diagram.LayoutManager = new LayoutManager()
{
    Layout = new DirectedTreeLayout()
    {
        Type = LayoutType.Hierarchical,
        Orientation = TreeOrientation.TopToBottom,
        HorizontalSpacing = 80,
        VerticalSpacing = 50,
    },
};

```

To learn more about the supported built-in layout, refer to the [Automatic Layouts](#) page.

*How to done Add, Remove, Reset and Move in DataSource*

As **DataSource** is a collection of any business objects or nodes, it has the functionalities of add, remove, reset, and move.

### C#

```
//Add new data in datasource
(Diagram.DataSourceSettings.DataSource as Employees).Add(new Employee()
{ Name = "Steven", EmpId = "8", ParentId = "2", Designation = "S/w Developer"
});
(Diagram.DataSourceSettings.DataSource as Employees).Insert(3, new
Employee() { Name = "William", EmpId = "8", ParentId = "2", Designation =
"S/w Developer" });
//Remove data from datasource
Employee emp = (Diagram.DataSourceSettings.DataSource as
Employees).ElementAt(5);
(Diagram.DataSourceSettings.DataSource as Employees).Remove(emp);
(Diagram.DataSourceSettings.DataSource as Employees).RemoveAt(5);
//Reset all data in datasource
Diagram.DataSourceSettings.DataSource = null;
Diagram.DataSourceSettings.DataSource = new Employees();
//Move data from one position to another in datasource
(Diagram.DataSourceSettings.DataSource as Employees).Move(2, 1);
```

### Root

By default, the node without parent is treated as root of the layout. Now, **DataSourceSettings** have option to specify the root node of the layout.

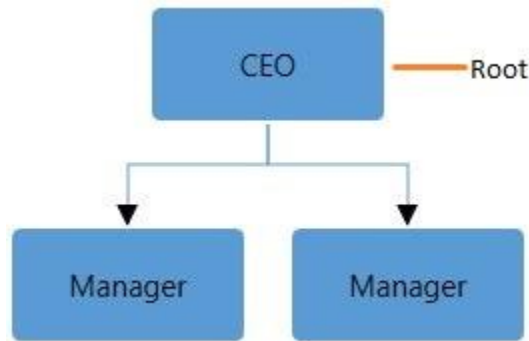
### XML

```
<!--Initializes the DataSourceSettings and set object with id "1" and name
"CEO"as root-->
<syncfusion:DataSourceSettings x:Key="DataSourceSettings" Id="EmpId"
ParentId="ParentId"
DataSource="{StaticResource employees}"
Root="1" />
```

### C#

```
//object with id "1" and name "CEO", is considered as root of tree layout.
diagram.DataSourceSettings.Root = "1";
```





### Layout with multiple parents

Tree layout and data sources will now support nodes having multiple parents by defining more than one parent ID to the `ParentID` property. The child node is arranged in center of the parent positions. The following code explains the defining of multiple parent ID's to single node.

### XML

```

<local:DataItems x:Key="DataItems">
  <local:ItemInfo Name="n11" RatingColor="#ff6329"/>
  <local:ItemInfo Name="n12" RatingColor="#ff6329"/>
  <local:ItemInfo Name="n13" RatingColor="#ff6329"/>
  <local:ItemInfo Name="n21" RatingColor="#941100">
    <local:ItemInfo.ReportingPerson>
      <local:StringList>
        <system:String>n11</system:String>
        <system:String>n12</system:String>
        <system:String>n13</system:String>
      </local:StringList>
    </local:ItemInfo.ReportingPerson>
  </local:ItemInfo>
</local:DataItems>
<syncfusion:SfDiagram>
  <syncfusion:SfDiagram.DataSourceSettings>
    <syncfusion:DataSourceSettings Id="Name" ParentId="ReportingPerson"
    DataSource="{StaticResource DataItems}"/>
  </syncfusion:SfDiagram.DataSourceSettings>
</syncfusion:SfDiagram>
  
```

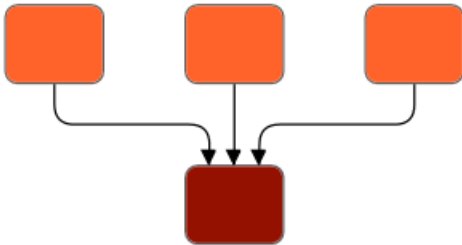
### C#

```

public class ItemInfo
{
  public ItemInfo(string name, string color)
  {
    this.Name = name;
    this.RatingColor = color;
  }
  public string RatingColor { get; set; }
  public string Name { get; set; }
  public List<string> ReportingPerson { get; set; }
}

DataItems data = new DataItems();
data.Add(new ItemInfo("n11", "#ff6329"));
data.Add(new ItemInfo("n12", "#ff6329"));
  
```

```
data.Add(new ItemInfo("n13", "#ff6329"));
data.Add(new ItemInfo("n21", "#941100")
{
    ReportingPerson = new List<string> { "n11", "n12", "n13" }
});
```



[View sample in GitHub](#)

FlowchartDataSourceSettings

[FlowchartDataSourceSettings](#) is the derived class of [DataSourceSettings](#), which contains the mapping properties. These properties are used to map the data member in the underlying data object to the datasource item.

**ContentMapping:** Maps the content in the underlying data object to data source item.

**ConnectorTextMapping:** Maps the ConnectorText in the underlying data object to data source item.

**ShapeMapping:** Maps the shape in the underlying data object to data source item.

**WidthMapping:** Maps the width in the underlying data object to data source item.

**HeightMapping:** Maps the height in the underlying data object to data source item.

## XML

```
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
    Source="/Syncfusion.SfDiagram.Wpf;component/Resources/BasicShapes.xaml"/>
</ResourceDictionary.MergedDictionaries>
<!-- Initializes the DataSource collection-->
<local:DataItems x:Key="Datas">
<local:ItemInfo Id="1" NodeShape="{StaticResource Terminator}"
    Width="80" Height="40"
    Name="Start"/>
<local:ItemInfo Id="2" NodeShape="{StaticResource Decision}"
    Width="100" Height="80"
    Name="Decision?">
<local:ItemInfo.ParentId>
<local:LabelList>
<sys:String>1</sys:String>
</local:LabelList>
</local:ItemInfo.ParentId>
</local:ItemInfo>
<local:ItemInfo Id="3" NodeShape="{StaticResource Process}"
    Width="80" Height="50"
```

```

Name="Process1">
<local:ItemInfo.ParentId>
<local:LabelList>
<sys:String>2</sys:String>
</local:LabelList>
</local:ItemInfo.ParentId>
<local:ItemInfo.Label>
<local:LabelList>
<sys:String>No</sys:String>
</local:LabelList>
</local:ItemInfo.Label>
</local:ItemInfo>
<local:ItemInfo Id="4" NodeShape="{StaticResource Process}"
Width="80" Height="50"
Name="Process2">
<local:ItemInfo.ParentId>
<local:LabelList>
<sys:String>2</sys:String>
</local:LabelList>
</local:ItemInfo.ParentId>
<local:ItemInfo.Label>
<local:LabelList>
<sys:String>Yes</sys:String>
</local:LabelList>
</local:ItemInfo.Label>
</local:ItemInfo>
<local:ItemInfo Id="5" NodeShape="{StaticResource Terminator}"
Width="80" Height="40"
Name="End">
<local:ItemInfo.ParentId>
<local:LabelList>
<sys:String>4</sys:String>
</local:LabelList>
</local:ItemInfo.ParentId>
</local:ItemInfo>
</local:DataItems>
<!--Initializes the Layout-->
<syncfusion:FlowchartLayout x:Key="Layout" Orientation="TopToBottom"
YesBranchDirection="LeftInFlow"
HorizontalSpacing="50"
VerticalSpacing="30"/>
<!--Initializes the DataSourceSettings -->
<syncfusion:FlowchartDataSourceSettings x:Key="DataSourceSettings"
DataSource="{StaticResource Datas}"
ParentId="ParentId" Id="Id"
ShapeMapping="NodeShape"
WidthMapping="Width"
HeightMapping="Height"
ConnectorTextMapping="Label"
ContentMapping="Name" />
<!--Initializes the LayoutManager-->
<syncfusion:LayoutManager x:Key="layoutmanager" Layout="{StaticResource
Layout}"/>
<!--Initializes the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram" LayoutManager="{StaticResource
layoutManager}"
DataSourceSettings="{StaticResource DataSourceSettings">

```

```

<!--Initializes the NodeCollection-->
<syncfusion:SfDiagram.Nodes>
<syncfusion:NodeCollection/>
</syncfusion:SfDiagram.Nodes>
<!--Initializes the ConnectorCollection-->
<syncfusion:SfDiagram.Connectors>
<syncfusion:ConnectorCollection/>
</syncfusion:SfDiagram.Connectors>
<syncfusion:SfDiagram.Theme>
<syncfusion:OfficeTheme/>
</syncfusion:SfDiagram.Theme>
</syncfusion:SfDiagram>

```

## C#

```

//Initialize Diagram
SfDiagram diagram = new SfDiagram();
//Initialize Node Collection
diagram.Nodes = new ObservableCollection<NodeViewModel>();
//Initialize Connector Collection
diagram.Connectors = new ObservableCollection<ConnectorViewModel>();
//Initialize DataSourceSettings for SfDiagram
diagram.DataSourceSettings = new FlowchartDataSourceSettings()
{
    ParentId = "ParentId",
    Id = "Id",
    DataSource = GetData(),
    ConnectorTextMapping = "Label",
    ContentMapping = "Name",
    ShapeMapping = "NodeShape",
    WidthMapping = "Width",
    HeightMapping = "Height"
};
//Initialize LayoutManager
LayoutManager layoutManager = new LayoutManager();
//Initialize Layout for SfDiagram
layoutManager.Layout = new FlowchartLayout()
{
    Orientation = FlowchartOrientation.TopToBottom,
    YesBranchDirection = BranchDirection.LeftInFlow,
    NoBranchDirection = BranchDirection.RightInFlow,
    HorizontalSpacing = 50,
    VerticalSpacing = 30
};
//initialize theming style for SfDiagram
diagram.Theme = new OfficeTheme();
//Initialize LayoutManager
diagram.LayoutManager = layoutManager;
//Adding SfDiagram as children to mainwindow grid.
WindowGrid.Children.Add(Diagram);
//Initializes the DataSource collection
private DataItems GetData()
{
    DataItems collection = new DataItems();
    collection.Add(new ItemInfo()
    {

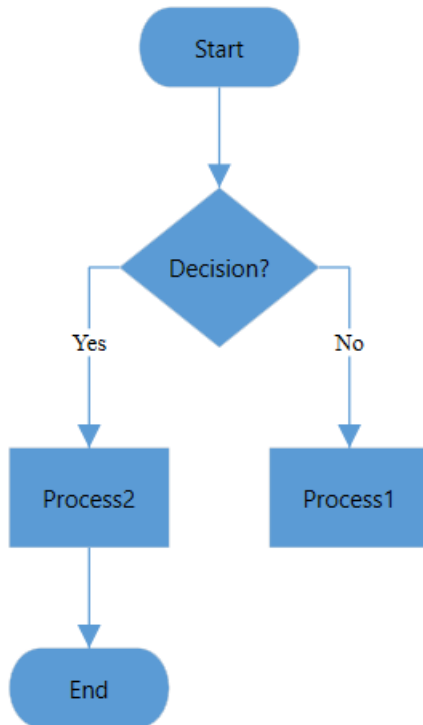
```

```

Id = "1",
NodeShape = App.Current.Resources["Terminator"] as string,
Name = "Start",
Height = 40,
Width = 100
});
collection.Add(new ItemInfo()
{
    Id = "2",
    ParentId = new List<string> { "1" },
    NodeShape = App.Current.Resources["Decision"] as string,
    Name = "Decision?",
    Height = 100,
    Width = 100
});
collection.Add(new ItemInfo()
{
    Id = "3", ParentId = new List<string> { "2" },
    Label = new List<string> { "No" },
    NodeShape = App.Current.Resources["Process"] as string,
    Name = "Process1",
    Height = 40,
    Width = 100
});
collection.Add(new ItemInfo()
{
    Id = "4",
    ParentId = new List<string> { "2" },
    Label = new List<string> { "Yes" },
    NodeShape = App.Current.Resources["Process"] as string,
    Name = "Process2",
    Height = 40,
    Width = 100
});
collection.Add(new ItemInfo()
{
    Id = "5",
    ParentId = new List<string> { "4" },
    NodeShape = App.Current.Resources["Terminator"] as string,
    Name = "End",
    Height = 40,
    Width = 100
});
return collection;
}
//Data Object Class
public class ItemInfo
{
    public string Name { get; set; }
    public string Id { get; set; }
    public List<string> ParentId { get; set; }
    public string NodeShape { get; set; }
    public List<string> Label { get; set; }
    public double Width { get; set; }
    public double Height { get; set; }
}
//Collection to hold the Data Object class

```

```
public class DataItems : ObservableCollection<ItemInfo>
{
}
```



[View sample in GitHub](#)

### Automatic Layout in WPF Diagram (SfDiagram)

SfDiagram provides a set of built-in automatic layout algorithms, which is used to arrange nodes automatically based on a predefined layout logic. SfDiagram supports the following built-in automatic layout algorithms:

- Organizational layout
- Flowchart layout
- MindMap tree layout
- Hierarchical tree layout
- Radial tree layout

Automatic layout algorithm uses the nodes and connectors defined in `NodeCollection` and `ConnectorCollection` or business objects defined in `DataSource` as input to generate the layout. To generate layout from `NodeCollection` and `ConnectorCollection`, you have to create all the nodes and connectors required for layout and add those items in `NodeCollection` and `ConnectorCollection` as defined in the following code snippet.

### XML

```
<!--Initializes the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
```

```

<!--Initialize Nodes-->
<syncfusion:SfDiagram.Nodes>
<syncfusion:NodeCollection>
<syncfusion:NodeViewModel ID="General Manager"
UnitHeight="40" UnitWidth="120"
Shape="{StaticResource Rectangle}"
Content="General Manager"/>
<syncfusion:NodeViewModel ID="Product Manager1"
UnitHeight="40" UnitWidth="120"
Shape="{StaticResource Rectangle}"
Content="Product Manager1"/>
<syncfusion:NodeViewModel ID="Product Manager2"
UnitHeight="40" UnitWidth="120"
Shape="{StaticResource Rectangle}"
Content="Product Manager2"/>
<syncfusion:NodeViewModel ID="Engineer1"
UnitHeight="40" UnitWidth="120"
Shape="{StaticResource Rectangle}"
Content="Engineer1"/>
<syncfusion:NodeViewModel ID="Engineer2"
UnitHeight="40" UnitWidth="120"
Shape="{StaticResource Rectangle}"
Content="Engineer2"/>
<syncfusion:NodeViewModel ID="Engineer3"
UnitHeight="40" UnitWidth="120"
Shape="{StaticResource Rectangle}"
Content="Engineer3"/>
<syncfusion:NodeViewModel ID="Engineer4"
UnitHeight="40" UnitWidth="120"
Shape="{StaticResource Rectangle}"
Content="Engineer4"/>
</syncfusion:NodeCollection>
</syncfusion:SfDiagram.Nodes>
<!--Initialize Connectors-->
<syncfusion:SfDiagram.Connectors>
<syncfusion:ConnectorCollection>
<syncfusion:ConnectorViewModel SourceNodeID="General Manager"
TargetNodeID="Product Manager1"/>
<syncfusion:ConnectorViewModel SourceNodeID="General Manager"
TargetNodeID="Product Manager2"/>
<syncfusion:ConnectorViewModel SourceNodeID="Product Manager1"
TargetNodeID="Engineer1"/>
<syncfusion:ConnectorViewModel SourceNodeID="Product Manager1"
TargetNodeID="Engineer2"/>
<syncfusion:ConnectorViewModel SourceNodeID="Product Manager2"
TargetNodeID="Engineer3"/>
<syncfusion:ConnectorViewModel SourceNodeID="Product Manager2"
TargetNodeID="Engineer4"/>
</syncfusion:ConnectorCollection>
</syncfusion:SfDiagram.Connectors>
</syncfusion:SfDiagram>

```

**C#**

```

//Create SfDiagram instance
SfDiagram diagram = new SfDiagram;

```

```

//Initialize Nodes and Connectors Collection
diagram.Nodes = new NodeCollection();
diagram.Connectors = new ConnectorCollection();
//Create and add Nodes and Connectors
(diagram.Nodes as NodeCollection).Add(CreateNode("General Manager"));
(diagram.Nodes as NodeCollection).Add(CreateNode("Product Manager1"));
(diagram.Nodes as NodeCollection).Add(CreateNode("Product Manager2"));
(diagram.Nodes as NodeCollection).Add(CreateNode("Engineer1"));
(diagram.Nodes as NodeCollection).Add(CreateNode("Engineer2"));
(diagram.Nodes as NodeCollection).Add(CreateNode("Engineer3"));
(diagram.Nodes as NodeCollection).Add(CreateNode("Engineer4"));
(diagram.Connectors as ConnectorCollection).Add(CreateConnector("General Manager", "Product Manager1"));
(diagram.Connectors as ConnectorCollection).Add(CreateConnector("General Manager", "Product Manager2"));
(diagram.Connectors as ConnectorCollection).Add(CreateConnector("Product Manager1", "Engineer1"));
(diagram.Connectors as ConnectorCollection).Add(CreateConnector("Product Manager1", "Engineer2"));
(diagram.Connectors as ConnectorCollection).Add(CreateConnector("Product Manager2", "Engineer3"));
(diagram.Connectors as ConnectorCollection).Add(CreateConnector("Product Manager2", "Engineer4"));
//Method to create Connectors
private ConnectorViewModel CreateConnector(string node1, string node2)
{
    ConnectorViewModel con = new ConnectorViewModel()
    {
        SourceNodeID = node1,
        TargetNodeID = node2,
    };
    return con;
}
//Method to create Nodes
private NodeViewModel CreateNode(string content)
{
    NodeViewModel node = new NodeViewModel()
    {
        ID = content,
        UnitHeight = 40,
        UnitWidth = 120,
        Shape = new RectangleGeometry() { Rect = new Rect(10, 10, 10, 10) },
        Content = content,
    };
    return node;
}

```

To generate layout from DataSource, you have to define the business object and add the necessary data to the DataSource collection. During the layout generation, nodes and connectors can be generated automatically with the information provided through data source and those items will be added to NodeCollection and ConnectorCollection respectively. Refer to the following code to generate the layout from data source.

#### **XML**

```

<!-- Initializes the employee collection-->

```



```

<local:Employees x:Key="employees">
<local:Employee EmpId = "1" ParentId="" Name="General Manager"/>
<local:Employee EmpId = "2" ParentId = "1" Name = "Product Manager1" />
<local:Employee EmpId = "3" ParentId = "1" Name = "Product Manager2"/>
<local:Employee EmpId = "4" ParentId = "2" Name = "Engineer1"/>
<local:Employee EmpId = "5" ParentId = "2" Name = "Engineer2"/>
<local:Employee EmpId = "6" ParentId = "3" Name = "Engineer3"/>
<local:Employee EmpId = "7" ParentId = "3" Name = "Engineer4"/>
</local:Employees>
<!--Initializes the DataSourceSettings -->
<syncfusion:DataSourceSettings x:Key="DataSourceSettings"
ParentId="ParentId" Id="EmpId"
DataSource="{StaticResource employees}"/>

```

## C#

```

Employees employee = new Employees();
employee.Add(new Employee() { EmpId = "1", ParentId = "", Name = "General
Manager" });
employee.Add(new Employee() { EmpId = "2", ParentId = "1", Name = "Product
Manager1" });
employee.Add(new Employee() { EmpId = "3", ParentId = "1", Name = "Product
Manager2" });
employee.Add(new Employee() { EmpId = "4", ParentId = "2", Name =
"Engineer1" });
employee.Add(new Employee() { EmpId = "5", ParentId = "2", Name =
"Engineer2" });
employee.Add(new Employee() { EmpId = "6", ParentId = "3", Name =
"Engineer3" });
employee.Add(new Employee() { EmpId = "7", ParentId = "3", Name =
"Engineer4" });
//Initializes the DataSourceSettings
diagram.DataSourceSettings = new DataSourceSettings()
{
    Id = "EmpId",
    ParentId = "ParentId",
    DataSource = employee,
};

```

## Defining layout

You can use the [LayoutManager.Layout](#) property to specify any one of the layouting algorithm.

## XML

```

<!--Initializes the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
<!--Initialize LayoutManager and Layout-->
<syncfusion:SfDiagram.LayoutManager>
<syncfusion:LayoutManager>
<syncfusion:LayoutManager.Layout>
<syncfusion:DirectedTreeLayout HorizontalSpacing="30"
VerticalSpacing="50"
AvoidSegmentOverlapping="False"
Orientation="TopToBottom"
Type="Hierarchical"/>
</syncfusion:LayoutManager.Layout>

```

```

</syncfusion:LayoutManager>
</syncfusion:SfDiagram.LayoutManager>
</syncfusion:SfDiagram>

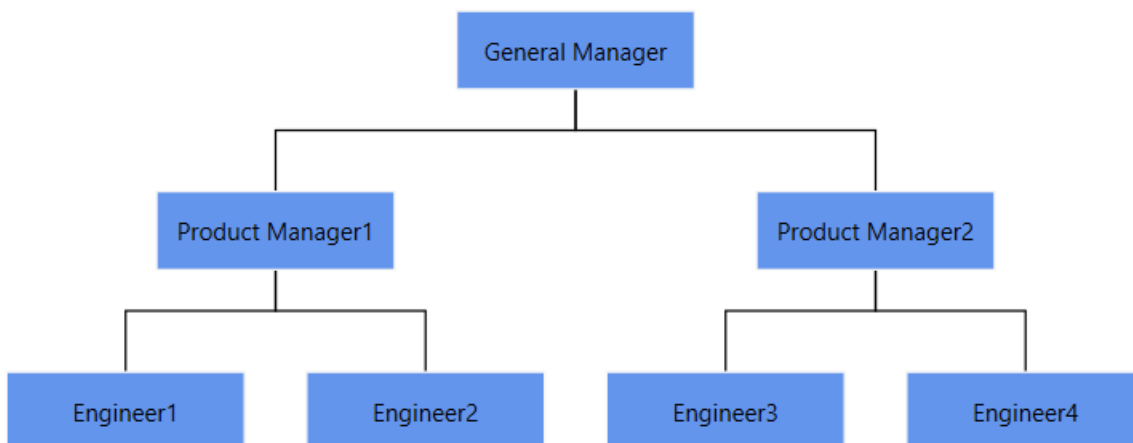
```

**C#**

```

//Initialize layout Manager
diagram.LayoutManager = new LayoutManager()
{
    Layout = new DirectedTreeLayout()
    {
        HorizontalSpacing = 30,
        VerticalSpacing = 50,
        Orientation = TreeOrientation.TopToBottom,
        Type = LayoutType.Hierarchical,
        AvoidSegmentOverlapping = false,
    },
};

```



## Updating layout

The [RefreshFrequency](#) property of LayoutManager is used to re-arrange the nodes in the diagram area when a node is added, deleted, moved, or resized. Also, you can decide when the nodes should be arranged for every diagram load or only for the first load. Find the description for each condition in the following table.

Refresh Frequencies	Description
---	---
Add	Used to update the layout after adding a new element to the datasource.
Remove	Used to update the layout after removing an existing element from datasource.
Move	Used to update the layout after moving element in the datasource.
Reset	Used to update the layout after resetting the datasource.

- | Load | Used to update the layout when loading the diagram. |
- | FirstLoad | Used to update the layout in the first load of the diagram. |
- | Resizing | Used to update the layout when resizing an element in the layout. |
- | Resized | Used to update the layout when resizing of an element is completed. |
- | ArrangeParsing | Used to update the layout when the operations like Add, Remove, Move, Reset, Resizing, and Resized are performed in layout. |

**XML**

```
<syncfusion:LayoutManager RefreshFrequency="ArrangeParsing"/>
```

**C#**

```
diagram.LayoutManager = new LayoutManager()  
{  
    RefreshFrequency = RefreshFrequency.ArrangeParsing,  
};
```

**Customize spacing between nodes in layout**

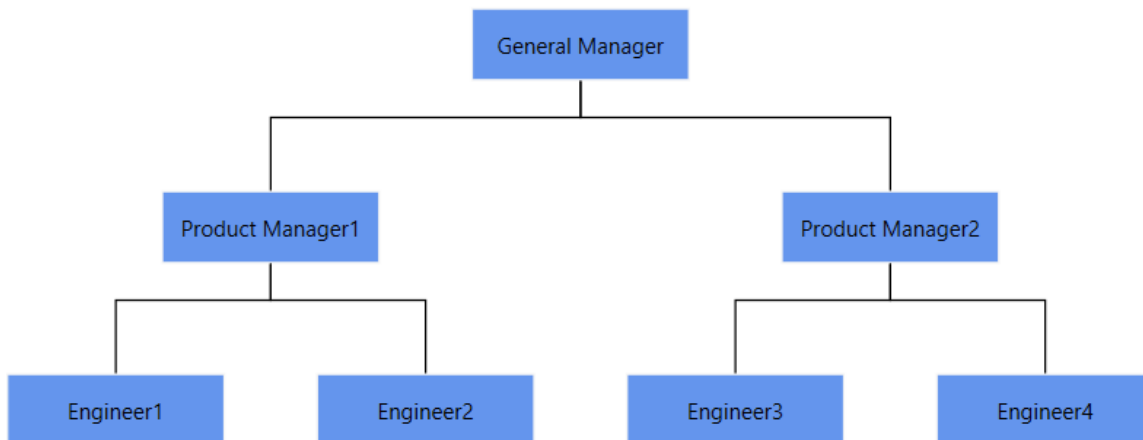
The Horizontal and Vertical spacing properties of Layouts are used to customize the space between successive nodes in both horizontally and vertically. The default value for horizontal spacing is 20 and vertical spacing is 50.

**XML**

```
<syncfusion:DirectedTreeLayout HorizontalSpacing="50" VerticalSpacing="60"/>
```

**C#**

```
diagram.LayoutManager = new LayoutManager()  
{  
    Layout = new DirectedTreeLayout()  
    {  
        HorizontalSpacing = 50,  
        VerticalSpacing = 60,  
    },  
};
```



Customize tree orientation in layout

[Orientation](#) of `DirectedTreeLayout` is used to arrange the tree layout based on the direction. Orientation is only valid for hierarchical and organization layout. The default value for orientation is `TopToBottom`. The different orientation types are defined in the following table:

Orientation Type	Description
TopToBottom	Aligns the tree layout from top to bottom. All the roots are placed at top of diagram.
LeftToRight	Aligns the tree layout from left to right. All the roots are placed at left of diagram.
BottomToTop	Aligns the tree layout from bottom to top. All the roots are placed at bottom of the diagram.
RightToLeft	Aligns the tree layout from right to left. All the roots are placed at right of the diagram.

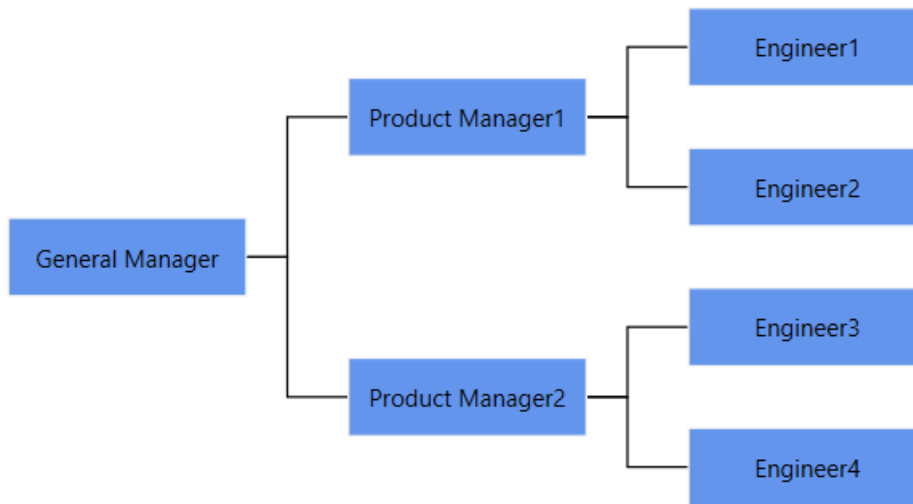
#### XML

```
<syncfusion:DirectedTreeLayout Orientation="LeftToRight" />
```

#### C#

```

diagram.LayoutManager = new LayoutManager()
{
    Layout = new DirectedTreeLayout()
    {
        Orientation = TreeOrientation.LeftToRight,
    },
};
  
```



---

N Orientation is not valid for RadialTreeLayout.

---

Avoiding connector segment overlapping in layout

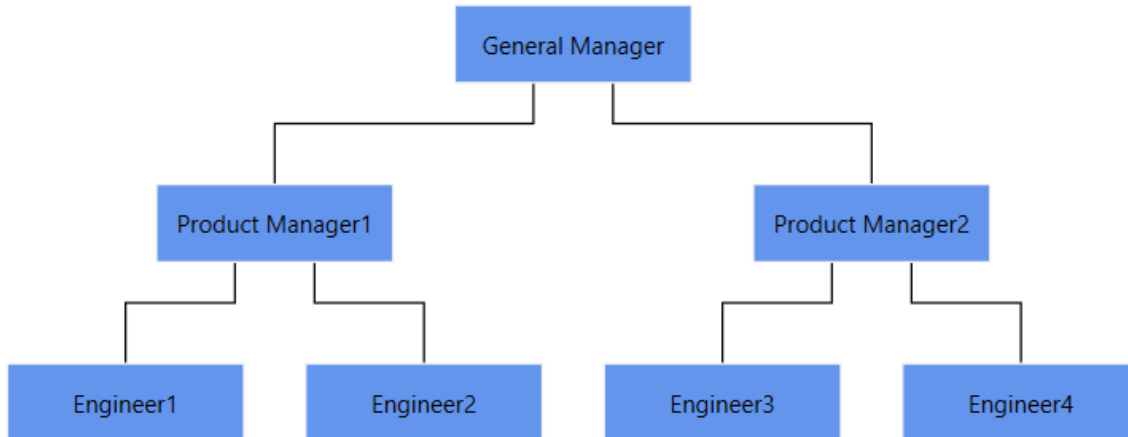
The [AvoidSegmentOverlapping](#) property of `DirectedTreeLayout` is used to decide whether segment of each connector from a single parent is distributed automatically or not. It is only valid for hierarchical and multi-parent layout.

#### XML

```
<syncfusion:DirectedTreeLayout AvoidSegmentOverlapping="True">
</Syncfusion:DirectedTreeLayout>
```

#### C#

```
diagram.LayoutManager = new LayoutManager()
{
    Layout = new DirectedTreeLayout()
    {
        AvoidSegmentOverlapping = true,
    },
};
```



---

N AvoidSegmentOverlapping is not valid for RadialTreeLayout.

---

#### Customize margin in layout

The [Margin](#) property of DirectedTreeLayout is used to provide space between the bounds of the tree layout to the diagram. The default margin value is 50.

#### XML

```
<syncfusion:DirectedTreeLayout Margin="200">  
</Syncfusion:DirectedTreeLayout>
```

#### C#

```
diagram.LayoutManager = new LayoutManager()  
{  
    Layout = new DirectedTreeLayout()  
    {  
        Margin = new Thickness(200),  
    },  
};
```

---

N Margin is not valid for RadialTreeLayout.

---

#### See Also

- [How to create parent and child relationship by drag and drop nodes?](#)
- [How to show assistants to the parent node of the organization layout?](#)

#### Overview of Essential WPF Diagram (SfDiagram)

Overview control is used to display a preview (overall view) of the entire content of a Diagram. This helps you to look overall picture of large diagram and easy to navigate (pan or zoom) to a particular position of the page.

### Usage scenario

When you work on a huge and complex diagram, you may not know the part where you are actually working, and navigating from one part to another might be difficult. To navigation, zoom out entire diagram and find where you are. This solution is not suitable when you need some frequent navigation.

Overview control solved this problem by displaying a preview (overall view) of the entire diagram with option to pan and zoom.

### Define overview

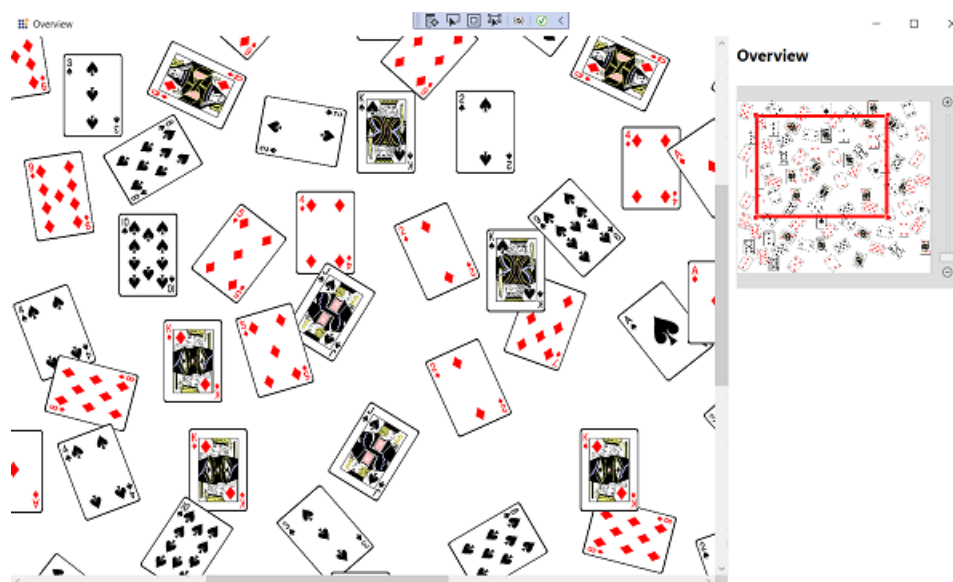
Overview control can be added to the application by dragging it from the toolbox and dropping it in Designer view. The required assembly references will be added automatically.

Steps to add Overview control manually in XAML:

1. Add the following required assembly reference to the project, Syncfusion.SfDiagram.WPF
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> or SfDiagram control namespace Syncfusion.UI.Xaml.Diagram.Controls in XAML page.
3. Declare Overview control in XAML page.

### XML

```
<Window x:Class="UserInteraction_Overview.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
WindowStartupLocation="CenterScreen"
Title="Overview" Height="720" Width="1200">
<!--Initialize the overview control and bind the diagram control elements to
overview-->
<syncfusion:Overview Source="{Binding ElementName=diagram}" Height="300"
Margin="0,25,0,0"/>
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram"/>
</Window>
```



[View Sample in GitHub.](#)

### ZoomSlider

The [ShowZoomSlider](#) property is used to show or hide the zoom slider in the overview control. By default, the ShowZoomSlider is true.

You can zoom in or zoom out the overview as well as the diagram by changing the slider or click on the zoom in or zoom out button.

The following code example explains how to hide the zoom slider.

### XML

```
<Window x:Class="UserInteraction_Overview.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
WindowStartupLocation="CenterScreen"
Title="Overview" Height="720" Width="1200">
<!--Initialize the overview control and bind the diagram control elements to overview-->
<syncfusion:Overview Source="{Binding ElementName=diagram}" Height="300"
Margin="0,25,0,0" ShowZoomSlider="False"/>
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram"/>
</Window>
```



### Interaction

Overview control allows Zoom and Pan interactions. The red rectangle indicates the area currently displayed on the diagram page. The red box can be moved within the panel to pan around the diagram. You can click and drag the corner of the rectangle to change the level of zooming on the diagram page or you can draw a new rectangle by clicking and dragging anywhere within the panel to zoom an area.

### XML

```
<!--Initialize the overview control with its constraint-->
<syncfusion:Overview Source="{Binding ElementName=diagram}"
```



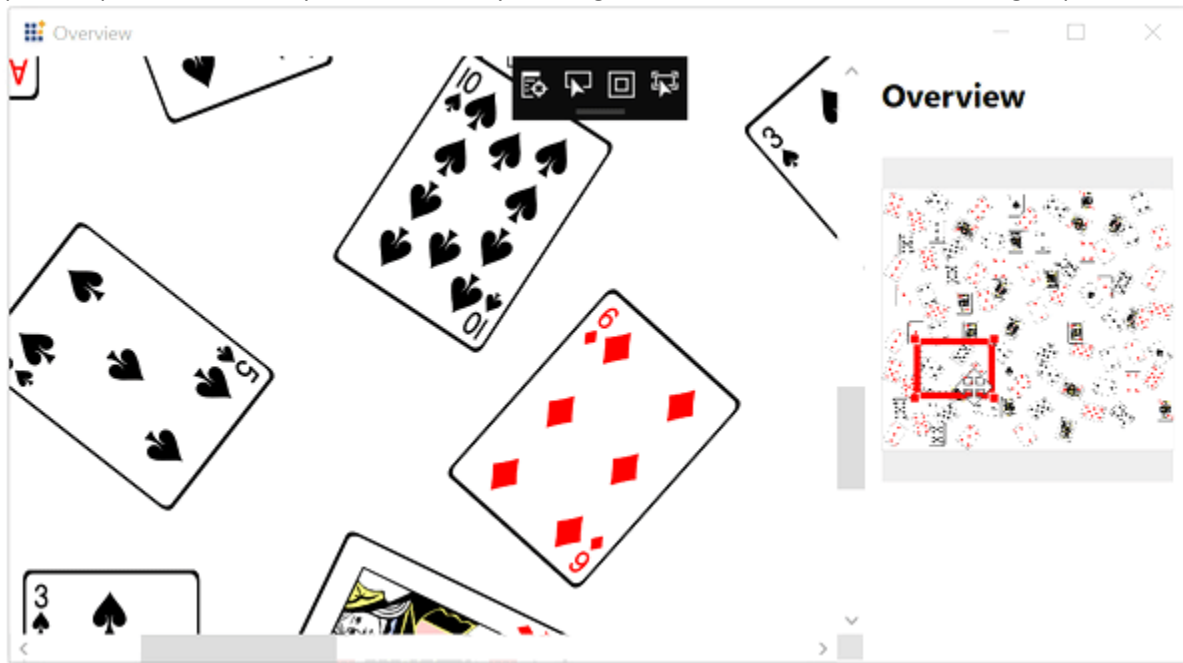
```
Constraint="Default"
Height="300" Margin="0,25,0,0"/>
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram"/>
```

The [Constraint](#) property of the Overview class allows you to control the Pan and Zoom interactions based the value assigned to that property. The following table explains the various values and their behaviors:

OverviewConstraints	Description	Output
--- --- ---		
Pan	Allows users to pan the diagram page by dragging the focused rectangle.	



| Zoom | Allows users to perform zoom by resizing the corners of the focused rectangle. |



| DrawFocus | Allows users to draw new focused rectangle anywhere within the overview panel, that is corresponding region in the diagram will be brought into the view. |



| TapFocus | Allows users move the focused rectangle to any area within the panel by just tapping it. | |

| Default | Allows users to perform drag, resize, draw, and tap the overview control. | |

| None | No interaction can be performed on the focused rectangle. |

### Deferred scrolling

Diagram supports the deferred scrolling behaviour to improve the zoom and pan performances.

To learn more about deferred scrolling, refer to [Deferred Scrolling](#).

### Event

The `OverviewChangedEvent` will notify the interactions in Overview control with [OverviewChangedEventArgs](#) as argument. This argument will provide the dragging and interaction state value of the overview.

[How to virtualize the diagram control](#)

[How to serialize the diagram control](#)

[How to localize the diagram control](#)

## Stencil in WPF Diagram (SfDiagram)

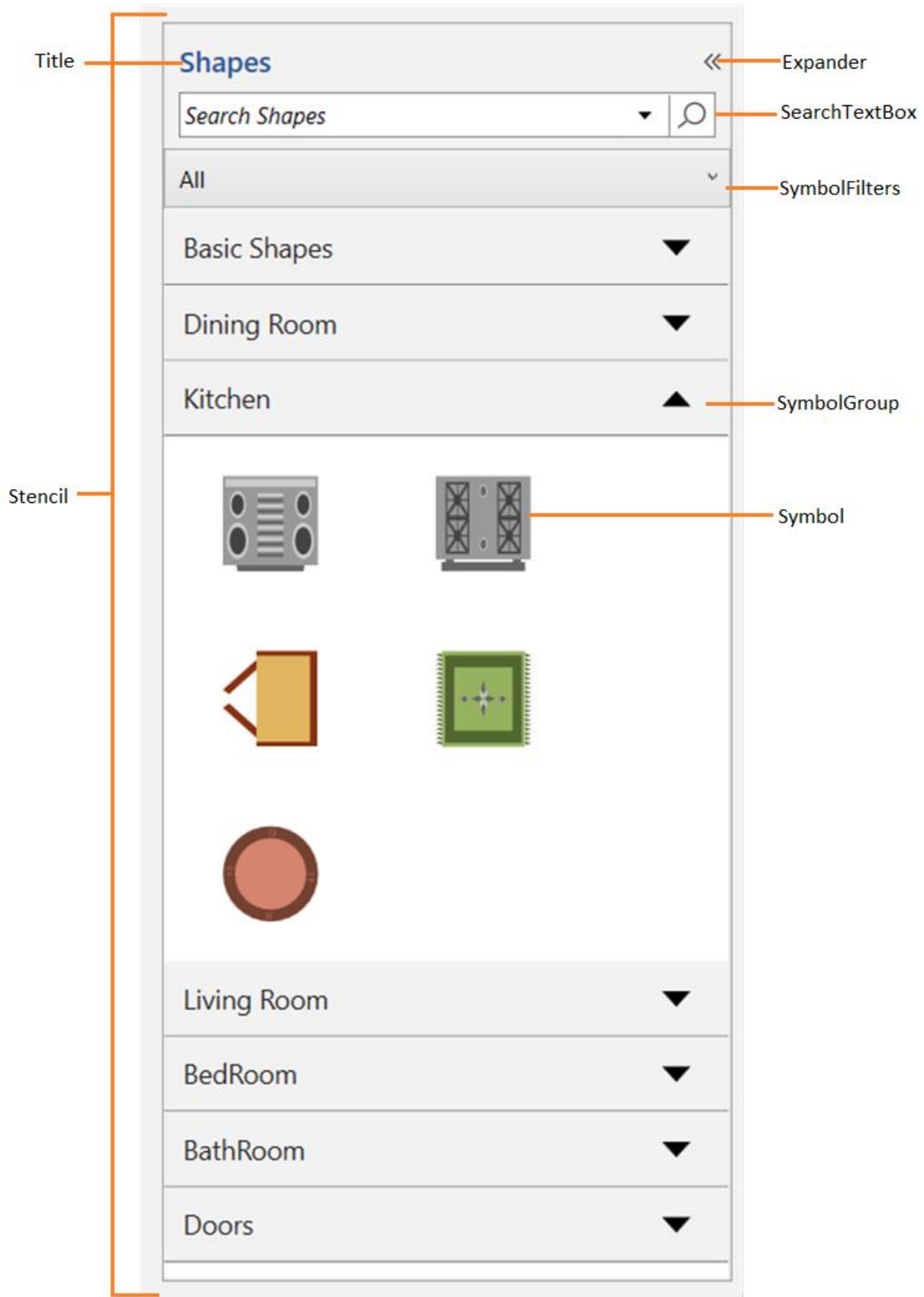
The [Stencil](#) is a gallery of reusable symbols and diagram elements that can be dragged and dropped on the diagram surface multiple times.

### XML

```
<!--Namespace for stencil-->
xmlns:stencil="clr-
namespace:Syncfusion.UI.Xaml.Diagram.Stencil;assembly=Syncfusion.SfDiagram.W
PF"
<!--Define a Stencil-->
<stencil:Stencil x:Name="stencil" ExpandMode="All" BorderBrush="Black"
BorderThickness="0,0,1,0" />
```

### C#

```
//Define a Stencil.
Stencil stencil = new Stencil()
{
    ExpandMode = ExpandMode.All,
    BorderThickness = new Thickness(0,0,1,0),
    BorderBrush = new SolidColorBrush(Colors.Black)
};
```



### Add symbols in a Stencil

The [Symbol](#) is used to visualize the elements in a stencil that can be created using the following ways:

- Using the diagram elements.
- Using the SymbolViewModel.

The Stencil's [SymbolSource](#) property is used to define the data source as a collection of objects (symbol, node, connector, and more) that needs to be populated as symbols.

### Using the Diagram Elements

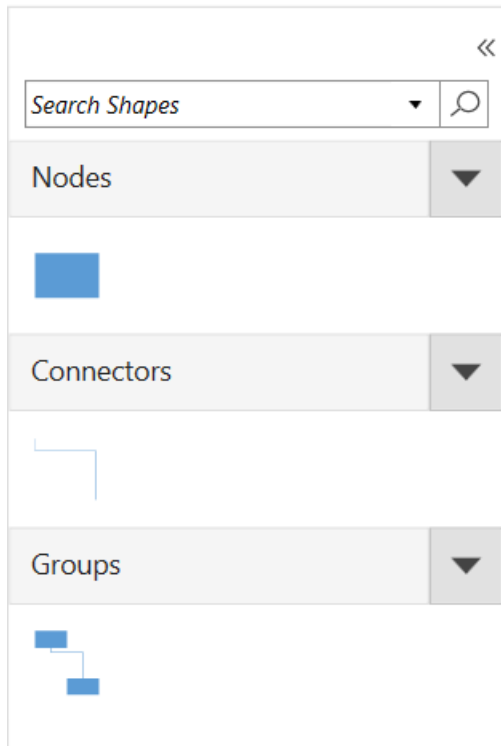
The diagram elements such as [NodeViewModel](#), [ConnectorViewModel](#), and [GroupViewModel](#) can be used to visualize the symbol.

### XML

```
<!--Initialize the SymbolSource-->
<stencil:Stencil.SymbolSource>
  <!--Define the SymbolCollection-->
  <local:SymbolCollection>
    <syncfusion:NodeViewModel x:Name="node" UnitHeight="70" UnitWidth="100"
      OffsetX="100" OffsetY="100" Shape="{StaticResource Rectangle}">
    </syncfusion:NodeViewModel>
    <syncfusion:ConnectorViewModel SourcePoint="100,100" TargetPoint="200,200"/>
  <!--Define the DiagramElement- Group-->
  <syncfusion:GroupViewModel>
    <!--Creates the Groupable Nodes-->
    <syncfusion:GroupViewModel.Nodes>
      <syncfusion:NodeCollection>
        <syncfusion:NodeViewModel UnitHeight="70" ID="srcnode" OffsetX="0"
          OffsetY="300"
          UnitWidth="100"
          Shape="{StaticResource Rectangle}">
        </syncfusion:NodeViewModel>
        <syncfusion:NodeViewModel UnitHeight="70"
          ID="tarnode"
          OffsetX="100"
          OffsetY="500"
          UnitWidth="100"
          Shape="{StaticResource Rectangle}">
        </syncfusion:NodeViewModel>
      </syncfusion:NodeCollection>
    </syncfusion:GroupViewModel.Nodes>
    <!--Creates the Groupable Connectors-->
    <syncfusion:GroupViewModel.Connectors>
      <syncfusion:ConnectorCollection>
        <syncfusion:ConnectorViewModel SourceNodeID="srcnode"
          TargetNodeID="tarnode"/>
      </syncfusion:ConnectorCollection>
    </syncfusion:GroupViewModel.Connectors>
  </syncfusion:GroupViewModel>
</local:SymbolCollection>
</stencil:Stencil.SymbolSource>
```

### C#

```
//Define the SymbolSource with SymbolCollection.
stencil.SymbolSource = new SymbolCollection();
//Initialize the diagram element.
NodeViewModel node = new NodeViewModel()
{
    UnitHeight = 70,
    UnitWidth = 100,
    OffsetX = 100, OffsetY = 100,
    Shape = App.Current.MainWindow.Resources["Rectangle"],
};
ConnectorViewModel cvm = new ConnectorViewModel()
{
    SourcePoint = new Point(100, 100),
    TargetPoint = new Point(200, 200),
};
GroupViewModel grp = new GroupViewModel()
{
    Nodes = new NodeCollection()
    {
        new NodeViewModel()
        {
            ID="srcnode",
            UnitHeight=70,
            UnitWidth=100,
            OffsetX=0,
            OffsetY=300,
            Shape=App.Current.Resources["Rectangle"]
        },
        new NodeViewModel()
        {
            ID="tarnode",
            UnitHeight=70,
            UnitWidth=100,
            OffsetX=100,
            OffsetY=500,
            Shape=App.Current.Resources["Rectangle"]
        }
    },
    Connectors = new ConnectorCollection()
    {
        new ConnectorViewModel()
        {
            SourceNodeID="srcnode",
            TargetNodeID="tarnode"
        }
    }
};
//Adding an element to the collection.
(stencil.SymbolSource as SymbolCollection).Add(node);
(stencil.SymbolSource as SymbolCollection).Add(cvm);
(stencil.SymbolSource as SymbolCollection).Add(grp);
//Adding the ISymbol to the SymbolCollection.
public class SymbolCollection : ObservableCollection<Object>
{
}
```



[View Sample in GitHub](#)

Using the `SymbolViewModel`

The `SymbolViewModel` has `Symbol` and `SymbolTemplate` properties to visualize the `Symbol` in the stencil.

### XML

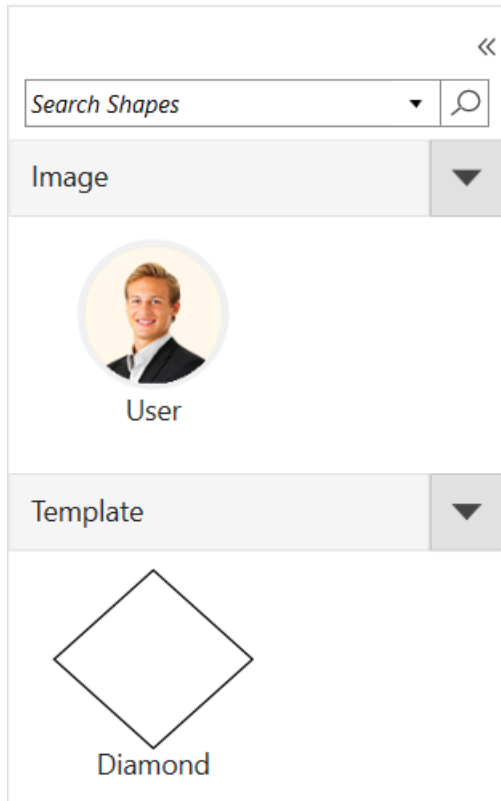
```
<DataTemplate x:Key="Diamond">
  <StackPanel>
    <Path Stretch="Fill"
      Data="M 397.784,287.875L 369.5,316.159L 341.216,287.875L 369.5,259.591L
      397.784,287.875 Z"
      Fill="White"
      Stroke="Black"
      StrokeThickness="1" />
    <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
      Text="Diamond" />
  </StackPanel>
</DataTemplate>
<DataTemplate x:Key="symboltemplate">
  <StackPanel>
    <Image Source="/Image/user_image.png" Width="100" Height="80" />
    <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
      Text="User" />
  </StackPanel>
</DataTemplate>
<stencil:Stencil.SymbolSource>
  <!--Define the SymbolCollection-->
  <local:SymbolCollection>
```

```
<Syncfusion:SymbolViewModel Symbol="User" SymbolTemplate="{StaticResource  
symboltemplate}"/>  
<Syncfusion:SymbolViewModel Symbol="Diamond" SymbolTemplate="{StaticResource  
Diamond}"/>  
</local:SymbolCollection>  
</stencil:Stencil.SymbolSource>
```

## C#

```
//Define the SymbolSource with the SymbolCollection.  
stencil.SymbolSource = new SymbolCollection();  
//Initialize the SymbolItem.  
SymbolViewModel imagenode = new SymbolViewModel()  
{  
    Symbol = "User",  
    SymbolTemplate = this.Resources["symboltemplate"] as DataTemplate  
};  
SymbolViewModel symbol = new SymbolViewModel()  
{  
    Symbol = "Diamond",  
    SymbolTemplate = this.Resources["Diamond"] as DataTemplate  
};  
//Adding the element to the collection.  
(stencil.SymbolSource as SymbolCollection).Add(imagenode);  
(stencil.SymbolSource as SymbolCollection).Add(symbol);  
//Adding the ISymbol to the SymbolCollection.  
public class SymbolCollection : ObservableCollection<Object>  
{  
}
```





### Constraints

The **Constraints** property of stencil allows you to enable or disable certain features. For more information about stencil constraints, refer to the [StencilConstraints](#).

[View Sample in GitHub](#)

See also

- [How to drag and drop elements from a treeview ?](#)
- [How to refresh the stencil when adding a new symbol in the symbol source ?](#)
- [How to host different UI elements as a node content ?](#)
- [How to notify stencil has been loaded ?](#)
- [How to get the base node interface while dropping a Symbol from Stencil to SfDiagram ?](#)
- [How to use different User Controls into Stencil?](#)

### Printing in WPF Diagram (SfDiagram)

SfDiagram provides support to print the content displayed in the diagram page using the [PrintingService.Print](#) method.

#### Direct print

Sfdiagram provides support to directly print the diagram pages using system default printer without opening the print preview window. To print the diagram by calling [SfDiagram.PrintingService.Print](#) method, refer to the following code sample.

#### C#

```
diagram.PrintingService.Print();
```

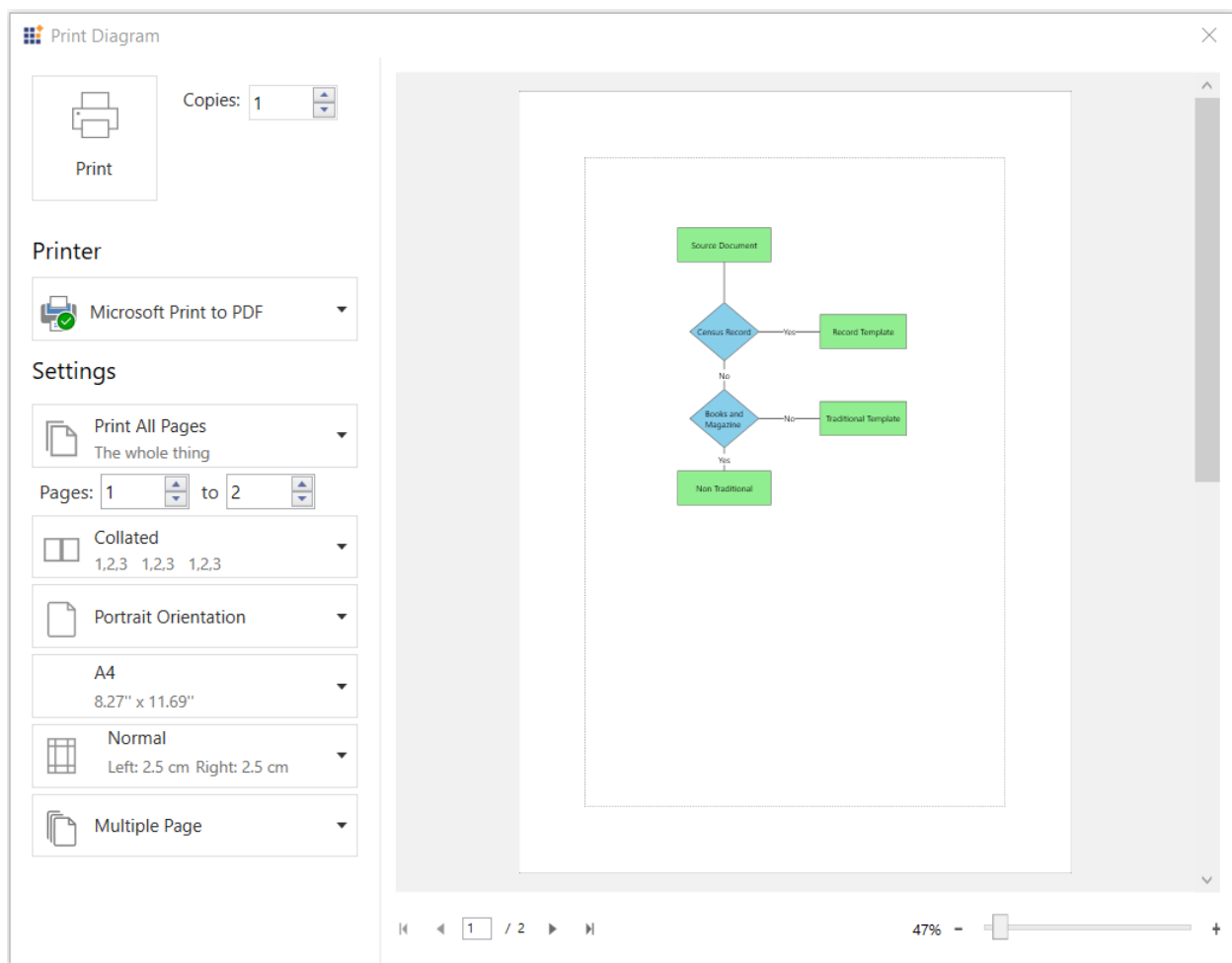
### Print preview

SfDiagram provides option to display print preview to review and customize the diagram in desired format before printing. Print preview window lets users to navigate through every page, zoom in and out on a page to determine the errors if any, which need to be resolved prior to printing.

Print preview window can be opened by setting [SfDiagram.PrintingService.ShowDialog](#) to true and calling the [SfDiagram.PrintingService.Print](#) method.

### C#

```
diagram.PrintingService.ShowDialog = true;  
diagram.PrintingService.Print();
```



### Print settings

SfDiagram provides various options to customize print preview settings using the [SfDiagram.PageSettings](#) and [PrintingService.PrintSettings](#) properties.

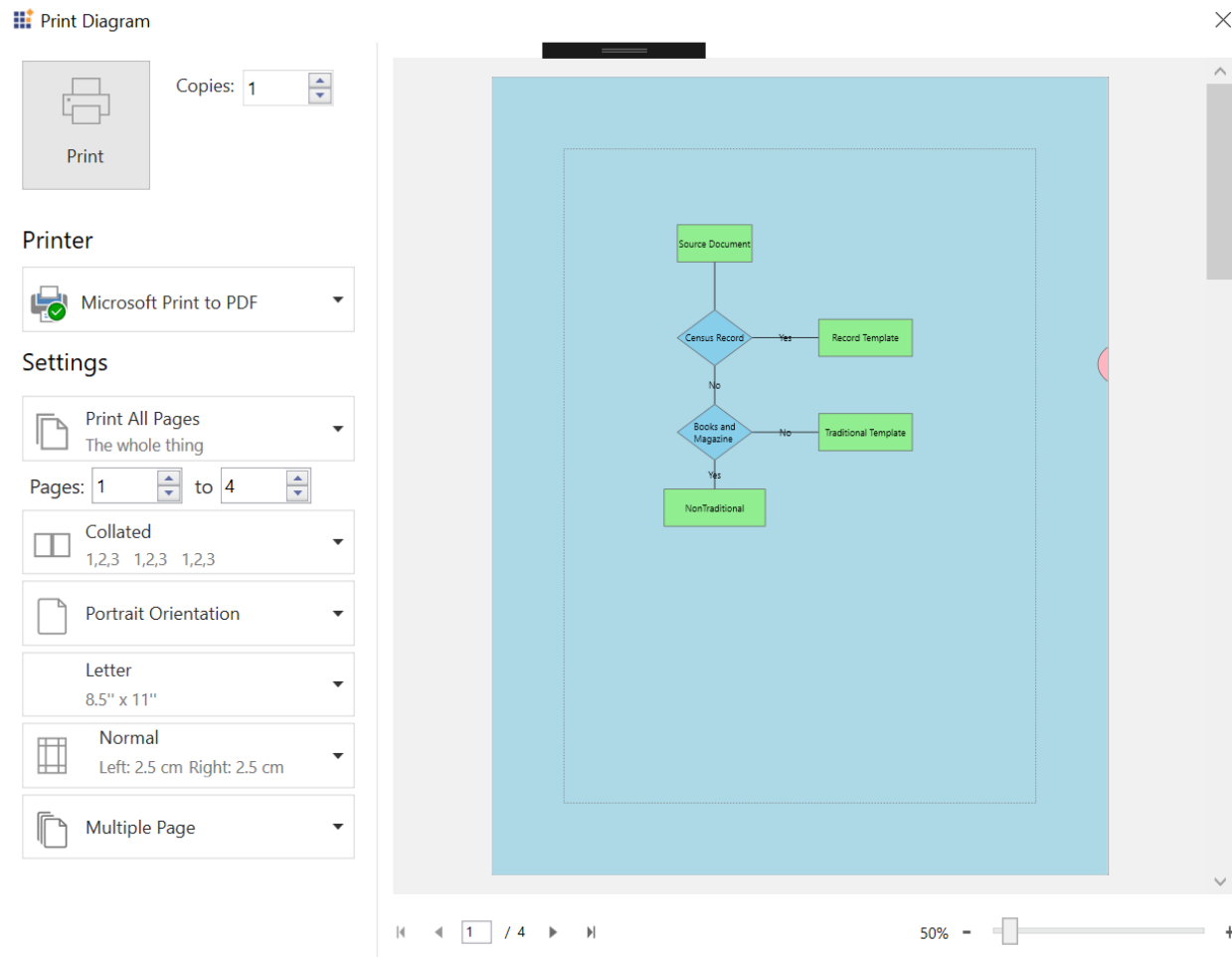
### C#

```
diagram.PageSettings=new PageSettings();  
diagram.PageSettings.PageWidth = 800;
```

```
diagram.PageSettings.PageHeight = 800;  
diagram.PageSettings.PageOrientation=PageOrientation.Landscape;  
diagram.PrintingService.PrintSettings.PageMargin=new Thickness(5);
```

### Print

To print a diagram from the selected printer, click the Print button at the top left corner of the Print Preview window. Also, Print Preview window has option to decide how many copies need to be printed.



**Note:** Copies will be effective only for real printers.

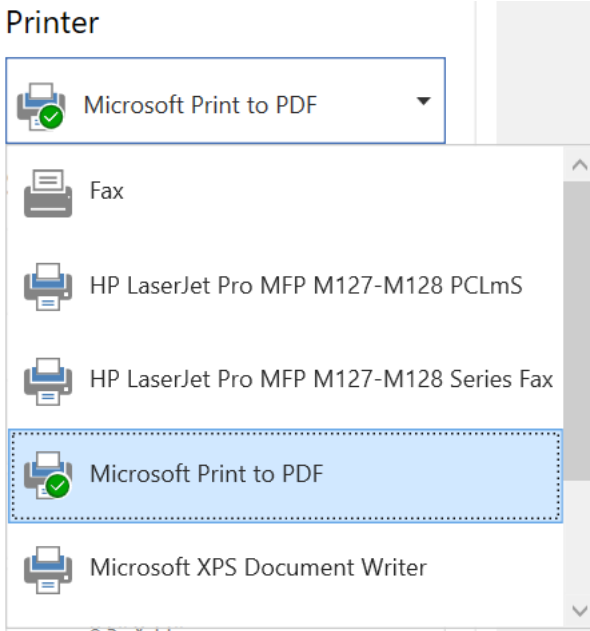
For sample, refer to [Printing](#).

### Printer

The printer option will list all the installed printers in your system. You can choose any printer from that list before start printing.

### How to save diagram in PDF Format

To save a diagram as PDF, choose any PDF printer (like "Microsoft Print To PDF") under Printer section.

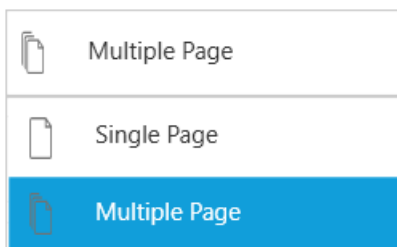


### Scaling

SfDiagram provides support to scale the diagram whether to print as single page or split into multiple pages. Scaling options can be changed by setting the `PrintingService.PrintManager.SelectedScaleIndex` property.

### C#

```
//Here, 0 denotes Single Page and 1 denotes Multiple Page.  
diagram.PrintingService.PrintManager.SelectedScaleIndex = 1;
```



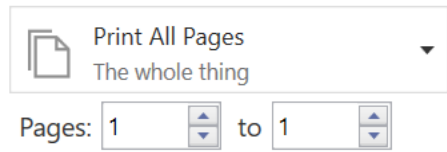
### PageRangeSelection

When the page scaling is multiple page, then you can decide whether to print all the pages or selected page range. Find the PageRangeSelection options below:

1) Print All Pages: All the pages to be printed. 2) Custom Print: A specified range of pages to be printed.

**PageRange** Allows you to specify a single page or a range of pages to be printed.

1) FromPage: Specifies the start page of printing. 2) ToPage: Specifies the end page of printing.



---

**Note:** Based on the FromPage and ToPage values, the PageRangeSelection option will change.

---

#### Collation

Collation is used to specify whether a printer collates output when it prints multiple copies of a multipage diagram.

1) Collate - Collated output. 2) UnCollate - UnCollated output.

---

**Note:** Collation applicable when more than one copy of multiple page diagram is printed in real printer.

---

#### Orientation

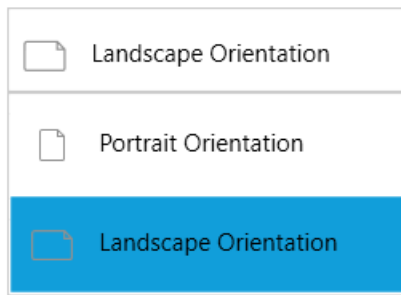
SfDiagram provides support to switch between Portrait and Landscape orientation while printing. Orientation can be changed by setting the `PageSettings.PageOrientation` Property.

#### C#

```
diagram.PageSettings=new PageSettings();  
diagram.PageSettings.PageOrientation = PageOrientation.Landscape;
```

The orientation can be changed in print preview window at runtime using orientation drop-down. When the value of orientation changed in print preview, then the same will reflect in Diagram orientation.

1) Portrait - Standard Orientation. 2) Landscape - Content of the imageable area is rotated on the page 90 degrees counterclockwise from standard (portrait) orientation.



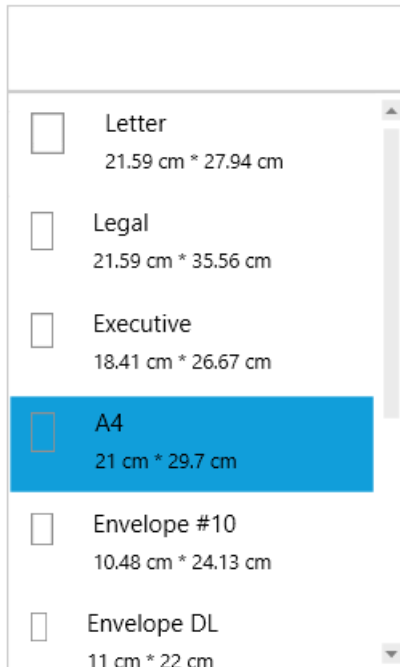
#### Paper size

SfDiagram provides support to change the page size. Page size can be changed by setting the `PageSettings.PageWidth` and `PageSettings.PageHeight` properties.

#### C#

```
diagram.PageSettings=new PageSettings();  
diagram.PageSettings.PageWidth = 800;  
diagram.PageSettings.PageHeight = 800;
```

Page size can be changed in print preview window also by selecting any value from the page-size drop-down, which displays the supported page sizes of a selected printer.



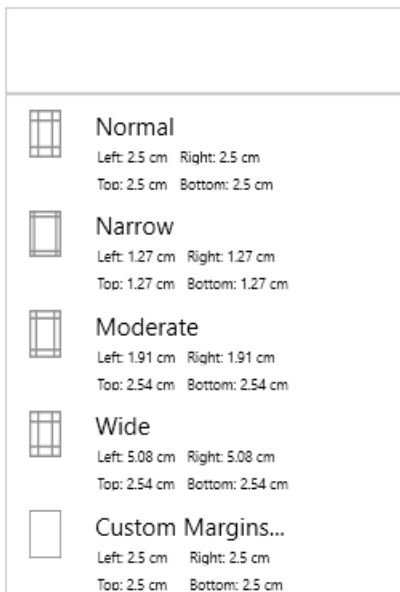
#### Page margin

SfDiagram provides support to change the page margins to adjust content in printed page. Page margin can be changed by setting `PrintingService.PrintSettings.PageMargin` property.

#### C#

```
diagram.PrintingService.PrintSettings.PageMargin=new Thickness(5);
```

Page margin can be changed in print preview also by selecting from predefined page margin from margin drop-down. You can manually enter custom margins in the editors below margin drop-down and press OK to apply the custom margin.



## Header and Footer

SfDiagram provides a way to display additional content at the top (Header) or bottom (Footer) of the page while printing. This can be achieved by setting the [PageHeaderHeight](#), [PageHeaderTemplate](#), [PageFooterHeight](#), and [PageFooterTemplate](#) properties in `PrintSettings`.

Steps for setting Header for printing:

1. Create `DataTemplate` in Resources and assign it to the `PageHeaderTemplate` and `PageFooterTemplate` properties.

### XML

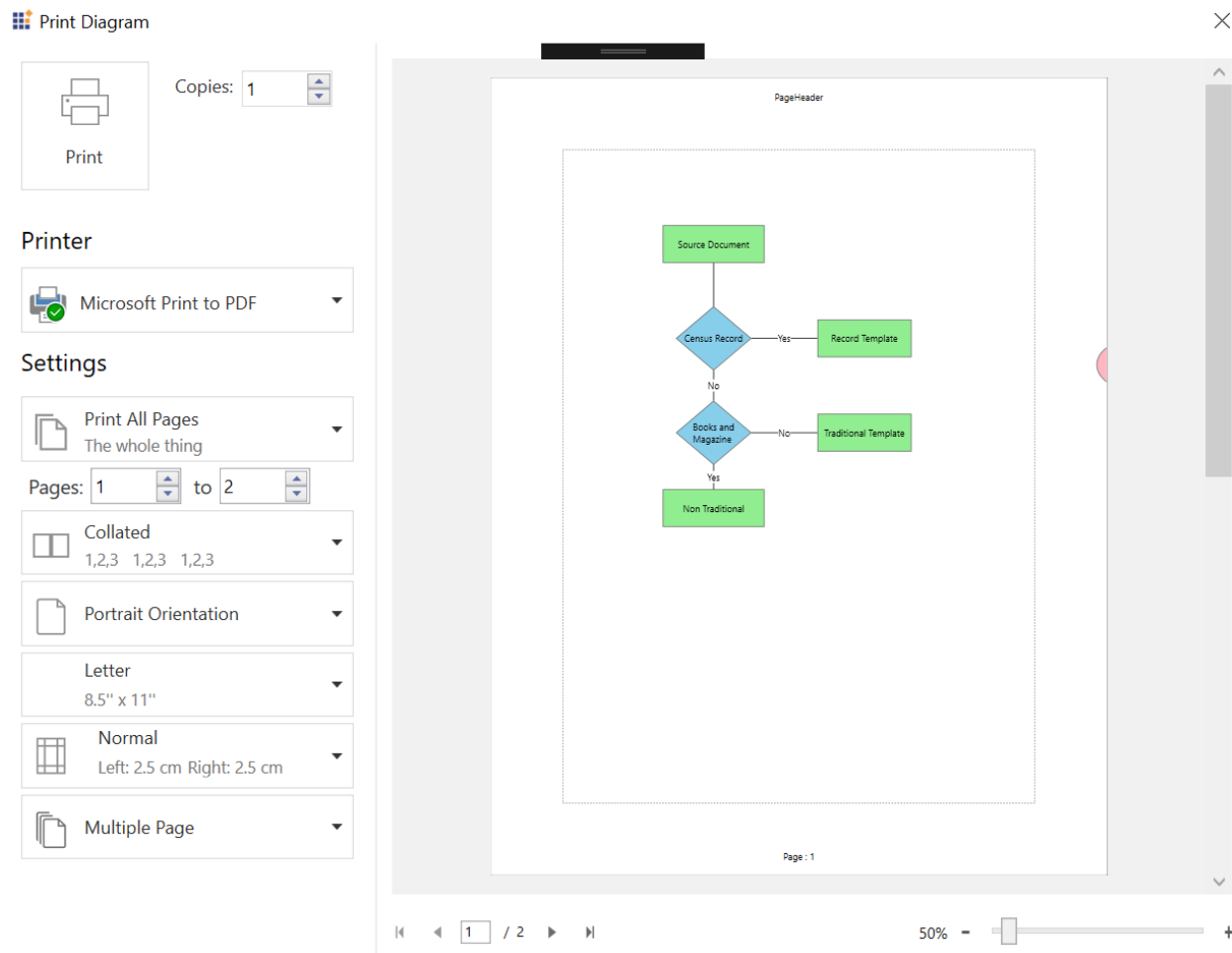
```
<DataTemplate x:Key="PrintHeaderTemplate">
<TextBlock Text="PageHeader" FontSize="12" Foreground="Black"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
</DataTemplate>
<DataTemplate x:Key="PrintFooterTemplate">
<TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
Foreground="Black" FontSize="12">
<TextBlock.Text>
<Binding Path="PageIndex"
RelativeSource="{RelativeSource Mode=FindAncestor, AncestorType={x:Type
Printing:PrintPageControl}}">
StringFormat="Page : {0}" />
</TextBlock.Text>
</TextBlock>
</DataTemplate>
```

2. Set the above defined `DataTemplate` to `PrintSettings.PageHeaderTemplate` and `PrintSettings.PageFooterTemplate`, then assign value for `PrintSettings.PageHeaderHeight` and `PrintSettings.PageFooterHeight` properties also.

### C#

```
diagram.PrintingService.PrintSettings.PageHeaderHeight = 50;
diagram.PrintingService.PrintSettings.PageHeaderTemplate =
this.Resources["PrintHeaderTemplate"] as DataTemplate;
diagram.PrintingService.PrintSettings.PageFooterHeight = 50;
diagram.PrintingService.PrintSettings.PageFooterTemplate
=this.Resources["PrintFooterTemplate"] as DataTemplate;
```

3. Now, run the application and you can see page header and footer in all the pages.



**Note:** [View Sample in GitHub](#)

### Skip empty pages

Sfdiagram provides support to skip the empty pages while navigating through preview and in printed document, thus reduces paper wastage.

The `GetPrintInfo` virtual method of `PrintingService` is used to cancel the empty pages. This method will execute for page navigation, printing each diagram pages, and changes made in the print preview area.

### C#

```
public class CustomPrintingService : PrintingService
{
    protected override void GetPrintInfo(PrintInfo args)
    {
        if (!(args.Elements as IEnumerable<object>).Any())
        {
            args.Cancel = true;
        }
        else
        {
            base.GetPrintInfo(args);
        }
    }
}
```



### Printing event

The **Printing** event will notify the different state of the printing with [PrintingEventArgs](#). The argument will provide the print dialog and print state value of the printing.

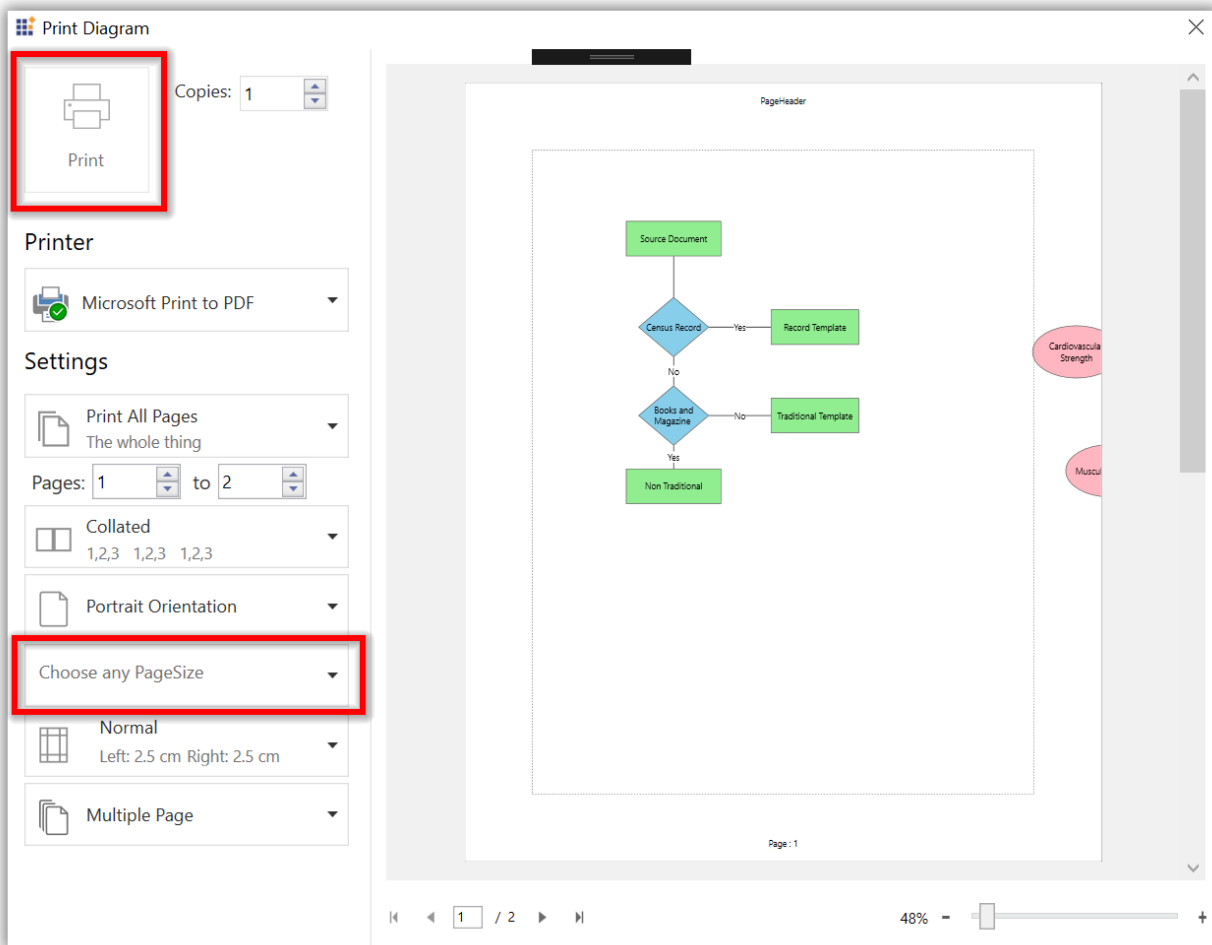
PrintStatus	Description
---	---
Started	Specifies that printing has been started.
Printing	Specifies the progress status of the printing.
Completed	Specifies the completed status of the printing.
PagePrepared	Specifies the completed status of the page preparation.
DocumentPrepared	Specifies the completed status of the document preparation.
Cancelled	Specifies the cancelled status of the printing.

### C#

```
(diagram.Info as IGraphInfo).Printing += MainWindow_Printing;  
private void MainWindow_Printing(object sender, PrintingEventArgs args)  
{  
}
```

### Custom paper size

Print preview window's page size drop down will display only the supported paper sizes of the selected printer. When the Diagram page size is not supported by the selected printer, then the Print button will be disabled and Page Size drop-down will ask you to choose the supported page size.



Using the `OnSelectedPrinterChanged` virtual method of `DiagramPrintManager`, you can add the custom pager sizes to the selected printer apart from the default supported paper sizes. Refer to the following code example.

### C#

```
public class CustomDiagramPrintManager : DiagramPrintManager
{
    public CustomDiagramPrintManager(PrintingService printingService) :
    base(printingService)
    {
    }
    public override void OnSelectedPrinterChanged(PrintQueue printQueue)
    {
        if (printQueue.Name.Contains("Microsoft Print to PDF"))
        {
            List<string> pagesizeName = PageSizeOptions.Select(c =>
            c.PageSizeName).ToList();
            if (!(pagesizeName.Contains("Ansi B")))
            {
                PageSizeOptions.Add(new Syncfusion.Windows.Controls.Printing.PrintPageSize()
                { PageSizeName = "Ansi B", Size = new Size(1055, 1632) });
            }
            if (!(pagesizeName.Contains("Ansi C")))
            {
                PageSizeOptions.Add(new Syncfusion.Windows.Controls.Printing.PrintPageSize()
                { PageSizeName = "Ansi C", Size = new Size(1055, 1632) });
            }
        }
    }
}
```

```
{
    PageSizeOptions.Add(new Syncfusion.Windows.Controls.Printing.PrintPageSize()
    { PageSizeName = "Ansi C", Size = new Size(1632, 2112) });
}
if (!(pagesizename.Contains("Ansi D")))
{
    PageSizeOptions.Add(new Syncfusion.Windows.Controls.Printing.PrintPageSize()
    { PageSizeName = "Ansi D", Size = new Size(2112, 3264) });
}
if (!(pagesizename.Contains("A0")))
{
    PageSizeOptions.Add(new Syncfusion.Windows.Controls.Printing.PrintPageSize()
    { PageSizeName = "A0", Size = new Size(3179, 4494) });
}
}
}
}
}
}
public class CustomPrintingService : PrintingService
{
    public CustomPrintingService()
    {
        this.PrintManager = new CustomDiagramPrintManager(this);
    }
}
```

Here, the **Microsoft Print to PDF** printer won't support Ansi B, Ansi C, Ansi D, and A0 paper sizes by default, so the above code will add the Ansi B, Ansi C, Ansi D, and A0 paper sizes in the Print Preview window's page size combo box.

However, we have added custom paper sizes manually in the page size drop-down. The printer won't print the diagram in the custom paper size, to print the diagram in custom page size, set the chosen custom paper size to **PrintDialog.PrintTicket.PageMediaSize** to true and enable **CanUseCustomPageMediaSize** in the printing event when the printing state is started. Refer to the following code example.

#### C#

```
private void MainWindow_Printing(object sender, PrintingEventArgs args)
{
    if (args.PrintState == PrintStatus.Started)
    {
        var customPages = new System.Collections.Generic.Dictionary<string, Size>();
        var printerName = args.PrintDialog.PrintQueue.Name;
        if (printerName.Contains("Microsoft Print to PDF"))
        {
            customPages.Add("Ansi B", new Size(1055, 1632));
            customPages.Add("Ansi C", new Size(1632, 2112));
            customPages.Add("Ansi D", new Size(2112, 3264));
        }
        else if (printerName.Contains("Microsoft XPS Document Writer"))
        {
            customPages.Add("A0", new Size(3179, 4494));
        }
        foreach (var customPage in customPages)
        {

```

```
if (args.SelectedPageMediaSizeName.Contains(customPage.Key))
{
    var pageSize = customPage.Value;
    var mediaSize = new PageMediaSize(PageMediaSizeName.Unknown, pageSize.Width,
    pageSize.Height);
    args.PrintDialog.PrintTicket.PageMediaSize = mediaSize;
    args.CanUseCustomPageMediaSize = true;
    break;
}
}
```

---

**Note:** Custom page size print is not applicable for any real printer. When you choose any custom paper size and try to print it in any real printer, then the printer will print the next supported paper size instead of the chosen one. The above mentioned option is only applicable for "Microsoft Print to PDF" printer.

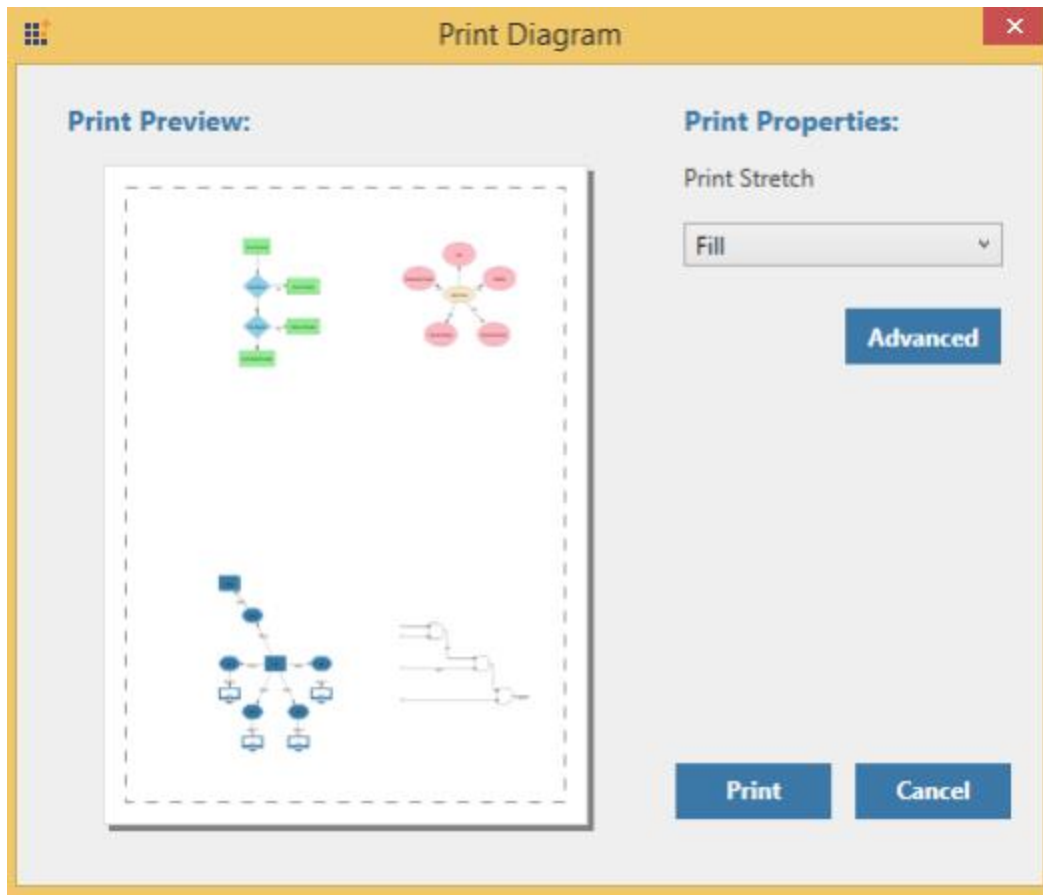
---

#### Classic PrintPreview

SfDiagram provides backward compatibility to show the older or classic print preview by calling the `ShowClassicPrintPreview` method instead of `Print()` method.

#### C#

```
diagram.PrintingService.ShowDialog = true;
diagram.PrintingService.ShowClassicPrintPreview();
```



See Also

- [How to customize the Header or Footer of the print preview?](#)

### Exporting in WPF Diagram (SfDiagram)

SfDiagram provides the support to export its content as image/XPS files using the [Export](#) method.

SfDiagram can be exported in the following File formats.

- PNG
- JPEG
- TIFF
- GIF
- BMP
- WDP
- XPS

The following code explains how to export the diagram as image.

#### C#

```
//Initialize the diagram  
SfDiagram diagram = new SfDiagram();  
//Method to export the SfDiagram
```

```
diagram.Export();
```

### Export settings

SfDiagram provides various options to customize the exported diagram using the [SfDiagram.ExportSettings](#) property of type [ExportSettings](#).

### Image format

You can use the [ExportBitmapEncoder](#) or [ExportType](#) properties to specify the type/format of the exported image file.

### C#

```
//Initialize the diagram
SfDiagram diagram = new SfDiagram();
//Specify the file format of the image
diagram.ExportSettings.ExportType = ExportType.PNG;
//Method to export the SfDiagram
diagram.Export();
```

### Image file name

You can save the exported image as stream or file system using the [ExportStream](#) or [FileName](#) properties of [ExportSettings](#) class respectively.

### C#

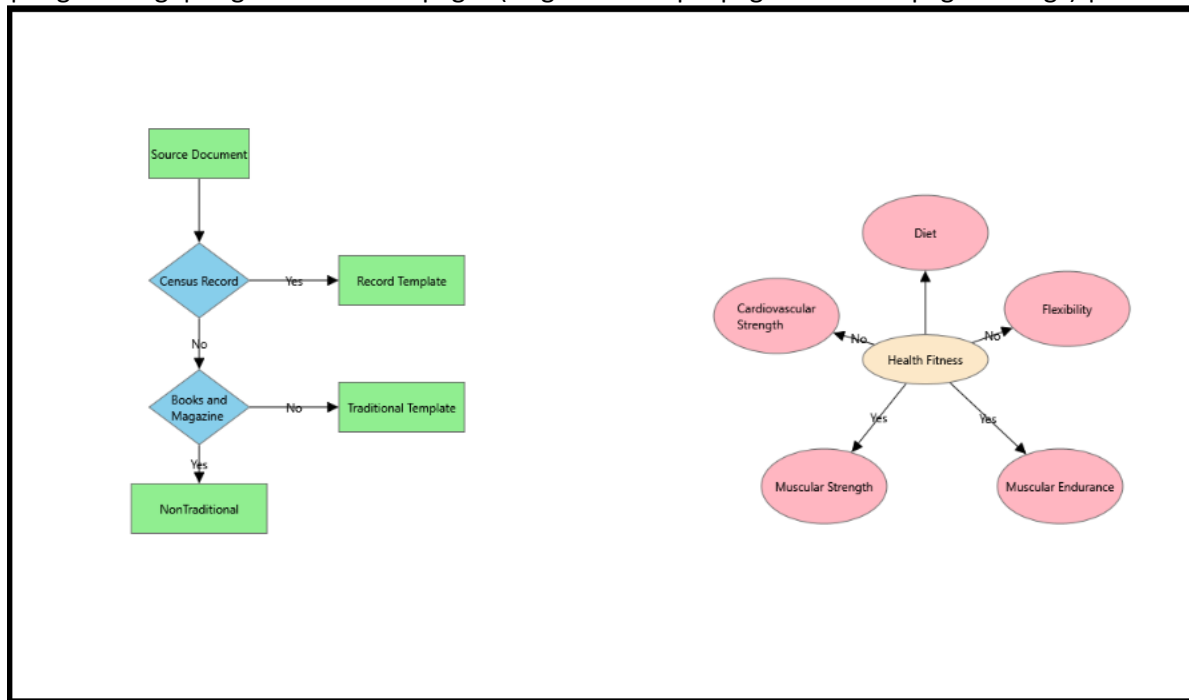
```
//Initialize the export settings
ExportSettings settings = new ExportSettings()
{
    FileName = "export.png",
};
diagram.ExportSettings = settings;
//Method to export the SfDiagram
diagram.Export();
```

### ExportMode

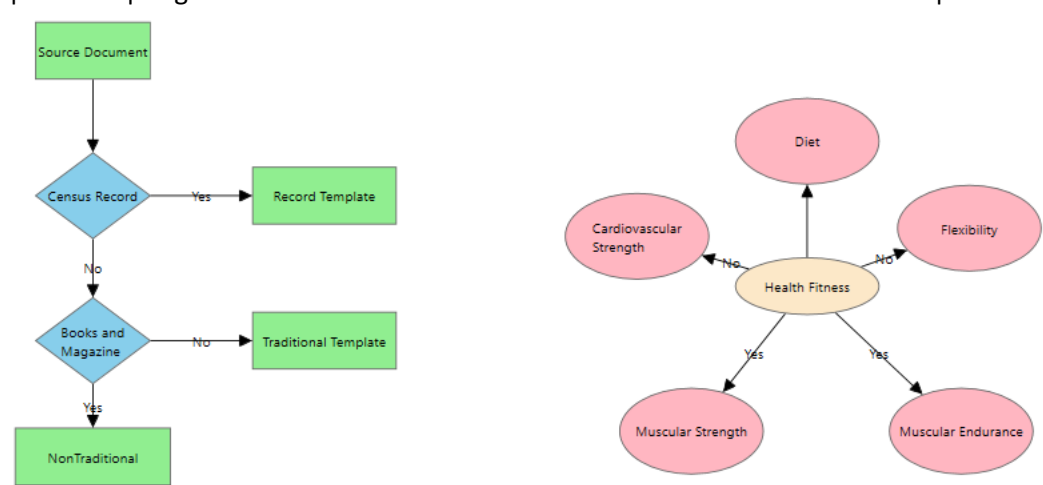
[ExportMode](#) specifies whether the complete page region of the diagram is to be exported or the content region alone. The exporting options are as follows:

ExportMode	Description	Output
---	---	---

| PageSettings | Region that fits all pages (single or multiple pages based on page settings) |



| Content | Region that fits all nodes and connectors that are added to model |



### XML

```
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Initialize the export settings-->
  <syncfusion:SfDiagram.ExportSettings>
    <syncfusion:ExportSettings ExportMode="PageSettings"/>
  </syncfusion:SfDiagram.ExportSettings>
</syncfusion:SfDiagram>
```

### C#

```
//Initialize the export settings
```

```
ExportSettings settings = new ExportSettings()
{
    ExportMode = ExportMode.PageSettings,
};
diagram.ExportSettings = settings;
//Method to export the SfDiagram
diagram.Export();
```

#### Export to XPS

SfDiagram has in-built support to export the diagram as XPS file instead of image file. To export diagram as XPS file, set the `IsSaveToXps` property of `ExportSettings` class to `true` and specify the file name with ".xps" extension.

#### C#

```
//Initialize the export settings
ExportSettings settings = new ExportSettings()
{
    IsSaveToXps = true,
    FileName = "export.xps",
};
diagram.ExportSettings = settings;
//Method to export the SfDiagram
diagram.Export();
```

#### Export to PDF

SfDiagram does not have the built-in support to convert the diagram to PDF file, but you can achieve this by exporting the diagram as XPS file and then convert the exported XPS file to PDF using [Syncfusion.XPS.XPSToPdfConverter](#).

#### Export specific region of diagram

SfDiagram provides the supports to export any specific region of the diagram by using the 'Clip' property of `ExportSettings` class.

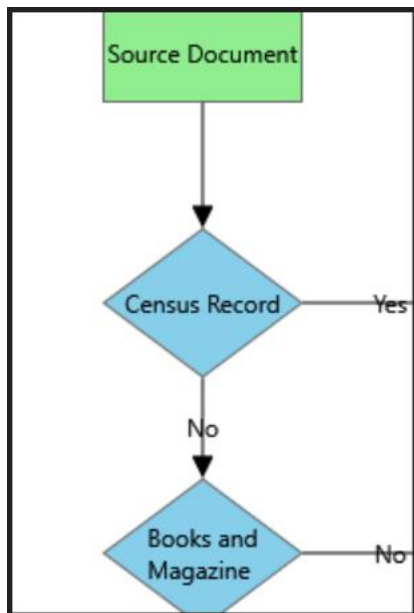
#### XML

```
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
<!--Initialize the export settings with clipping area-->
<syncfusion:SfDiagram.ExportSettings>
<syncfusion:ExportSettings Clip="200, 200, 200, 300"/>
</syncfusion:SfDiagram.ExportSettings>
</syncfusion:SfDiagram>
```

#### C#

```
//Initialize the export settings with clipping area
ExportSettings settings = new ExportSettings()
{
    Clip = new Rect(200, 0, 200, 500),
};
diagram.ExportSettings = settings;
//Method to export the SfDiagram
diagram.Export();
```





*Change the background of the exported files*

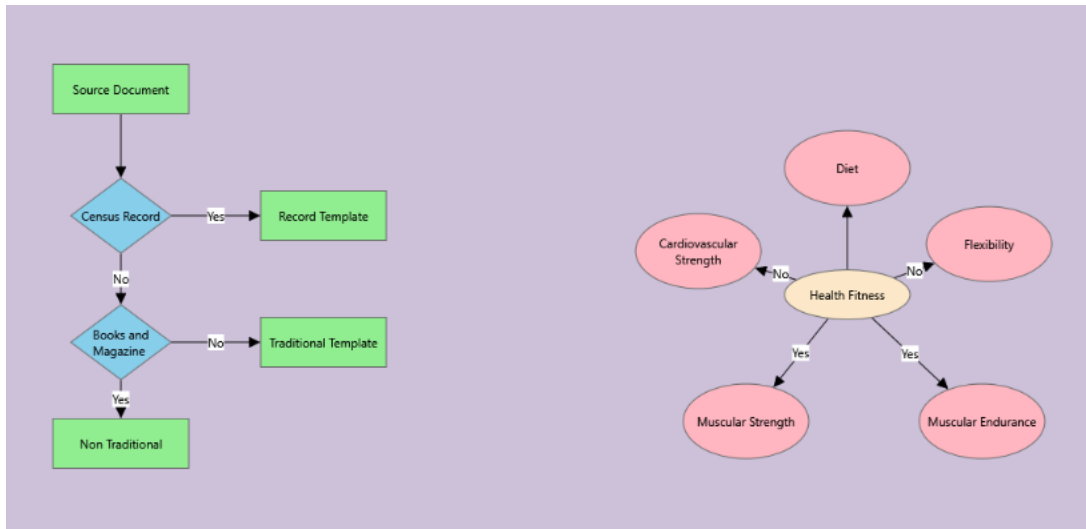
SfDiagram provides the supports to change the background color of the exported image using the 'ExportBackground' property of `ExportSettings` class.

#### XML

```
<!--Initialize the SfDiagram-->
<syncfusion:SfDiagram x:Name="diagram">
  <!--Initialize the export settings with clipping area-->
  <syncfusion:SfDiagram.ExportSettings>
    <syncfusion:ExportSettings ExportBackground="Blue"/>
  </syncfusion:SfDiagram.ExportSettings>
</syncfusion:SfDiagram>
```

#### C#

```
//Initialize the export settings with clipping area
ExportSettings settings = new ExportSettings()
{
    ExportBackground = new SolidColorBrush(Colors.Blue),
};
diagram.ExportSettings = settings;
//Method to export the SfDiagram
diagram.Export();
```



[View Sample in GitHub](#)

See Also

[How to export the Diagram as PDF?](#)

## Localization in WPF Diagram (SfDiagram)

Localization is the process of configuring the application to a specific language. SfDiagram provides support to localize all the static text used for annotation and context menu contents.

Name	Value
BoardDecision	Conseil décide de procéder
ContextMenu_Copy	Exemplaire
ContextMenu_Cut	Couper
ContextMenu_Paste	Pâte
ContextMenu_SelectAll	Sélectionner tout
Decision	Processus de nouvelles idées de logiciels
End	Mettre en œuvre et Livrer
Meeting	Rencontre avec carte
NewIdea	Identifié nouveau idée
No	Aucun
project	Spécification d'écriture
Reject	Rejeter et écrire rapport
Resources	Embaucher de nouveaux Ressources
ToolTip_Delete	Effacer
ToolTip_Draw	Dessiner
ToolTip_Duplicate	Dupliquer
Yes	Oui

You can refer [Localization](#) to add [resource file](#) in the application.

## Localize the Annotations using ResourceManager

SfDiagram has support to localize the Annotations of Nodes/Connector. The following code illustrates how to provide localization support for Annotation.

### XML

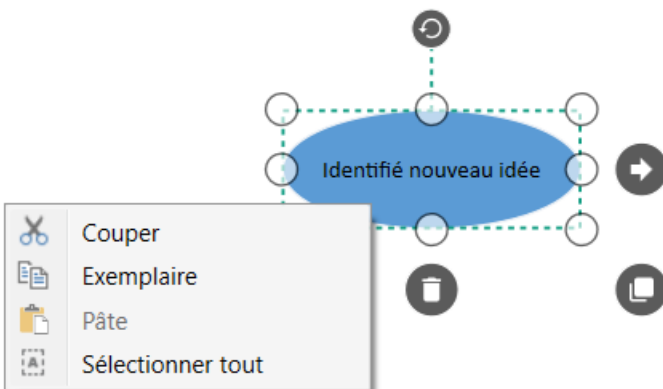
```

<!--Style for Node-->
<Style TargetType="Path" x:Key="Nodestyle">
  <Setter Property="Stretch" Value="Fill"/>
  <Setter Property="Fill" Value="#FF5B9BD5"/>
</Style>
  
```

```
</Style>
```

**C#**

```
public MainWindow()
{
    System.Resources.ResourceManager manager;
    //Get CultureInfo
    System.Threading.Thread.CurrentThread.CurrentUICulture = new
    System.Globalization.CultureInfo("fr");//French
    //Initialize Assembly
    Assembly assembly = Application.Current.GetType().Assembly;
    manager = new
    System.Resources.ResourceManager("Localization.Resources.Syncfusion.SfDiagram.WPF", assembly);
    InitializeComponent();
    //Creating Diagram
    SfDiagram diagram = new SfDiagram();
    //Adding diagram to the main grid of the mainwindow.
    RootGrid.Children.Add(diagram);
    diagram.Nodes = new ObservableCollection<NodeViewModel>();
    NodeViewModel node = new NodeViewModel();
    node.UnitWidth = 150;
    node.UnitHeight = 60;
    node.OffsetX = 345;
    node.OffsetY = 140;
    node.Annotations = new ObservableCollection<IAnnotation>()
    {
        new AnnotationEditorViewModel()
        {
            //localizing annotation using ResourceManager
            Content = manager.GetString("NewIdea"),
        }
    };
    node.Shape = new EllipseGeometry() { RadiusX = 100, RadiusY = 100 };
    node.ShapeStyle = this.Resources["Nodestyle"] as Style;
    (diagram.Nodes as ObservableCollection<NodeViewModel>).Add(node);
}
```



**Note:** You have to define the textual descriptions of the context menu items for your custom cultures.

[View Sample in GitHub](#)

## Tools in WPF Diagram (SfDiagram)

### 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 – ContinuousDraw / 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 are applied, a tool is activated based on the precedence given in the following table.

Tools	Description
ContinuousDraw	Allows you to draw the Nodes or Connectors continuously. Once it is activated, you cannot perform any other interaction in the Diagram.
DrawOnce	Allows you to draw single Node or Connector. Once you complete the DrawOnce action, SingleSelect and MultipleSelect tools are automatically enabled.
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.
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.
SingleSelect	Allows you to select individual or Connectors.
None	Disables all tools.

You can set the desired tool to the [Tool](#) property of the Diagram. The following code illustrates how to enable single/multiple tools.

### C#

```
//To Enable Single Tool
diagram.Tool = Tool.SingleSelect;
//To Enable multiple tools
diagram.Tool = Tool.SingleSelect | Tool.ZoomPan;
```

**Note:** [View Sample in GitHub](#)

### Drawing Tools

[DrawingTool](#) allow you to draw any kind of node/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 **Tool** property and you need to set the event for **GetDrawType**.

#### XML

```
<Style TargetType="Path" x:Key="shapestyle">
<Setter Property="Fill" Value="#fbe172"/>
<Setter Property="Stroke" Value="Black"/>
<Setter Property="Stretch" Value="Fill"/>
</Style>
<Style TargetType="{x:Type diagram:Node}">
<Setter Property="Shape" Value="M13.560 67.524 L 21.941 41.731 L 0.000
25.790 L
27.120 25.790 L 35.501 0.000 L 43.882 25.790 L 71.000
25.790 L 49.061 41.731 L 57.441 67.524 L 35.501
51.583 z"/>
<Setter Property="ShapeStyle" Value="{StaticResource shapestyle}"/>
</Style>
```

#### C#

```
//GetDrawType event is used to specify which item have to be drawn by the
user.
(diagram.Info as IGraphInfo).GetDrawType += MainWindow_GetDrawType;
diagram.DrawingTool = DrawingTool.Node;
diagram.Tool = Tool.ContinuesDraw;
private void MainWindow_GetDrawType(object sender, DrawTypeEventArgs args)
{
args.DrawItem = new TextBlock()
{
Text="Path",
HorizontalAlignment = HorizontalAlignment.Center,
VerticalAlignment = VerticalAlignment.Center
};
}
```



- **GetDrawType** event will invoke when start drawing and get **DrawItem** (i.e which item you will draw) from the user. To explore about arguments, please refer to the [DrawTypeEventArgs](#).

### Text

Diagram allows you to create a text Node as soon as you click on the Diagram page. The following code illustrates how to draw a text.

#### XML

```
<Syncfusion:SfDiagram x:Name="diagram" Tool="ContinuesDraw"
DrawingTool="TextNode">
<Syncfusion:SfDiagram.Nodes>
<Syncfusion:NodeCollection />
</Syncfusion:SfDiagram.Nodes>
<Syncfusion:SfDiagram.Connectors>
<Syncfusion:ConnectorCollection />
</Syncfusion:SfDiagram.Connectors>
</Syncfusion:SfDiagram>
```

**C#**

```
diagram.DrawingTool = DrawingTool.TextNode;
diagram.Tool = Tool.ContinuesDraw;
```

*Connectors*

To draw Connectors, you have to set the Connector to **DrawingTool** property. The **DrawingTool** can be activated by using the **Tool** property as shown. The following code example illustrates how to draw a straight line Connector.

**XML**

```
<Style x:Key="decoratorstyle" TargetType="Path">
<Setter Property="Stroke" Value="Black" />
<Setter Property="Fill" Value="Black" />
<Setter Property="StrokeThickness" Value="1" />
</Style>
<Style TargetType="Path" x:Key="connectorstyle">
<Setter Property="Stroke" Value="Black"></Setter>
<Setter Property="StrokeThickness" Value="2"></Setter>
</Style>
<Style TargetType="{x:Type diagram:Connector}">
<Setter Property="TargetDecoratorStyle" Value="{StaticResource
decoratorstyle1}" />
<Setter Property="ConnectorGeometryStyle" Value="{StaticResource
connectorstyle}" />
</Style>
```

**C#**

```
diagram.DrawingTool = DrawingTool.Connector;
diagram.Tool = Tool.DrawOnce;
```

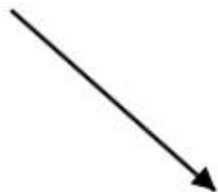


Diagram allows you to establish connection with Node/Port as soon as you click on the Node/Port.

- **ObjectDrawn** event will invoke with drawing state. To explore about arguments, please refer to the [ObjectDrawnEventArgs](#).

### FreeHand drawing

Free hand connections will be drawn by using **DrawingTool** property.

### XML

```
<Syncfusion:SfDiagram x:Name="diagram" Tool="ContinuesDraw"
DrawingTool="FreeHand">
<Syncfusion:SfDiagram.Nodes>
<Syncfusion:NodeCollection />
</Syncfusion:SfDiagram.Nodes>
<Syncfusion:SfDiagram.Connectors>
<Syncfusion:ConnectorCollection />
</Syncfusion:SfDiagram.Connectors>
</Syncfusion:SfDiagram>
```

### C#

```
// Enable the FreeHand drawing
diagram.DrawingTool = DrawingTool.FreeHand;
```

[FreeFormEvent](#) will notify the current drawing Connector and drawing State in [FreeFormDrawingEventArgs](#).

### Ellipse

Diagram allows you to create a ellipse shaped node as soon as you click and drag on the Diagram page. The following code illustrates how to draw an ellipse shaped node.

### XML

```
<Syncfusion:SfDiagram x:Name="diagram" Tool="ContinuesDraw"
DrawingTool="Ellipse">
<Syncfusion:SfDiagram.Nodes>
<Syncfusion:NodeCollection />
</Syncfusion:SfDiagram.Nodes>
```

```
<Syncfusion:SfDiagram.Connectors>
<Syncfusion:ConnectorCollection />
</Syncfusion:SfDiagram.Connectors>
</Syncfusion:SfDiagram>
```

**C#**

```
diagram.DrawingTool = DrawingTool.Ellipse;
diagram.Tool = Tool.ContinuesDraw;
```

*Rectangle*

Diagram allows you to create a rectangle shaped node as soon as you click and drag on the Diagram page. The following code illustrates how to draw a rectangle shaped node.

**XML**

```
<Syncfusion:SfDiagram x:Name="diagram" Tool="ContinuesDraw"
DrawingTool="Rectangle">
<Syncfusion:SfDiagram.Nodes>
<Syncfusion:NodeCollection />
</Syncfusion:SfDiagram.Nodes>
<Syncfusion:SfDiagram.Connectors>
<Syncfusion:ConnectorCollection />
</Syncfusion:SfDiagram.Connectors>
</Syncfusion:SfDiagram>
```

**C#**

```
diagram.DrawingTool = DrawingTool.Rectangle;
diagram.Tool = Tool.ContinuesDraw;
```

*How to override the default tool of diagram elements*

Each objects in diagram control have default actions while interact on them. Those default actions can be customized by overriding the virtual method [SetTool](#) of the [SfDiagram](#) class. The [SetTool](#) method takes the [SetToolArgs](#) as an argument that is used to know the objects under the mouse when modifying the tools of them.

- [Source](#) – To know the object on which item the mouse is interacting.
- [Action](#) - To customize the tools of the diagram object.

**XML**

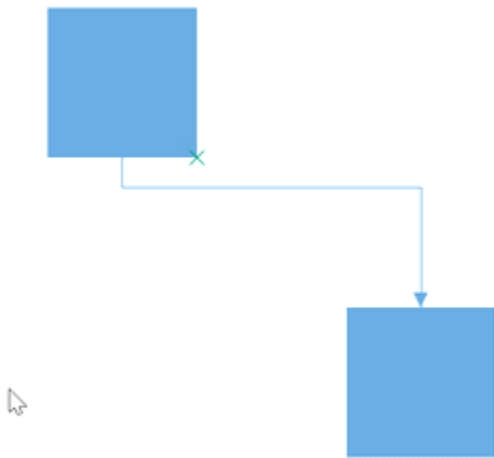
```
<!--Create new diagram for custom diagram class-->
<local:CustomClass PortVisibility="Visible" x:Name="diagram"/>
```

**C#**

```
//Create custom class of SfDiagram to override the tools.
public class CustomClass:SfDiagram
{
//Override method to customize the default tools of diagram objects
protected override void SetTool(SetToolArgs args)
```



```
{  
    if (args.Source is INode)  
    {  
        args.Action = ActiveTool.Pan;  
    }  
    else if (args.Source is IConnector)  
    {  
        args.Action = ActiveTool.Draw;  
    }  
    else if (args.Source is IPort)  
    {  
        args.Action = ActiveTool.Drag;  
    }  
    else  
    {  
        base.SetTool(args);  
    }  
}
```



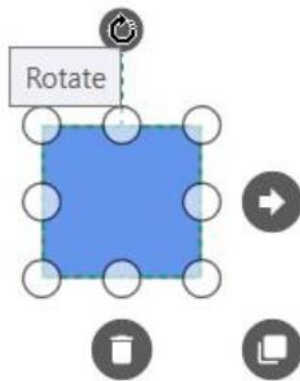
---

**Note:** [View Sample in GitHub](#)

---

#### How to override the default cursors while interaction

While mouse hovers on the diagramming objects, different cursors will be appearing on each object for different actions. For example, when we mouse hover on the rotator thumb, then rotator cursor will be shown.



These cursors can be customized by overriding the virtual method [SetCursor\(\)](#) of the [SfDiagram](#) class. The [SetCursor\(\)](#) method takes the [SetCursorArgs](#) as an argument that is used to know the objects under the mouse cursor when modifying the cursors of them.

- [Action](#) – To know the action tool of the element.
- [ControlPointType](#) – To know the control point of the object.
- [Cursor](#) – To customize the cursor of the object.
- [Source](#) – To know the object on which item the mouse is interacting.
- [SourceType](#) – To know the parent element of the object.

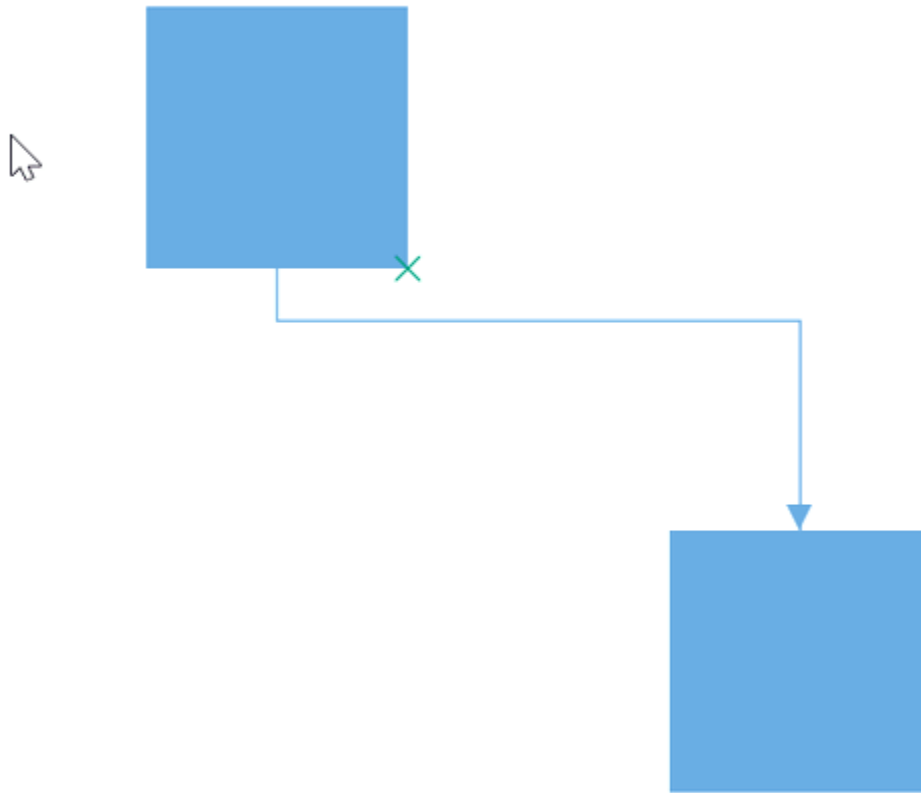
### XML

```
<!--Create new diagram for custom diagram class-->
<local:CustomClass PortVisibility="Visible" x:Name="diagram"/>
```

### C#

```
//Create custom class of SfDiagram to override the cursor.
public class CustomClass:SfDiagram
{
    //Override method to customize the default cursors of diagram objects
    protected override void SetCursor(SetCursorArgs args)
    {
        if (args.Source is INode)
        {
            args.Cursor = Cursors.No;
        }
        else if (args.Source is IConnector)
        {
            args.Cursor = Cursors.Hand;
        }
        else if (args.Source is IPort)
        {
            args.Cursor = Cursors.SizeAll;
        }
        else
        {
            base.SetCursor(args);
        }
    }
}
```

```
}  
}
```



---

**Note:** [View Sample in GitHub](#)

---

[How to override the default cursors while interaction?](#)

### Constraints in WPF Diagram (SfDiagram)

**Constraints** are used to enable/disable certain behaviors of the [diagram](#), [node](#), [connector](#), [port](#) and [annotation](#). Constraints are provided as flagged enumerations, so that multiple behaviors can be enabled/disabled with bitwise operators (&, |, ~, <<, etc.).

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

#### Graph Constraints

[GraphConstraints](#) allows to enable or disable the following behaviors. By default, [Zoomable](#), [Pannable](#), [PanRails](#), [Relationship](#), [Events](#), [AutoScroll](#), [PageEditing](#) constraints are enabled for diagram.

| Constraints | Description |

|--|--|

|AllowPan | Enables or disables diagram panning|

|AutomaticPortCreation | Enables or disables the automatic port creation while start or end the connection on or over the node or connector|

|AutoScroll | Enables or disables the autoscrolling behavior of diagram|

|Bridging | Enables or disables the line bridging while two connectors intersecting with each other.|

|Commands | Enables or disables the default diagram commands such as.|

|Connectable | Enables or disables the connector creation in the diagram.|

|ContextMenu | Enables or disables the opening of context menu during right-clicking on diagram or node or connector.|

|Default | Enables or disables all the default behaviors of the diagram.|

|Draggable | Enables or disables the drag of element from one diagram to the other.|

|DrawingTool | Enables or disables to draw any kind of node/connector during runtime by clicking and dragging on the Diagram page.|

|Drop | Enables or Disables the dropping of element from one diagram to the other.|

|Events | Enables or disables all events of the control.|

|FloatElements | Enables or disables the dragging of objects between multiple Diagrams.|

|None | Disable all SfDiagram constraints|

|Outline | Enable or disables the outline of diagram elements while loading.|

|PageEditing| Enables or disables the page editing options such as Selectable, Draggable, Connectable, Drop, Resizable, Rotatable, ContextMenu, DrawingTool, Commands.|

|Pannable | Enables or disables the panning of diagram over both X and Y axis.|

|PannableX | Enables or disables the panning of diagram over X-axis.|

|PannableY | Enables or disables the panning of diagram over Y -axis.|

|PanRails | Enables or disables the panning actions on the x-axis (horizontal panning) and y-axis (vertical panning) in SfDiagram.|

|PanRailsX | Enables or disables the pan rails of diagram in X-axis|

|PanRailsY | Enables or disables the pan rails of diagram in Y -axis|

|Relationship | Enables or disables the properties based on Node and Connector relationships on dragging at run time.|

|Resizable | Enables or disables the resize action on diagram or node.|

|Rotatable | Enables or disables the rotation action on diagram or node.|

|Routing | Enables or disables the line routing when node intersects with the connector.|

Selectable	Enables or disable the selection action of diagram and its child elements.
Undoable	Enables or disables the undo or redo action on diagram.
Virtualize	Enables or disables the virtualizing behavior of diagram.
Zoomable	Enables or disables the zooming of diagram.

The following code example illustrates how to disable page editing.

#### C#

```
diagram.Constraints = GraphConstraints.Default &
~GraphConstraints.PageEditing;
```

#### Node Constraints

[NodeConstraints](#) allows to enable or disable the following behaviors of Node. By default Selectable, Connectable, Inherit, RoutingObstacle, PivotDraggable, Delete and ThemeStyle constraints are enabled for Node.

Constraints	Description
-- --	
AllowDrop	Enables or disables interaction events(DragEnter,DragOver,ItemDrop and DragLeave) for diagram nodes.
AllowPan	Enables or disables the panning action for node.
AspectRatio	Enables or disables the node to be resized proportionally.
AutomaticPortCreation	Enables or disables the automatic port creation while start or end the connection on or over the node
Connectable	Enables or disables the node to be connected with connector.
Default	Enables Selectable, Draggable, Resizable, Rotatable, Connectable, Inherit, RoutingObstacle, PivotDraggable, Delete, ThemeStyle constraints.
Delete	Enables or disables the delete operation on node.
DragAnnotation	Enables or disables the node's annotation to be draggable.
Draggable	Enables or disables the node to be dragged from one point to other.
DynamicPortConnection	Enable or disables to create connector with port point when mouse hover on Node's bounds.
ExcludeFromLayout	Enables or disables to exclude the node from layout.
InConnect	Enables or disables the connection creation from or to the incoming Connector.
Inherit	Enables or disables to inherit all the Dragging, Resizing, Rotating, Snapping, SnapToObject, and PortVisibility from diagram.
InheritAllowPan	Enables or disables a node to pan from graph level

InheritAutomaticPortCreation	Enables or disables to inherit the value for automatic port creation from diagram.
InheritDraggable	Enables or disables to inherit the value for draggable from diagram.
InheritHitPadding	Enables or disables to inherit the value for hit-padding from diagram.
InheritMenu	Enables or disables to inherit the context menu from diagram.
InheritPortVisibility	Enables or disables to inherit the value of SfDiagram.PortVisibility.
InheritResizable	Enables or disables to inherit the value for resizable from diagram.
InheritRotatable	Enables or disables to inherit the value for rotatable from diagram.
InheritSnapping	Enables or disables to inherit the value of snap-to-lines and snap angle from diagram by using SnapSettings.SnapConstraints property value.
InheritSnapToObject	Enables or disables to inherit the value to SnapToObject from diagram by using SnapSettings.SnapConstraints property value.
Menu	Enables or disables to create new context menu on the node.
None	Disable all Node Constraints.
OutConnect	Enables or disables the connection creation to or from outgoing connector.
PivotDraggable	Enables or disables a pivot thumb dragging for node.
Resizable	Enables or disables the node to be resizable.
ResizeEast	Enables or disables the node to be resized in east direction.
ResizeNorth	Enables or disables the node to be resized in North direction.
ResizeNorthEast	Enables or disables the node to be resized in NorthEast direction.
ResizeNorthWest	Enables or disables the node to be resized in NorthWest direction.
ResizeSouth	Enables or disables the node to be resized in South direction.
ResizeSouthEast	Enables or disables the node to be resized in SouthEast direction.
ResizeSouthWest	Enables or disables the node to be resized in SouthWest direction.
ResizeWest	Enables or disables the node to be resized in West direction.
Rotatable	Enables or disables the node to be rotated.
RoutingObstacle	Enables or disables the node to be treated as obstacle while in routing.
Selectable	Enables or disables the node to be selected.
SnapAngle	Enables or disables the node to be snapped while rotating.
SnapToHorizontalLines	Enables or disables the node to be snapped to horizontal gridlines.
SnapToLines	Enables or disables the nodes to be snapped to gridlines.
SnapToVerticalLines	Enables or disables the node to be snapped to vertical gridlines.

**ThemeStyle** | Enables or disables the theme style for node. |

The following code example illustrates how to disable rotation.

**C#**

```
//Create NodeViewModel collection
diagram.nodes = new ObservableCollection<NodeViewModel>();
//Create NodeViewModel
NodeViewModel node = new NodeViewModel()
{
    Constraints = NodeConstraints.Default &
    ~(NodeConstraints.Rotatable|NodeConstraints.InheritRotatable),
    UnitWidth = 50,
    UnitHeight = 50,
    OffsetX = 100,
    OffsetY = 100,
};
//Add NodeViewModel to Nodes Collection
(diagram.Nodes as ObservableCollection<NodeViewModel>).Add(node);
```

### Connector Constraints

[ConnectorConstraints](#) allows to enable or disable the following behaviors of Connectors. By default Selectable, EndDraggable, Inherit, Thumbs, Connectable, Delete, BridgeObstacle and ThemeStyle constraints are enabled for connector.

**Constraints** | Description |

--|--|

**AllowDrop** | Enables or disables interaction events(DragEnter,DragOver,ItemDrop and DragLeave) for diagram connectors. |

**AllowPan** | Enables or disables the panning action for connector. |

**AutomaticPortCreation** | Enables or disables the automatic port creation while start or end the connection on or over the connector |

**BridgeObstacle** | Enables or disables bridge can be created over a connector. |

**Bridging** | Enables or disables the line bridging while two connectors intersecting with each other. |

**Connectable** | Enables or disables the node or port to be connected to the connector. |

**Default** | Enables Selectable, EndDraggable, Inherit, Thumbs, Connectable, Delete, BridgeObstacle, ThemeStyle behaviors for connector. |

**Delete** | Enable or disables the connector to be deleted. |

**DragAnnotation** | Enables or disable the connector's annotation to be dragged. |

**Draggable** | Enables or disables the connector to be dragged from one point to another in diagram. |

**EndDraggable** | Enables or disables the connector source or target end to be dragged. |

**EndThumbs** | Enables or disables the connector's end thumbs to be dragged. |

- |InConnect | Enables or disables the connection creation from the incoming connector. |
- |Inherit | Enables or disables to inherit the Snapping, SnapToObject, Bridging, Smoothness, Menu, PortVisibility, Pan, HitPadding and Routing from SfDiagram. |
- |InheritAllowPan | Enable or disables the panning of connector over graph. |
- |InheritAutomaticPortCreation | Enables or disables to inherit the value of automatic port creation from diagram. |
- |InheritBridging | Enables or disables to inherit the value of bridging from diagram. |
- |InheritHitPadding | Enables or disables to inherit the value of SfDiagram.HitPadding. |
- |InheritMenu | Enables or disables to inherit the context menu from diagram. |
- |InheritPortVisibility | Enables or disables to inherit the value from SfDiagram.PortVisibility. |
- |InheritRouting | Enables or disables to inherit the value of routing from diagram. |
- |InheritSmoothness | Enables or disables to inherit the value of Smoothness from diagram. |
- |InheritSnapping | Enables or disables to inherit the value of SnapToLines from diagram by using SnapSettings.SnapConstraints. |
- |InheritSnapToObject | Enables or disables to inherit the value of SnapToObject from diagram by using SnapSettings.SnapConstraints. |
- |Menu | Enables or disables to create new context menu for connector. |
- |None | Disable all behaviors for connector. |
- |OutConnect | Enables or disables the connection from the outgoing connector. |
- |Routing | Enables or disables the connector to be routed |
- |SegmentThumbs | Enables or disables the control point and end point of every segment in a connector for editing. |
- |Selectable | Enables or disables the connector to be selected. |
- |SnapToHorizontalLines | Enables or disables the connector to be snapped to horizontal gridlines. |
- |SnapToLines | Enables or disables the connector to be snapped to gridlines. |
- |SnapToVerticalLines | Enables or disables the connector to be snapped to vertical gridlines. |
- |SourceDraggable | Enables or disables the connector's source end to be dragged |
- |TargetDraggable | Enables or disables the connector's target end to be dragged. |
- |ThemeStyle | Enables or disables the theme style for connector. |
- |Thumbs | Enables or disables the connector's thumbs to be dragged. |

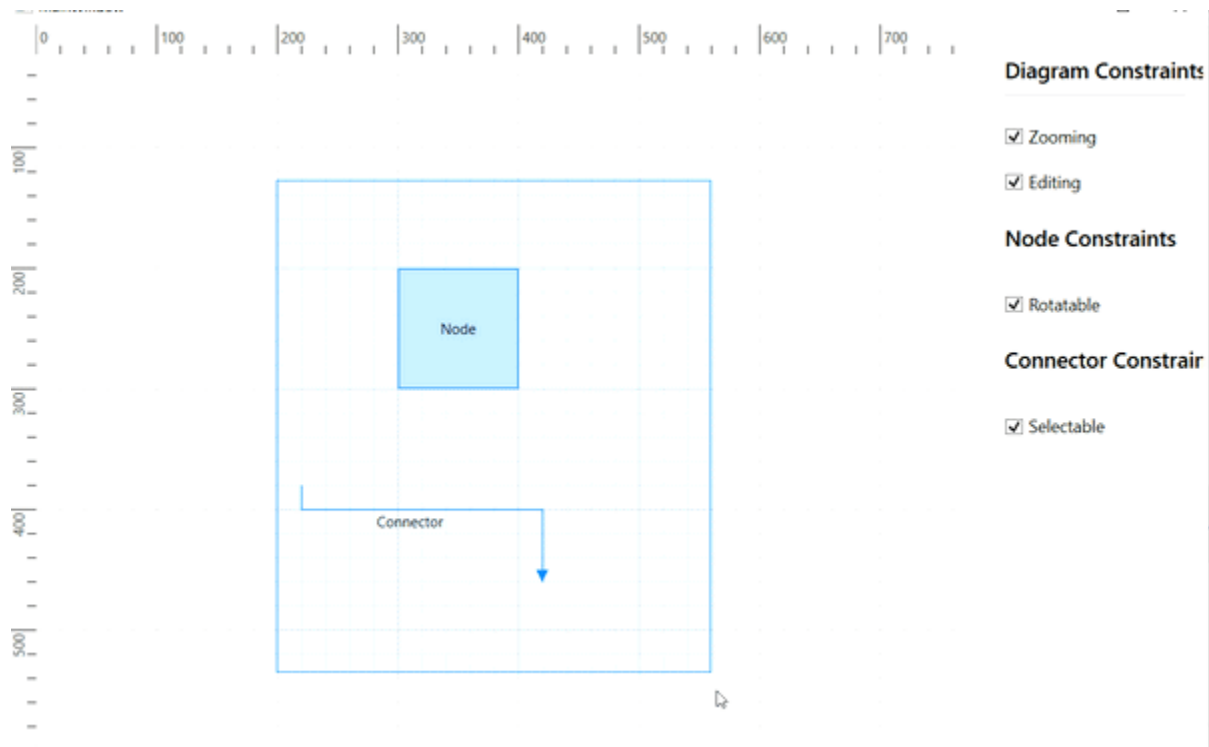
The following code example illustrates how to disable selection.

#### C#

```
diagram.connectors = new ObservableCollection<ConnectorViewModel>();
```



```
//Create ConnectorViewModel
ConnectorViewModel connector1 = new ConnectorViewModel()
{
    Constraints = ConnectorConstraints.Default &
    ~ConnectorConstraints.Selectable
};
(diagram.Connectors as
ObservableCollection<ConnectorViewModel>).Add(connector1);
```



[View Sample in GitHub](#)

### Port Constraints

[PortConstraints](#) allows to enable or disable the following behaviors of port. By default Inherit constraints are enabled for port.

Constraints	Description
-- --	
<b>Connectable</b>	Enables or disables the connector creation from port.
<b>ConnectionDirection</b>	Decides the connection direction of port based on <code>PortBase.ConnectionDirection</code>
<b>Default</b>	Enables or disables the Inheritable constraints.
<b>Draggable</b>	Enables or disables whether port to be dragged at boundaries of node
<b>Dynamic</b>	Defines whether port to be connected at boundaries of node and pointed to the port.
<b>InConnect</b>	Enables or disables the connection creation to the incoming Connector.

**|Inherit** | Enables or disables to inherit the value of Node.PortVisibility, Node.HitPadding and NodeConstraints.Connectable from Node. |

**|InheritConnectable** | Enables or disables to inherit the value of Connectable from Node. |

**|InheritHitPadding** | Enables or disables to inherit the value of SfDiagram.HitPadding. |

**|InheritPortVisibility** | Enables or disables to inherit the value of PortVisibility from diagram or Node. |

**|None** | Disable all port Constraints. |

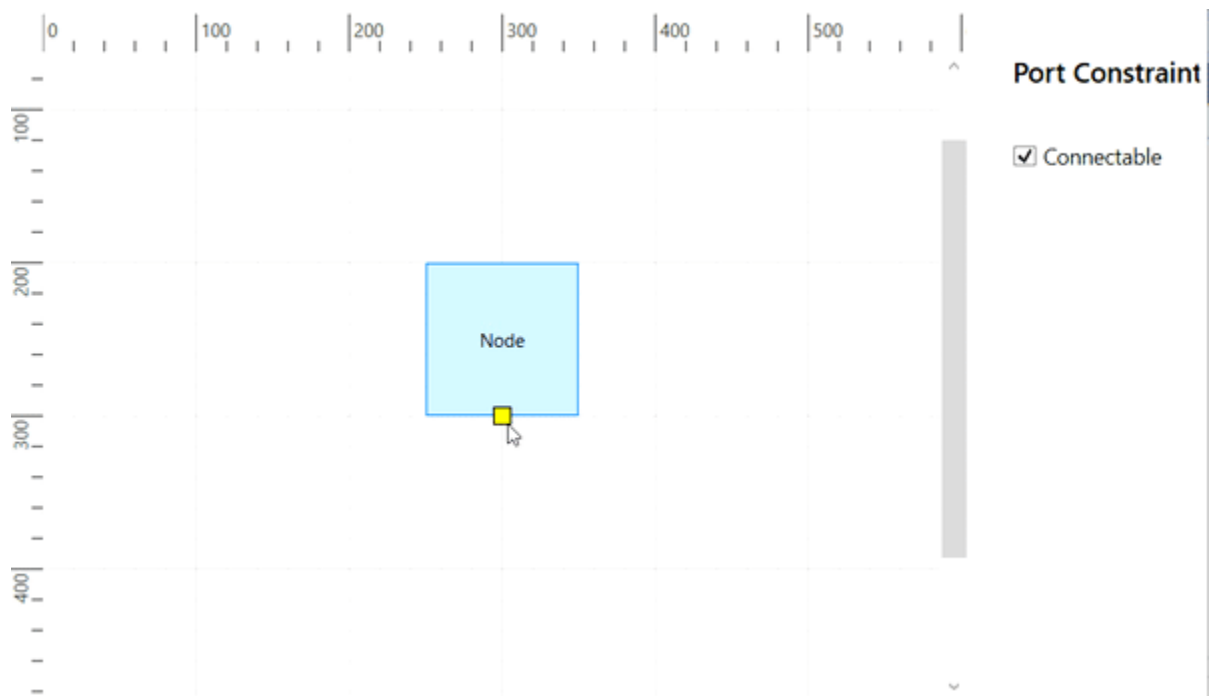
**|OutConnect** | Enables or disables the connection creation from the outgoing Connector. |

**|Snapping** | Enables or disables snapping behavior of port on node |

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

### C#

```
NodePortViewModel port= new NodePortViewModel()
{
    UnitWidth= 10,
    UnitHeight= 10,
    Constraints = PortConstraints.Default &
    ~PortConstraints.Connectable
};
```



[View Sample in GitHub](#)

### Annotation Constraints

[AnnotationConstraints](#) allows to enable or disable the following behaviors of Annotation. By default Inherit and Editable constraints are enabled for annotation.

Constraints	Description
-------------	-------------

|--|--|

|Default | Enables all Annotation constraints. |

|Draggable | Enables or disables whether the annotation to be dragged. |

|DragLimit | Specifies whether the annotation to be dragged with in the specified limit |

|Editable | Enables or disables the annotation to be edited. |

|Inherit | Enables or disables to inherit the Dragging, Rotation, Selection, Resize and Editable constraints from Node or Connector |

|InheritDraggable | Enables or disables to inherit the value of Draggable from Node or Connector. |

|InheritEditable | Enables or disables to inherit the value of Editable from Node or Connector. |

|InheritResizable | Enables or disables to inherit the value of Resizable from Node or Connector. |

|InheritRotatable | Enables or disables to inherit the value of Rotatable from Node or Connector. |

|InheritSelectable | Enables or disables to inherit the value of Selectable from Node or Connector. |

|None | Disables all behaviors of annotation. |

|Resizable | Enables or disables the annotation to be resized. |

|ResizeEast | Enables or disables the annotation to be resized in east direction. |

|ResizeNorth | Enables or disables annotation to be resized in north direction. |

|ResizeNorthEast | Enables or disables the annotation to be resized in northeast direction. |

|ResizeNorthWest | Enables or disables the annotation to be resized in northwest direction. |

|ResizeSouth | Enables or disables the annotation to be resized in south direction. |

|ResizeSouthEast | Enables or disables the annotation to be resized in southeast direction. |

|ResizeSouthWest | Enables or disables the annotation to be resized in southwest direction. |

|ResizeWest | Enables or disables the annotation to be resized in west direction. |

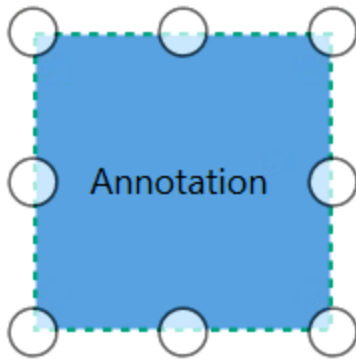
|Rotatable | Enables or disables whether the annotation to be rotated. |

|Selectable | Enables or disables whether the annotation to be selectable. |

The following code example illustrates how to enable annotation dragging.

### C#

```
AnnotationEditorViewModel anno = new AnnotationEditorViewModel()
{
    Content="Annotation",
    //Assign Constraint to Drag
    Constraints=AnnotationConstraints.Draggable,
};
```



[View Sample in GitHub](#)

### Selector Constraints

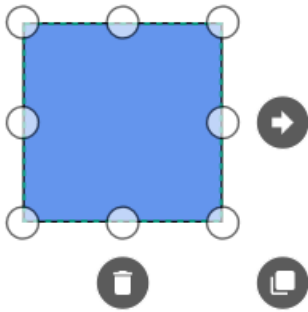
Selector visually represents the selected elements with certain editable thumbs. The visual representation of the thumbs can be controlled with [SelectorConstraints](#). The part of selector is categorized as follows.

Constraints	Description
-- --	
Default	Enables all behaviors for selector.
HideDisabledResizer	Enables or disables the visibility of disabled resizer thumbs.
None	Disables all the behaviors for Selector.
Pivot	Enables or disables the visibility of the pivot.
QuickCommands	Enables or disables the visibility of the QuickCommands.
Resizer	Enables or disables the Resizer of the selected node.
Rotator	Enables or disables the Rotator of the selected node.
Tooltip	Enables or disables the Tooltip of the selected node.
TooltipAngle	Enables or disables the Tooltip when rotate the node at runtime.
TooltipPosition	Enables or disables Tooltip when drag the node at runtime.
TooltipSize	Enables or disables the Tooltip when resize the node at runtime.

The following code example illustrates how to hide rotator.

### C#

```
(diagram.SelectedItems as SelectorViewModel).SelectorConstraints =
(diagram.SelectedItems as SelectorViewModel).SelectorConstraints &
~SelectorConstraints.Rotator;
```



[View Sample in GitHub](#)

### Snap Constraints

[SnapConstraints](#) control the visibility of gridlines and enable or disable snapping. Snap constraints allow to set the following behaviors.

Constraints	Description
<code>-- --</code>	
<code>All</code>	Enables or disables all Snap Constraints.
<code>HorizontalLines</code>	Enables or disables the node to be snapped to horizontal lines.
<code>None</code>	Disables all the behaviors for Snapping.
<code>Rotation</code>	Enables or disables the node to be snapped to rotation.
<code>ShowLines</code>	Enables or disables the node to be snapped to horizontal or vertical lines.
<code>SnapToHorizontalLines</code>	Enables or disables the node to be snapped to vertical gridlines.
<code>SnapToLines</code>	Enables or disables the node to be snapped to gridlines
<code>SnapToVerticalLines</code>	Enables or disables the node to be snapped to horizontal gridlines.
<code>VerticalLines</code>	Enables or disables the node to be snapped to vertical lines.

The following code example illustrates how to show only Horizontal Gridlines

### C#

```
diagram.SnapSettings.SnapConstraints =
SnapConstraints.SnapToHorizontalLines;
```

Refer [Snapping](#) for details.

### Inherit behaviors

Some of the behaviors can be defined through both the specific object (Node/Connector) and Diagram. When the behaviors are contradictorily defined through both, the actual behavior is set through inherit options.

The following code example illustrates how to inherit the line bridging behavior from the Diagram.

### C#

```
diagram.Constraints = GraphConstraints.Default | GraphConstraints.Bridging;
ObservableCollection<ConnectorViewModel> connectors = new
ObservableCollection<ConnectorViewModel>();
ConnectorViewModel connector1 = new ConnectorViewModel()
{
    SourcePoint=new Point(100,100),
    TargetPoint=new Point(200,200),
    Constraints = ConnectorConstraints.Default |
    ConnectorConstraints.InheritBridging
};
connectors.Add(connector1);
diagram.Connectors = connectors;
```

### Bitwise Operations

Bitwise Operations are used to manipulate the flagged enumerations [enum]. In the section, Bitwise Operations are illustrated by using Node Constraints. The same is applicable while working with Node Constraints, Connector Constraints, or Port Constraints.

#### Add Operation

You can add or enable multiple values at a time by using Bitwise **|** (OR) operator or Add() method of constraints.

### C#

```
//Using OR operator
node.Constraints = NodeConstraints.Selectable | NodeConstraints.Rotatable;
//Using Add method to add constraints
node.Constraints = node.Constraints.Add(NodeConstraints.Rotatable);
```

In the above example, you can do both the selection and rotation.

#### Remove Operation

You can remove or disable values by using Bitwise **&~** (XOR) operator or Remove() method of constraints.

### C#

```
//Using XOR operator
node.Constraints = node.Constraints & ~(NodeConstraints.Rotatable);
//Using Remove method to remove constraints
node.Constraints = node.Constraints.Remove(NodeConstraints.Rotatable);
```

In the above example, Rotation is disabled but other constraints are enabled.

#### Check Operation

You can check any value by using Bitwise **&** (AND) operator.

### C#

```
if ((node.Constraints & (NodeConstraints.Rotatable)) ==
(NodeConstraints.Rotatable))
```

In the above example, you can check whether the rotate constraints are enabled in a Node. When `NodeConstraints` have rotate constraints, the expression returns a rotate constraint.

### Context Menu in WPF Diagram (SfDiagram)

In graphical user interface (GUI), a `ContextMenu` is a type of Menu that appears when you perform right-click operation. Nested level of Context Menu items can be created. Diagram provided some in-built `ContextMenu` items and allows to define custom menu items.

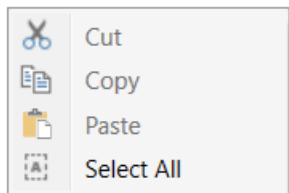
#### Default Context Menu

The `ContextMenu` Constraint helps you to enable/disable the context menu. The following code illustrates how to enable/disable the default context menu items.

#### C#

```
//Disable context menu
diagram.Constraints = GraphConstraints.Default &
~GraphConstraints.ContextMenu;
//Enable context menu
diagram.Constraints = GraphConstraints.Default |
GraphConstraints.ContextMenu;
```

Diagram provides some default context menu items to ease the execution of some frequently used commands.



#### Customize Context Menu

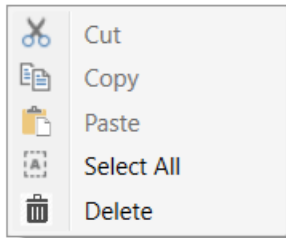
- Apart from the default `ContextMenu` items, you can define some additional menu items by using `Menu` property of `SfDiagram`, `Node` and `Connector`. Those additional items have to be defined and added to `MenuItems` Property.
- The `Content` property allows you to set Content for the context menu item.
- The `Icon` property allows you to set icon for the context menu item.
- The `Command` property of the Context menu item allows you to define command for it.
- The `IsSeparator` 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 `IsSeparator` property.

The following code example illustrates how to add custom context menu items to `Menu` property of `SfDiagram`.

#### C#

```
DiagramMenuItem menu = new DiagramMenuItem()
{
    Content = "Delete",
```

```
Command = (diagram.Info as IGraphInfo).Commands.Delete,
Icon = @"pack://application:,,,/delete.ico"
};
diagram.Menu.MenuItems.Add(menu);
```

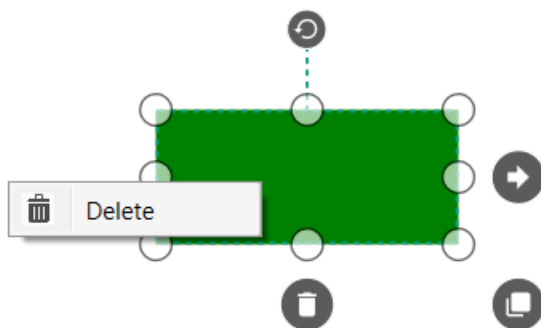


### *Menu for Node and Connector*

The default value of Menu property for Node and Connector is null. The following code example illustrates how to set ContextMenu and ContextMenuItems to Node.

### **C#**

```
node.Constraints = node.Constraints | NodeConstraints.Menu;
node.Constraints = node.Constraints & ~NodeConstraints.InheritMenu;
node.Menu = new DiagramMenu();
node.Menu.MenuItems=new ObservableCollection<DiagramMenuItem>();
DiagramMenuItem mi = new DiagramMenuItem()
{
    Content = "Delete",
    Command = (diagram.Info as IGraphInfo).Commands.Delete,
    Icon = @"pack://application:,,,/delete.ico"
};
(node.Menu.MenuItems as ICollection<DiagramMenuItem>).Add(mi);
```



[View Sample in GitHub](#)

### Events

- **MenuItemClickedEvent** will invoke when you click the menu items. To explore about arguments, refer to [MenuItemClickedEventArgs](#)

The following code example illustrates how to define those events.



**C#**

```
(diagram.Info as IGraphInfo).MenuItemClickedEvent +=
MainPage_MenuItemClickedEvent;
void MainPage_MenuItemClickedEvent(object sender,
MenuItemClickedEventArgs args)
{
    //Source - in which object Event get fired
    //Item - MenuItem
}
```

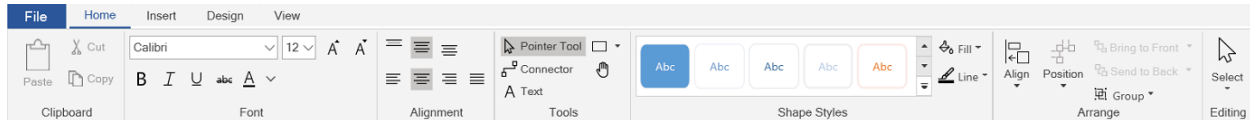
- **MenuOpening** event will notify when you perform right click on Diagram/Node/Connector.To explore about arguments, refer to [MenuOpeningEventArgs](#).

See Also

- [How to customize the context menu?](#)

## Diagram Ribbon in WPF Diagram (SfDiagram)

The diagram ribbon control is a user interface that hosts a Quick Access Toolbar, Application Menu, and Tabs to provide the most common features and settings for the WPF Diagram. The diagram ribbon control contains the UI elements that allow end-users to load and save diagrams, add diagram items to the canvas, format text within the diagram items, rearrange and recolor shapes, change the canvas size and orientation, and perform a copy and paste operations.



The following table lists actions that end-users can perform using the Diagram Ribbon.

Ribbon Tab   Ribbon Bar   Actions
---   ---   ---
Home   Clipboard   Cut, Copy, and Paste operations.
Font   Change the font settings of the selected shape.
Alignment   Realign the text within the selected shape.
Tools   Select a tool.
Shape Styles   Change the background and border colors for the currently selected shape using a palette of predefined patterns or by selecting the colors individually using color pickers.
Arrange   Change the position or alignment of the selected shapes and move the selected shapes in and out in the Z-order.
Select   Select all the diagram items or by specific type.
Insert   Diagram Parts   Allows you to add an image item or connection line to the canvas.
Design   Page Setup   Toggle between portrait and landscape page orientation, set the page size.

| | Themes | The diagram theme give your diagram a variety of dynamic looks, ranging from professional and modern to stylish. By combining colors, fonts, and effects in a unified and purposeful manner, each theme to make the diagram polished look. |

| | Variants | Variant styles are set of styles that are applied to the Diagram elements. |

| | Connectors | Change the appearance of the connection lines in the canvas. |

| View | Show | Toggle the visibility of the Zoom Pan window, Rulers, Gridlines, and Page Breaks. |

| | Zoom | Zoom the diagram so that the entire page either fits the window or is as wide as the window. |

### Assembly Deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

You can find more details about installing the NuGet package in a WPF application in the following link:

[How to install nuget packages](#)

### Adding Diagram Ribbon

The following section helps you to add the SfDiagramRibbon in your application with the SfDiagram.

The SfDiagramRibbon control can be added to the application by dragging it from the toolbox and dropping it in the designer view. The required assembly references will be added automatically.



Steps to add the Diagram ribbon control manually is given below with its code example.

1. Add the following required assembly reference to the project **Syncfusion.SfDiagramRibbon.WPF**.
2. Import the Syncfusion WPF schema **http://schemas.syncfusion.com/wpf** or the SfDiagramRibbon control namespace **Syncfusion.UI.Xaml.DiagramRibbon** in your application.
3. Declare the SfDiagramRibbon control in your application.

### XML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:SfDiagram_WPF"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="SfDiagram_WPF.MainWindow"
mc:Ignorable="d" Title="MainWindow" Height="350" Width="525">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*/>
</Grid.RowDefinitions>
<syncfusion:SfDiagramRibbon DataContext="{Binding ElementName=Diagram}"
Grid.Row="0"/>
<syncfusion:SfDiagram x:Name="Diagram" Grid.Row="1">
<syncfusion:SfDiagram.Theme>
<syncfusion:OfficeTheme/>
</syncfusion:SfDiagram.Theme>
<syncfusion:SfDiagram.Nodes>
<syncfusion:NodeCollection/>
</syncfusion:SfDiagram.Nodes>
<syncfusion:SfDiagram.Connectors>
<syncfusion:ConnectorCollection/>
</syncfusion:SfDiagram.Connectors>
</syncfusion:SfDiagram>
</Grid>
</Window>

```

## C#

```

using Syncfusion.UI.Xaml.Diagram;
using Syncfusion.UI.Xaml.Diagram.Theming;
using Syncfusion.UI.Xaml.DiagramRibbon;
namespace SfDiagram_WPF
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            // Initialize, arrange and add a diagram and Diagram ribbon in the
            application
            SfDiagram sfDiagram = new SfDiagram() { Theme = new OfficeTheme(), Nodes =
            new NodeCollection(), Connectors = new ConnectorCollection() };
            SfDiagramRibbon sfDiagramRibbon = new SfDiagramRibbon() { DataContext =
            SfDiagram };
            SfDiagram.SetValue(Grid.RowProperty, 1);
            sfDiagramRibbon.SetValue(Grid.RowProperty, 0);
            Root_Grid.Children.Add(SfDiagram);
            Root_Grid.Children.Add(sfDiagramRibbon);
        }
    }
}

```

## Simple Diagram Designer

This section describes how to build a simple diagram designer application with diagram ribbon, and diagram control.

**Note:** You have to set the diagram as DataContext for the Diagram ribbon control, which will help the users to interact with the diagram by using the Diagram ribbon control.

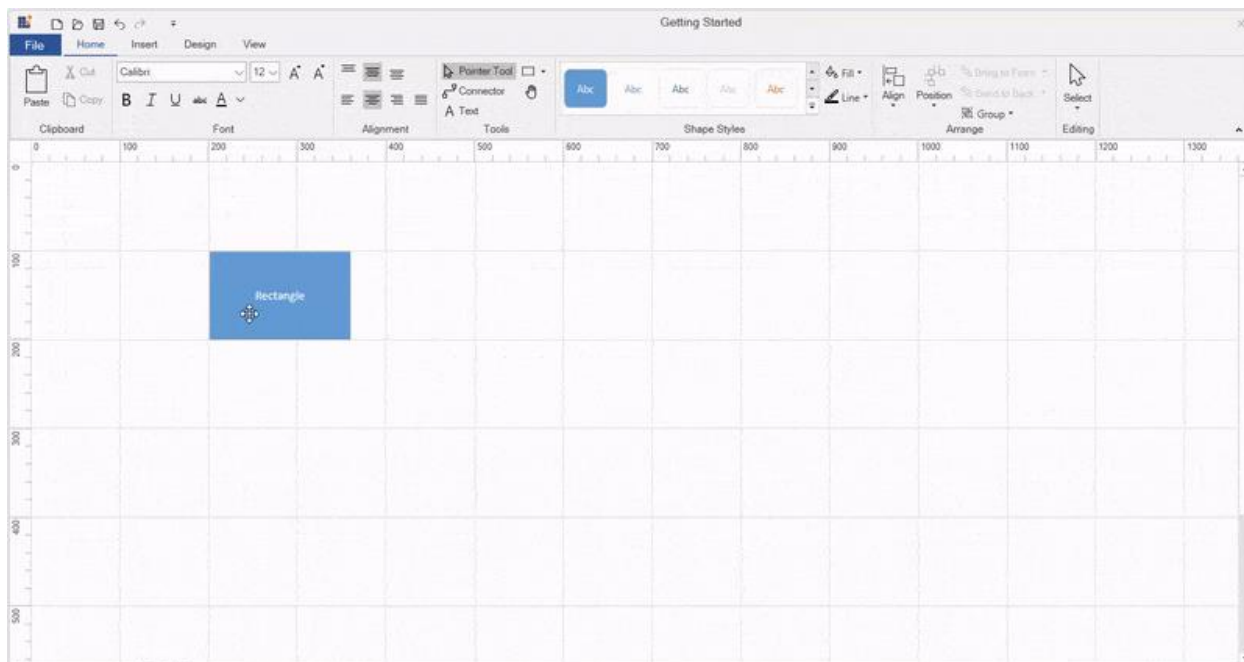
### XML

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*/>
</Grid.RowDefinitions>
<syncfusion:SfDiagramRibbon x:Name="DiagramRibbon" Grid.Row="0"
DataContext="{Binding ElementName=Diagram}"/>
<syncfusion:SfDiagram x:Name="Diagram" Grid.Row="1"
Constraints="Undoable,Default">
<syncfusion:SfDiagram.Theme>
<syncfusion:OfficeTheme/>
</syncfusion:SfDiagram.Theme>
<syncfusion:SfDiagram.Nodes>
<syncfusion:NodeCollection/>
</syncfusion:SfDiagram.Nodes>
<syncfusion:SfDiagram.Connectors>
<syncfusion:ConnectorCollection/>
</syncfusion:SfDiagram.Connectors>
<syncfusion:SfDiagram.Groups>
<syncfusion:GroupCollection/>
</syncfusion:SfDiagram.Groups>
<syncfusion:SfDiagram.SnapSettings>
<syncfusion:SnapSettings SnapConstraints="All"/>
</syncfusion:SfDiagram.SnapSettings>
<syncfusion:SfDiagram.HorizontalRuler>
<syncfusion:Ruler Orientation="Horizontal"/>
</syncfusion:SfDiagram.HorizontalRuler>
<syncfusion:SfDiagram.VerticalRuler>
<syncfusion:Ruler Orientation="Vertical"/>
</syncfusion:SfDiagram.VerticalRuler>
</syncfusion:SfDiagram>
</Grid>
```

### C#

```
SfSkinManager.SetTheme(this, new Syncfusion.SfSkinManager.Theme() {
ThemeName = "Office2019Colorful" });
// Initialize, arrange and add a diagram and Diagram ribbon in the
application
SfDiagram Sfdiagram = new SfDiagram()
{
Theme = new OfficeTheme(),
Nodes = new NodeCollection(),
Connectors = new ConnectorCollection(),
Groups = new GroupCollection(),
Constraints = GraphConstraints.Default | GraphConstraints.Undoable,
SnapSettings = new SnapSettings() { SnapConstraints = SnapConstraints.All },
HorizontalRuler = new Syncfusion.UI.Xaml.Diagram.Controls.Ruler() {
Orientation = Orientation.Horizontal },
VerticalRuler = new Syncfusion.UI.Xaml.Diagram.Controls.Ruler() {
Orientation = Orientation.Vertical },
```

```
};
SfDiagramRibbon sfDiagramRibbon = new SfDiagramRibbon() { DataContext =
Sfdiagram };
Sfdiagram.SetValue(Grid.RowProperty, 1);
sfDiagramRibbon.SetValue(Grid.RowProperty, 0);
Root_Grid.Children.Add(Sfdiagram);
Root_Grid.Children.Add(sfDiagramRibbon);
```



[View sample in GitHub](#)

### Ribbon Customization

The Ribbon customization can be done in two ways.

#### *Using Control Template*

An user can customize the ribbon items by overriding the template of the [SfDiagramRibbon](#).

#### *Using Tabs property in Events*

By invoking the SfDiagramRibbon loaded event, a user can add or delete the ribbon tabs, and ribbon items with the help of [Tabs Property](#) in the [SfDiagramRibbon](#).

#### Adding RibbonTab

To add a custom ribbon tab with the default tabs in the [SfDiagramRibbon](#).

#### C#

```
DiagramRibbon.Loaded += DiagramRibbon_Loaded;
private void DiagramRibbon_Loaded(object sender, RoutedEventArgs e)
{
    // Add a new Ribbon tab.
    RibbonTab FormatTab = new RibbonTab() { Caption = "Format" };
    (sender as SfDiagramRibbon).Tabs.Add(FormatTab);
}
```

### Removing Ribbon Tab

To remove a ribbon tab from the default tabs in **SfDiagramRibbon**.

#### C#

```
DiagramRibbon.Loaded += DiagramRibbon_Loaded;
private void DiagramRibbon_Loaded(object sender, RoutedEventArgs e)
{
    // Remove the Insert tab.
    (sender as SfDiagramRibbon).Tabs.RemoveAt(1);
}
```

### Adding RibbonItems

To add a custom ribbon item with the default items in an already existing ribbon tab.

#### C#

```
DiagramRibbon.Loaded += DiagramRibbon_Loaded;
private void DiagramRibbon_Loaded(object sender, RoutedEventArgs e)
{
    // Add new Ribbon Item.
    RibbonTab InsertTab = (sender as SfDiagramRibbon).Tabs.ElementAt(1);
    InsertTab.Items.Add(new RibbonBar() { Header = "Links" });
}
```

### Removing Ribbon Tab

To remove an already existing ribbon item from the ribbon tab.

#### C#

```
DiagramRibbon.Loaded += DiagramRibbon_Loaded;
private void DiagramRibbon_Loaded(object sender, RoutedEventArgs e)
{
    // Remove the alignment bar from the Home Tab.
    RibbonTab HomeTab = (sender as SfDiagramRibbon).Tabs.ElementAt(0);
    HomeTab.Items.RemoveAt(2);
}
```

## SfHeatMap

### WPF HeatMap (SfHeatMap) Overview

**Essential HeatMap WPF** represents tabular data values as gradient colors instead of numbers. Low and high values are different colors with different gradients.

Product	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
Alice Mutton	3	4	3	4	10	6	7	6	8	5
Boston Crab Meat	10	11	10	12	15	13	11	10	12	11
Raclette Courdavault	12	12	15	18	19	20	22	21	22	25
Gorgonzola Telino	1	1	2	3	2	2	3	2	3	3
Chartreuse verte	15	14	14	13	10	8	9	9	8	8
Flotemysost	3	3	4	5	4	6	8	2	5	7
Carnarvon Tigers	20	21	20	20	20	21	20	20	21	22
Vegie-spread	18	19	20	21	22	23	24	25	25	25
Tarte au sucre	1	2	3	3	4	4	7	10	12	16
Konbu	10	8	8	7	8	10	11	12	11	13
Valkoinen suklaa	20	20	19	20	15	12	6	3	3	3
Perth Pasties	12	13	13	15	15	18	19	19	22	22

**Key features:**

- 2 types of data source mapping (TableMapping, CellMapping)
- Color mapping
- Legend
- Virtualization

**Getting Started with WPF HeatMap (SfHeatMap)**

Initialize the SfHeatMap

The SfHeatMap exists in the Syncfusion.UI.Xaml.HeatMap namespace. Initialize SfHeatMap to the XAML Page as shown in the following code sample.

**XML**

```
<Window x:Class="GettingStarted.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:GettingStarted"
mc:Ignorable="d"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="350" Width="525">
<Grid>
<syncfusion:SfHeatMap>
</syncfusion:SfHeatMap>
</Grid>
</Window>
```

**Note:** Starting with v16.2.0.x, if you reference Syncfusion assemblies from trial setup or from the NuGet feed, you also have to include a license key in your projects. Please refer to this [link](#) to know about registering Syncfusion license key in your WPF application to use our components.

Prepare data

Create a class to store Product information to be visualized using SfHeatMap

### C#

```
public class Product
{
    public string ProductName { get; set; }
    public double Y2007 { get; set; }
    public double Y2008 { get; set; }
    public double Y2009 { get; set; }
    public double Y2010 { get; set; }
    public double Y2011 { get; set; }
    public double Y2012 { get; set; }
    public double Y2013 { get; set; }
    public double Y2014 { get; set; }
    public double Y2015 { get; set; }
    public double Y2016 { get; set; }
    public Product()
    {
    }
}

public class Products : ObservableCollection<Product>
{
}
```

Populate data

Populate product information in a collection

### XML

```
<local:Products x:Key="productsData">
<local:Product ProductName="Alice Mutton" Y2007="3" Y2008="4" Y2009="3"
Y2010="4" Y2011="10"
Y2012="6" Y2013="7" Y2014="6" Y2015="8" Y2016="5"/>
<local:Product ProductName="Boston Crab Meat" Y2007="10" Y2008="11"
Y2009="10" Y2010="12"
Y2011="15" Y2012="13" Y2013="11" Y2014="10" Y2015="12" Y2016="11"/>
<local:Product ProductName="Raclette Courdavault" Y2007="12" Y2008="12"
Y2009="15" Y2010="18"
Y2011="19" Y2012="20" Y2013="22" Y2014="21" Y2015="22" Y2016="25"/>
<local:Product ProductName="Gorgonzola Telino" Y2007="1" Y2008="1" Y2009="2"
Y2010="3"
Y2011="2" Y2012="2" Y2013="3" Y2014="2" Y2015="3" Y2016="3"/>
<local:Product ProductName="Chartreuse verte" Y2007="15" Y2008="14"
Y2009="14" Y2010="13"
Y2011="10" Y2012="8" Y2013="9" Y2014="9" Y2015="8" Y2016="8"/>
<local:Product ProductName="Fløtemysost" Y2007="3" Y2008="3" Y2009="4"
Y2010="5" Y2011="4"
Y2012="6" Y2013="8" Y2014="2" Y2015="5" Y2016="7"/>
<local:Product ProductName="Carnarvon Tigers" Y2007="20" Y2008="21"
Y2009="20" Y2010="20"
Y2011="20" Y2012="21" Y2013="20" Y2014="20" Y2015="21" Y2016="22"/>
<local:Product ProductName="Vegie-spread" Y2007="18" Y2008="19" Y2009="20"
Y2010="21"
Y2011="22" Y2012="23" Y2013="24" Y2014="25" Y2015="25" Y2016="25"/>
```



```

<local:Product ProductName="Tarte au sucre" Y2007="1" Y2008="2" Y2009="3"
Y2010="3" Y2011="4"
Y2012="4" Y2013="7" Y2014="10" Y2015="12" Y2016="16"/>
<local:Product ProductName="Konbu" Y2007="10" Y2008="8" Y2009="8" Y2010="7"
Y2011="8"
Y2012="10" Y2013="11" Y2014="12" Y2015="11" Y2016="13"/>
<local:Product ProductName="Valkoinen suklaa" Y2007="20" Y2008="20"
Y2009="19" Y2010="20"
Y2011="15" Y2012="12" Y2013="6" Y2014="3" Y2015="3" Y2016="3"/>
<local:Product ProductName="Perth Pasties" Y2007="12" Y2008="13" Y2009="13"
Y2010="15"
Y2011="15" Y2012="18" Y2013="19" Y2014="19" Y2015="22" Y2016="22"/>
</local:Products>

```

### Map data into SfHeatMap

Now data is ready, next we need to configure data source and map rows and columns to visualize.

- Prepare ItemsMapping add it in resource.

### XML

```

<syncfusion:TableMapping x:Key="ItemsMapping">
<syncfusion:TableMapping.HeaderMapping>
<syncfusion:ColumnMapping PropertyName="ProductName" DisplayName="Product
Name"/>
</syncfusion:TableMapping.HeaderMapping>
<syncfusion:TableMapping.ColumnMapping>
<syncfusion:ColumnMapping PropertyName="Y2007" DisplayName="Y2007"/>
<syncfusion:ColumnMapping PropertyName="Y2008" DisplayName="Y2008"/>
<syncfusion:ColumnMapping PropertyName="Y2009" DisplayName="Y2009"/>
<syncfusion:ColumnMapping PropertyName="Y2010" DisplayName="Y2010"/>
<syncfusion:ColumnMapping PropertyName="Y2011" DisplayName="Y2011"/>
<syncfusion:ColumnMapping PropertyName="Y2012" DisplayName="Y2012"/>
<syncfusion:ColumnMapping PropertyName="Y2013" DisplayName="Y2013"/>
<syncfusion:ColumnMapping PropertyName="Y2014" DisplayName="Y2014"/>
<syncfusion:ColumnMapping PropertyName="Y2015" DisplayName="Y2015"/>
<syncfusion:ColumnMapping PropertyName="Y2016" DisplayName="Y2016"/>
</syncfusion:TableMapping.ColumnMapping>
</syncfusion:TableMapping>

```

- Set items source and mapping

### XML

```

<syncfusion:SfHeatMap ItemsSource="{StaticResource productsData}"
ItemsMapping="{StaticResource itemsMapping}">
</syncfusion:SfHeatMap>

```

- This will show a grid with data as in following image.

Product	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
Alice Mutton	3	4	3	4	10	6	7	6	8	5
Boston Crab Meat	10	11	10	12	15	13	11	10	12	11
Raclette Courdavault	12	12	15	18	19	20	22	21	22	25
Gorgonzola Telino	1	1	2	3	2	2	3	2	3	3
Chartreuse verte	15	14	14	13	10	8	9	9	8	8
Fløtemysost	3	3	4	5	4	6	8	2	5	7
Carnarvon Tigers	20	21	20	20	20	21	20	20	21	22
Vegie-spread	18	19	20	21	22	23	24	25	25	25
Tarte au sucre	1	2	3	3	4	4	7	10	12	16
Konbu	10	8	8	7	8	10	11	12	11	13
Valkoinen suklaa	20	20	19	20	15	12	6	3	3	3
Perth Pasties	12	13	13	15	15	18	19	19	22	22

### Color Mapping

Next we can configure color range for these values using color mapping

- Configure items mapping based on items source.

### XML

```
<syncfusion:ColorMappingCollection x:Key="colorMapping">
  <syncfusion:ColorMapping Value="0" Color="#8ec8f8"/>
  <syncfusion:ColorMapping Value="30" Color="#0d47a1"/>
</syncfusion:ColorMappingCollection>
```

- Set ColorMapping

### XML

```
<syncfusion:SfHeatMap ItemsSource="{StaticResource productsData}"
  ItemsMapping="{StaticResource itemsMapping}"
  ColorMappingCollection="{StaticResource colorMapping}">
```

- This will show the grid data with color based on the range given.

Product	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
Alice Mutton	3	4	3	4	10	6	7	6	8	5
Boston Crab Meat	10	11	10	12	15	13	11	10	12	11
Raclette Courdavault	12	12	15	18	19	20	22	21	22	25
Gorgonzola Telino	1	1	2	3	2	2	3	2	3	3
Chartreuse verte	15	14	14	13	10	8	9	9	8	8
Flotemysost	3	3	4	5	4	6	8	2	5	7
Carnarvon Tigers	20	21	20	20	20	21	20	20	21	22
Vegie-spread	18	19	20	21	22	23	24	25	25	25
Tarte au sucre	1	2	3	3	4	4	7	10	12	16
Konbu	10	8	8	7	8	10	11	12	11	13
Valkoinen suklaa	20	20	19	20	15	12	6	3	3	3
Perth Pasties	12	13	13	15	15	18	19	19	22	22

### Legend

A legend control is used to represent range value in a gradient, create a legend with the same color mapping as shown below.

### XML

```
<syncfusion:SfHeatMapLegend ColorMappingCollection="{StaticResource colorMapping}" />
```

Final MainPage.cs looks like this.

### C#

```
namespace GettingStarted
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
    public class Product
    {
        public string ProductName { get; set; }
        public double Y2007 { get; set; }
        public double Y2008 { get; set; }
        public double Y2009 { get; set; }
        public double Y2010 { get; set; }
        public double Y2011 { get; set; }
        public double Y2012 { get; set; }
        public double Y2013 { get; set; }
        public double Y2014 { get; set; }
        public double Y2015 { get; set; }
    }
}
```

```

public double Y2016 { get; set; }
public Product()
{
}
}
public class Products : ObservableCollection<Product>
{
}
}

```

Final MainPage.xaml looks like this.

### XML

```

<Window x:Class="GettingStarted.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:GettingStarted"
mc:Ignorable="d"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
Title="MainWindow" Height="350" Width="525">
<Window.Resources>
<local:Products x:Key="productsData">
<local:Product ProductName="Alice Mutton" Y2007="3" Y2008="4" Y2009="3"
Y2010="4" Y2011="10" Y2012="6" Y2013="7" Y2014="6" Y2015="8" Y2016="5"/>
<local:Product ProductName="Boston Crab Meat" Y2007="10" Y2008="11"
Y2009="10" Y2010="12" Y2011="15" Y2012="13" Y2013="11" Y2014="10" Y2015="12"
Y2016="11"/>
<local:Product ProductName="Raclette Courdavault" Y2007="12" Y2008="12"
Y2009="15" Y2010="18" Y2011="19" Y2012="20" Y2013="22" Y2014="21" Y2015="22"
Y2016="25"/>
<local:Product ProductName="Gorgonzola Telino" Y2007="1" Y2008="1" Y2009="2"
Y2010="3" Y2011="2" Y2012="2" Y2013="3" Y2014="2" Y2015="3" Y2016="3"/>
<local:Product ProductName="Chartreuse verte" Y2007="15" Y2008="14"
Y2009="14" Y2010="13" Y2011="10" Y2012="8" Y2013="9" Y2014="9" Y2015="8"
Y2016="8"/>
<local:Product ProductName="Fløtemysost" Y2007="3" Y2008="3" Y2009="4"
Y2010="5" Y2011="4" Y2012="6" Y2013="8" Y2014="2" Y2015="5" Y2016="7"/>
<local:Product ProductName="Carnarvon Tigers" Y2007="20" Y2008="21"
Y2009="20" Y2010="20" Y2011="20" Y2012="21" Y2013="20" Y2014="20" Y2015="21"
Y2016="22"/>
<local:Product ProductName="Veggie-spread" Y2007="18" Y2008="19" Y2009="20"
Y2010="21" Y2011="22" Y2012="23" Y2013="24" Y2014="25" Y2015="25"
Y2016="25"/>
<local:Product ProductName="Tarte au sucre" Y2007="1" Y2008="2" Y2009="3"
Y2010="3" Y2011="4" Y2012="4" Y2013="7" Y2014="10" Y2015="12" Y2016="16"/>
<local:Product ProductName="Konbu" Y2007="10" Y2008="8" Y2009="8" Y2010="7"
Y2011="8" Y2012="10" Y2013="11" Y2014="12" Y2015="11" Y2016="13"/>
<local:Product ProductName="Valkoinen suklaa" Y2007="20" Y2008="20"
Y2009="19" Y2010="20" Y2011="15" Y2012="12" Y2013="6" Y2014="3" Y2015="3"
Y2016="3"/>
<local:Product ProductName="Perth Pasties" Y2007="12" Y2008="13" Y2009="13"
Y2010="15" Y2011="15" Y2012="18" Y2013="19" Y2014="19" Y2015="22"
Y2016="22"/>

```

```

</local:Products>
<syncfusion:TableMapping x:Key="itemsMapping">
<syncfusion:TableMapping.HeaderMapping>
<syncfusion:ColumnMapping PropertyName="ProductName" DisplayName="Product"/>
</syncfusion:TableMapping.HeaderMapping>
<syncfusion:TableMapping.ColumnMapping>
<syncfusion:ColumnMapping PropertyName="Y2007" DisplayName="2007"/>
<syncfusion:ColumnMapping PropertyName="Y2008" DisplayName="2008"/>
<syncfusion:ColumnMapping PropertyName="Y2009" DisplayName="2009"/>
<syncfusion:ColumnMapping PropertyName="Y2010" DisplayName="2010"/>
<syncfusion:ColumnMapping PropertyName="Y2011" DisplayName="2011"/>
<syncfusion:ColumnMapping PropertyName="Y2012" DisplayName="2012"/>
<syncfusion:ColumnMapping PropertyName="Y2013" DisplayName="2013"/>
<syncfusion:ColumnMapping PropertyName="Y2014" DisplayName="2014"/>
<syncfusion:ColumnMapping PropertyName="Y2015" DisplayName="2015"/>
<syncfusion:ColumnMapping PropertyName="Y2016" DisplayName="2016"/>
</syncfusion:TableMapping.ColumnMapping>
</syncfusion:TableMapping>
<syncfusion:ColorMappingCollection x:Key="colorMapping">
<syncfusion:ColorMapping Value="0" Color="#8ec8f8"/>
<syncfusion:ColorMapping Value="30" Color="#0d47a1"/>
</syncfusion:ColorMappingCollection>
</Window.Resources>
<Grid HorizontalAlignment="Center" VerticalAlignment="Center">
<Grid.RowDefinitions>
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<syncfusion:SfHeatMap ItemsSource="{StaticResource productsData}"
ItemsMapping="{StaticResource itemsMapping}"
ColorMappingCollection="{StaticResource colorMapping}">
</syncfusion:SfHeatMap>
<syncfusion:SfHeatMapLegend Grid.Row="1"
Margin="20"
MaxWidth="300"
ColorMappingCollection="{StaticResource colorMapping}" />
</Grid>
</Window>

```

## Theme

SfHeatMap supports various built-in themes. Refer to the below links to apply themes for the SfHeatMap,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)

Product Name	Y2007	Y2008	Y2009	Y2010	Y2011	Y2012	Y2013	Y2014	Y2015
Alice	7	44	49	23	17	40	26	25	35
Boston	26	17	19	17	37	48	9	25	31
Raclette	9	4	10	29	32	9	43	42	36
Telino	45	38	30	3	34	38	43	7	23
Verte	17	35	34	0	19	29	31	23	39
Fløtemy	0	25	36	45	35	3	33	19	42
Tigers	48	34	40	18	25	38	38	16	33
Vegie	4	2	1	30	19	24	21	10	9
Tarte	29	19	2	42	16	19	2	36	18
Konbu	5	24	15	16	3	33	46	15	26

### Items Mapping in WPF HeatMap (SfHeatMap) control

External data source can be mapped with HeatMap using `ItemsMapping` property. It supports 2 kind of data source.

- In `TableMapping` rows represents an objects in collection, columns represents numerical properties of that object.
- In `CellMapping` each cell represent an object in collection, this collection is grouped based on specific property to form as rows and columns.

Let us see the difference between two types of mapping. Following table represents two different data structure to represent the same HeatMap.

CellMapping	TableMapping
<code>{% highlight C# %}public class ProductInfo{ public string ProductName { get; set; } public int Year { get; set; } public double Value { get; set; }}{% endhighlight %}</code>	<code>{% highlight C# %}public class ProductInfo{ public string ProductName { get; set; } public double Y2010 { get; set; } public double Y2011 { get; set; } public double Y2012 { get; set; } public double Y2013 { get; set; } public double Y2014 { get; set; } public double Y2015 { get; set; }}{% endhighlight %}</code>
Here, a single <code>ProductInfo</code> object represent a value for a particular product in a particular year	Here, a single <code>ProductInfo</code> object represents value for a particular product from year 2010 to 2015.
<code>{% highlight xaml %} {% endhighlight %}</code>	<code>{% highlight xaml %} {% endhighlight %}</code>
<code>{% include image.html url="Images/ItemsMapping.png"%}</code>	<code>{% include image.html url="Images/ItemsMapping.png"%}</code>

### Color Mapping in WPF HeatMap (SfHeatMap) control

Color mapping is used to indicate values as colors instead of numerical values. For example, if a HeatMap represents a data from 0 to 100. `ColorMapping` is used to specify a color for lower value and higher value. For any value between two values, a medium color will be automatically be chosen.

In color mapping, when white color is set to value 0 and red color is set for value 30, as shown below.

### XML

```
<syncfusion:ColorMappingCollection x:Key="colorMapping">
<syncfusion:ColorMapping Value="0" Color="White"/>
<syncfusion:ColorMapping Value="30" Color="Red"/>
</syncfusion:ColorMappingCollection>
```

Resultant HeatMap will be as shown below.

Product	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
Alice Mutton	3	4	3	4	10	6	7	6	8	5
Boston Crab Meat	10	11	10	12	15	13	11	10	12	11
Raclette Courdavault	12	12	15	18	19	20	22	21	22	25
Gorgonzola Telino	1	1	2	3	2	2	3	2	3	3
Chartreuse verte	15	14	14	13	10	8	9	9	8	8
Flateemysost	3	3	4	5	4	6	8	2	5	7
Carnarvon Tigers	20	21	20	20	20	21	20	20	21	22
Vegie-spread	15	19	20	21	22	23	24	25	25	25
Tarte au sucre	1	2	3	3	4	4	7	10	12	16
Konbu	10	8	6	7	8	10	11	12	11	13
Valkoinen suklaa	20	20	19	20	15	12	6	3	3	3
Perth Pasties	12	13	13	15	15	18	19	19	22	22

### Legend in WPF HeatMap (SfHeatMap) control

Legend is a control used to summarize the range of colors in HeatMap. This gives visual guideline for mapping between value and color.

#### Create Legend

Legend can be created with color mapping as shown below,

### XML

```
<syncfusion:ColorMappingCollection x:Key="colorMapping">
<syncfusion:ColorMapping Value="0" Color="#fe0002" Label="Poor"/>
<syncfusion:ColorMapping Value="3" Color="#ffff01" Label="Average"/>
<syncfusion:ColorMapping Value="6" Color="#13ab11" Label="Good"/>
<syncfusion:ColorMapping Value="10" Color="#005ba2" Label="Excellent"/>
</syncfusion:ColorMappingCollection>
<syncfusion:SfHeatMapLegend ColorMappingCollection="{StaticResource
colorMapping}"/>
```

Resultant legend will be like following image,



### Legend Mode

There are two modes for Legend

- Gradient
- List

#### Gradient:

### XML

```
<syncfusion:SfHeatMapLegend
LegendMode="Gradient"
ColorMappingCollection="{StaticResource colorMapping}"/>
```



List:

#### XML

```
<syncfusion:SfHeatMapLegend
  LegendMode="List"
  ColorMappingCollection="{StaticResource colorMapping}"/>
```



#### Orientation

There are 2 types of Orientation, applicable for Gradient and List Mode

- Horizontal
- Vertical

Horizontal:

#### XML

```
<syncfusion:SfHeatMapLegend
  LegendMode="List"
  Orientation="Horizontal"
  ColorMappingCollection="{StaticResource colorMapping}"/>
```



Vertical:

#### XML

```
<syncfusion:SfHeatMapLegend
  LegendMode="List"
  Orientation="Vertical"
  ColorMappingCollection="{StaticResource colorMapping}"/>
```





## SfDigitalGauge

### WPF Digital Gauge (SfDigitalGauge) Overview

The Digital Gauge control is used to display alphanumeric characters in digital (LED Display) mode. Digital gauge is used to display a range of values that uses character in combination with numbers.

#### Use Cases

- Displays alpha-numeric values as a virtual digital display.
- Displays the current time in a virtual digital clock.
- Displays the speed and distance in a digital speedometers.

### Key Features in WPF Digital Gauge (SfDigitalGauge)

A digital gauge is composed of segments which are a major UI component of the digital gauge. **SfDigitalGauge** comprises of the following segments to display the digital characters.

- 7 Segment Display
- 14 Segment Display
- 16 Segment Display
- 8\*8 Dot Matrix Display

#### 7-Segment Display

These type of digital gauge displays the digital characters in 7 segments. These are mainly used to display the numbers.

#### 14-Segment Display

These type of digital gauge displays the digital characters in 14 segments. These are mainly used to display both the numbers and alphabets.

#### 16-Segment Display

These type of digital gauge displays the digital characters in 16 segments. These type of digital gauge are also mainly used to display both the numbers and alphabets.

#### 8 \* 8 Dot Matrix Display

These type of digital gauge displays the digital character in 8\*8 dot matrix segments where the characters are spotted by the regular brush and remaining dot are spotted by the dimmed brush. These are mainly used to display numbers, characters and special characters.

#### Easy to use

**SfDigitalGauge** is available in Visual Studio toolbox itself, you can easily drag and drop the control from toolbox.

### Getting Started with WPF Digital Gauge (SfDigitalGauge)

This section explains you the steps required to configure the **SfDigitalGauge** and also explains the steps to add basic elements of **SfDigitalGauge** through various API's available within it.

#### Adding gauge references

You can add gauge reference using one of the following methods:

### Method 1: Adding gauge reference from nuget.org

Syncfusion WPF components are available in [nuget.org](https://www.nuget.org/packages/Syncfusion.SfGauge.WPF). To add gauge to your project, open the NuGet package manager in Visual Studio, search for [Syncfusion.SfGauge.WPF](https://www.nuget.org/packages/Syncfusion.SfGauge.WPF), and then install it.

![Adding gauge reference from NuGet](Getting-Started\_images/Adding gauge reference.png)

### Method 2: Adding gauge reference from toolbox

You can drag the digital gauge control from the toolbox and drop it to the designer. It will add the required assembly references automatically, and add the namespace to the page.

### Method 3: Adding gauge assemblies manually from the installed location

If you prefer to manually reference the assemblies instead referencing from NuGet, add the following assemblies in respective projects.

Location: {Installed location}/{version}/WPF/Assemblies

You can refer to [this](#) link to know about the assemblies required for adding gauge to your project.

#### Initialize gauge

You can initialize the [SfDigitalGauge](#) control with a required optimal name using the included namespace.

#### XML

```
xmlns:gauge="clr-namespace:Syncfusion.UI.Xaml.Gauges;assembly=Syncfusion.SfGauge.Wpf"
```

#### C#

```
using Syncfusion.UI.Xaml.Gauges;
```

initialize digital gauge control.

#### XML

```
<syncfusion:SfDigitalGauge/>
```

#### C#

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
this.Content = digitalgauge;
```

Run the above code and now the default **SfDigitalGauge** can be displayed as follows. UI component of the digital gauge can be customized by adding segments and passing Values which will be explained in the next section.

#### Displaying Values

The [SfDigitalGauge](#) control provides options to display special characters or values using the [Value](#) property.

#### XML

```
<syncfusion:SfDigitalGauge Value="1 2 3 4" />
```

### C#

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "1 2 3 4";  
this.Content = digitalgauge;
```

### Setting character type

By using the [CharacterType](#) property, you can set the segments for digital gauge. The digital characters can be drawn in the following four different segments:

- EightCrossEightDotMatrix
- SegmentFourteen
- SegmentSeven
- SegmentSixteen

### XML

```
<gauge:SfDigitalGauge Value="1 2 3 4"  
CharacterType="EightCrossEightDotMatrix"/>
```

### C#

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.CharacterType = CharacterType.EightCrossEightDotMatrix;  
this.Content = digitalgauge;
```

### Configuring properties

The [CharacterHeight](#), [CharacterWidth](#), and [CharacterStroke](#) properties are used to display characters, which can be customized as shown in the following code snippets

### XML

```
<gauge:SfDigitalGauge CharacterHeight="60" CharacterWidth="25"  
CharacterStroke="#146CED"/>
```

### C#

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.CharacterHeight = 60;  
digitalgauge.CharacterWidth = 25;  
digitalgauge.CharacterStroke = (SolidColorBrush) new  
BrushConverter().ConvertFrom("#146CED");  
this.Content = digitalgauge;
```

The following code example is the complete code of the previous configurations.

### XML

```
<gauge:SfDigitalGauge Value="11:59:50 PM"  
Height="100"  
Width="375"/>
```

```
DimmedBrush="#F2F2F2"
CharacterHeight="60"
CharacterWidth="25"
HorizontalAlignment="Center"
VerticalAlignment="Center"
CharacterType="EightCrossEightDotMatrix"
CharacterStroke="#146CED" />
```

**C#**

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();
digitalgauge.Height = 100;
digitalgauge.Width = 375;
digitalgauge.Value = "11:59:50 PM";
digitalgauge.CharacterHeight = 60;
digitalgauge.CharacterWidth = 25;
digitalgauge.HorizontalAlignment = HorizontalAlignment.Center;
digitalgauge.VerticalAlignment = VerticalAlignment.Center;
digitalgauge.CharacterType = CharacterType.EightCrossEightDotMatrix;
digitalgauge.CharacterStroke = (SolidColorBrush)new
BrushConverter().ConvertFrom("#146CED");
digitalgauge.DimmedBrush = (SolidColorBrush)new
BrushConverter().ConvertFrom("#F2F2F2");
this.Content = digitalgauge;
```

The following screenshot illustrates the result of the previous codes.

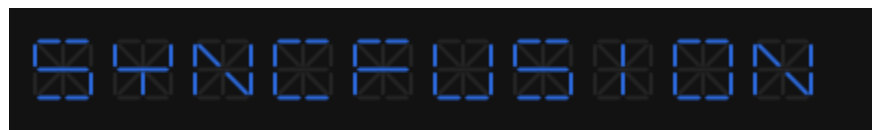


You can find the complete getting started sample from this [link](#).

**Theme**

SfDigitalGauge supports various built-in themes. Refer to the below links to apply themes for the SfDigitalGauge,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



See also

[How to apply themes for SfDigitalGauge](#)

**Digital Characters in WPF Digital Gauge (SfDigitalGauge)**

The digital characters in the digital gauge can be viewed in different types of segments. These digital characters are set to the digital gauge through the [Value](#) property of type string.

### XML

```
<syncfusion:SfDigitalGauge Value="GAUGE" />
```

### C#

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "GAUGE";  
this.Grid.Children.Add(digitalgauge);
```



### 7-Segments

The digital characters which are set as the value property of the digital gauge are displayed by default 7-segments. Rather than using the alphabets, these are mainly used to display numbers. The type of segments can be set by the [CharacterType](#) property.

### XML

```
<syncfusion:SfDigitalGauge Value="12345" CharacterType="SegmentSeven" />
```

### C#

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "12345";  
digitalgauge.CharacterType = CharacterType.SegmentSeven;  
this.Grid.Children.Add(digitalgauge);
```



### 14-Segments

The digital characters which are set as the value property of the digital gauge are displayed by 14 segments. These type of characters are used to display both alphabets and numbers.

### XML

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION"  
CharacterType="SegmentFourteen" />
```

**C#**

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "SYNCFUSION";  
digitalgauge.CharacterType = CharacterType.SegmentFourteen;  
this.Grid.Children.Add(digitalgauge);
```

**16-Segments**

The digital characters which are set as the value property of the digital gauge are displayed by 16 segments. These type of characters are also used to display both alphabets and numbers.

**XML**

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION" CharacterType="SegmentSixteen"  
/>
```

**C#**

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "SYNCFUSION";  
digitalgauge.CharacterType = CharacterType.SegmentSixteen;  
this.Grid.Children.Add(digitalgauge);
```

**8\*8 Dot Matrix Segments**

The digital characters which are set as the value property of the digital gauge are displayed by eight cross eight dot matrix segments. These type of characters are used to display special characters along with the alphabets and numbers.

**XML**

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION"  
CharacterType="EightCrossEightDotMatrix" />
```

**C#**

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "SYNCFUSION";  
digitalgauge.CharacterType = CharacterType.EightCrossEightDotMatrix;  
this.Grid.Children.Add(digitalgauge);
```

{% endtabs%}



### Transformation of Characters in WPF Digital Gauge (SfDigitalGauge)

The digital characters in the digital gauge can be transformed by setting certain properties in the digital gauge. Two kinds of transformations are done using this property. They are:

- Scaling
- Skewing

#### Scaling

The value of the digital characters is scaled by altering the height and width of the digital characters. It is achieved by setting the [CharacterHeight](#) and [CharacterWidth](#) property in the digital gauge.

#### *CharacterHeight*

##### **XML**

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION" CharacterHeight="70"  
CharacterType="SegmentFourteen" />
```

##### **C#**

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "SYNCFUSION";  
digitalgauge.CharacterHeight = 70;  
digitalgauge.CharacterType = CharacterType.SegmentFourteen;  
this.Grid.Children.Add(digitalgauge);
```



### CharacterWidth

#### XML

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION" CharacterWidth="60"  
CharacterType="SegmentFourteen" />
```

#### C#

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "SYNCFUSION";  
digitalgauge.CharacterWidth = 60;  
digitalgauge.CharacterType = CharacterType.SegmentFourteen;  
this.Grid.Children.Add(digitalgauge);
```



### Skewing

The digital gauge also performs skew transformation for the digital characters. It can be done on both x-axis and y-axis through [SkewAngleX](#) and [SkewAngleY](#) properties respectively.

#### SkewAngleX

##### XML

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION" SkewAngleX="35"  
CharacterType="SegmentFourteen" />
```

#### C#

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "SYNCFUSION";  
digitalgauge.SkewAngleX = 35;  
digitalgauge.CharacterType = CharacterType.SegmentFourteen;  
this.Grid.Children.Add(digitalgauge);
```



#### SkewAngleY

##### XML

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION" SkewAngleY="30"  
CharacterType="SegmentFourteen" />
```



**C#**

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "SYNCFUSION";  
digitalgauge.SkewAngleY = 30;  
digitalgauge.CharacterType = CharacterType.SegmentFourteen;  
this.Grid.Children.Add(digitalgauge);
```

**Settings in WPF Digital Gauge (SfDigitalGauge)**

There are some other elements/behavior in SfDigitalGauge also can be customized.

They are:

- Character Spacing
- Segment Thickness
- RTL (Right to Left) support
- Character Stroke
- Dimmed Brush stroke
- Dimmed Brush opacity

**Character Spacing**

The distance between the characters can be set by using the [CharacterSpacing](#) property.

**XML**

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION" CharactersSpacing="50"/>
```

**C#**

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "SYNCFUSION";  
digitalgauge.CharacterSpacing = 50;  
this.Grid.Children.Add(digitalgauge);
```

**Character Stroke**

The Stroke of the character can be changed by [CharacterStroke](#) property.

### XML

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION"  
CharacterType="SegmentFourteen"  
CharacterStroke="Blue" />
```

### C#

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "SYNCFUSION";  
digitalgauge.CharacterStroke = new SolidColorBrush(Colors.Blue);  
digitalgauge.CharacterType = CharacterType.SegmentFourteen;  
this.Grid.Children.Add(digitalgauge);
```



### Segment Thickness

Using [SegmentThickness](#) property, you can adjust the thickness of the segment.

### XML

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION"  
CharacterType="SegmentFourteen"  
SegmentThickness="5" />
```

### C#

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();  
digitalgauge.Value = "SYNCFUSION";  
digitalgauge.SegmentThickness = 5;  
digitalgauge.CharacterType = CharacterType.SegmentFourteen;  
this.Grid.Children.Add(digitalgauge);
```



### RTL (Right to Left) support

The Characters are aligned using [EnableRTLFormat](#) property. The default value of `EnableRTLFormat` is `false`.

**XML**

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION"
CharacterType="SegmentFourteen"
EnableRTLFormat="True" />
```

**C#**

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();
digitalgauge.Value = "SYNCFUSION";
digitalgauge.EnableRTLFormat= true;
digitalgauge.CharacterType = CharacterType.SegmentFourteen;
this.Grid.Children.Add(digitalgauge);
```



Dimmed Brush stroke

[DimmedBrush](#) property is used to apply brushes to the dimmed segment. This property is used to suit the background of the Digital gauge

**XML**

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION"
CharacterType="SegmentFourteen"
DimmedBrush="SkyBlue" />
```

**C#**

```
SfDigitalGauge digitalgauge = new SfDigitalGauge();
digitalgauge.Value = "SYNCFUSION";
digitalgauge.DimmedBrush = new SolidColorBrush(Colors.SkyBlue);
digitalgauge.CharacterType = CharacterType.SegmentFourteen;
this.Grid.Children.Add(digitalgauge);
```



Dimmed Brush opacity

[DimmedBrushOpacity](#) property is used to set the opacity of the brushes to the dimmed segment.

**XML**

```
<syncfusion:SfDigitalGauge Value="SYNCFUSION" DimmedBrush="Blue"
DimmedBrushOpacity="20" CharacterType="SegmentFourteen" />
```

**C#**

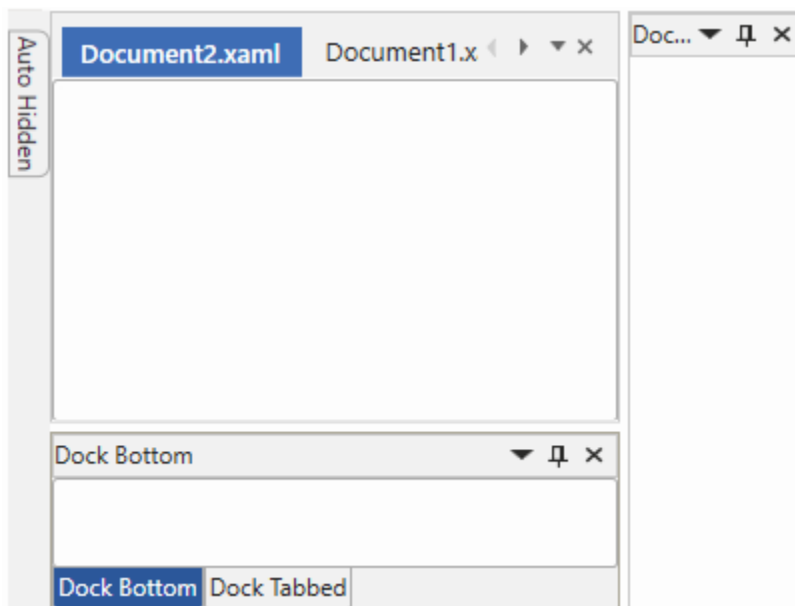
```
SfDigitalGauge digitalgauge = new SfDigitalGauge();
digitalgauge.Value = "SYNCFUSION";
digitalgauge.DimmedBrush = new SolidColorBrush(Colors.Blue);
digitalgauge.CharacterType = CharacterType.SegmentFourteen;
digitalgauge.DimmedBrushOpacity = 20;
this.Grid.Children.Add(digitalgauge);
```



## DockingManager

### WPF Docking (DockingManager) Overview

The [DockingManager](#) control implements an architecture that allows child controls to be docked at any part of a window as in Microsoft Visual Studio. The dock panels containing the child elements can be interactively dragged to any area within the window. These windows can also be floated, tabbed, and auto hidden at run time.



### Key features

**DockState** : Supports the different dock states such as Docking, Floating, AutoHide and Document.

**MDI** : Allows the multiple windows to reside under a single parent window in the Docking Manager.

**TDI** : Allows adding the child window as a document tab in the Docking Manager.

**Themes** : Supports several built-in themes such as Metro, Office2016, and more.

**Tab Switchers** : Supports the various window switchers for navigating between the tabs.

**Customization** : Supports to customize the appearance of Float, Document, AutoHidden and Tabbed windows.

**Nested Layout** : Provides complete support to add the docking manager as a child window to another docking manager.

**Serialization** : Allows you to save and load dock panels states in different formats such as binary and XML.

**ContextMenu** : Provides support to the built-in context menu option for float, close, pin and tab docking windows or child.

**Window Dragging** : Provides support for various styles of window dragging like border, shadow and content dragging.

**Tooltip** : Provides the tooltip support for all default buttons and docking window header.

**Localization** : Supports complete localization to any desired language of header and context menus of the docking windows.

### Getting Started with WPF Docking (DockingManager)

This section explains how to implement a similar UI as Visual Studio by using the [WPF DockingManager](#) in your project.

#### Assembly deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the [DockingManager](#) control in any application.

You can find more details about installing the NuGet package in a WPF application in the following link:

<<<<<< HEAD

[How to install nuget packages](#)

=====

[How to install nuget packages](#)

---

>>>>> 75e024c6dd1a443a80d7a9cfa02fb694db1b406a

---

#### Creating Application with DockingManager control

In this walk through, user will create a WPF application with [DockingManager](#) control.

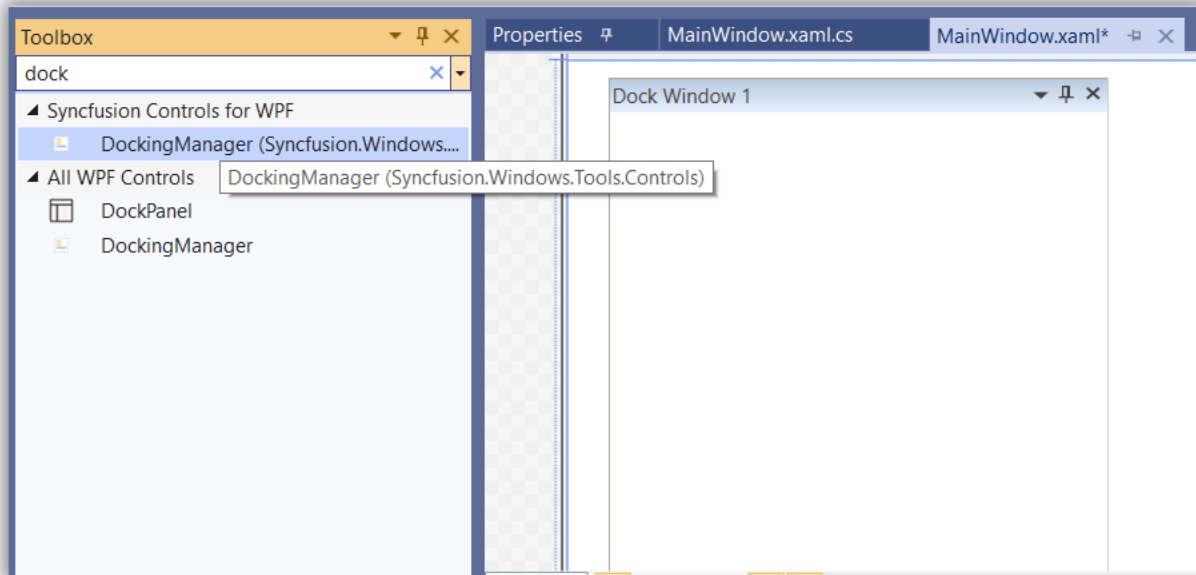
1. [Creating project](#)
2. [Adding control via designer](#)
3. [Adding control manually in XAML](#)
4. [Adding control manually in C#](#)

### Creating project

Below section provides detailed information to create new project in Visual Studio with [DockingManager](#) control.

### Adding control via designer

The [DockingManager](#) control can be added to the application by dragging it from Toolbox and dropping it in designer. The required assembly will be added automatically.



### Adding control manually in XAML

In order to add [DockingManager](#) control manually in XAML, do the below steps,

1. Add the below required assembly references to the project,
  - Syncfusion.Shared.Wpf
  - Syncfusion.Tools.Wpf
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in XAML page or Syncfusion.Windows.Tools.Controls namespace.
3. Declare [DockingManager](#) in XAML page.

### XML

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:GettingStartedComboBox"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="GettingStartedComboBox.MainWindow"
mc:Ignorable="d"
Title="MainWindow" Height="450" Width="800">
<Grid>
<syncfusion:DockingManager x:Name="dockingManager">
<ContentControl x:Name="SolutionExplorer"/>
```

```
<ContentControl x:Name="ToolBox"/>
<ContentControl x:Name="Properties"/>
<ContentControl x:Name="Output"/>
<ContentControl x:Name="StartPage"/>
</syncfusion:DockingManager>
</Grid>
</Window>
```

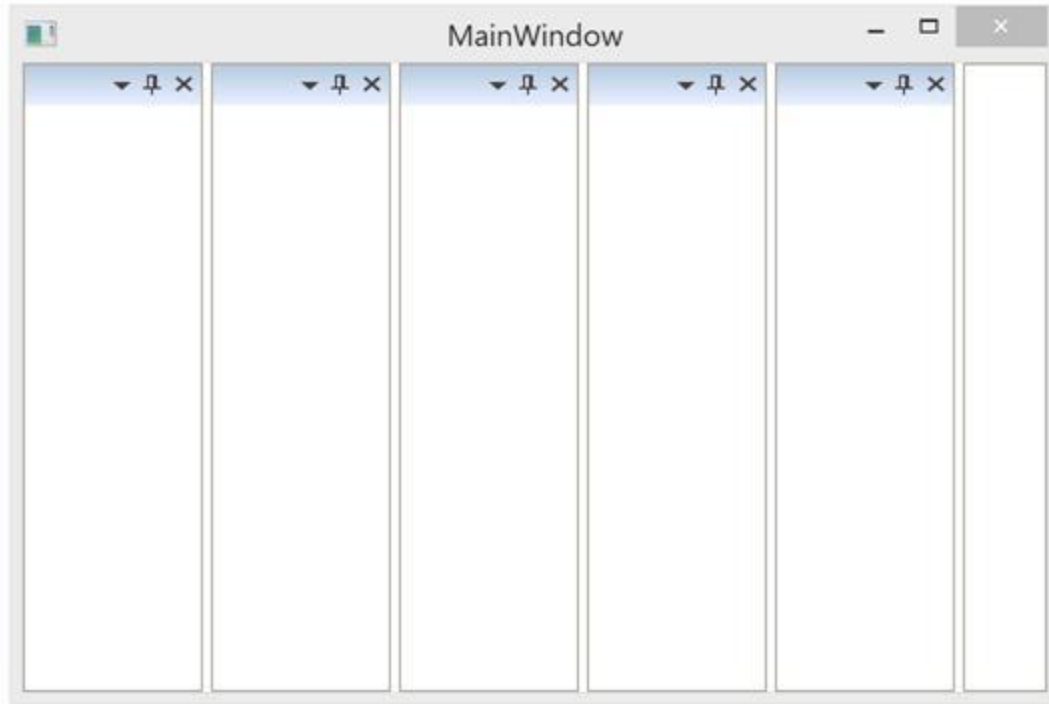
### Adding control manually in C#

In order to add [DockingManager](#) control manually in C#, do the below steps,

1. Add the below required assembly references to the project,
  - Syncfusion.Shared.Wpf
  - Syncfusion.Tools.Wpf
2. Import DockingManager namespace **Syncfusion.Windows.Tools.Controls**.
3. Create DockingManager control instance and add it to the page.

### C#

```
using System.Windows;
using Syncfusion.Windows.Tools.Controls;
namespace DockingManager_Sample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            DockingManager dockingManager = new DockingManager();
            ContentControl SolutionExplorer = new ContentControl();
            ContentControl ToolBox = new ContentControl();
            ContentControl Properties = new ContentControl();
            ContentControl Output = new ContentControl();
            ContentControl StartPage = new ContentControl();
            //Add content controls as child of DockingManager
            dockingManager.Children.Add(SolutionExplorer);
            dockingManager.Children.Add(ToolBox);
            dockingManager.Children.Add(Properties);
            dockingManager.Children.Add(Output);
            dockingManager.Children.Add(StartPage);
            this.Content = dockingManager;
        }
    }
}
```



Set Header for each child window

You can set title for each child windows in [DockingManager](#) using [Header](#) attached property.

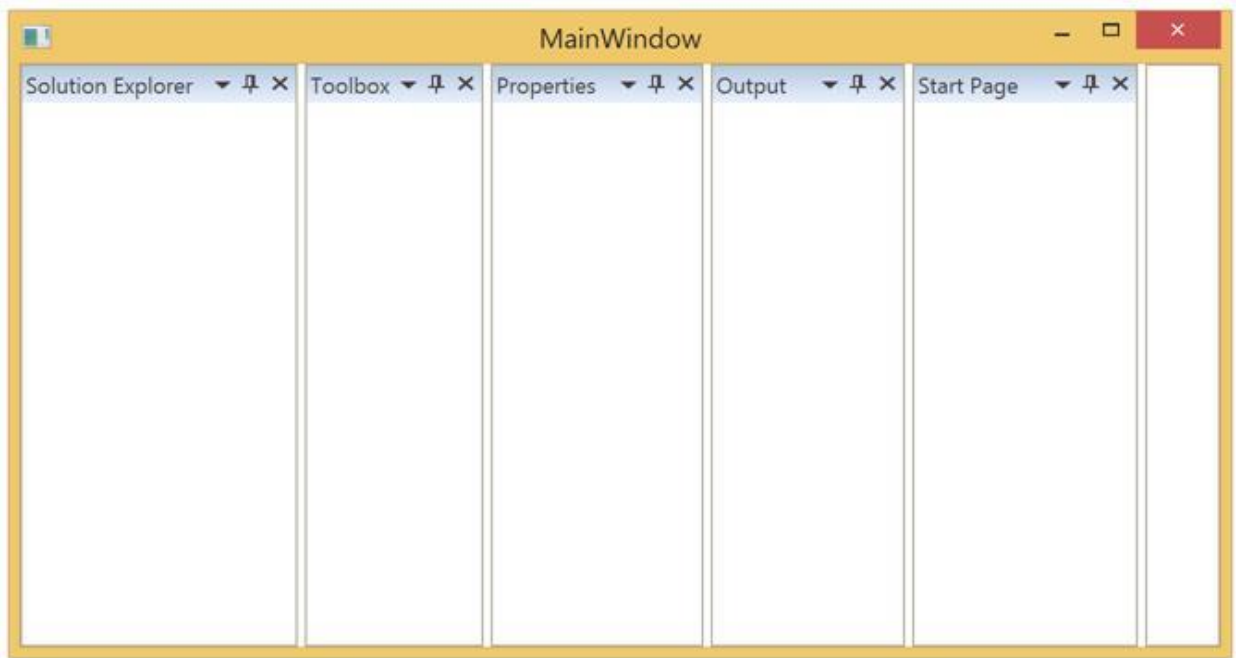
### XML

```
<syncfusion:DockingManager x:Name="dockingManager"
    HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
    <ContentControl x:Name="SolutionExplorer"
        syncfusion:DockingManager.DesiredWidthInDockedMode="150"
        syncfusion:DockingManager.Header="Solution Explorer" />
    <ContentControl x:Name="ToolBox"
        syncfusion:DockingManager.DesiredWidthInDockedMode="110"
        syncfusion:DockingManager.Header="Toolbox" />
    <ContentControl x:Name="Properties"
        syncfusion:DockingManager.DesiredWidthInDockedMode="110"
        syncfusion:DockingManager.Header="Properties" />
    <ContentControl x:Name="Output"
        syncfusion:DockingManager.DesiredWidthInDockedMode="110"
        syncfusion:DockingManager.Header="Output" />
    <ContentControl x:Name="StartPage"
        syncfusion:DockingManager.DesiredWidthInDockedMode="110"
        syncfusion:DockingManager.Header="Start Page" />
</syncfusion:DockingManager>
```

### C#

```
//Set header of each Content Control
DockingManager.SetHeader(SolutionExplorer, "Solution Explorer");
DockingManager.SetHeader(ToolBox, "Toolbox");
DockingManager.SetHeader(Properties, "Properties");
DockingManager.SetHeader(Output, "Output");
DockingManager.SetHeader(StartPage, "StartPage");
```





#### Set States for each child window

You can set the states of each children window in [DockingManager](#) using the [State](#) attached property. Since **Dock** is the default value, initially all the children are docked Docking Window.

The value of State property can be customized with below,

- **AutoHidden** : Auto hides the assigned child window.
- **Dock** : Docks the assigned child window in DockingManager.
- **Float** : Allows child window to float the assigned child window.
- **Document** : Docks the assigned child window as tab item in DocumentContainer.

---

**Note:** To set **Document** state for any child window [UseDocumentContainer](#) property of DockingManager must be enabled.

---

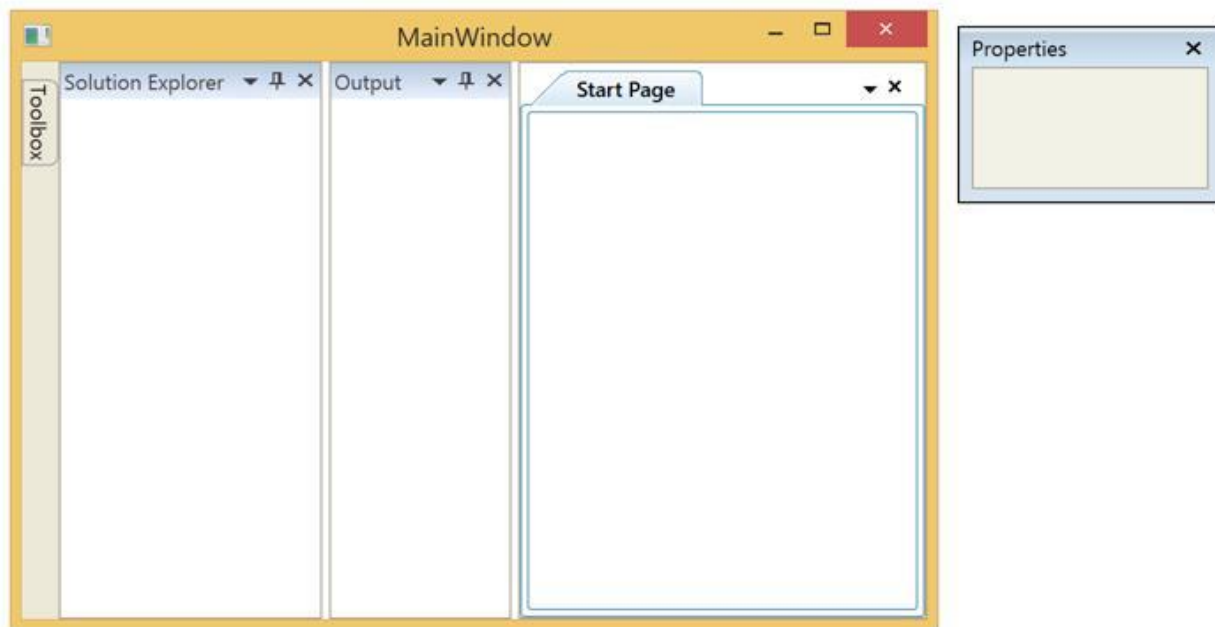
#### XML

```
<syncfusion:DockingManager x:Name="dockingManager"
UseDocumentContainer="True">
  <ContentControl x:Name="SolutionExplorer"
syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.State="Dock"
/>
  <ContentControl x:Name="ToolBox"
syncfusion:DockingManager.Header="Toolbox"
syncfusion:DockingManager.State="AutoHidden"
/>
  <ContentControl x:Name="Properties"
syncfusion:DockingManager.Header="Properties"
syncfusion:DockingManager.State="Float"
/>
```

```
<ContentControl x:Name="Output"
syncfusion:DockingManager.Header="Output"/>
<ContentControl x:Name="StartPage"
syncfusion:DockingManager.Header="Start Page"
syncfusion:DockingManager.State="Document"
/>
</syncfusion:DockingManager>
```

**C#**

```
// Enable UseDocumentContainer to have a document state windows
dockingManager.UseDocumentContainer = true;
//Set State
DockingManager.SetState(SolutionExplorer, DockState.Dock);
DockingManager.SetState(ToolBox, DockState.AutoHidden);
DockingManager.SetState(Properties, DockState.Float);
DockingManager.SetState(Output, DockState.Dock);
DockingManager.SetState(StartPage, DockState.Document);
```

**Set Sides for children**

You can customize the position of child windows in [DockingManager](#) using the [SideInDockMode](#) attached property. Since **Left** is the default value, initially all the windows are docked at left side.

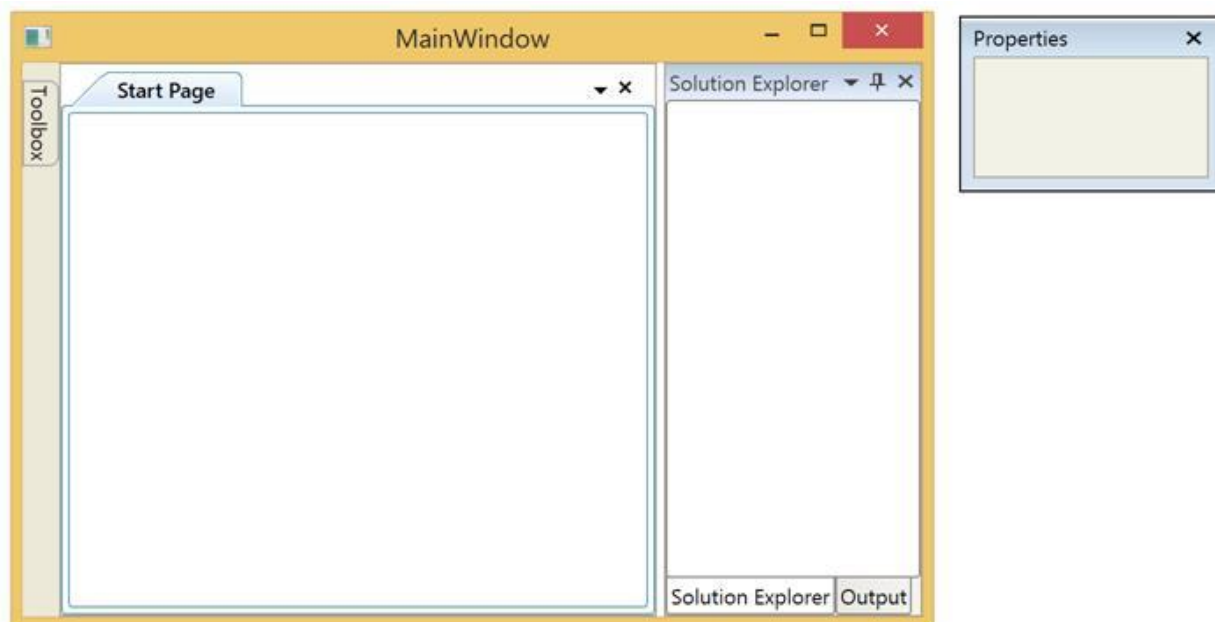
**XML**

```
<syncfusion:DockingManager x:Name="dockingManager"
UseDocumentContainer="True">
<ContentControl syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.SideInDockedMode="Right"
x:Name="SolutionExplorer"/>
<ContentControl x:Name="ToolBox" syncfusion:DockingManager.Header="Toolbox"
syncfusion:DockingManager.State="AutoHidden" />
```

```
<ContentControl x:Name="Properties"
syncfusion:DockingManager.Header="Properties"
syncfusion:DockingManager.State="Float" />
<ContentControl syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="SolutionExplorer"/>
<ContentControl x:Name="StartPage" syncfusion:DockingManager.Header="Start
Page" syncfusion:DockingManager.State="Document"/>
</syncfusion:DockingManager>
```

## C#

```
// Dock at right side
DockingManager.SetSideInDockedMode(SolutionExplorer, DockSide.Right);
// For Tabbed Mode
DockingManager.SetTargetNameInDockedMode(Output, "SolutionExplorer");
DockingManager.SetSideInDockedMode(Output, DockSide.Tabbed);
```



Save / Load the layout

You can automatically save the current layout of [DockingManager](#) while closing the MainWindow by enabling the [PersistState](#) property.

## XML

```
<syncfusion:DockingManager x:Name="dockingManager"
UseDocumentContainer="True" PersistState="True">
<ContentControl syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.SideInDockedMode="Right"/>
<ContentControl syncfusion:DockingManager.Header="Toolbox"
syncfusion:DockingManager.State="AutoHidden" />
<ContentControl syncfusion:DockingManager.Header="Properties"
syncfusion:DockingManager.State="Float" />
```

```
<ContentControl syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Right"/>
<ContentControl syncfusion:DockingManager.Header="Start Page"
syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>
```

### C#

```
SyncDockingManager.PersistState = true;
```

The saved state can be reload by calling the [LoadDockState](#) method, whenever it is required to load the states.

### C#

```
this.SyncDockingManager.LoadDockState();
```

### VB.NET

```
Me.SyncDockingManager.LoadDockState()
```

### Set Visual Styles

You can customize the visual styles of [DockingManager](#) using [SfSkinManager](#). To apply Visual Studio style on the current layout, refer to the following steps.

- Refer the following assemblies in your project.
  1. Syncfusion.SfSkinManager.Wpf
  2. Syncfusion.Themes.VisualStudio2013.Wpf
- Include an namespace for the [SfSkinManager](#) assembly to the MainWindow.

### XML

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:syncfusionskin="clr-
namespace:Syncfusion.SfSkinManager;assembly=Syncfusion.SfSkinManager.WPF"
x:Class="WpfApplication7.MainWindow"
Title="MainWindow" Height="350" Width="525" />
```

### C#

```
using Syncfusion.SfSkinManager;
```

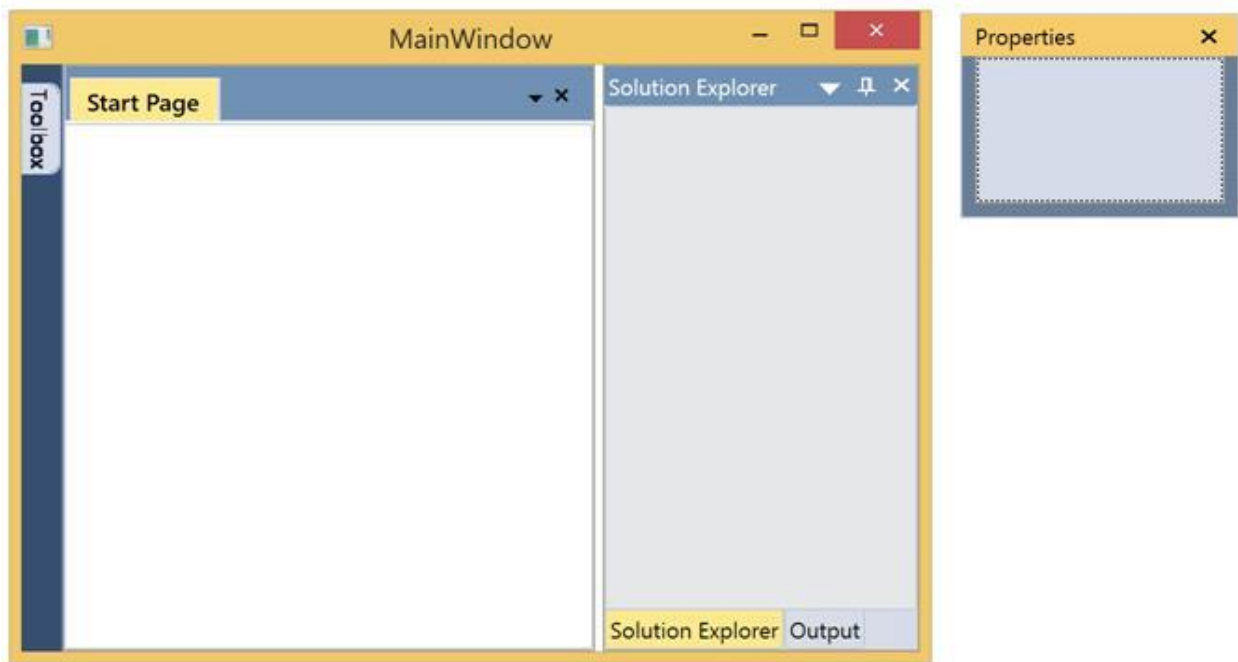
- Now apply the value as [VisualStudio2013](#) to the [VisualStyle](#) property of the SfSkinManager for the DockingManager control.

### XML

```
<syncfusion:DockingManager UseDocumentContainer="True" PersistState="True"
syncfusionskin:SfSkinManager.VisualStudioStyle="VisualStudio2013">
  <ContentControl syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.SideInDockedMode="Right"
x:Name="SolutionExplorer" />
  <ContentControl syncfusion:DockingManager.Header="Toolbox" x:Name="ToolBox"
syncfusion:DockingManager.State="AutoHidden" />
  <ContentControl syncfusion:DockingManager.Header="Properties"
x:Name="Properties" syncfusion:DockingManager.State="Float" />
  <ContentControl syncfusion:DockingManager.Header="Output" x:Name="Output"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="SolutionExplorer"/>
  <ContentControl syncfusion:DockingManager.Header="Start Page"
syncfusion:DockingManager.State="Document" x:Name="StartPage" />
</syncfusion:DockingManager>
```

### C#

```
//Set VisualStyle
SfSkinManager.SetVisualStyle(SyncDockingManager, VisualStyles.VisualStudio2013);
```



### ToolTip for child window

You can show ToolTip when hover the mouse over the header of child windows in [DockingManager](#) using the [CaptionToolTip](#) attached property in XAML and [SetCaptionToolTip](#) method in code behind.

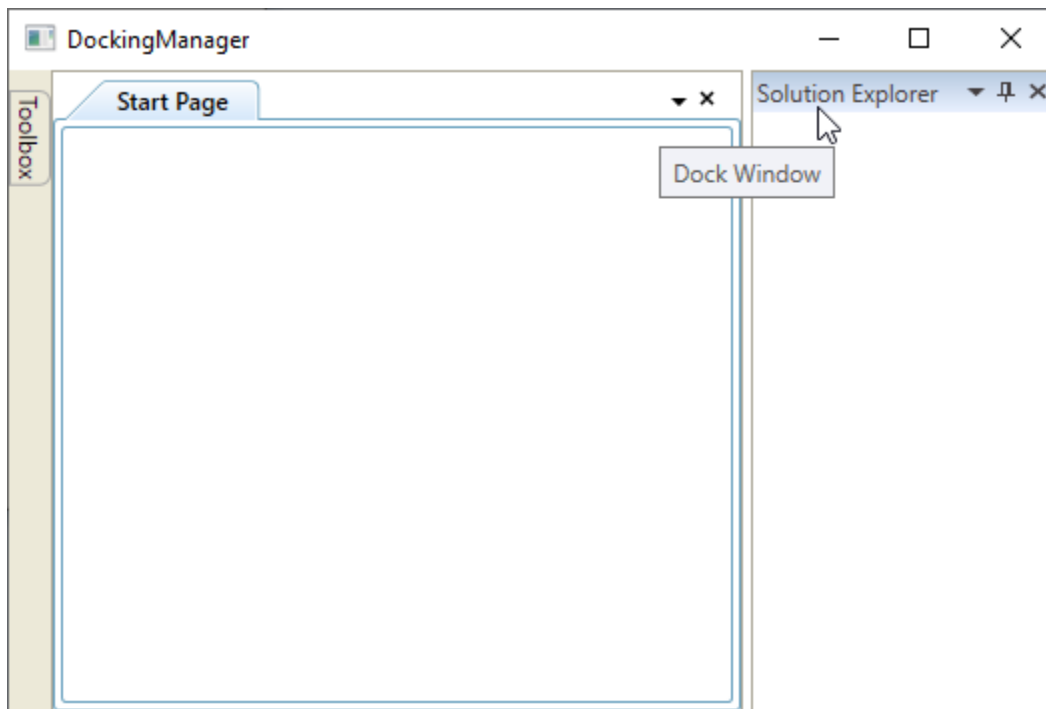
### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True">
```

```
<ContentControl syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.DesiredWidthInDockedMode="175"
x:Name="solutionExplorer" syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.CaptionToolTip="Dock Window"/>
<ContentControl syncfusion:DockingManager.Header="Start Page"
x:Name="startPage" syncfusion:DockingManager.State="Document"
syncfusion:DockingManager.CaptionToolTip="Document Window"/>
<ContentControl syncfusion:DockingManager.Header="Toolbox" x:Name="toolBox"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.CanSerialize="False"
syncfusion:DockingManager.CaptionToolTip="AutoHidden Window"/>
</syncfusion:DockingManager>
```

## C#

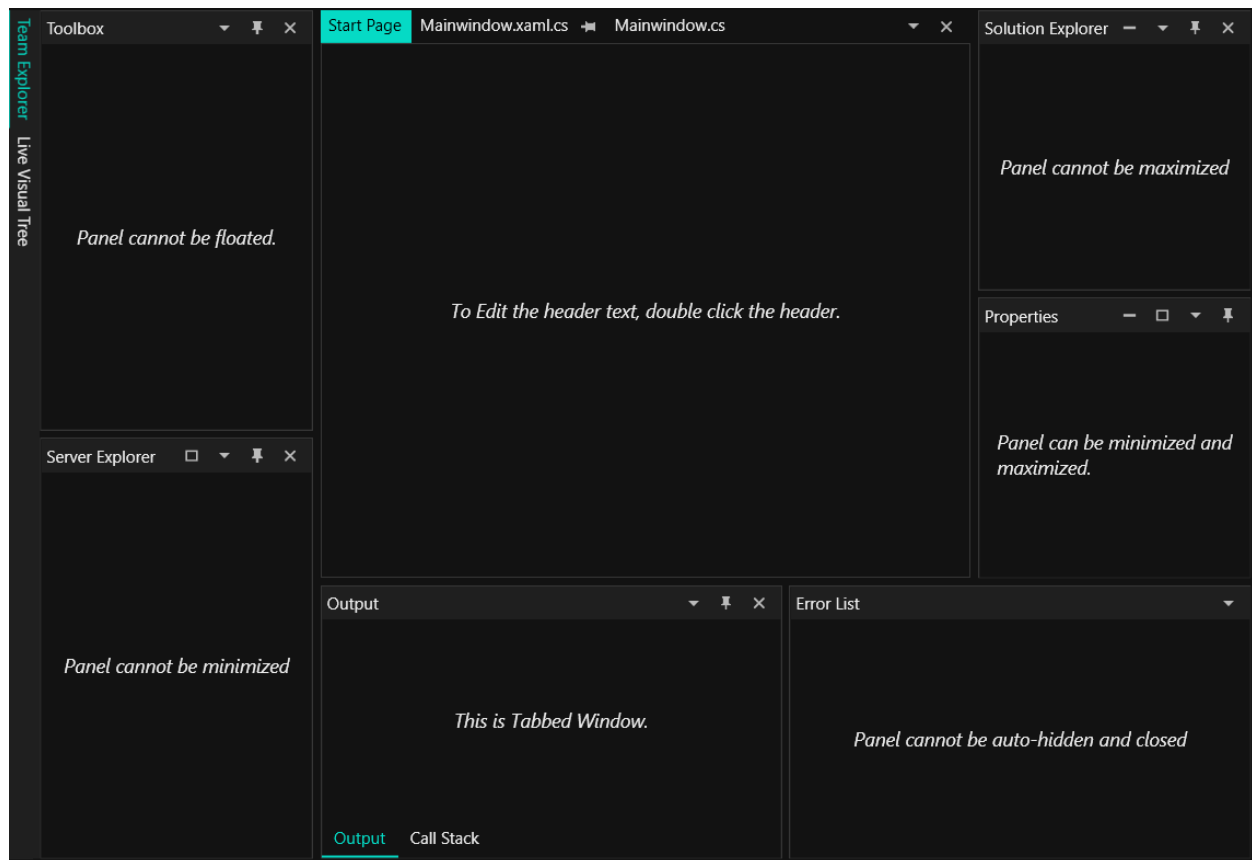
```
DockingManager.SetCaptionToolTip(solutionExplorer, "Dock Window");
DockingManager.SetCaptionToolTip(startPage, "Document Window");
DockingManager.SetCaptionToolTip(toolBox, "AutoHidden Window");
```



## Theme

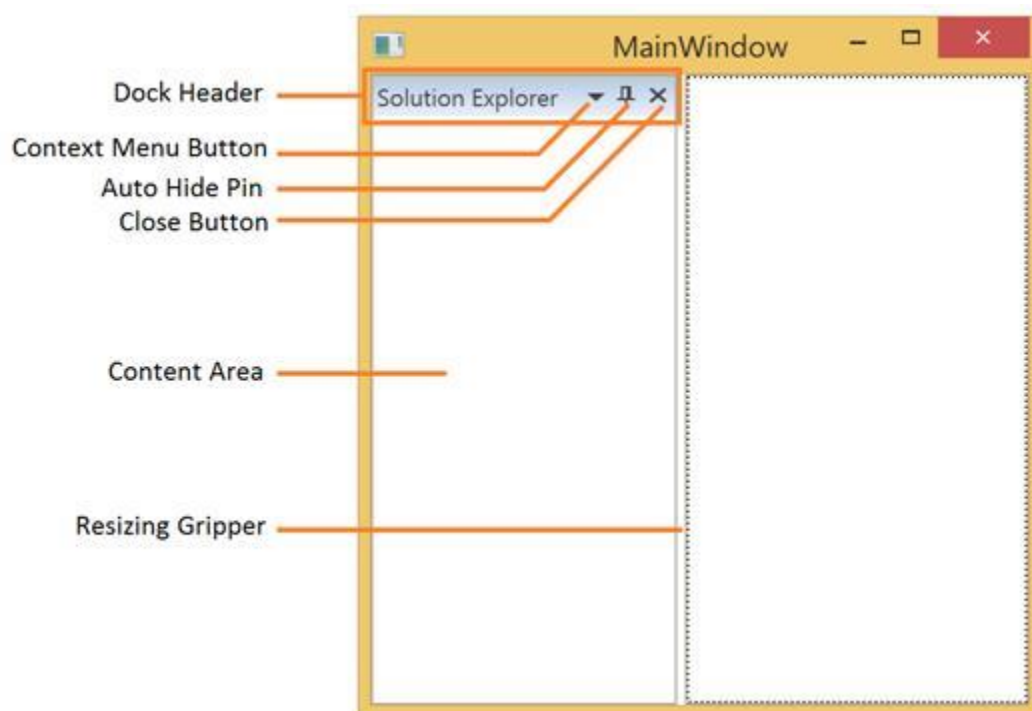
[DockingManager](#) supports various built-in themes. Refer to the below links to apply themes for the DockingManager,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



## Docking Window in WPF Docking (DockingManager)

Docking windows is one of the state of DockingManager. Since `Dock` is the default value, so initially all the children stay as Docking Window



### Configuring window in Different Sides

The five sides that can be docked are

- Left
- Right
- Top
- Bottom
- Tabbed

To dock 4 children of a DockingManager in 4 different sides, then use [SideInDockedMode](#) property with the required values.

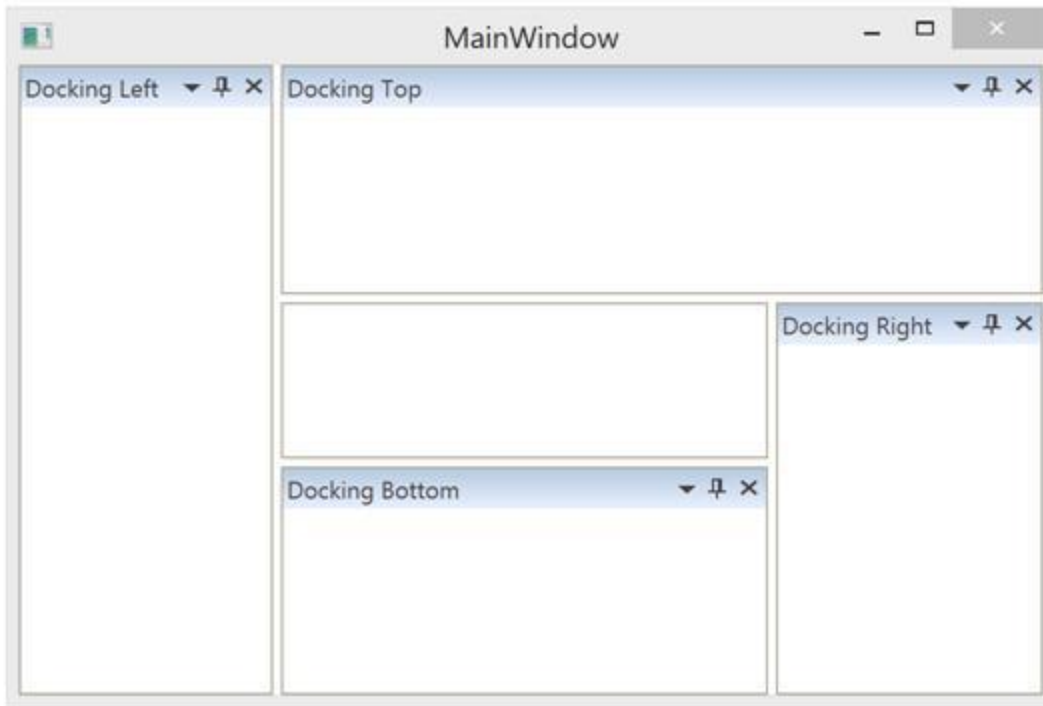
#### XML

```
<ContentControl syncfusion:DockingManager.Header="Docking Left"
syncfusion:DockingManager.SideInDockedMode= "Left" />
<ContentControl syncfusion:DockingManager.Header="Docking Top"
syncfusion:DockingManager.SideInDockedMode= "Top"/>
<ContentControl syncfusion:DockingManager.Header="Docking Right"
syncfusion:DockingManager.SideInDockedMode= "Right"/>
<ContentControl syncfusion:DockingManager.Header="Docking Bottom"
syncfusion:DockingManager.SideInDockedMode="Bottom" />
```

#### C#

```
//Dock at Left
DockingManager.SetSideInDockedMode(dockWindow1, DockSide.Left);
//Dock at Top
DockingManager.SetSideInDockedMode(dockWindow2, DockSide.Top);
//Dock at Right
DockingManager.SetSideInDockedMode(dockWindow3, DockSide.Right);
//Dock at Bottom
DockingManager.SetSideInDockedMode(dockWindow4, DockSide.Bottom);
```





### Docking window in various Targets

Docking window can also be docked at any side of the Target Docking Window through an attached property named [TargetNameInDockedMode](#).

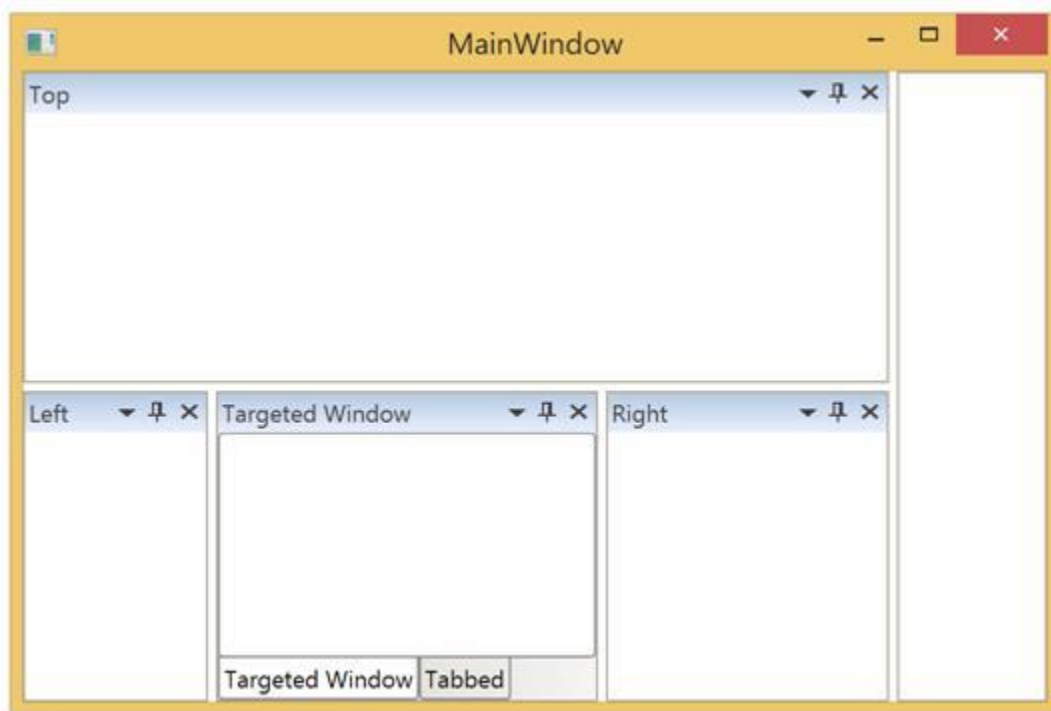
Also to set as Tabbed Window, the window should aware of a Target window name. The following code helps to arrange children of DockingManager that targets a single Docking window docked along Left, Top, Right and Tabbed.

### XML

```
<ContentControl syncfusion:DockingManager.Header="Targeted Window"
x:Name="DockingWindow1"/>
<!--Targeted to Docking Window1 on Top Side-->
<ContentControl syncfusion:DockingManager.Header="Top"
syncfusion:DockingManager.SideInDockedMode="Top"
syncfusion:DockingManager.TargetNameInDockedMode="DockingWindow1"/>
<!--Targeted to DockingWindow1 on Right Side-->
<ContentControl syncfusion:DockingManager.Header="Right"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.TargetNameInDockedMode="DockingWindow1"/>
<!--Targeted to DockingWindow1 on Left Side-->
<ContentControl syncfusion:DockingManager.Header="Left"
syncfusion:DockingManager.SideInDockedMode="Left"
syncfusion:DockingManager.TargetNameInDockedMode="DockingWindow1"/>
<!--Targeted to DockingWindow to tab-->
<ContentControl syncfusion:DockingManager.Header="Tabbed"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="DockingWindow1"/>
```

### C#

```
dockWindow1.Name = "DockingWindow1";
DockingManager.SetTargetNameInDockedMode(dockWindow2, "DockingWindow1");
DockingManager.SetSideInDockedMode(dockWindow2, DockSide.Top);
DockingManager.SetTargetNameInDockedMode(dockWindow3, "DockingWindow1");
DockingManager.SetSideInDockedMode(dockWindow3, DockSide.Right);
DockingManager.SetTargetNameInDockedMode(dockWindow4, "DockingWindow1");
DockingManager.SetSideInDockedMode(dockWindow4, DockSide.Left);
DockingManager.SetTargetNameInDockedMode(dockWindow5, "DockingWindow1");
DockingManager.SetSideInDockedMode(dockWindow5, DockSide.Tabbed);
```



**Note:** View [Sample](#) in GitHub

### Maximize/Minimize Support

This feature helps to Maximize/Minimize Docked Windows for better usage of each window. This support is enabled only when the parent container of a specific element contains more than one host.

- It helps a particular docked window to provide a maximized view
- It can minimize a docked window and can be restored when needed

### Enabling Maximization feature

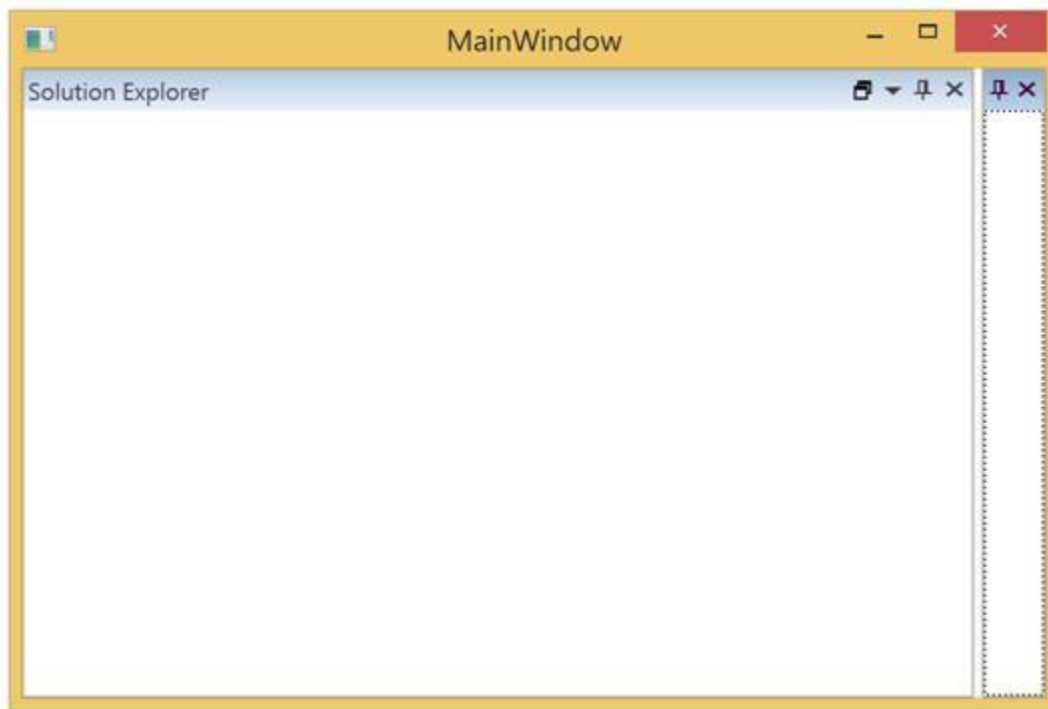
To enable maximizing feature of Docking Window, set [MaximizeButtonEnabled](#) to **True**

### XML

```
<syncfusion:DockingManager MaximizeButtonEnabled="True">
  <ContentControl syncfusion:DockingManager.Header="Solution Explorer"/>
  <ContentControl syncfusion:DockingManager.Header="Toolbox"/>
</syncfusion:DockingManager>
```

**C#**

```
SyncDockingManager.MaximizeButtonEnabled = true;
```



*Maximize Docking Window to full screen*

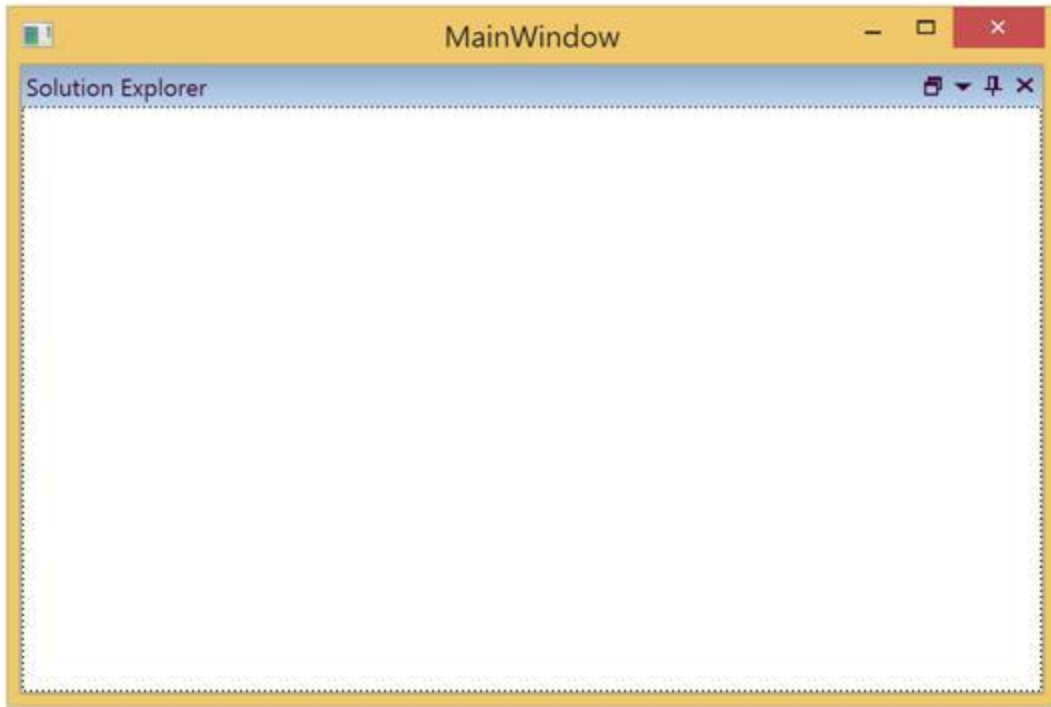
[MaximizeMode](#) helps to change the maximization behavior of DockingWindow. Docking Windows occupies entire screen when [MaximizeMode](#) set as FullScreen and DockingWindow in the Maximized state.

**XML**

```
<syncfusion:DockingManager MaximizeButtonEnabled="True"
MaximizeMode="FullScreen"/>
```

**C#**

```
SyncDockingManager.MaximizeButtonEnabled = true;
SyncDockingManager.MaximizeMode = MaximizeMode.FullScreen;
```



#### Enabling Minimization feature

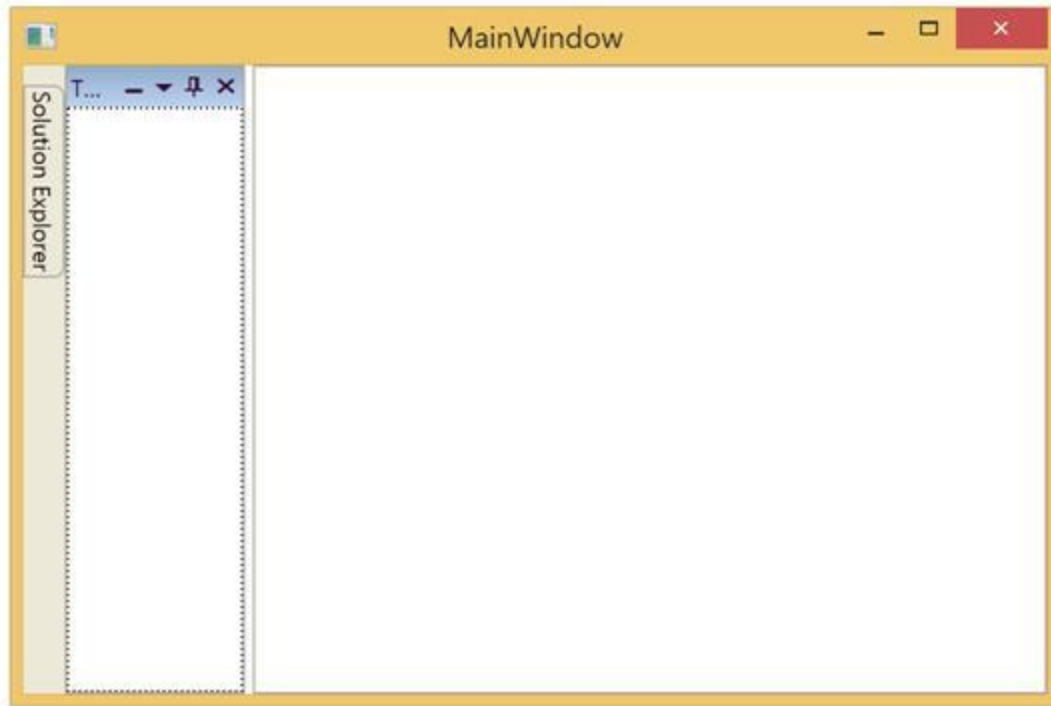
To enable minimizing feature of DockingWindow, set [MinimizeButtonEnabled](#) to **True**

#### XML

```
<syncfusion:DockingManager x:Name="SyncDockingManager"
MinimizeButtonEnabled="True">
  <ContentControl x:Name="SolutionExplorer"
syncfusion:DockingManager.Header="Solution Explorer"/>
  <ContentControl x:Name="ToolBox"
syncfusion:DockingManager.Header="Toolbox"/>
</syncfusion:DockingManager>
```

#### C#

```
SyncDockingManager.MinimizeButtonEnabled = true;
```



#### *Restrict Maximization or Minimization for a specific children*

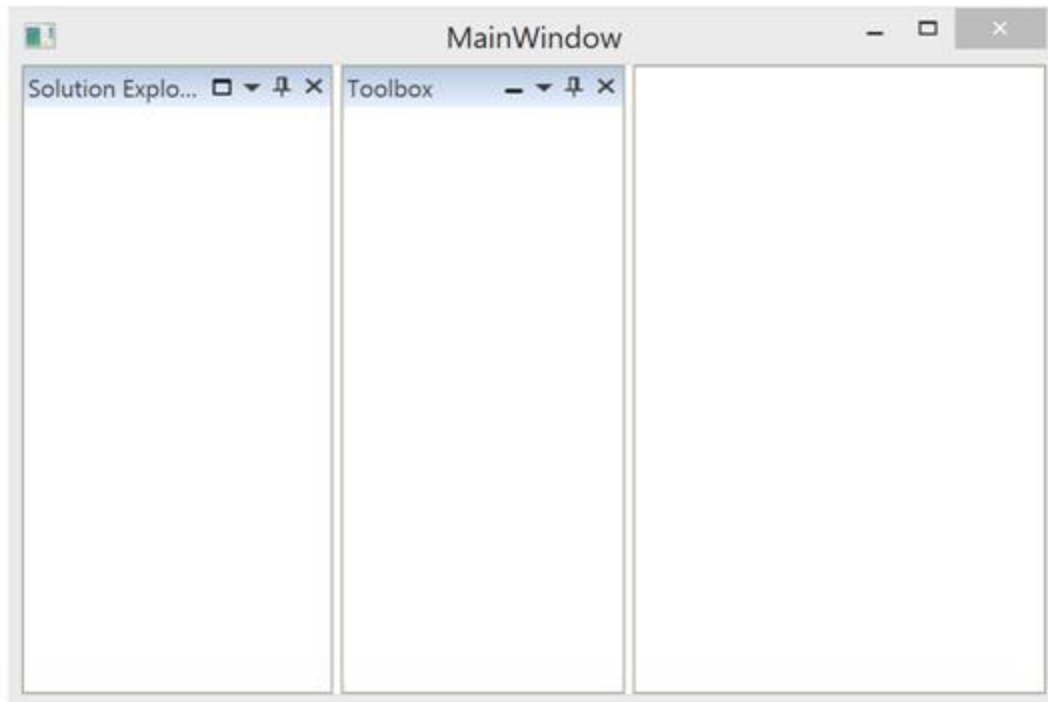
DockingManager provides two attached property named [CanMaximize](#) and [CanMinimize](#) to enable or disable Maximizing and Minimizing buttons respectively to the specific window.

#### **XML**

```
<syncfusion:DockingManager MaximizeButtonEnabled="True"
MinimizeButtonEnabled="True">
  <ContentControl syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.CanMinimize="False" />
  <ContentControl syncfusion:DockingManager.Header="Toolbox"
syncfusion:DockingManager.CanMaximize="False"/>
</syncfusion:DockingManager>
```

#### **C#**

```
SyncDockingManager.MinimizeButtonEnabled = true;
SyncDockingManager.MaximizeButtonEnabled = true;
//Restrict the minimize option for Solution Explorer window
DockingManager.SetCanMinimize(SolutionExplorer, false);
//Restrict the maximize option Toolbox window
DockingManager.SetCanMaximize(Toolbox, false);
```



### Hot Drag the window

The DockWindow Header can be highlighted when the mouse is hovered on an active Docking window by [IsEnableHotTracking](#) property. Default value of IsEnableHotTracking is **False**, to enable this functionality turn its value to **True**.

#### XML

```
<syncfusion:DockingManager x:Name="SyncDockingManager"
    IsEnableHotTracking="True"/>
```

#### C#

```
SyncDockingManager.IsEnableHotTracking = true;
```

### Enabling or Disabling the Dock functionality

The [CanDock](#) property can help to enable or disable the docking functionality by setting its value as **True** or **False**. By default its value is **True**, to disable this functionality turn its value to **False**.

#### XML

```
<ContentControl syncfusion:DockingManager.CanDock="False" />
```

#### C#

```
DockingManager.SetCanDock(dockWindow1, false);
```

### Enabling or Disabling the Header Visibility

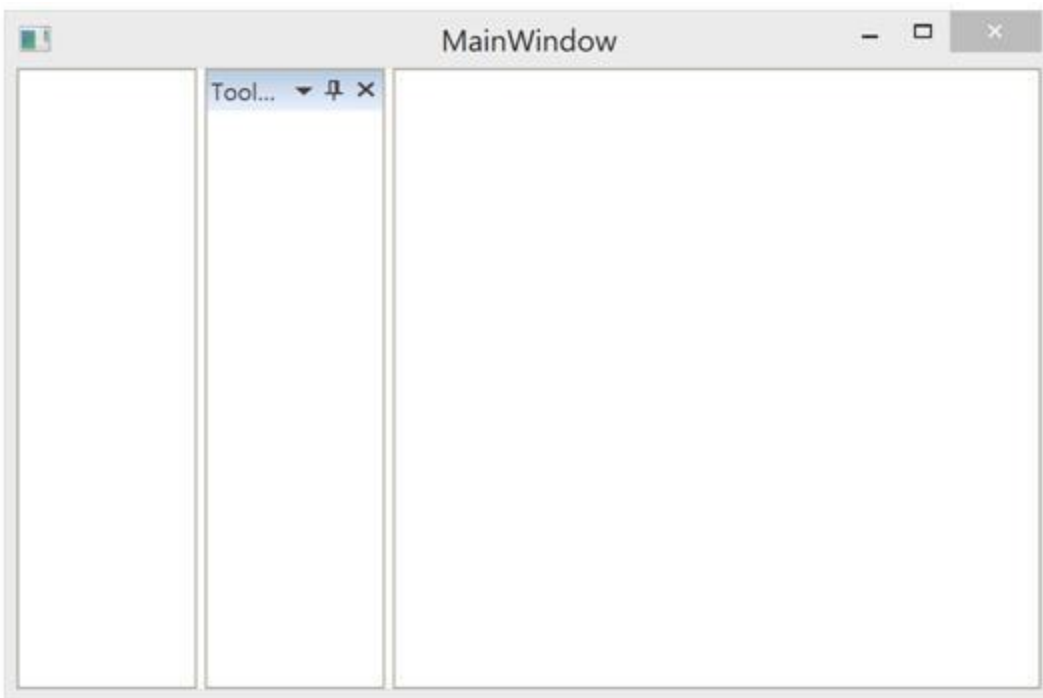
[NoHeader](#) is an attached property, that is used to hide the header of DockWindow. Default value of NoHeader is `False`, to hide the Header turn its value to `True`.

#### XML

```
<ContentControl syncfusion:DockingManager.Header="Toolbox"/>
<!--NoHeader enabled to this child-->
<ContentControl syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.NoHeader="True" />
```

#### C#

```
DockingManager.SetNoHeader(SolutionExplorer, true);
```



### Custom context menu items for docking window

You can add the custom context menu items for docking window by using the [FloatWindowContextMenuItems](#) property. You can also add sub menu items for custom context menu item by adding that sub `CustomMenuItem` to the parent [CustomMenuItem](#). You can check or uncheck the `CustomMenuItem` interactively or programmatically by using the `CustomMenuItem.IsChecked` property.

You can collapse the default docking context menu and show only the custom docking context menu items by setting the [CollapseDefaultContextMenuItemsInDock](#) property to `true`. The default value of `CollapseDefaultContextMenuItemsInDock` property is `false`.

#### XML

```
<syncfusion:DockingManager
DockFill="True"
```

```

Name="dockingManager" >
<!--Adding custom context menu items for docking windows-->
<syncfusion:DockingManager.DockWindowContextMenuItems>
<syncfusion:CustomMenuItemCollection>
<!--Adding custom context menu items-->
<syncfusion:CustomMenuItem Header="Menu 1"/>
<syncfusion:CustomMenuItem Header="Menu 2"/>
<!--Adding sub custom context menu items-->
<syncfusion:CustomMenuItem Header="SubMenu 1"/>
<syncfusion:CustomMenuItem Header="SubMenu 2"/>
<syncfusion:CustomMenuItem Header="SubMenu 3" IsChecked="True"/>
</syncfusion:CustomMenuItem>
</syncfusion:CustomMenuItemCollection>
</syncfusion:DockingManager.DockWindowContextMenuItems>
<ContentControl syncfusion:DockingManager.Header="Targeted Window"
syncfusion:DockingManager.CollapseDefaultContextMenuItemsInDock="True"
x:Name="DockingWindow1"/>
<!--Targeted to Docking Window1 on Bottom Side-->
<ContentControl syncfusion:DockingManager.Header="Bottom"
syncfusion:DockingManager.SideInDockedMode="Bottom"
syncfusion:DockingManager.TargetNameInDockedMode="DockingWindow1"/>
<!--Targeted to DockingWindow1 on Right Side-->
<ContentControl syncfusion:DockingManager.Header="Right"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.TargetNameInDockedMode="DockingWindow1"/>
<!--Targeted to DockingWindow1 on Left Side-->
<ContentControl syncfusion:DockingManager.Header="Left"
syncfusion:DockingManager.SideInDockedMode="Left"
syncfusion:DockingManager.TargetNameInDockedMode="DockingWindow1"/>
<!--Targeted to DockingWindow to tab-->
<ContentControl syncfusion:DockingManager.Header="Tabbed"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="DockingWindow1"/>
</syncfusion:DockingManager>

```

**C#**

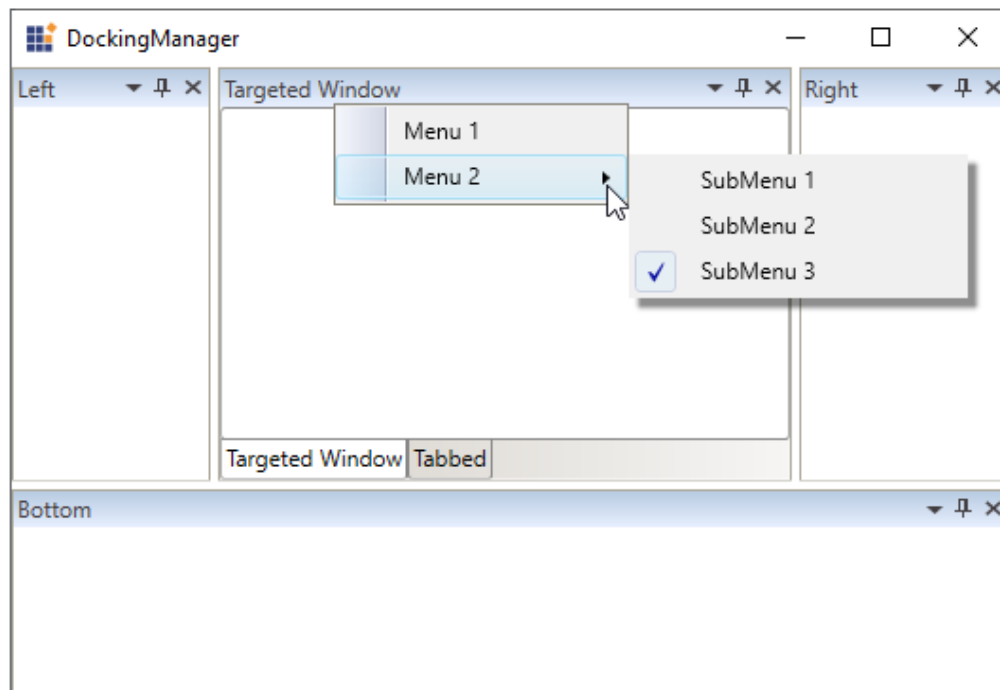
```

//Creating custom context menu items
CustomMenuItem menu1 = new CustomMenuItem();
menu1.Header = "Menu 1";
CustomMenuItem menu2 = new CustomMenuItem();
menu2.Header = "Menu 2";
//Creating custom sub context menu items
CustomMenuItem customMenuItem1 = new CustomMenuItem() { Header = "SubMenu 1"
};
CustomMenuItem customMenuItem2 = new CustomMenuItem() { Header = "SubMenu 2"
};
CustomMenuItem customMenuItem3 = new CustomMenuItem() { Header = "SubMenu
3", IsChecked= true };
menu2.Items.Add(customMenuItem1);
menu2.Items.Add(customMenuItem2);
menu2.Items.Add(customMenuItem3);
//Adding custom context menu items with sub menu items
dockingManager.FloatWindowContextMenuItems.Add(menu1);
dockingManager.FloatWindowContextMenuItems.Add(menu2);

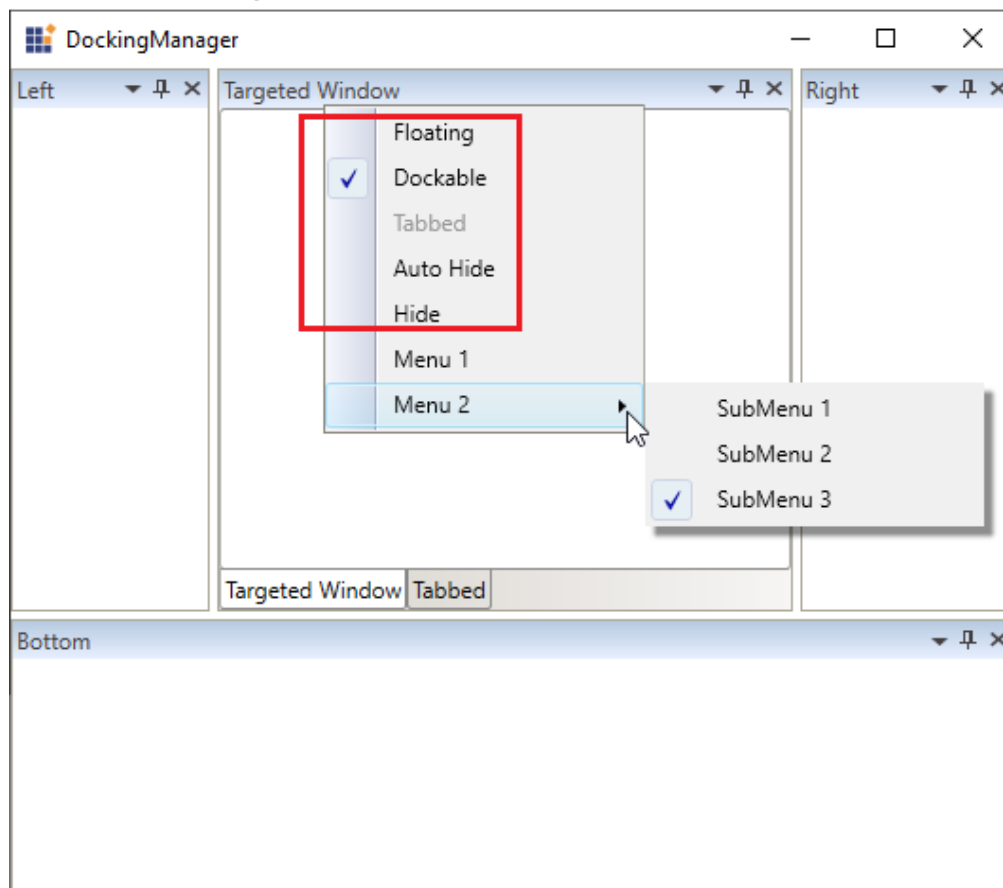
```



CollapseDefaultContextMenuItemsInDock = "True"



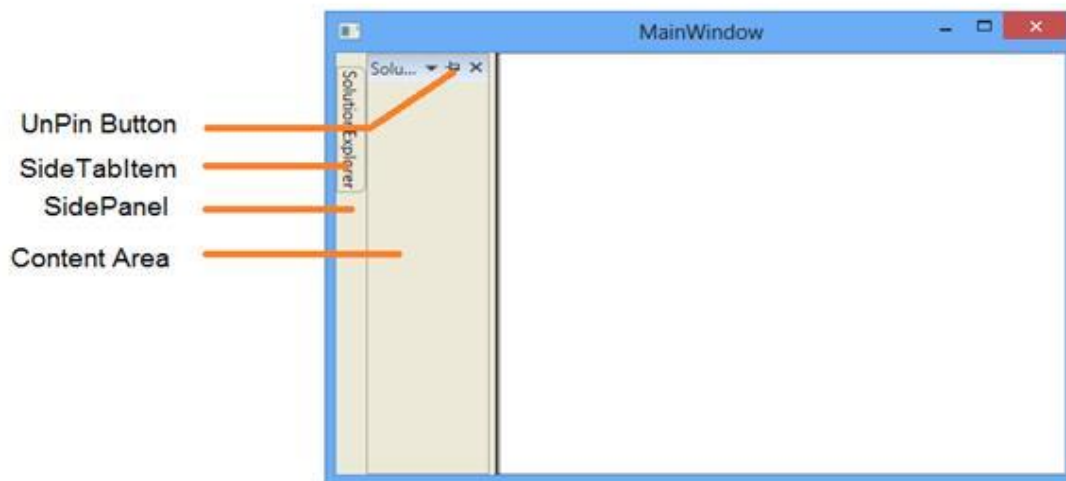
CollapseDefaultContextMenuItemsInDock = "False"



**Note:** [View Sample in GitHub](#)

### Auto Hide Window in WPF Docking (DockingManager)

AutoHide window is one of the state in the DockingManager. To enable Auto hidden for DockingManager's children, set its **State** value as **AutoHidden**.



### Configuring window in Different Side

AutoHidden window can be placed in four different sides such as Top, Bottom, Left and Right. To place the four auto hidden children in four different sides, set [SideInDockedMode](#) property according to its corresponding values in the DockingManager.

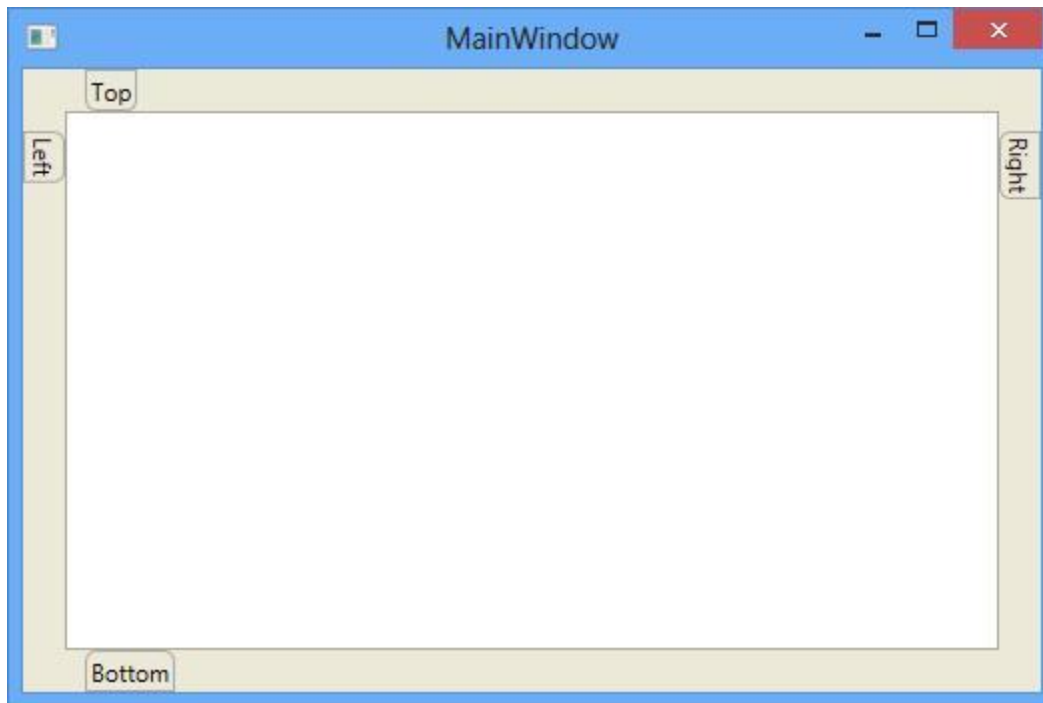
#### XML

```
<ContentControl syncfusion:DockingManager.Header="Top"
x:Name="AutoHideWindow1"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.SideInDockedMode="Top" />
<ContentControl syncfusion:DockingManager.Header="Left"
x:Name="AutoHideWindow2"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.SideInDockedMode="Left" />
<ContentControl syncfusion:DockingManager.Header="Right"
x:Name="AutoHideWindow3"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.SideInDockedMode="Right" />
<ContentControl syncfusion:DockingManager.Header="Bottom"
x:Name="AutoHideWindow4"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.SideInDockedMode="Bottom" />
```

#### C#

```
//To place the auto hidden children in four different sides
DockingManager.SetSideInDockedMode(AutoHideWindow1, DockSide.Top);
DockingManager.SetSideInDockedMode(AutoHideWindow2, DockSide.Left);
DockingManager.SetSideInDockedMode(AutoHideWindow3, DockSide.Right);
```

```
DockingManager.SetSideInDockedMode(AutoHideWindow4, DockSide.Bottom);
```



The AutoHideWindow can be placed on a required target window through the [TargetNameInDockedMode](#) property of the DockingManager. DockingWindow will auto hidden in place according to its Parent position, if any target exist. For example: Here "Output" docked at bottom of "SolutionExplorer" which docked at left side. While auto hiding Output window, it will auto hide at left due to it's TargetWindow side.

### XML

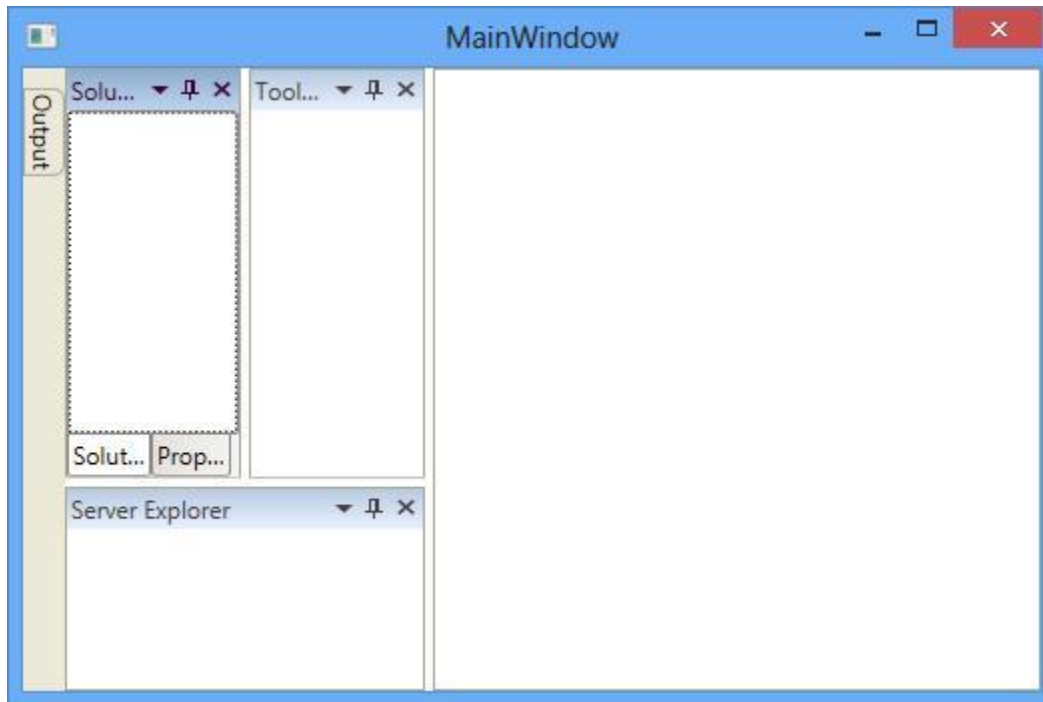
```
<syncfusion:DockingManager>
  <ContentControl syncfusion:DockingManager.Header="SolutionExplorer"
  x:Name="SolutionExplorer"/>
  <ContentControl x:Name="ServerExplorer"
  syncfusion:DockingManager.Header="Server Explorer"
  syncfusion:DockingManager.SideInDockedMode="Bottom"
  syncfusion:DockingManager.TargetNameInDockedMode="SolutionExplorer" />
  <ContentControl x:Name="ToolBox" syncfusion:DockingManager.Header="ToolBox"
  syncfusion:DockingManager.SideInDockedMode="Right"
  syncfusion:DockingManager.TargetNameInDockedMode="SolutionExplorer" />
  <ContentControl x:Name="Output" syncfusion:DockingManager.Header="Output"
  syncfusion:DockingManager.SideInDockedMode="Bottom"
  syncfusion:DockingManager.TargetNameInDockedMode="SolutionExplorer" />
  <ContentControl x:Name="Properties"
  syncfusion:DockingManager.Header="Properties"
  syncfusion:DockingManager.SideInDockedMode="Tabbed"
  syncfusion:DockingManager.TargetNameInDockedMode="SolutionExplorer" />
</syncfusion:DockingManager>
```

### C#

```

DockingManager.SetSideInDockedMode(ServerExplorer, DockSide.Bottom);
DockingManager.SetTargetNameInDockedMode(ServerExplorer,
"SolutionExplorer");
DockingManager.SetSideInDockedMode(ToolBox, DockSide.Right);
DockingManager.SetTargetNameInDockedMode(ToolBox, "SolutionExplorer");
DockingManager.SetSideInDockedMode(Output, DockSide.Bottom);
DockingManager.SetTargetNameInDockedMode(Output, "SolutionExplorer");
DockingManager.SetSideInDockedMode(Properties, DockSide.Tabbed);
DockingManager.SetTargetNameInDockedMode(Properties, "SolutionExplorer");

```



### Side panel Customization

The side panel and side panel header can be customized by applying its Background, BorderBrush, BorderThickness and SidePanelSize through [SidePanelBackground](#), [SidePanelBorderBrush](#), [SidePanelBorderThickness](#), [SidePanelSize](#) properties of the DockingManager. You can also refer to this [sample](#) which demonstrate the side panel customization.

### XML

```

<syncfusion:DockingManager SidePanelBackground="Brown"
SidePanelBorderBrush="Yellow" SideItemsBackground="Green"
SidePanelBorderThickness="2,2,2,2" SideItemsBorderBrush="BlueViolet"
SidePanelSize="40">
  <ContentControl syncfusion:DockingManager.Header="SolutionExplorer"
syncfusion:DockingManager.State="AutoHidden" />
  <ContentControl syncfusion:DockingManager.Header="ToolBox" />
</syncfusion:DockingManager>

```

### C#

```

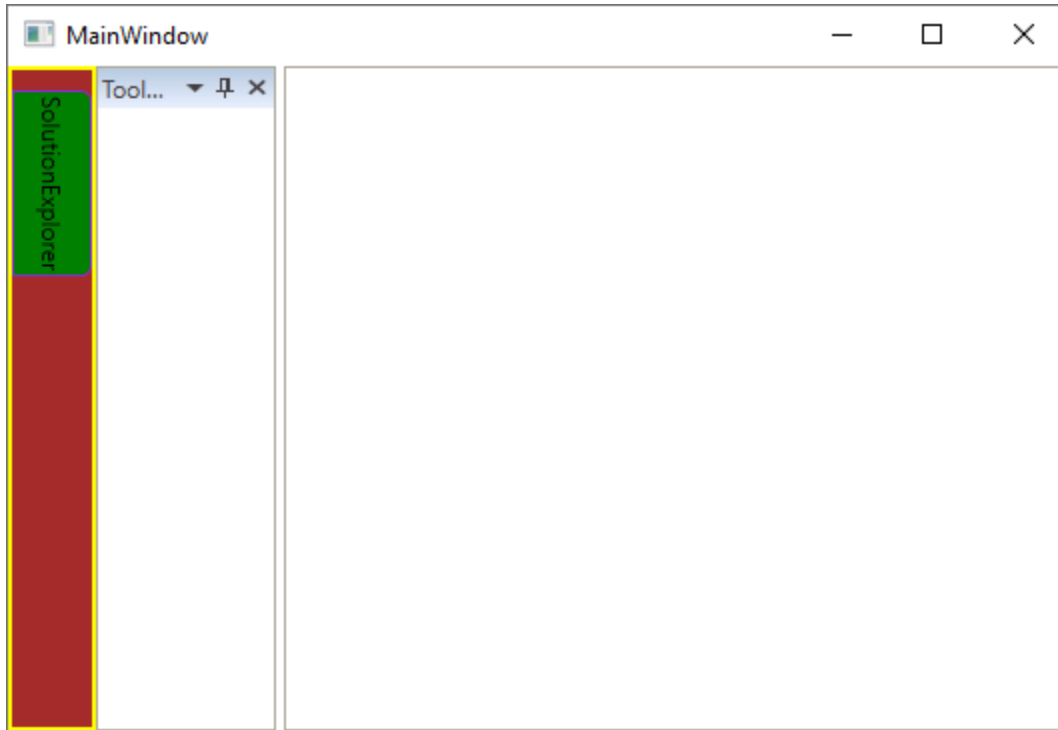
//Set Customization colors
SyncDockingManager.SidePanelBackground = new SolidColorBrush(Colors.Brown);

```

```

SyncDockingManager.SidePanelBorderBrush = new
SolidColorBrush(Colors.Yellow);
SyncDockingManager.SideItemsBackground = new SolidColorBrush(Colors.Green);
SyncDockingManager.SideItemsBorderBrush = new
SolidColorBrush(Colors.Violet);
SyncDockingManager.SidePanelBorderThickness = new Thickness(2, 2, 2, 2);
SyncDockingManager.SidePanelSize = 40;

```



#### *SideTabItem Customization*

The SideTabItem can be customized using the attached properties [SideTabItemForeground](#) and [SideTabItemBackground](#) of DockingManager.

#### **XML**

```

<syncfusion:DockingManager x:Name="DockingManager">
  <ContentControl syncfusion:DockingManager.Header="SolutionExplorer"
    syncfusion:DockingManager.State="AutoHidden"
    syncfusion:DockingManager.SideTabItemForeground="Blue"
    syncfusion:DockingManager.SideTabItemBackground="Pink"/>
</syncfusion:DockingManager>

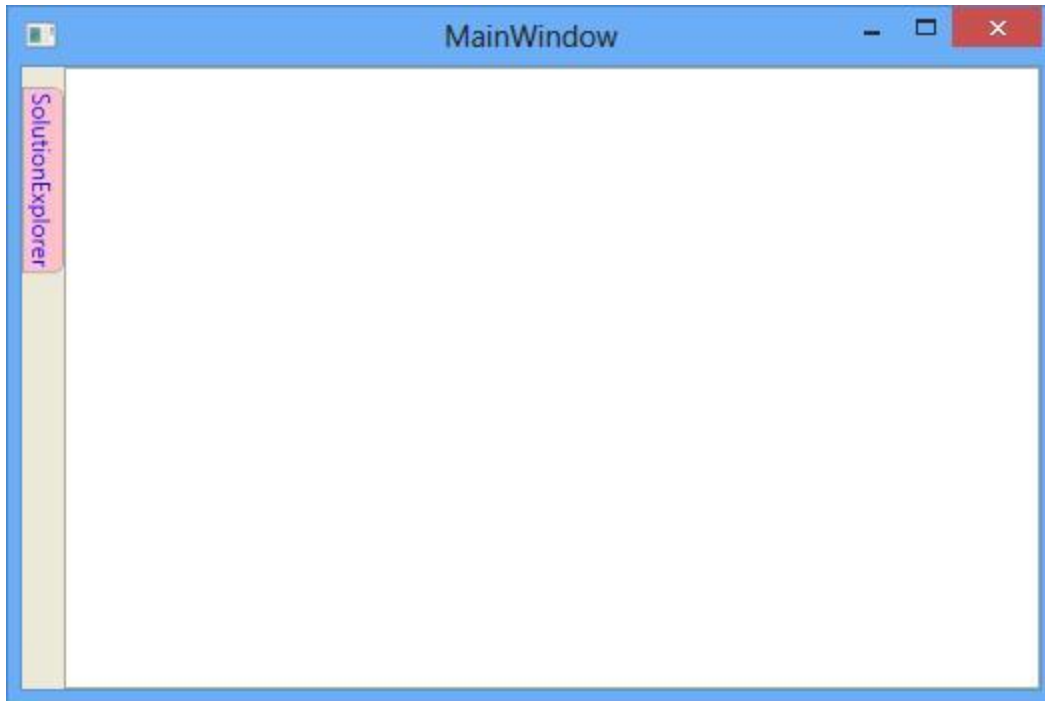
```

#### **C#**

```

//SideTabItem customization
DockingManager.SetSideTabItemBackground(content1, (new
SolidColorBrush(Colors.Pink)));
DockingManager.SetSideTabItemForeground(content1, (new
SolidColorBrush(Colors.Blue)));

```



#### Excel-like Scrollable panel

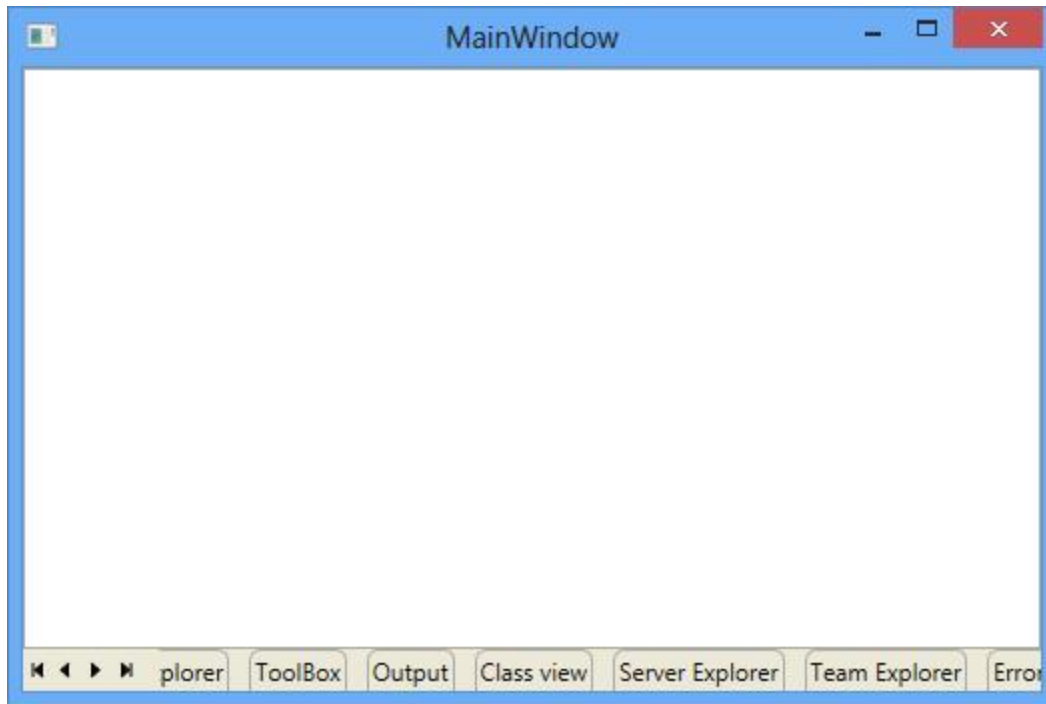
The [EnableScrollableSidePanel](#) feature is used to provide scroll support when Auto Hidden tab items overflow onto the side panel. By default [EnableScrollableSidePanel](#) value is `False`, this feature can be enabled by set its value to `True`.

#### XML

```
<syncfusion:DockingManager x:Name="dockingManager"
EnableScrollableSidePanel="True">
<ContentControl syncfusion:DockingManager.SideInDockedMode="Bottom"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.Header="Solution Explorer" />
<ContentControl syncfusion:DockingManager.SideInDockedMode="Bottom"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.Header="ToolBox" />
<ContentControl syncfusion:DockingManager.SideInDockedMode="Bottom"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.Header="Output" />
<ContentControl syncfusion:DockingManager.SideInDockedMode="Bottom"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.Header="Class view" />
<ContentControl syncfusion:DockingManager.SideInDockedMode="Bottom"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.Header="Server Explorer" />
<ContentControl syncfusion:DockingManager.SideInDockedMode="Bottom"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.Header="Team Explorer" />
<ContentControl syncfusion:DockingManager.SideInDockedMode="Bottom"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.Header="ErrorList" />
</syncfusion:DockingManager >
```

**C#**

```
//For Scroll Support  
dockingManager.EnableScrollableSidePanel = true;
```



## Changing pinning behavior

Auto Hidden Tabbed window provides two different pinning behaviors `AutoHideActive` and `AutoHideGroup` modes.

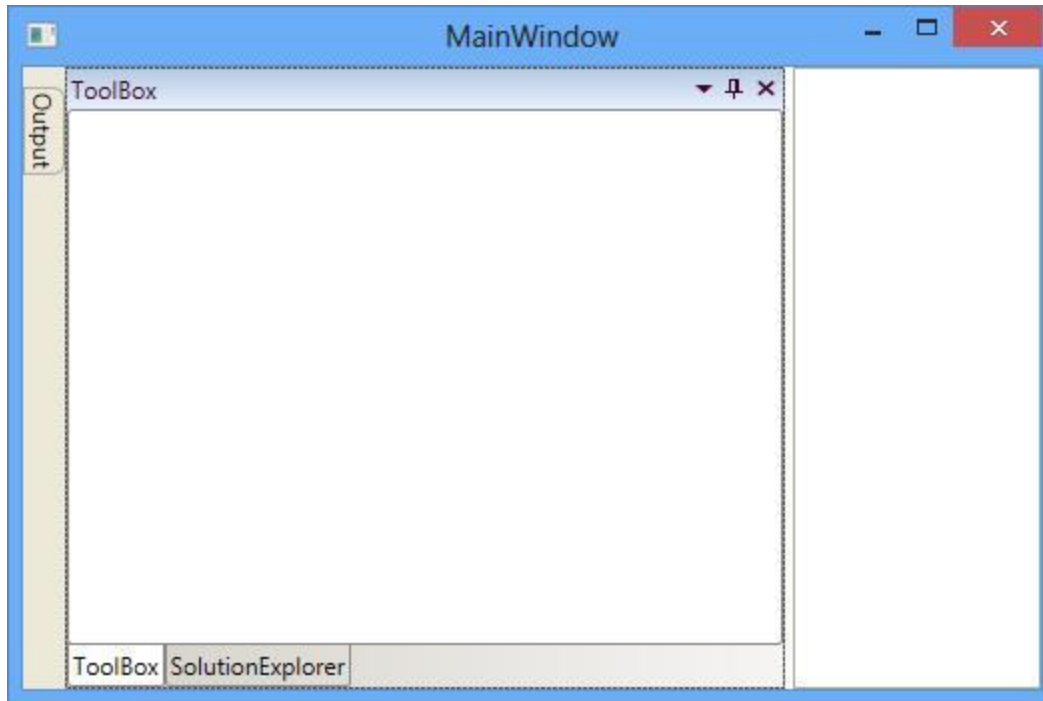
**AutoHideActive** – Used to auto-hide current active element of tabbed dock window.

**XML**

```
<syncfusion:DockingManager x:Name="dockingManager"  
AutoHideTabsMode="AutoHideActive">  
  <ContentControl syncfusion:DockingManager.Header="SolutionExplorer" />  
  <ContentControl syncfusion:DockingManager.Header="ToolBox" />  
  <ContentControl syncfusion:DockingManager.Header="Output" />  
</syncfusion:DockingManager>
```

**C#**

```
//To auto-hide current active element of tabbed dock window  
dockingManager.AutoHideTabsMode = AutoHideTabsMode.AutoHideActive;
```



**AutoHideGroup** – Used to auto-hide the entire tabbed dock window as a group.

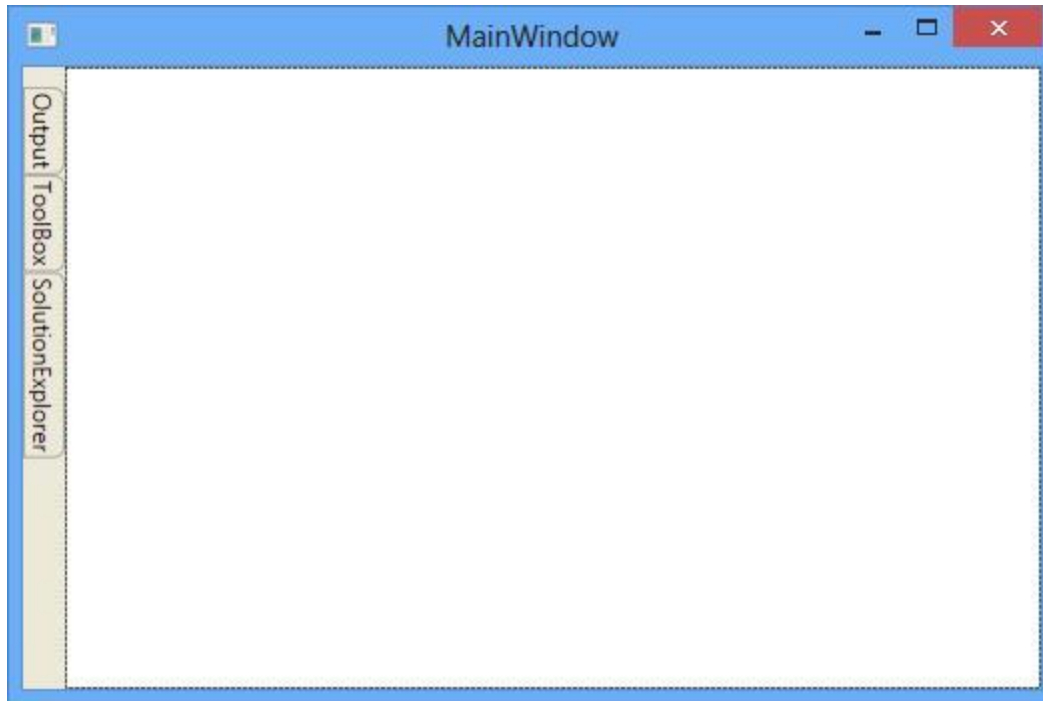
#### XML

```
<syncfusion:DockingManager x:Name="dockingManager"
AutoHideTabsMode="AutoHideGroup">
<ContentControl syncfusion:DockingManager.Header="SolutionExplorer" />
<ContentControl syncfusion:DockingManager.Header="ToolBox" />
<ContentControl syncfusion:DockingManager.Header="Output" />
</syncfusion:DockingManager>
```

#### C#

```
//To auto-hide current active element of tabbed dock window
dockingManager.AutoHideTabsMode = AutoHideTabsMode.AutoHideGroup;
```





### Configuring Auto Hide Animation

The Animation speed while auto hiding a window can be configured by setting required time delay in [AnimationDelay](#) property.

#### XML

```
<ContentControl syncfusion:DockingManager.Header="item1"
syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.AnimationDelay="100"/>
```

#### C#

```
//Set Animation delay
Docking.SetAnimationDelay(autoHideWindow1, new
Duration(TimeSpan.FromMilliseconds(100)));
```

### Making different animation for AutoHideWindow

DockingManager supports three different built-in animations while auto-hiding the windows such as Fade, scale and slide that can be set through the property [AutoHideAnimationMode](#).

**Fade** – AutoHideWindow fades while auto hiding

#### XML

```
<syncfusion:DockingManager AutoHideAnimationMode="Fade">
<ContentControl syncfusion:DockingManager.Header="SolutionExplorer"
syncfusion:DockingManager.State="AutoHidden" />
</syncfusion:DockingManager>
```

#### C#

```
//Set AnimationMode
```

```
Docking.AutoHideAnimationMode = AutoHideAnimationMode.Fade;
```

**Scale** – AutoHideWindow scale while auto hiding

#### XML

```
<syncfusion:DockingManager AutoHideAnimationMode="Scale">  
<ContentControl syncfusion:DockingManager.Header="SolutionExplorer"  
syncfusion:DockingManager.State="AutoHidden"/>  
</syncfusion:DockingManager>
```

#### C#

```
//Set AnimationMode  
Docking.AutoHideAnimationMode = AutoHideAnimationMode.Scale;
```

**Slide** – AutoHideWindow slides while auto hiding.

#### XML

```
<syncfusion:DockingManager AutoHideAnimationMode="Slide">  
<ContentControl syncfusion:DockingManager.Header="SolutionExplorer"  
syncfusion:DockingManager.State="AutoHidden"/>  
</syncfusion:DockingManager>
```

#### C#

```
//Set AnimationMode  
Docking.AutoHideAnimationMode = AutoHideAnimationMode.Slide;
```

### Enabling and disabling the AutoHide functionality

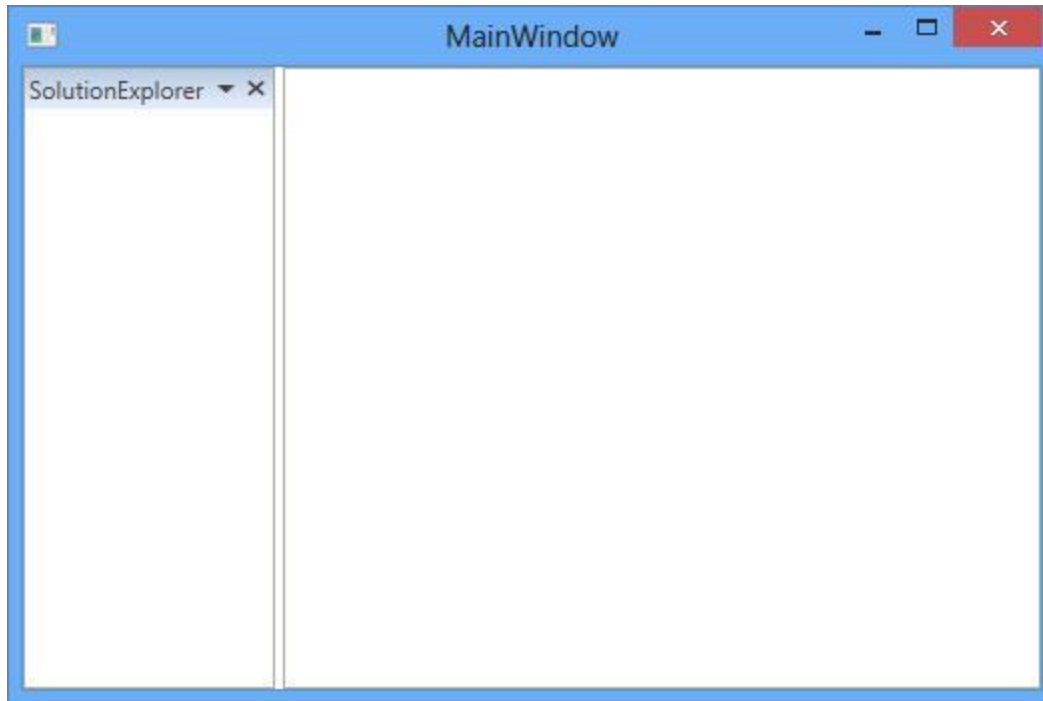
The Pin button that performs Auto Hide functionality can be visible by default. It can be invisible to disable the AutoHide functionality through [AutoHideVisibility](#) property.

#### XML

```
<syncfusion:DockingManager AutoHideVisibility="False">  
<ContentControl x:Name="AutoHideWindow"  
syncfusion:DockingManager.Header="SolutionExplorer"/>  
</syncfusion:DockingManager>
```

#### C#

```
//Set Visibility  
SyncDockingManager.AutoHideVisibility = false;
```



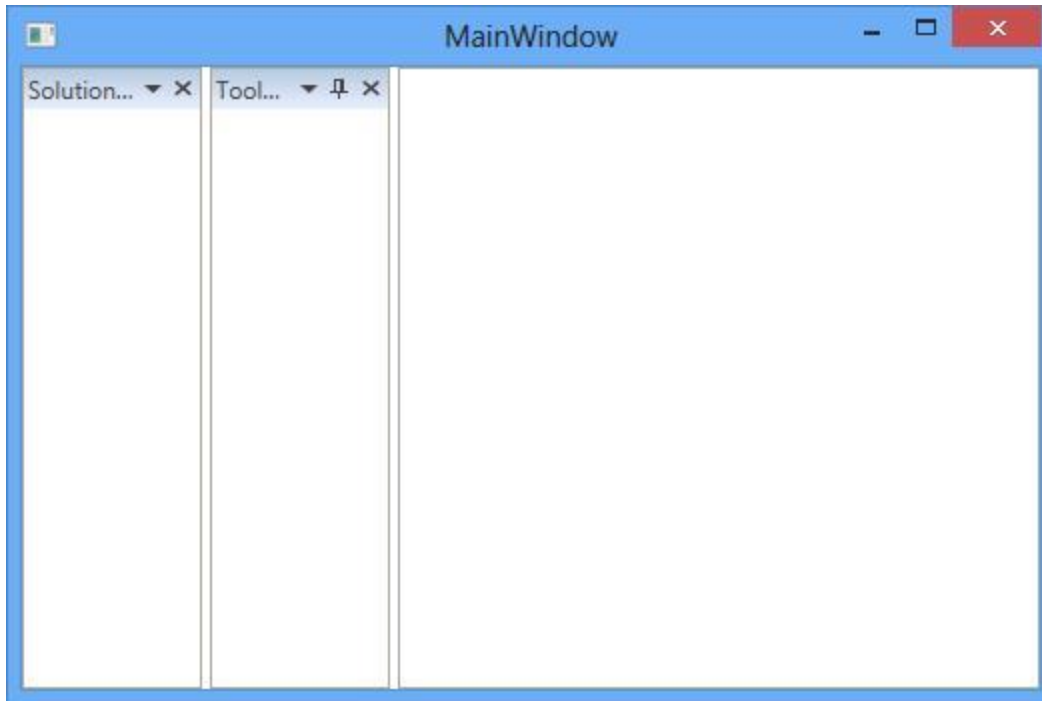
To enable or disable the AutoHide functionality for a specific child in the DockingManager, [CanAutoHide](#) can be used. By default its value is `true`, this functionality can be disabled by setting its value as `false`.

#### XML

```
<ContentControl syncfusion:DockingManager.Header="SolutionExplorer"
syncfusion:DockingManager.CanAutoHide="False" />
<ContentControl syncfusion:DockingManager.Header="ToolBox"
syncfusion:DockingManager.CanAutoHide="True" />
```

#### C#

```
//Disable the Auto hide functionality
DockingManager.SetCanAutoHide(SolutionExplorer, false);
//Enable the Auto hide functionality
DockingManager.SetCanAutoHide(ToolBox, true);
```



### Change AutoHide behavior like Visual Studio 2013

AutoHide panel open and close behavior can be changed as Visual Studio 2013. SidePanel can be opened by clicking on SideTabItem and already opened side panel can be closed by again click on the same item. This behavior of DockingManager can be enabled by setting its [IsVs2013SidePanelEnable](#) property to **True** and [IsAnimationEnabledOnMouseOver](#) property to **False**.

### XML

```
<syncfusion:DockingManager x:Name="Docking1"
IsAnimationEnabledOnMouseOver="False"
IsVs2013SidePanelEnable="True" >
<ContentControl syncfusion:DockingManager.Header="Content1"
syncfusion:DockingManager.State="AutoHidden" />
<ContentControl syncfusion:DockingManager.Header="Content2"
syncfusion:DockingManager.State="AutoHidden" />
</syncfusion:DockingManager>
```

### C#

```
DockingManager Docking1 = new DockingManager();
Docking1.IsAnimationEnabledOnMouseOver = false;
Docking1.IsVs2013SidePanelEnable = true;
ContentControl _content1 = new ContentControl();
DockingManager.SetHeader(_content1, "Content1");
DockingManager.SetState(_content1, DockState.AutoHidden);
ContentControl _content2 = new ContentControl();
DockingManager.SetHeader(_content2, "Content2");
DockingManager.SetState(_content2, DockState.AutoHidden);
Docking1.Children.Add(_content1);
Docking1.Children.Add(_content2);
Grid1.Children.Add(Docking1);
```

**VB.NET**

```
Dim Docking1 As DockingManager = New DockingManager()  
Docking1.IsAnimationEnabledOnMouseOver = False  
Docking1.IsVS2013SidePanelEnable = True  
Dim _content1 As ContentControl = New ContentControl()  
DockingManager.SetHeader(_content1, "Content1")  
DockingManager.SetState(_content1, DockState.AutoHidden)  
Dim _content2 As ContentControl = New ContentControl()  
DockingManager.SetHeader(_content2, "Content2")  
DockingManager.SetState(_content2, DockState.AutoHidden)  
Docking1.Children.Add(_content1)  
Docking1.Children.Add(_content2)  
Grid1.Children.Add(Docking1)
```

## AutoHide Animation enabled on Mouse Click

On mouse over the AutoHidden tab, the auto hide animation starts. To disable this functionality, set the property [IsAnimationEnabledOnMouseOver](#) as `False` that changes the auto hide tab start animation behavior. By default, its values is `True`.

**XML**

```
<syncfusion:DockingManager IsAnimationEnabledOnMouseOver="False">  
  <ContentControl x:Name="AutoHideWindow1"  
    syncfusion:DockingManager.Header="Solution Explorer"  
    syncfusion:DockingManager.State="AutoHidden" />  
</syncfusion:DockingManager>
```

**C#**

```
SyncDockingManager.IsAnimationEnabledOnMouseOver = false;
```

## Allow or restrict dragging the AutoHide Window

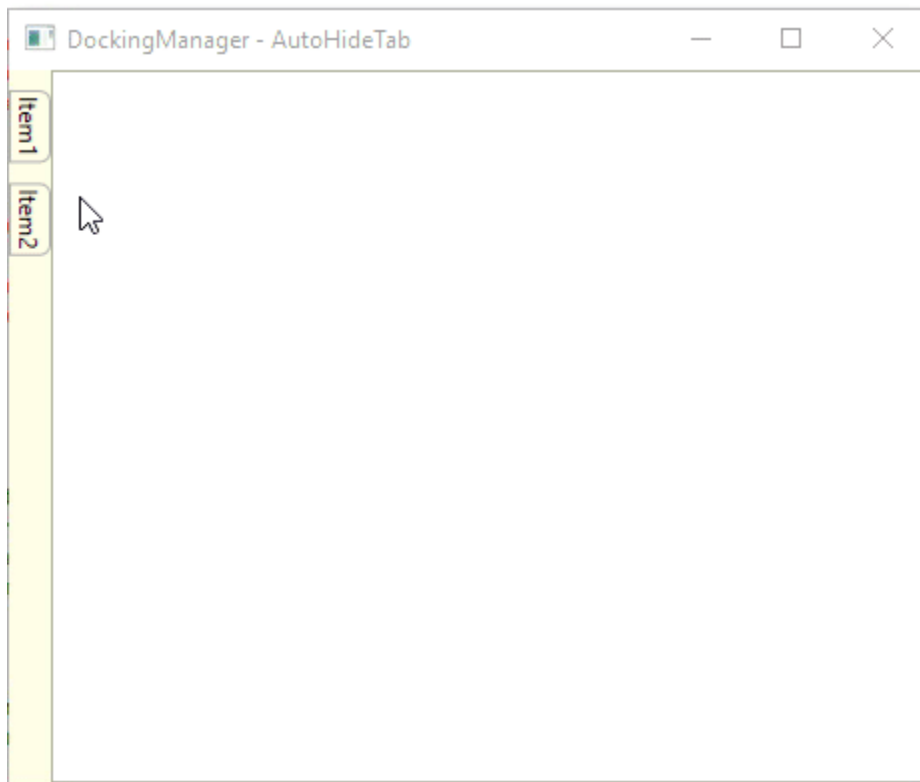
You can allow dragging of the auto hide window in the `DockingManager` by using the [CanDragAutoHidden](#) property value as `true`. You can restrict it by setting the `CanDragAutoHidden` property value as `false`. The default value of `CanDragAutoHidden` property is `false`.

**XML**

```
<syncfusion:DockingManager Name="dockingManager">  
  <!--Disable dragging-->  
  <ContentControl Content="First Window"  
    syncfusion:DockingManager.Header="Item1"  
    syncfusion:DockingManager.State="AutoHidden"  
    syncfusion:DockingManager.CanDragAutoHidden="False"  
    Name="AutoHideWindow1" />  
  <!--Enable dragging-->  
  <ContentControl Content="Second Window"  
    syncfusion:DockingManager.Header="Item2"  
    syncfusion:DockingManager.State="AutoHidden"  
    syncfusion:DockingManager.CanDragAutoHidden="True"  
    Name="AutoHideWindow2" />  
</syncfusion:DockingManager>
```

**C#**

```
// Disable dragging
ContentControl AutoHideWindow1 = new ContentControl() { Content = "First Window" };
DockingManager.SetHeader(AutoHideWindow1, "Item1");
DockingManager.SetState(AutoHideWindow1, DockState.AutoHidden);
DockingManager.SetCanDragAutoHidden(AutoHideWindow1, false);
// Enable dragging
ContentControl AutoHideWindow2 = new ContentControl() { Content = "Second Window" };
DockingManager.SetHeader(AutoHideWindow2, "Item2");
DockingManager.SetState(AutoHideWindow2, DockState.AutoHidden);
DockingManager.SetCanDragAutoHidden(AutoHideWindow2, true);
//Adding content control to the dockingmanager
DockingManager dockingManager = new DockingManager();
dockingManager.Children.Add(AutoHideWindow1);
dockingManager.Children.Add(AutoHideWindow2);
```



**Note:** View [Sample](#) in GitHub

### Pinning / Unpinning All Window

To auto hide all docked windows in the DockingManager, call [AutoHideAllDockWindow](#) method of the Docking Manager.

**C#**

```
DockingManager.AutoHideAllDockWindow();
```

**VB.NET**

```
DockingManager.AutoHideAllDockWindow();
```

To unpin all auto hide windows in the DockingManager, call [UnPinAllAutoHide](#) method of the Docking Manager.

**C#**

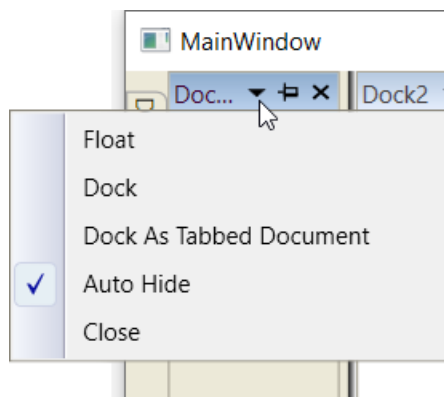
```
DockingManager.UnPinAllAutoHide();
```

**VB.NET**

```
DockingManager.UnPinAllAutoHide();
```

## Changing dock state using context menu

You can change the dock state of AutoHidden window by opening the context menu and selecting the required state in context menu items. The items in context menu is enabled and disabled based on [CanDock](#), [CanFloat](#) and [CanDocument](#) properties value.



## Floating Window in WPF Docking (DockingManager)

Floating window is one of the state in the DockingManager. To make children of the DockingManager as Float, set its [State](#) values as **Float**.

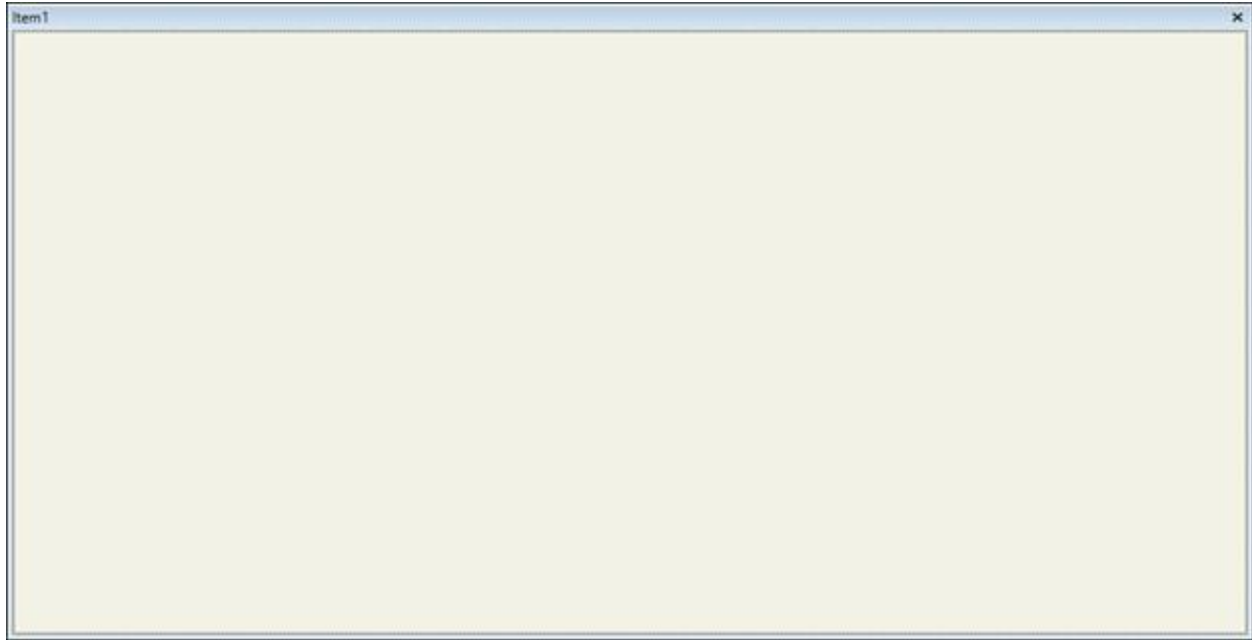
Floating window is like a Popup and it has some limitation in resizing. To avoid this limitation, set the [UseNativeFloatWindow](#) property of the DockingManager as **True**.

**XML**

```
<syncfusion:DockingManager x:Name="SyncDockingManager"
    UseNativeFloatWindow="True">
    <ContentControl syncfusion:DockingManager.Header="Item1"
        syncfusion:DockingManager.State="Float"/>
</syncfusion:DockingManager>
```

**C#**

```
SyncDockingManager.UseNativeFloatWindow = true;
```



### Rolling Up support

The float window is rolled up to top using the property [IsRollUpTopProperty](#). To enable this functionality set its value as `True`, by default its value is `False`.

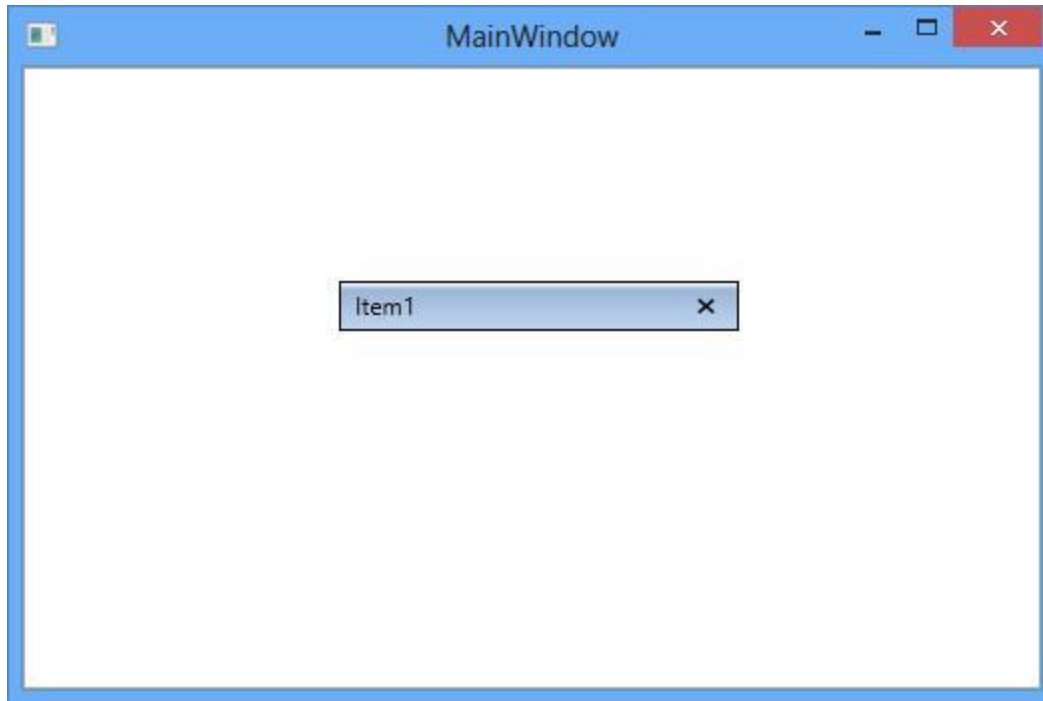
### XML

```
<syncfusion:DockingManager IsRollupFloatWindow="True">  
  <ContentControl syncfusion:DockingManager.Header="Item1 "  
    syncfusion:DockingManager.State="Float"/>  
</syncfusion:DockingManager>
```

### C#

```
SyncDockingManager.IsRollupFloatWindow = true;
```





#### Displaying Float Windows in Taskbar

Taskbar displays the icon of running applications for the purpose of switching between applications. Similarly `DockingManager` allows to display [NativeFloatWindow](#) in taskbar. It can be set using [ShowFloatWindowInTaskbar](#) property for all the [NativeFloatWindow](#) in `DockingManager` and default is false. Only [NativeFloatWindow](#) icon can be displayed in taskbar so it is necessary to set [UseNativeFloatWindow](#) to true.

#### XML

```
<syncfusion:DockingManager ShowFloatWindowInTaskbar="True"
UseNativeFloatWindow="True">
  <ContentControl syncfusion:DockingManager.Header="Toolbox"
syncfusion:DockingManager.State="Float"
Content="No items to display in toolbox"/>
  <ContentControl syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.State="Float"
Content="Loading failed"/>
</syncfusion:DockingManager>
```

#### C#

```
SyncDockingManager.UseNativeFloatWindow = true;
SyncDockingManager.ShowFloatWindowInTaskbar = true;
```

#### Show or Hide the Taskbar support for Selective Windows

To enable or disable the taskbar support for particular window, use the attached property [ShowInTaskbar](#) of `DockingManager` and the default is true. It is necessary to set [ShowFloatWindowInTaskbar](#) property of `DockingManager` to true for displaying even a single `NativeFloatWindow` in taskbar.

**XML**

```
<syncfusion:DockingManager ShowFloatWindowInTaskbar="True"
UseNativeFloatWindow="True">
<ContentControl syncfusion:DockingManager.Header="Toolbox"
syncfusion:DockingManager.State="Float"
syncfusion:DockingManager.ShowInTaskbar="False"
Content="No items to display in toolbox"/>
<ContentControl syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.State="Float"
Content="Loading failed"/>
</syncfusion:DockingManager>
```

**C#**

```
SyncDockingManager.UseNativeFloatWindow = true;
SyncDockingManager.ShowFloatWindowInTaskbar = true;
DockingManager.SetShowInTaskbar (Toolbox, false);
```

## Multiple Monitor functionalities

The default behavior of the float window remains as Popup in the DockingManager.

On using MultiMonitor scenario, the FloatWindow behavior as follows:

- FloatWindow can be resized both in primary and secondary monitor.
- On showing half of the float window between primary and secondary monitor, the float window remains in half between the monitor while plugged in and plugged out of the secondary monitor.
- When float window is moved to secondary monitor from primary monitor, the float window remains at the secondary monitor itself while plugged in.

To make float window behave like a WPF window in multiple monitor, set the property

[UseNativeFloatWindow](#) as **True**.

## Enabling or Disabling the float functionality

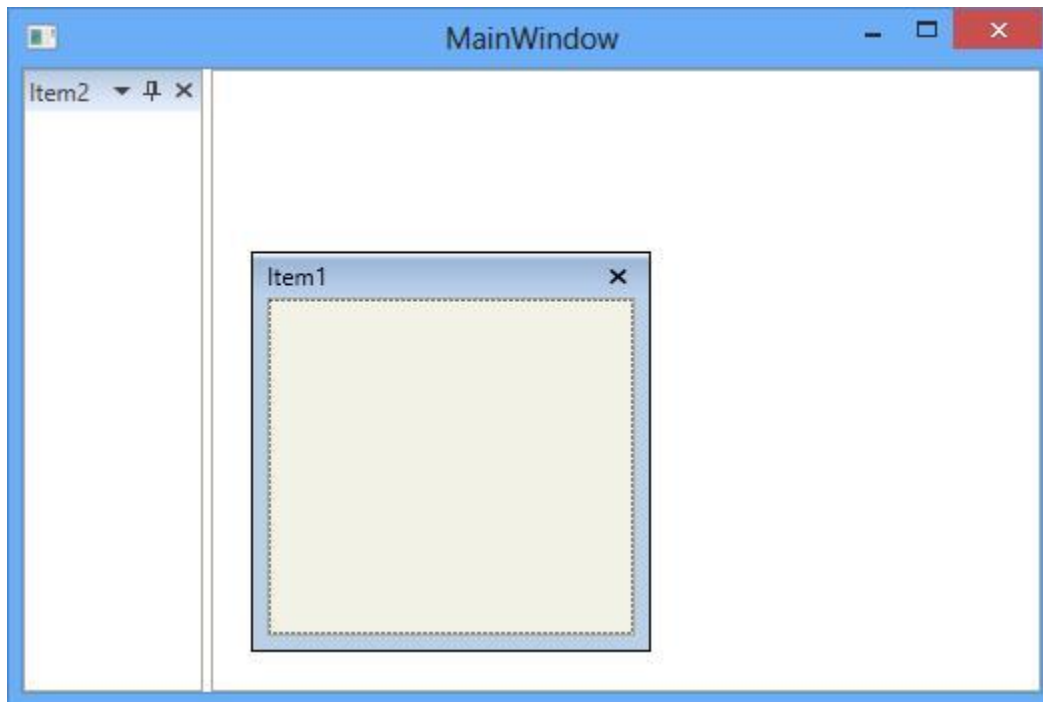
The [CanFloat](#) property helps to enable or disable the floating functionality by setting its value as **True** or **False** respectively. By default its value is **True**, to disable this functionality turn its value to **False**.

**XML**

```
<syncfusion:DockingManager x:Name="DockingManager1" >
<ContentControl syncfusion:DockingManager.Header="Item1"
syncfusion:DockingManager.CanFloat="True"/>
<ContentControl syncfusion:DockingManager.Header="Item2"
syncfusion:DockingManager.CanFloat="False"/>
</syncfusion:DockingManager>
```

**C#**

```
//To Enable the Float Functionality
DockingManager.SetCanFloat (Item1, true);
//To disable the Float Functionality
DockingManager.SetCanFloat (Item2, false);
```



### Enabling and Disabling the float functionality Operation on Double Click

The float window changes its state to **Dock** when double click its header by default. To disable this functionality for the specific child, set [NoDock](#) property as **True**.

#### XML

```
<syncfusion:DockingManager x:Name="DockingManager1">
  <ContentControl syncfusion:DockingManager.Header="Item1" x:Name="Content1"
    syncfusion:DockingManager.State="Float"
    syncfusion:DockingManager.NoDock="True"/>
</syncfusion:DockingManager>
```

#### C#

```
DockingManager.SetNoDock(Content1, true);
```

### Maximize/Minimize float window

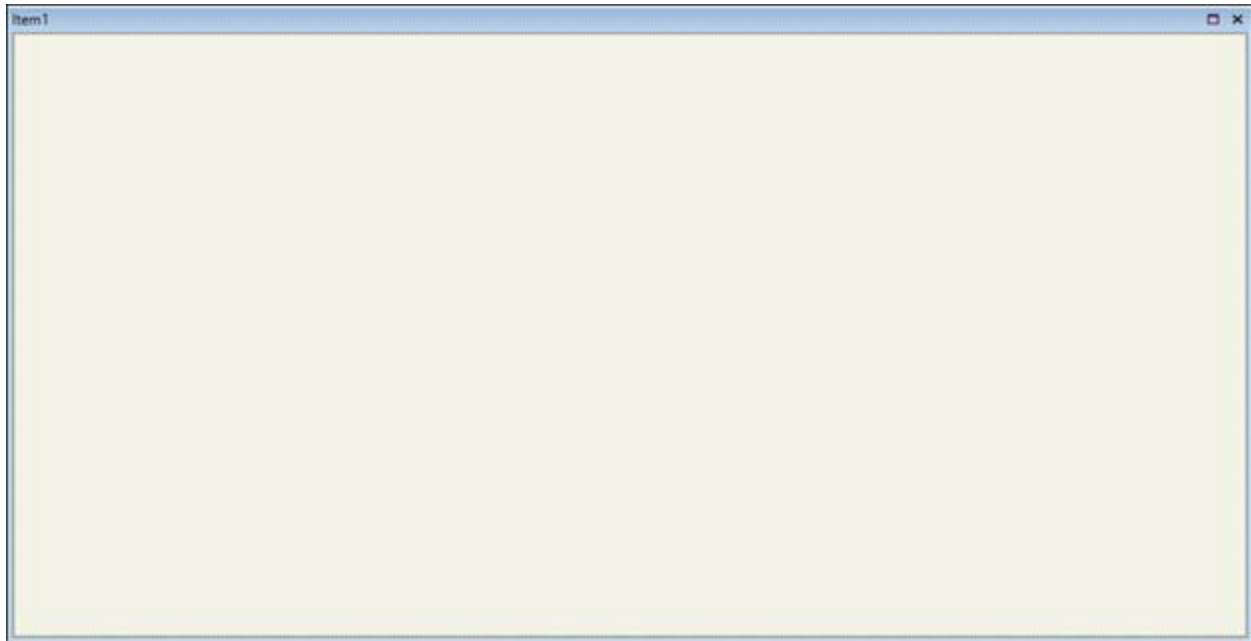
The float window provides Maximization support for better usability and it is only applicable for **NativeFloatWindow**. To enable the maximizing feature for the Float window, set [CanFloatMaximize](#) for the specific child as **True**. By default, its value is **False**.

#### XML

```
<syncfusion:DockingManager UseNativeFloatWindow="True">
  <ContentControl syncfusion:DockingManager.Header="Item1" x:Name="Content1"
    syncfusion:DockingManager.State="Float"
    syncfusion:DockingManager.CanFloatMaximize="True"/>
</syncfusion:DockingManager>
```

### C#

```
SyncDockingManager.UseNativeFloatWindow = true;  
DockingManager.SetCanFloatMaximize(Content1, true);
```



Float or Maximize on double clicking the header

The [DoubleClickAction](#) property of DockingManager decides whether the NativeFloatWindow can be maximized or moved to dock state while double clicking on header. If the property [DoubleClickAction](#) is [DockOrFloat](#), the NativeFloatWindow will moved to dock state while double clicking on header. Also, if the property [DoubleClickAction](#) is [MaximizeOrRestore](#), the NativeFloatWindow will moved to maximized or restored window state while double clicking on header only when its [CanFloatMaximize](#) attached property is true. The default value of the [DoubleClickAction](#) property is [DockOrFloat](#).

### XML

```
<syncfusion:DockingManager x:Name="dockingManager"  
UseNativeFloatWindow="True" DoubleClickAction="DockOrFloat">  
  <ContentControl x:Name="Dock1" syncfusion:DockingManager.Header="Dock1"  
    syncfusion:DockingManager.State="Float"/>  
  <ContentControl x:Name="Dock2" syncfusion:DockingManager.Header="Dock2"/>  
  <ContentControl x:Name="Dock3" syncfusion:DockingManager.Header="Dock3"/>  
  <ContentControl x:Name="Dock4" syncfusion:DockingManager.Header="Dock4"/>  
</syncfusion:DockingManager>
```

### C#

```
dockingManager.DoubleClickAction = DoubleClickAction.DockOrFloat;
```

**Note:** The [DoubleClickAction](#) property works only when [UseNativeFloatWindow](#) property is true in DockingManager.

### Positioning on desire location

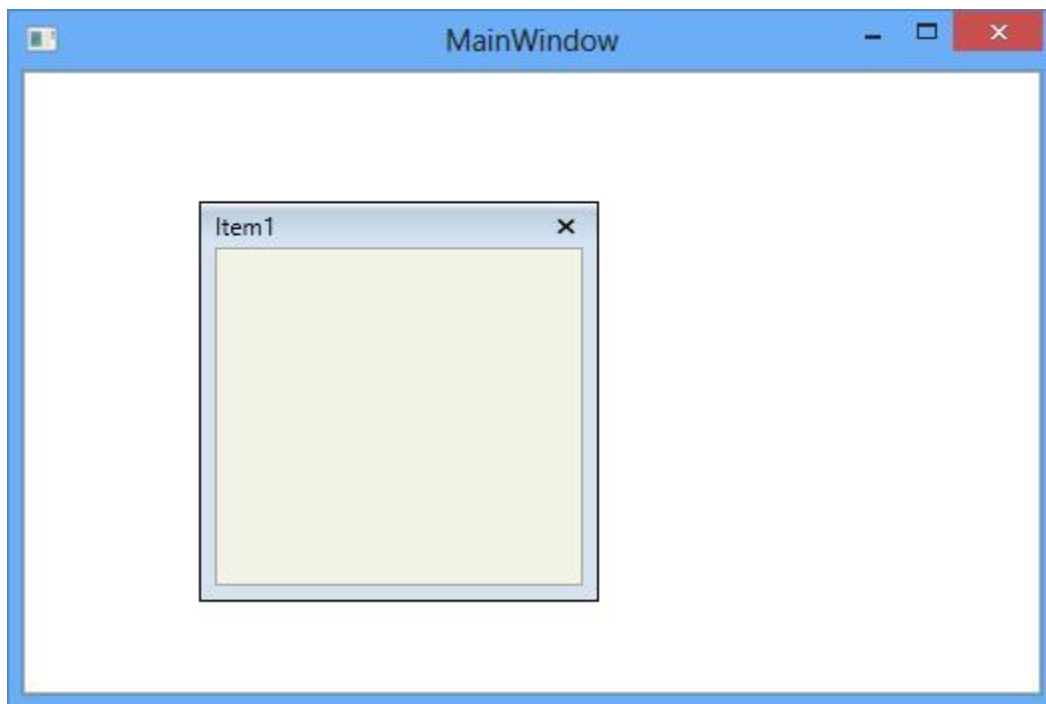
The FloatWindow can be placed at any desired location. To position the FloatWindow at the desired location with the required Rect Bounds, call [SetFloatingWindowRect](#) method of the DockingManager.

#### C#

```
DockingManager.SetFloatingWindowRect(Content1, new Rect(200, 200, 200, 200));
```

#### VB.NET

```
DockingManager.SetFloatingWindowRect(Content1, New Rect(200, 200, 200, 200))
```



### Snapping Float window

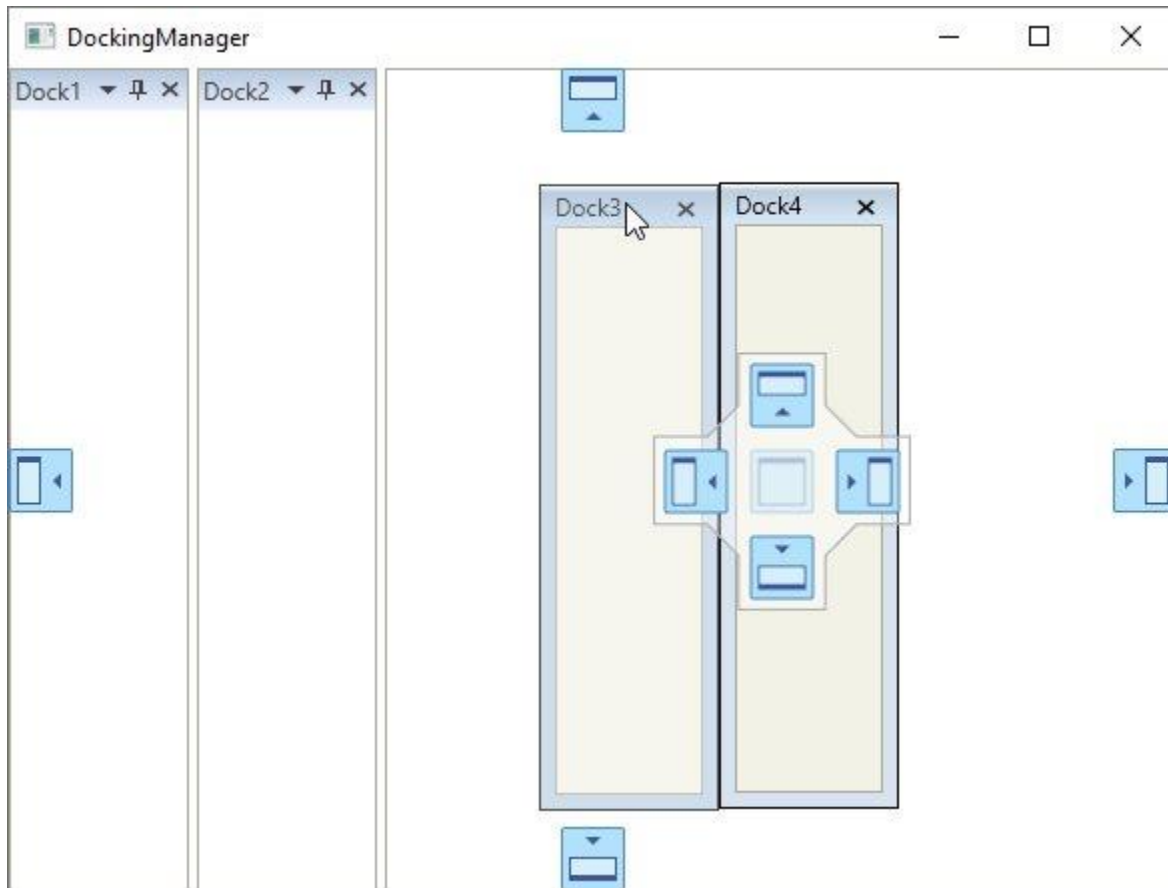
Float window can be snapped with the edge of another float window and moving all together in **DockingManager**. To enable snapping window feature for the Float window, set [AllowSnap](#) for the specific child as **True** and set [EnableSnappingFloatWindow](#) as **True** in **DockingManager**. By default, its value is **False**

#### XML

```
<syncfusion:DockingManager EnableSnappingFloatWindow="True">
  <ContentControl x:Name="Dock1" syncfusion:DockingManager.Header="Dock1"
    syncfusion:DockingManager.AllowSnap="True"/>
  <ContentControl x:Name="Dock2" syncfusion:DockingManager.Header="Dock2"
    syncfusion:DockingManager.AllowSnap="True"/>
  <ContentControl x:Name="Dock3" syncfusion:DockingManager.Header="Dock3"
    syncfusion:DockingManager.AllowSnap="True"/>
  <ContentControl x:Name="Dock4" syncfusion:DockingManager.Header="Dock4"
    syncfusion:DockingManager.AllowSnap="True"/>
</syncfusion:DockingManager>
```

**C#**

```
SyncDockingManager.EnableSnappingFloatWindow = true;  
DockingManager.SetAllowSnap(Dock1, true);  
DockingManager.SetAllowSnap(Dock2, true);  
DockingManager.SetAllowSnap(Dock3, true);  
DockingManager.SetAllowSnap(Dock4, true);
```



We can get the snapped windows collection for specific float child using [GetSnappedWindows](#) method.

**C#**

```
DockingManager.GetSnappedWindows(Dock1);
```

### Custom context menu items for floating window

You can add the custom context menu items for floating window by using the [FloatWindowContextMenuItems](#) property. You can also add sub menu items for custom context menu item by adding that sub [CustomMenuItem](#) to the parent [CustomMenuItem](#). You can check or uncheck the [CustomMenuItem](#) interactively or programmatically by using the [CustomMenuItem.IsChecked](#) property.

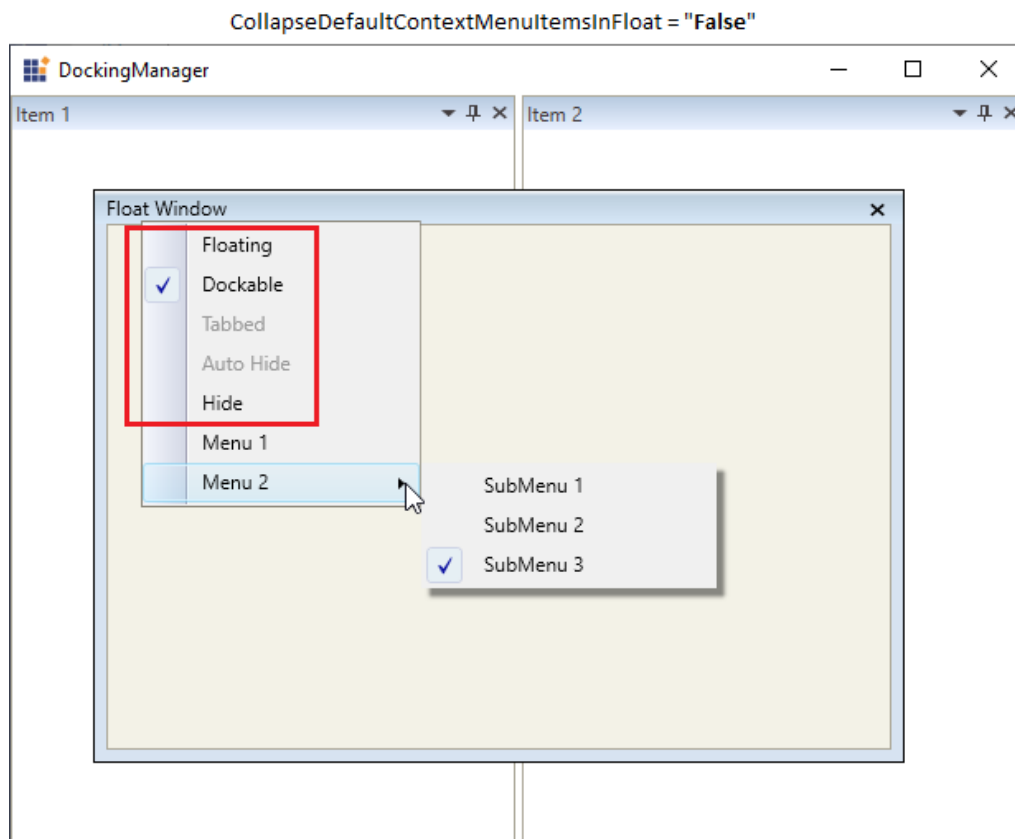
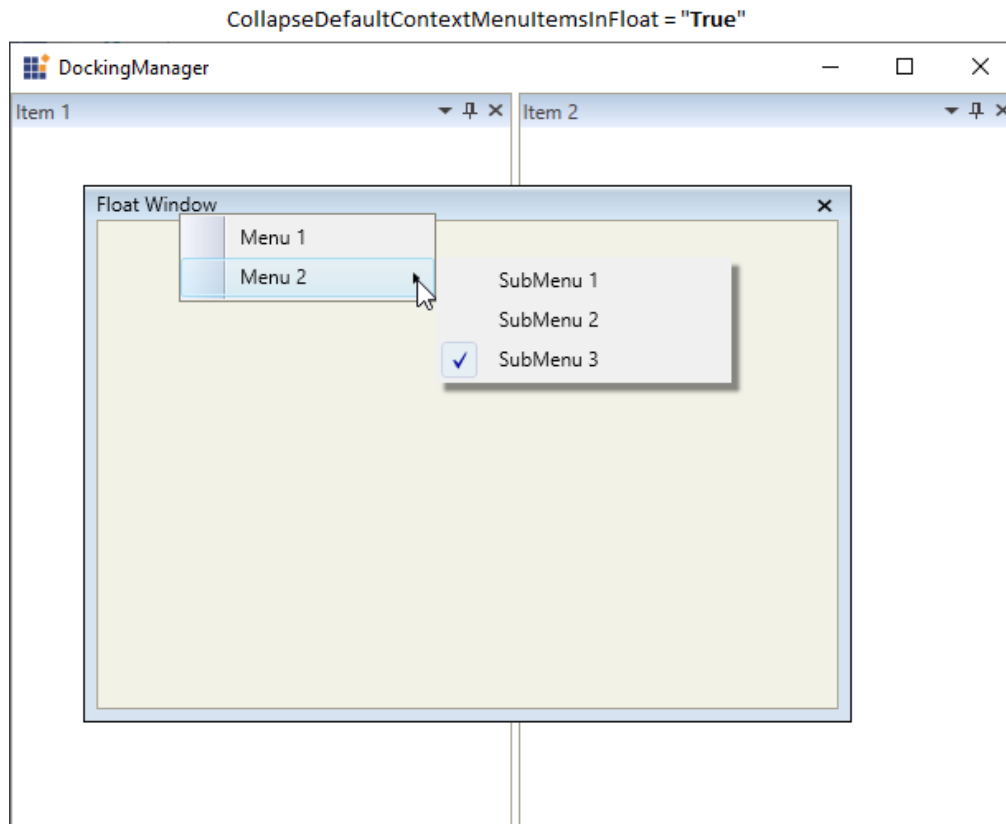
You can collapse the default floating context menu and show only the custom floating context menu items by setting the [CollapseDefaultContextMenuItemsInFloat](#) property to [true](#). The default value of [CollapseDefaultContextMenuItemsInFloat](#) property is [false](#).

## XML

```
<syncfusion:DockingManager
    DockFill="True"
    Name="dockingManager" >
    <!--Adding custom context menu items for float windows-->
    <syncfusion:DockingManager.FloatWindowContextMenuItems>
    <syncfusion:CustomMenuItemCollection>
    <!--Adding custom context menu items-->
    <syncfusion:CustomMenuItem Header="Menu 1"/>
    <syncfusion:CustomMenuItem Header="Menu 2">
    <!--Adding sub custom context menu items-->
    <syncfusion:CustomMenuItem Header="SubMenu 1"/>
    <syncfusion:CustomMenuItem Header="SubMenu 2"/>
    <syncfusion:CustomMenuItem Header="SubMenu 3" IsChecked="True"/>
    </syncfusion:CustomMenuItem>
    </syncfusion:CustomMenuItemCollection>
    </syncfusion:DockingManager.FloatWindowContextMenuItems>
    <ContentControl syncfusion:DockingManager.Header="Item 1"
    syncfusion:DockingManager.State="Dock"/>
    <ContentControl syncfusion:DockingManager.Header="Item 2"
    syncfusion:DockingManager.State="Dock"/>
    <ContentControl syncfusion:DockingManager.Header="Float Window"
    syncfusion:DockingManager.CollapseDefaultContextMenuItemsInFloat="True"
    syncfusion:DockingManager.State="Float"/>
    </syncfusion:DockingManager>
```

## C#

```
//Creating custom context menu items
CustomMenuItem menu1 = new CustomMenuItem();
menu1.Header = "Menu 1";
CustomMenuItem menu2 = new CustomMenuItem();
menu2.Header = "Menu 2";
//Creating custom sub context menu items
CustomMenuItem customMenuItem1 = new CustomMenuItem() { Header = "SubMenu 1"
};
CustomMenuItem customMenuItem2 = new CustomMenuItem() { Header = "SubMenu 2"
};
CustomMenuItem customMenuItem3 = new CustomMenuItem() { Header = "SubMenu
3", IsChecked = true };
menu2.Items.Add(customMenuItem1);
menu2.Items.Add(customMenuItem2);
menu2.Items.Add(customMenuItem3);
//Adding custom context menu items with sub menu items
dockingManager.FloatWindowContextMenuItems.Add(menu1);
dockingManager.FloatWindowContextMenuItems.Add(menu2);
```





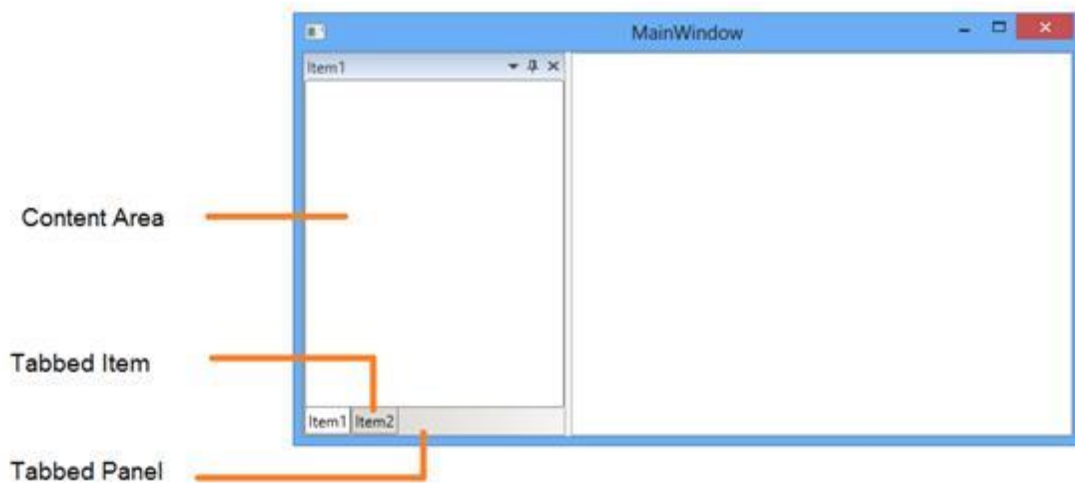
---

**Note:** [View Sample in GitHub](#)

---

### Tabbed Window in WPF Docking (DockingManager)

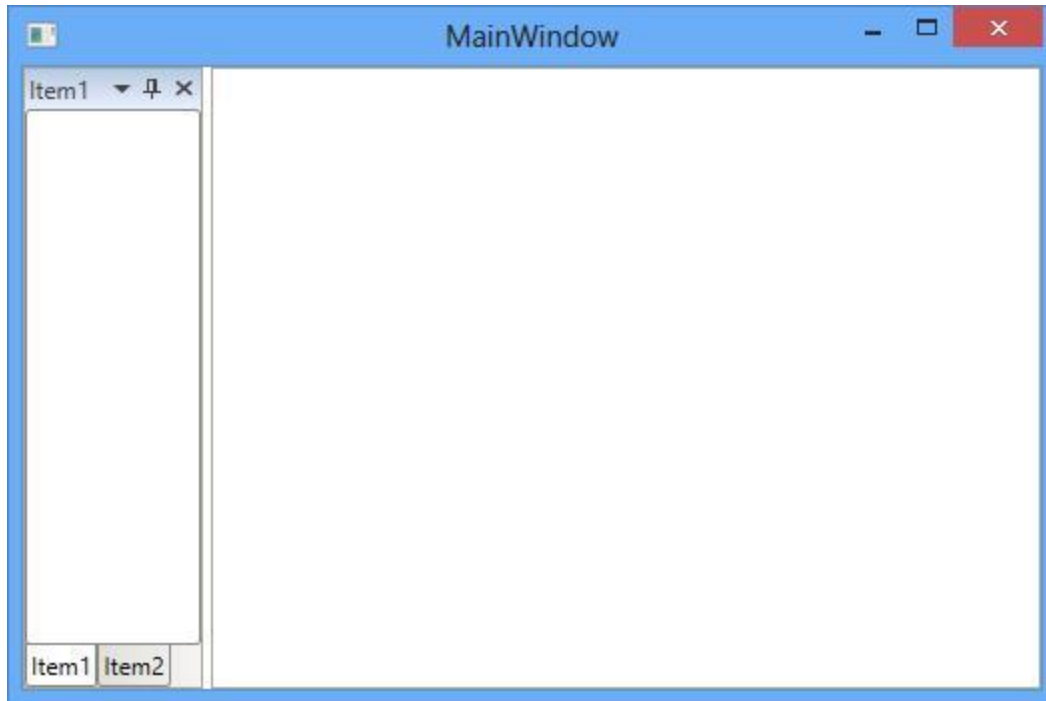
Child window can be arranged as Tabbed windows by setting TargetName and side value as **Tabbed** using the property [SideInDockedMode](#).



### Tab alignments

The tabs of the Docked window are placed at the bottom, by default. To place the tabs of the docked window at different sides set the property [DockTabAlignment](#) with desired values such as Top, Bottom, Left and Right.

- DockTabAlignment as **Bottom**



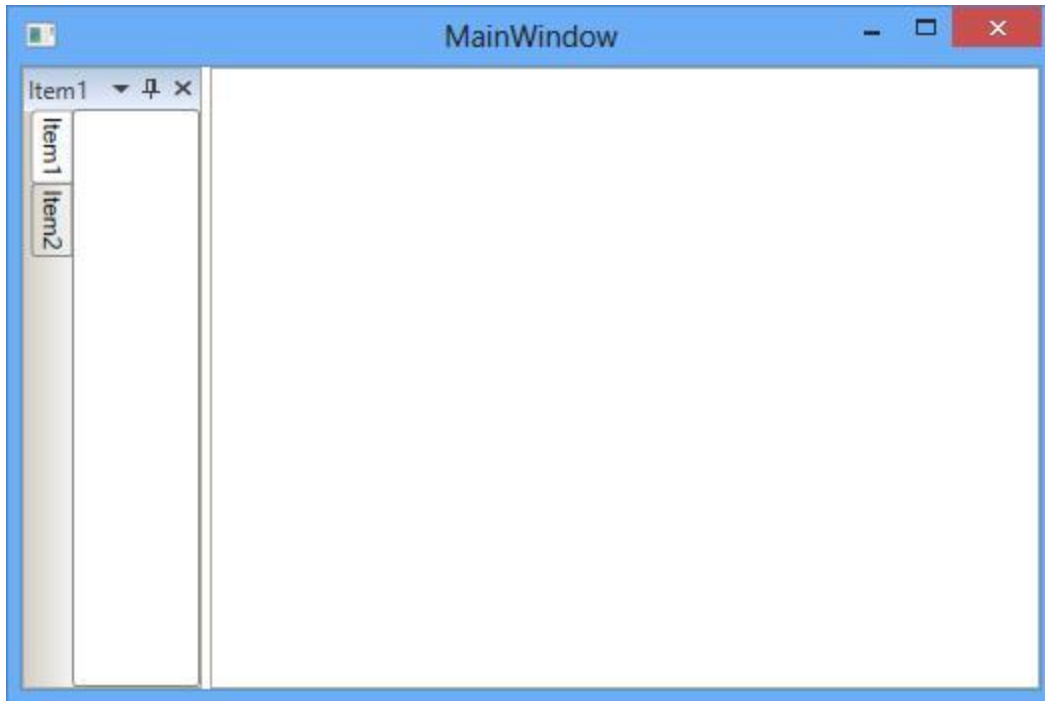
- Setting DockTabAlignment as **Left**.

#### XML

```
<syncfusion:DockingManager x:Name="DockingManager1" DockTabAlignment="Left">
  <ContentControl syncfusion:DockingManager.Header="Item1"
    x:Name="TabbedWindow1" />
  <ContentControl syncfusion:DockingManager.Header="Item2"
    x:Name="TabbedWindow2"
    syncfusion:DockingManager.SideInDockedMode="Tabbed"
    syncfusion:DockingManager.TargetNameInDockedMode="TabbedWindow1"/>
</syncfusion:DockingManager>
```

#### C#

```
DockingManager1.DockTabAlignment = Dock.Left;
```



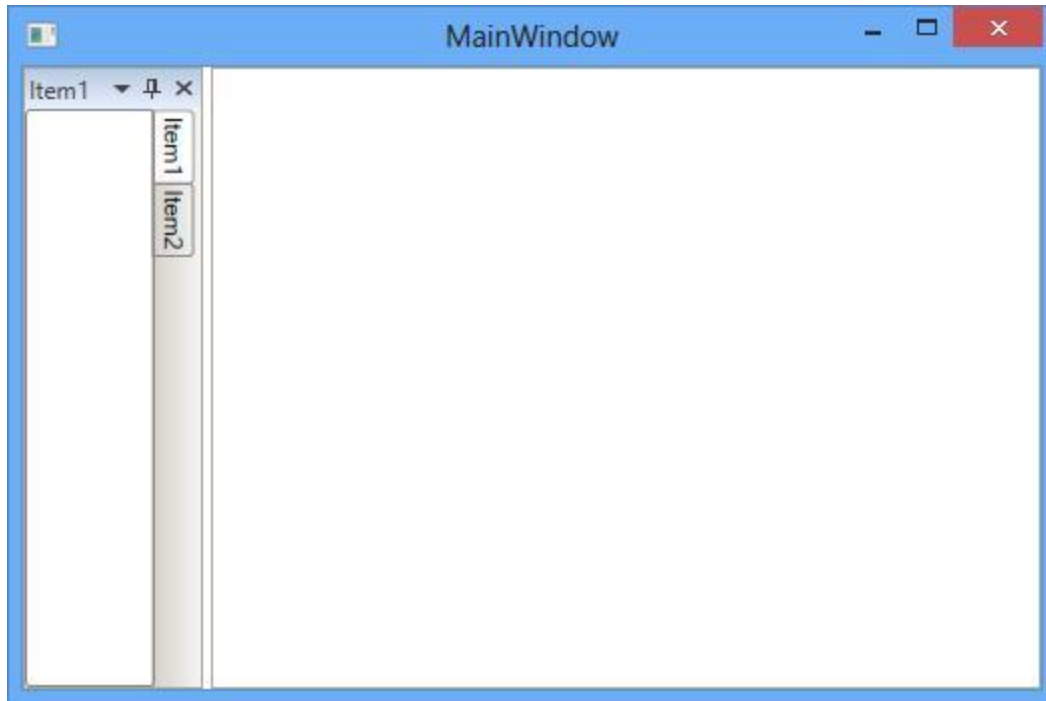
- Setting DockTabAlignment as Right

#### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
DockTabAlignment="Right">
  <ContentControl syncfusion:DockingManager.Header="Item1"
x:Name="TabbedWindow1"/>
  <ContentControl syncfusion:DockingManager.Header="Item2"
x:Name="TabbedWindow2"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="TabbedWindow1"/>
</syncfusion:DockingManager>
```

#### C#

```
DockingManager1.DockTabAlignment = Dock.Right;
```



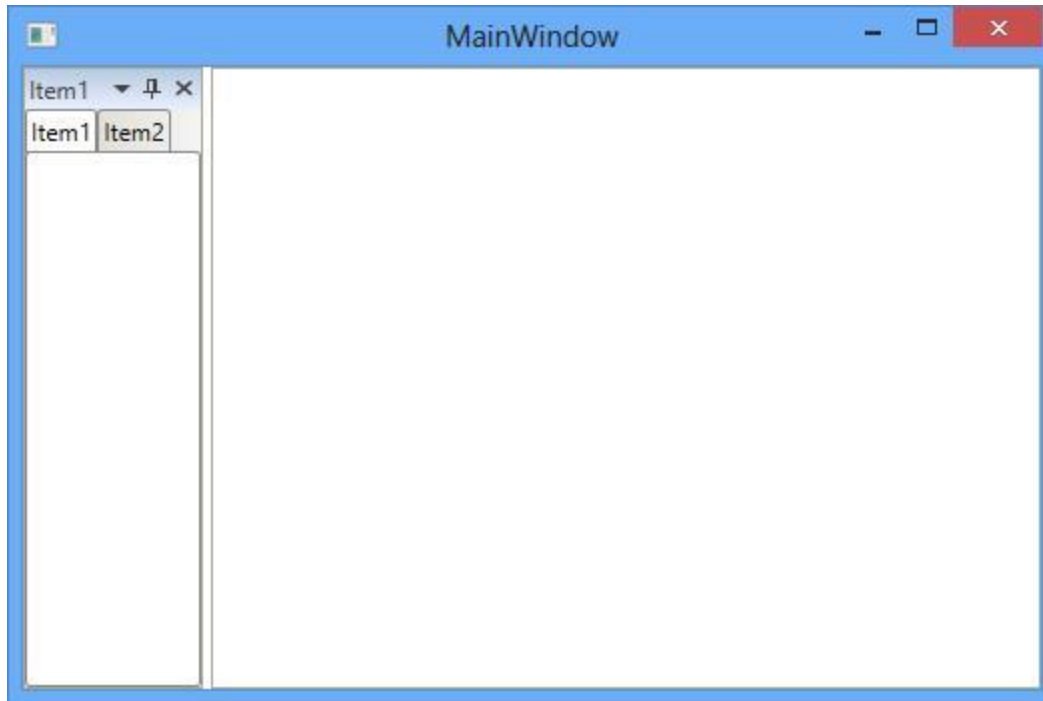
- Setting DockTabAlignment as **Top**

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1" DockTabAlignment="Top">
  <ContentControl syncfusion:DockingManager.Header="Item1"
  x:Name="TabbedWindow1"/>
  <ContentControl syncfusion:DockingManager.Header="Item2"
  x:Name="TabbedWindow2"
  syncfusion:DockingManager.SideInDockedMode="Tabbed"
  syncfusion:DockingManager.TargetNameInDockedMode="TabbedWindow1"/>
</syncfusion:DockingManager>
```

### C#

```
SyncDockingManager.DockTabAlignment = Dock.Top;
```



Closing a Tabbed window

Tabbed window provides two different closing behaviors. They are CloseActive and CloseAll modes of [CloseTabs](#) property.

**CloseActive** – Used to close the active element of Tabbed window.

#### XML

```
<syncfusion:DockingManager x:Name="DockingManager1" CloseTabs="CloseActive">
  <ContentControl syncfusion:DockingManager.Header="Item1"
  x:Name="TabbedWindow1" />
  <ContentControl syncfusion:DockingManager.Header="Item2"
  x:Name="TabbedWindow2"
  syncfusion:DockingManager.SideInDockedMode="Tabbed"
  syncfusion:DockingManager.TargetNameInDockedMode="TabbedWindow1"/>
  <ContentControl syncfusion:DockingManager.Header="Item3"
  x:Name="TabbedWindow3"
  syncfusion:DockingManager.SideInDockedMode="Tabbed"
  syncfusion:DockingManager.TargetNameInDockedMode="TabbedWindow1"/>
</syncfusion:DockingManager>
```

#### C#

```
//To close the active element of Tabbed window.
SyncDockingManager.CloseTabs = CloseTabsMode.CloseActive;
```

**CloseAll** – Used to close all the Tabbed window.

#### XML

```
<syncfusion:DockingManager x:Name="DockingManager1" CloseTabs="CloseAll">
```

```
<ContentControl syncfusion:DockingManager.Header="Item1"
x:Name="TabbedWindow1"/>
<ContentControl syncfusion:DockingManager.Header="Item2"
x:Name="TabbedWindow2"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="TabbedWindow1"/>
<ContentControl syncfusion:DockingManager.Header="Item3"
x:Name="TabbedWindow3"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="TabbedWindow1"/>
</syncfusion:DockingManager>
```

**C#**

```
//To close all the Tabbed window.
SyncDockingManager.CloseTabs = CloseTabsMode.CloseAll;
```

## Tabbed window order changed notification

You will be notified when the tabbed window's order is changed by using the [TabOrderChanged](#) event. You can get the order changed tabbed window with its old and new index values by using the [TargetItem](#), [OldIndex](#) and [NewIndex](#) properties.

**Note:** The [TabOrderChanged](#) event occurs only during drag and drop operation. Not occurs when add or remove items interactively or using code behind.

**XML**

```
<syncfusion:DockingManager TabOrderChanged="DockingManager1_TabOrderChanged"
DockTabAlignment="Bottom"
x:Name="dockingManager1">
<ContentControl syncfusion:DockingManager.Header="Item1"
x:Name="tabbedWindow1" />
<ContentControl syncfusion:DockingManager.Header="Item2"
x:Name="tabbedWindow2"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="tabbedWindow1"/>
<ContentControl syncfusion:DockingManager.Header="Item3"
x:Name="tabbedWindow3"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="tabbedWindow2"/>
</syncfusion:DockingManager>
```

**C#**

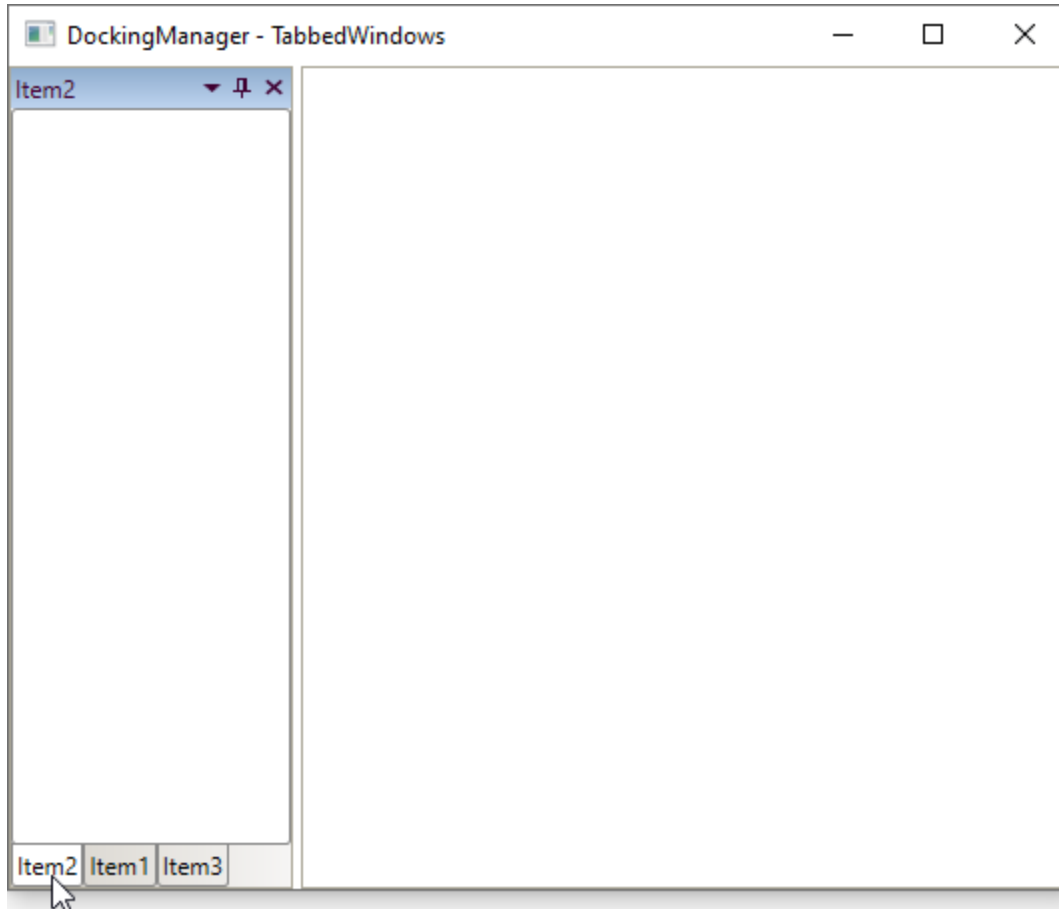
```
dockingManager1.DockTabAlignment = Dock.Bottom;
dockingManager1.TabOrderChanged += DockingManager1_TabOrderChanged;
```

You can handle the event as follows,

**C#**

```
private void DockingManager1_TabOrderChanged(object sender,
TabOrderChangedEventArgs e)
{
```

```
var dragged_Item = e.TargetItem;  
var oldIndex = e.OldIndex;  
var newIndex = e.NewIndex;  
}
```



**Note:** [View Sample in GitHub](#)

Restrict tabbed window reordering

If you want to restrict the user to reordering the tabbed window by drag and drop operation, use the [TabOrderChanging](#) event and set `Cancel` property value as `true`.

**Note:** The `TabOrderChanging` event occurs only during drag and drop operation. Not occurs when add or remove items using code behind.

#### XML

```
<syncfusion:DockingManager  
  TabOrderChanging="DockingManager1_TabOrderChanging"  
  DockTabAlignment="Bottom"  
  x:Name="DockingManager1">  
  <ContentControl syncfusion:DockingManager.Header="Item1"  
    x:Name="TabbedWindow1" />  
  <ContentControl syncfusion:DockingManager.Header="Item2"  
    x:Name="TabbedWindow2"
```

```
syncfusion:DockingManager.SideInDockedMode="Tabbed"  
syncfusion:DockingManager.TargetNameInDockedMode="TabbedWindow1"/>  
<ContentControl syncfusion:DockingManager.Header="Item3"  
x:Name="TabbedWindow3"  
syncfusion:DockingManager.SideInDockedMode="Tabbed"  
syncfusion:DockingManager.TargetNameInDockedMode="TabbedWindow2"/>  
</syncfusion:DockingManager>
```

### C#

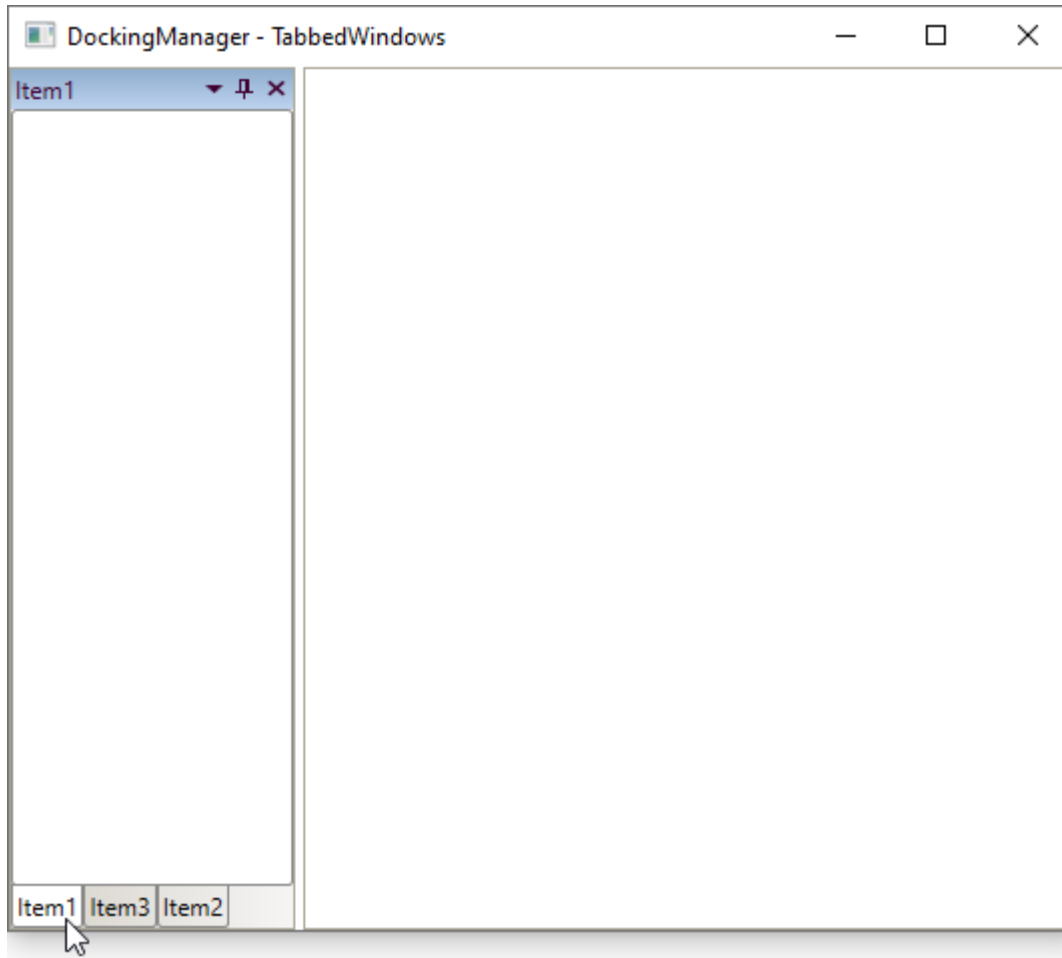
```
dockingManager1.DockTabAlignment = Dock.Bottom;  
dockingManager1.TabOrderChanging += DockingManager1.TabOrderChanging;
```

You can handle the event as follows,

### C#

```
private void DockingManager1_TabOrderChanging(object sender,  
Syncfusion.Windows.Tools.Controls.TabOrderChangedEventArgs e)  
{  
    // Restrict the Tab item order changing  
    e.Cancel = true;  
    var dragged_Item = e.TargetItem;  
    var oldIndex = e.OldIndex;  
    var newIndex = e.NewIndex;  
}
```





---

**Note:** [View Sample in GitHub](#)

---

### Data Binding in WPF Docking (DockingManager)

Data Binding is the process of establishing a connection between the application UI and business logic. Data Binding can be unidirectional (Source -> Target or Target -> Source) or bidirectional (Source <-> target).

Adding Docking Window child through ItemsSource:

**DockingManager** [ItemsSource](#) property allows to bind the Observable Collection of [DockItem](#). [DockItem](#) class which contains all attached properties of **DockingManager**.

The following code snippet explains how to use the [ItemsSource](#) property

#### XML

```
<syncfusion:DockingManager x:Name="docking" ItemsSource="{Binding  
DockCollections }" />
```

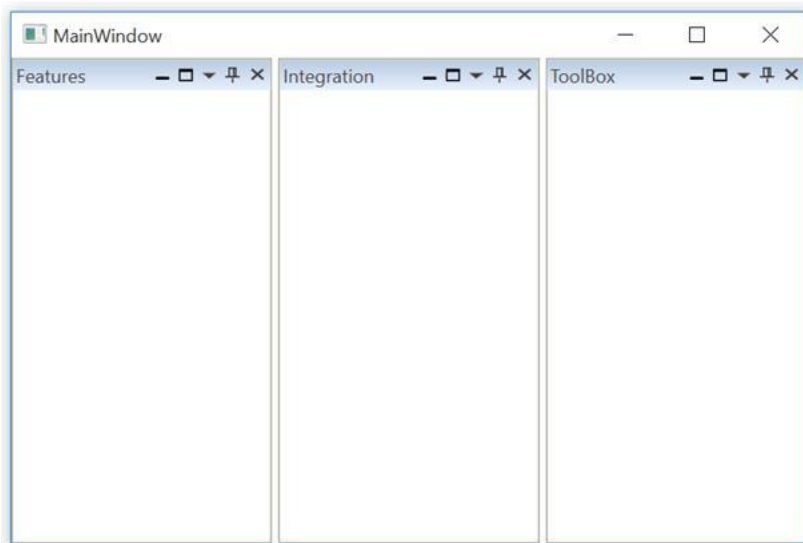
#### C#

```
DockingManager docking = new DockingManager();  
docking.ItemsSource = DockCollections;
```

Adding `DockItem` to collection:

**C#**

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        DockCollections = new ObservableCollection<DockItem>();
        DockCollections.Add(new DockItem() { Header = "ToolBox" });
        DockCollections.Add(new DockItem() { Header = "Integration" });
        DockCollections.Add(new DockItem() { Header = "Features" });
        this.DataContext = this;
    }
    private ObservableCollection<DockItem> _dockcollection;
    public ObservableCollection<DockItem> DockCollections
    {
        get
        {
            return _dockcollection;
        }
        set
        {
            _dockcollection = value;
        }
    }
}
```



Docking Window in Different side

We have docked the docking child window in five sides these are,

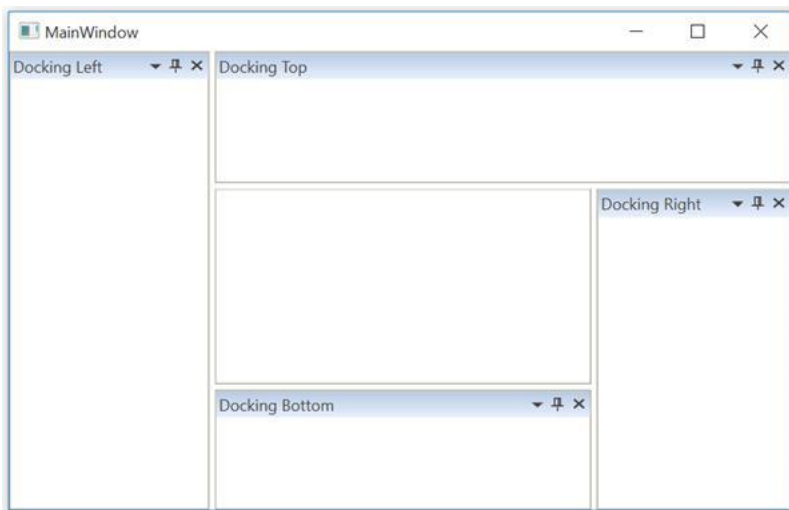
- Left
- Right
- Top
- Bottom

- Tabbed

To dock the children of `DockingManager` in different side, use `SideInDockedMode` mode property of `DockItem` class.

### C#

```
DockCollections.Add(new DockItem() { Header = "Docking Left",
SideInDockedMode = DockSide.Left });
DockCollections.Add(new DockItem() { Header = "Docking Top",
SideInDockedMode = DockSide.Top });
DockCollections.Add(new DockItem() { Header = "Docking
Right", SideInDockedMode=DockSide.Right });
DockCollections.Add(new DockItem() { Header = "Docking Bottom",
SideInDockedMode = DockSide.Bottom });
```



Configure the Docking window through ItemsSource

Docking window can also be docked at any side of the Target Docking Window through an attached property named `TargetNameInDockedMode`

Also to set as Tabbed Window, the window should aware of a `DockItem` Name of corresponding target window. The following code helps to arrange children of `DockingManager` that targets a single Docking window docked along Left, Top, Right and Tabbed.

**Note:** The following code snippet explains how to use all attached properties of `DockingManager` using `DockItem` class.

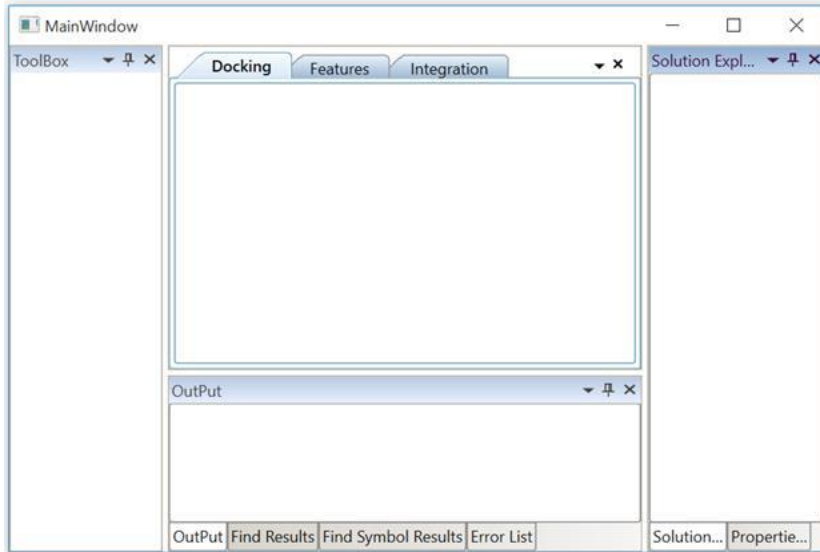
### C#

```
DockCollections = new ObservableCollection<DockItem>();
DockCollections.Add(new DockItem() { Header = "ToolBox", State =
DockState.Dock, DesiredWidthInDockedMode = 300d });
DockCollections.Add(new DockItem() { Header = "Integration", State =
DockState.Document});
DockCollections.Add(new DockItem() { Header = "Features", State =
DockState.Document });
DockCollections.Add(new DockItem() { Header = "Docking", State =
DockState.Document});
```

```

DockCollections.Add(new DockItem() { Header = "Solution Explorer", Name =
"solution", State = DockState.Dock, SideInDockedMode = DockSide.Right,
DesiredWidthInDockedMode = 300d });
DockCollections.Add(new DockItem() { Header = "Properties Window", Name =
"Properties", State = DockState.Dock, SideInDockedMode = DockSide.Tabbed,
TargetNameInDockedMode = "solution" });
DockCollections.Add(new DockItem() { Header = "OutPut", Name = "Output",
State = DockState.Dock, SideInDockedMode =
DockSide.Bottom, DesiredHeightInDockedMode=200d});
DockCollections.Add(new DockItem() { Header = "Error List", State =
DockState.Dock, SideInDockedMode = DockSide.Tabbed, TargetNameInDockedMode =
"Output" });
DockCollections.Add(new DockItem() { Header = "Find Symbol Results", State =
DockState.Dock, SideInDockedMode = DockSide.Tabbed, TargetNameInDockedMode =
"Output" });
DockCollections.Add(new DockItem() { Header = "Find Results", State =
DockState.Dock, SideInDockedMode = DockSide.Tabbed, TargetNameInDockedMode =
"Output" });

```



## Dealing with Windows in WPF Docking (DockingManager)

### Activating a window

A particular child window can be activated in DockingManager using its name or reference through the property [ActiveWindow](#) and [ActivateWindow](#) method that passes the element as argument to activate.

#### C#

```

DockingManager1.ActiveWindow = Content1;
DockingManager1.ActivateWindow("Content1");

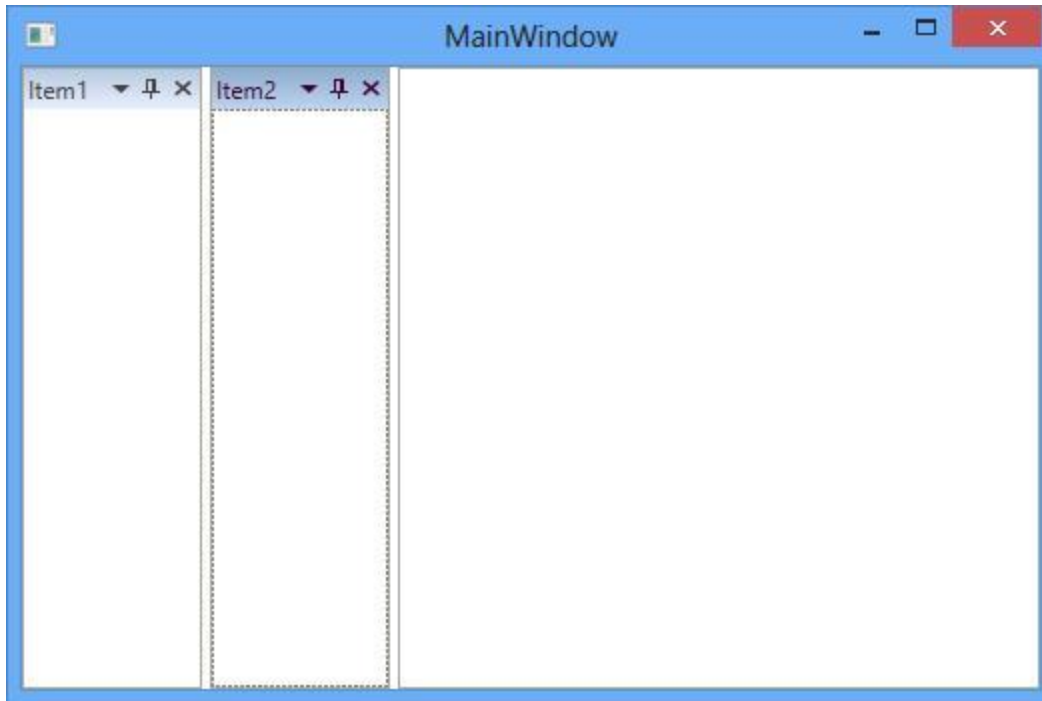
```

#### VB.NET

```

DockingManager1.ActiveWindow = Content1
DockingManager1.ActivateWindow("Content1")

```



### Adding Window Programmatically

Any UI element can be added inside the DockingManager as its child windows. The windows is added as Dock windows, since the default value of the state is **Dock**. The UI element is added in the DockingManager using the **Add** method of the Children property of the DockingManager.

For example, ContentControl is added as a window for DockingManager

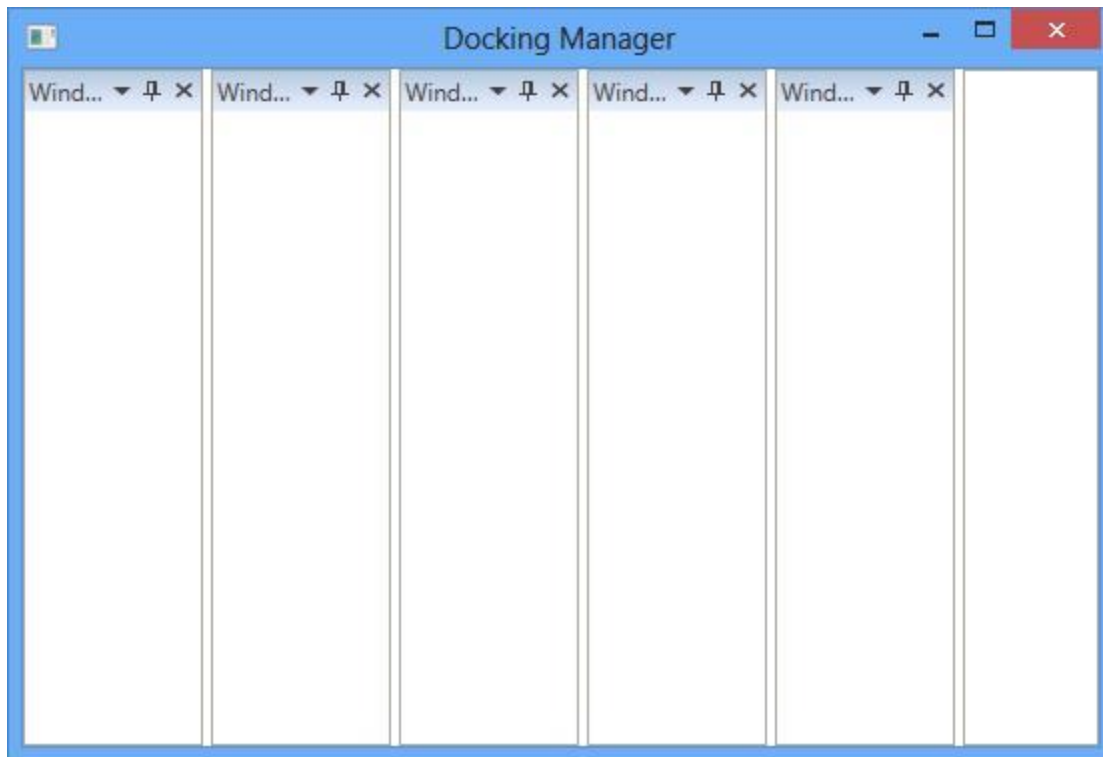
#### C#

```
DockingManager DockingManager1 = new DockingManager();
ContentControl content1 = new ContentControl();
DockingManager.SetHeader(content1, "Window1");
ContentControl content2 = new ContentControl();
DockingManager.SetHeader(content2, "Window2");
ContentControl content3 = new ContentControl();
DockingManager.SetHeader(content3, "Window3");
ContentControl content4 = new ContentControl();
DockingManager.SetHeader(content4, "Window4");
ContentControl content5 = new ContentControl();
DockingManager.SetHeader(content5, "window5");
DockingManager1.Children.Add(content1);
DockingManager1.Children.Add(content2);
DockingManager1.Children.Add(content3);
DockingManager1.Children.Add(content4);
DockingManager1.Children.Add(content5);
```

#### VB.NET

```
Dim DockingManager1 As New DockingManager()
Dim content1 As New ContentControl()
DockingManager.SetHeader(content1, "Window1")
Dim content2 As New ContentControl()
```

```
DockingManager.SetHeader(content2, "Window2")
Dim content3 As New ContentControl()
DockingManager.SetHeader(content3, "Window3")
Dim content4 As New ContentControl()
DockingManager.SetHeader(content4, "Window4")
Dim content5 As New ContentControl()
DockingManager.SetHeader(content5, "Window5")
DockingManager1.Children.Add(content1)
DockingManager1.Children.Add(content2)
DockingManager1.Children.Add(content3)
DockingManager1.Children.Add(content4)
DockingManager1.Children.Add(content5)
```



#### *Setting State for Window programmatically*

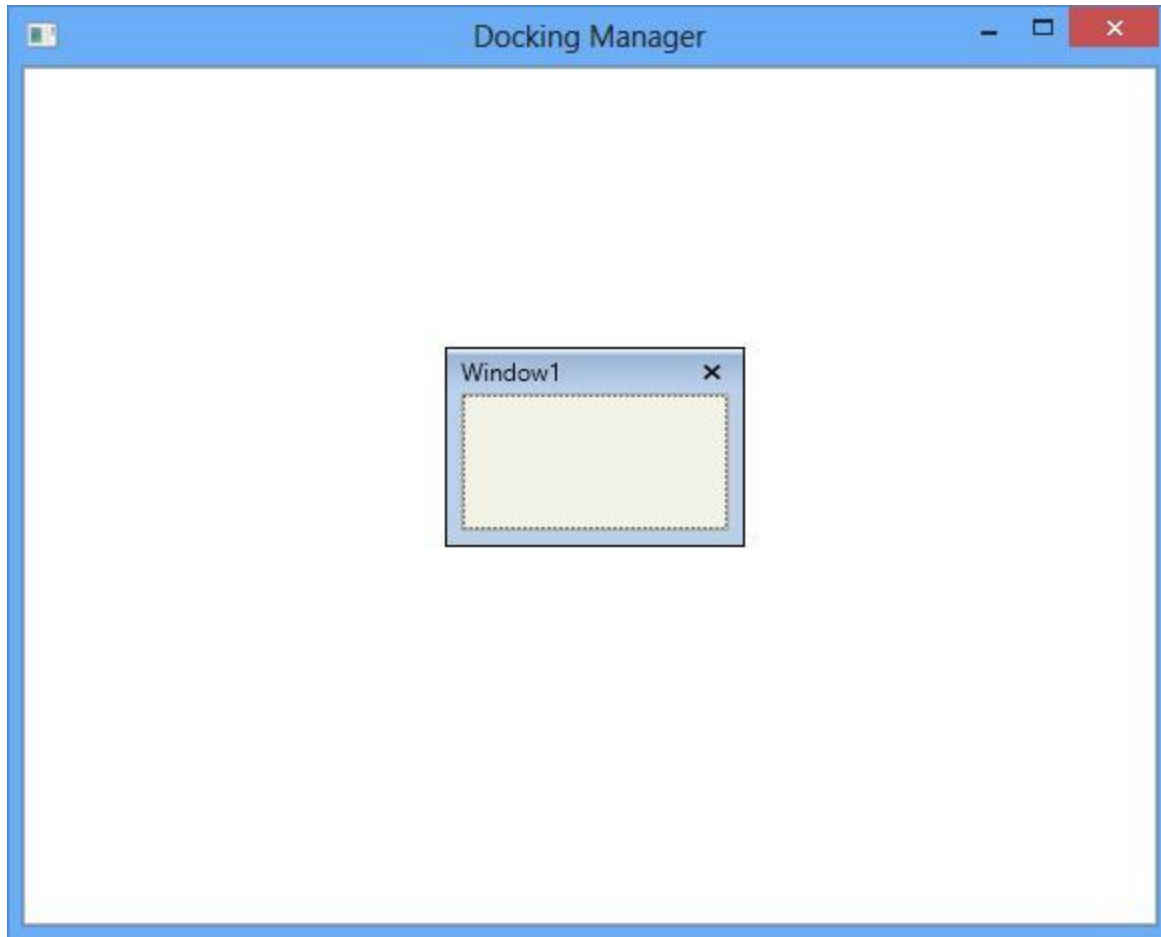
The state for the particular child window can be set programmatically using the [SetState](#) method of DockingManager.

#### **C#**

```
DockingManager.SetState(content1, DockState.Float);
```

#### **VB.NET**

```
DockingManager.SetState(content1, DockState.Float)
```



- Setting state as Document – To create document window in the DockingManager, set [UseDocumentContainer](#) as True for DockingManager and set its state as Document.

#### **C#**

```
DockingManager1.UseDocumentContainer = true;  
DockingManager.SetState(content1, DockState.Document);
```

#### **VB.NET**

```
DockingManager1.UseDocumentContainer = True  
DockingManager.SetState(content1, DockState.Document)
```



- Setting state as AutoHidden

**C#**

```
DockingManager.SetState(content1, DockState.AutoHidden);
```

**VB.NET**

```
DockingManager.SetState(content1, DockState.AutoHidden)
```



Making Window AutoHide programmatically

To auto hide the window programmatically call [ExecuteAutoHide](#) method of DockingManager.

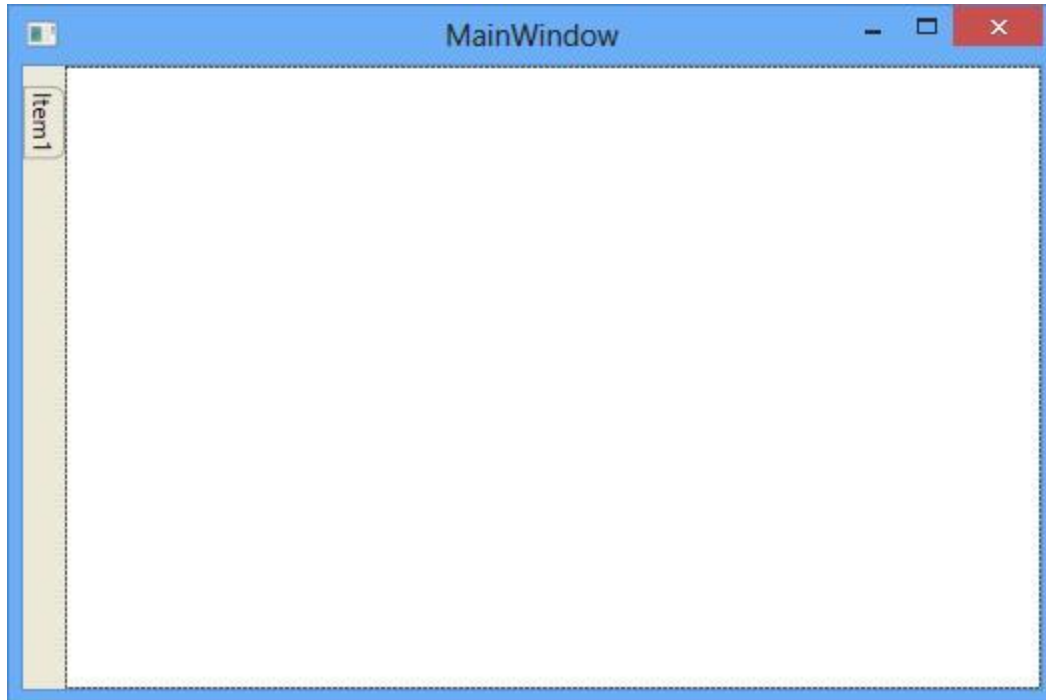
**C#**



```
DockingManager1.ExecuteAutoHide(Content1);
```

### VB.NET

```
DockingManager1.ExecuteAutoHide(Content1)
```



### Making Window Float and Document programmatically

The docking window can be made to float and document using the [SetState](#) method with its DockState value as **Float** and **Document** respectively

### C#

```
//to make the content1 float  
DockingManager.SetState(Content1, DockState.Float);  
//to make the content1 as Document window  
DockingManager.SetState(Content1, DockState.Document);
```

### VB.NET

```
'to make the content1 float  
DockingManager.SetState(Content1, DockState.Float)  
'to make the content1 as Document window  
DockingManager.SetState(Content1, DockState.Document)
```

### Hiding Window Programmatically

To hide the window, set [State](#) property of the DockingManager as **Hidden**.

### C#

```
DockingManager.SetState(Content1, DockState.Hidden);
```

**VB.NET**

```
DockingManager.SetState(Content1, DockState.Hidden)
```

To hide the window programmatically, call the [ExecuteClose](#) method with argument which refer the window need to be close.

**C#**

```
//Hide the element that passed as argument  
DockingManager1.ExecuteClose(Content1);
```

**VB.NET**

```
'Hide the element that passed as argument  
DockingManager1.ExecuteClose(Content1)
```

**Restore Window Programmatically**

To restore the closed window in the DockingManager, call [ExecuteRestore](#) method.

**C#**

```
//Restore the passed element with the state value as its argument  
DockingManager1.ExecuteRestore(Content1, DockState.Float);
```

**VB.NET**

```
'Restore the passed element with the state value as its argument  
DockingManager1.ExecuteRestore(Content1, DockState.Float)
```

**Detect the closing of a DockingManager child**

[WindowClosing](#) and [CloseButtonClick](#) are the two events, which can be used to get notification when the child windows are being closed.

**Dock Window Closing**

[WindowClosing](#) event raised whenever a child in [Float](#), [Dock](#) and [AutoHidden](#) windows are closing.

**Document Window Closing**

[CloseButtonClick](#) event raised only when close button of the [Document](#) child clicked. The following code describes how to handle the closing of all the children in [DockingManager](#).

**XML**

```
<syncfusion:DockingManager x:Name="dockingManager"  
CloseButtonClick="dockingManager_CloseButtonClick"  
WindowClosing="dockingManager_WindowClosing">  
  <ContentControl syncfusion:DockingManager.Header="Left"  
    syncfusion:DockingManager.SideInDockedMode="Left"  
    syncfusion:DockingManager.State="AutoHidden" />  
</syncfusion:DockingManager>
```

**C#**

```
// For windows in Document state
dockingManager.CloseButtonClick += Docking_CloseButtonClick;
// For windows in Float, Dock and AutoHidden state
dockingManager.WindowClosing += Docking_WindowClosing;
private void dockingManager_CloseButtonClick(object sender,
CloseButtonEventArgs e)
{
}
private void dockingManager_WindowClosing(object sender,
WindowClosingEventArgs e)
{
}
```

## Removing Window Programmatically

The windows for the DockingManager can be added using the Children collection. To remove the windows from the children collection, pass the window element that need to be remove using **Remove()** method of children property in DockingManager.

**C#**

```
DockingManager dockingmanager = new DockingManager();
ContentControl content1 = new ContentControl();
DockingManager.SetHeader(content1, "Dockwindow");
ContentControl content2 = new ContentControl();
DockingManager.SetHeader(content2, "Dockwindow2");
dockingmanager.Children.Add(content1);
dockingmanager.Children.Add(content2);
//The following code describes how to remove the child window
programmatically using the remove method.
dockingmanager.Children.Remove(content2);
Grid1.Children.Add(dockingmanager);
```

**VB.NET**

```
Dim dockingmanager As New DockingManager()
Dim content1 As New ContentControl()
DockingManager.SetHeader(content1, "Dockwindow")
Dim content2 As New ContentControl()
DockingManager.SetHeader(content2, "Dockwindow2")
dockingmanager.Children.Add(content1)
dockingmanager.Children.Add(content2)
'The following code describes how to remove the child window
programmatically using the remove method.
dockingmanager.Children.Remove(content2)
Grid1.Children.Add(dockingmanager)
```

## Event to notify when a child is added or removed

If you want to know while docking child added or removed from the **DockingManager**, it can be notified by using the [ChildrenCollectionChanged](#) event. It receives an argument of type **NotifyCollectionChangedEventArgs** containing the following information about the event.

Event Data	Description
Action	Gets an action that determines either an item is added to the children collection or item is removed from the children collection.
NewItems	Gets the newly added items in the children collection.
OldItems	Gets the removal of the items from the children collection.
NewStartingIndex	Gets the index location of the newly added items from the children collection.
OldStartingIndex	Gets the index location of the recently removed items from the children collection.

**XML**

```
<syncfusion:DockingManager
ChildrenCollectionChanged="DockingManager_ChildrenCollectionChanged"
Name="dockingManager" >
<ContentControl syncfusion:DockingManager.Header="Solution Explorer"
Name="SolutionExplorer" />
</syncfusion:DockingManager>
```

**XML**

```
dockingManager.ChildrenCollectionChanged +=
DockingManager_ChildrenCollectionChanged;
```

**C#**

```
//Add a docking child item in the children collection of DockingManager
ContentControl contentControl = new ContentControl();
dockingManager.Children.Add(contentControl)
```

You can handle the event as follows,

**C#**

```
private void DockingManager_ChildrenCollectionChanged(object sender,
NotifyCollectionChangedEventArgs e) {
//Occurs when children collection changed
MessageBox.Show("Children collection was changed");
if(e.Action== NotifyCollectionChangedEventArgs.Add) {
var added_Item = e.NewItems;
int addedIndex = e.NewStartingIndex;
}
else if (e.Action == NotifyCollectionChangedEventArgs.Remove) {
var removed_item = e.OldItems;
int removedIndex = e.OldStartingIndex;
}
}
```

### Restricting Docking in Float Window

The float window allows to dock another float window inside it by default. This behavior can be restricted by set [CanDockOnFloat](#) as False for that particular window.

#### XML

```
<syncfusion:DockingManager x:Name="DockingManager1">
  <ContentControl syncfusion:DockingManager.Header="Item1"
    syncfusion:DockingManager.State="Float"
    syncfusion:DockingManager.CanDockonFloat="False"/>
</syncfusion:DockingManager>
```

#### C#

```
DockingManager.SetCanDockonFloat(Item1, false);
```

### Customizing a window

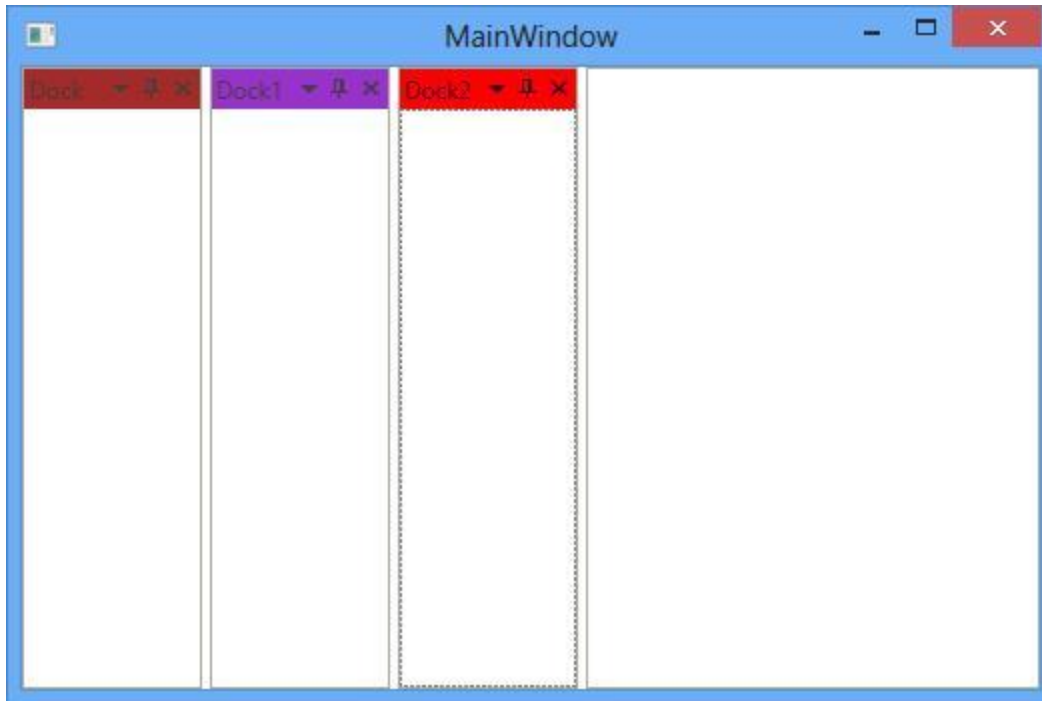
A Docking window can be customized using the property [HeaderBackground](#), [SelectedHeaderBackground](#), [HeaderMouseHoverBackground](#) with the desired brush values respectively.

#### XML

```
<syncfusion:DockingManager SelectedHeaderBackground="Red"
  HeaderBackground="Brown" HeaderMouseOverBackground="DarkOrchid" >
  <ContentControl syncfusion:DockingManager.Header="Dock"/>
  <ContentControl syncfusion:DockingManager.Header="Dock1"/>
  <ContentControl syncfusion:DockingManager.Header="Dock2"/>
</syncfusion:DockingManager>
```

#### C#

```
SyncDockingManager.SelectedHeaderBackground = new
SolidColorBrush(Colors.Red);
SyncDockingManager.HeaderBackground = new SolidColorBrush(Colors.Brown);
SyncDockingManager.HeaderMouseOverBackground = new
SolidColorBrush(Colors.DarkOrchid);
```



### Customizing FloatWindow

The float window can be customized by setting [FloatWindowHeaderBackground](#), [FloatWindowHeaderForeground](#), [FloatWindowSelectedHeaderBackground](#), [FloatWindowSelectedBorderBrush](#) and [FloatWindowMouseOverHeaderBackground](#) properties with the required brush values respectively.

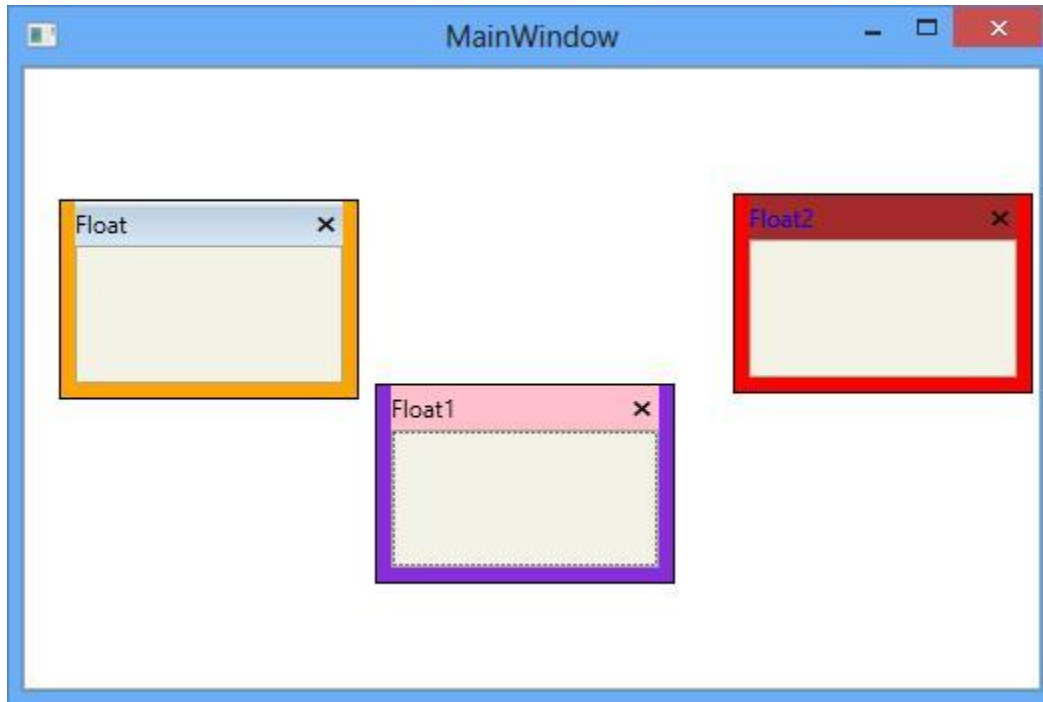
### XML

```
<syncfusion:DockingManager FloatWindowHeaderBackground="Brown"
FloatWindowHeaderForeground="Blue"
FloatWindowMouseOverBorderBrush="Orange"
FloatWindowSelectedHeaderBackground="Pink"
FloatWindowBorderBrush="Red" FloatWindowSelectedBorderBrush="BlueViolet" >
  <ContentControl syncfusion:DockingManager.Header="Float"
syncfusion:DockingManager.State="Float"/>
  <ContentControl syncfusion:DockingManager.Header="Float1"
syncfusion:DockingManager.State="Float"/>
  <ContentControl syncfusion:DockingManager.Header="Float2"
syncfusion:DockingManager.State="Float"/>
</syncfusion:DockingManager>
```

### C#

```
SyncDockingManager.FloatWindowHeaderBackground = new
SolidColorBrush(Colors.Brown);
SyncDockingManager.FloatWindowHeaderForeground = new
SolidColorBrush(Colors.Blue);
SyncDockingManager.FloatWindowMouseOverBorderBrush = new
SolidColorBrush(Colors.Orange);
SyncDockingManager.FloatWindowSelectedHeaderBackground = new
SolidColorBrush(Colors.Pink);
SyncDockingManager.FloatWindowBorderBrush = new SolidColorBrush(Colors.Red);
```

```
SyncDockingManager.FloatWindowSelectedBorderBrush = new
SolidColorBrush(Colors.BlueViolet);
```



### Enable/Disable Dragging a Window

The attached property [CanDrag](#) that helps to enable or disable the dragging functionality of a window by setting its value as True or False respectively. By default its value is **True**, to disable this functionality turn its value to **False**.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1" >
<ContentControl syncfusion:DockingManager.Header="Item1"
syncfusion:DockingManager.CanDrag="False"/>
</syncfusion:DockingManager>
```

### C#

```
DockingManager.SetCanDrag(Item1, false);
```

### Drag Shadow of a Window

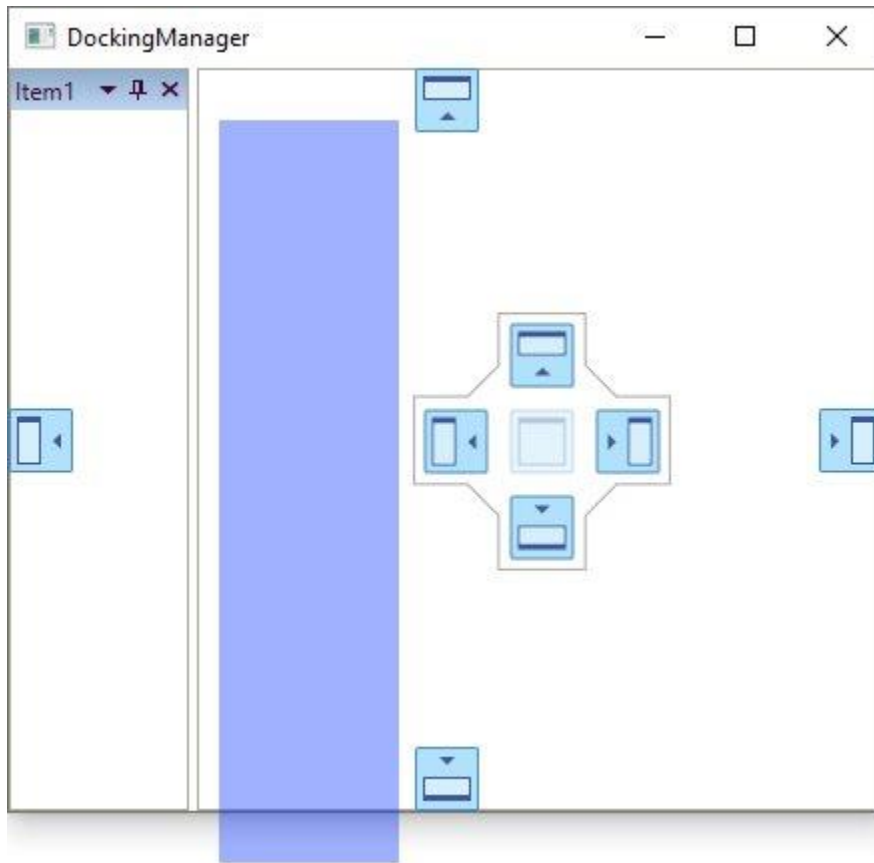
To drag child window in Shadow mode, set the [DraggingType](#) property of DockingManager as **ShadowDragging**.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
DraggingType="ShadowDragging" >
<ContentControl syncfusion:DockingManager.Header="Item1"/>
</syncfusion:DockingManager>
```

**C#**

```
SyncDockingManager.DraggingType = DraggingType.ShadowDragging;
```

**Drag Border of a Window**

To drag child window in Border mode, set the [DraggingType](#) property of DockingManager as **BorderDragging**.

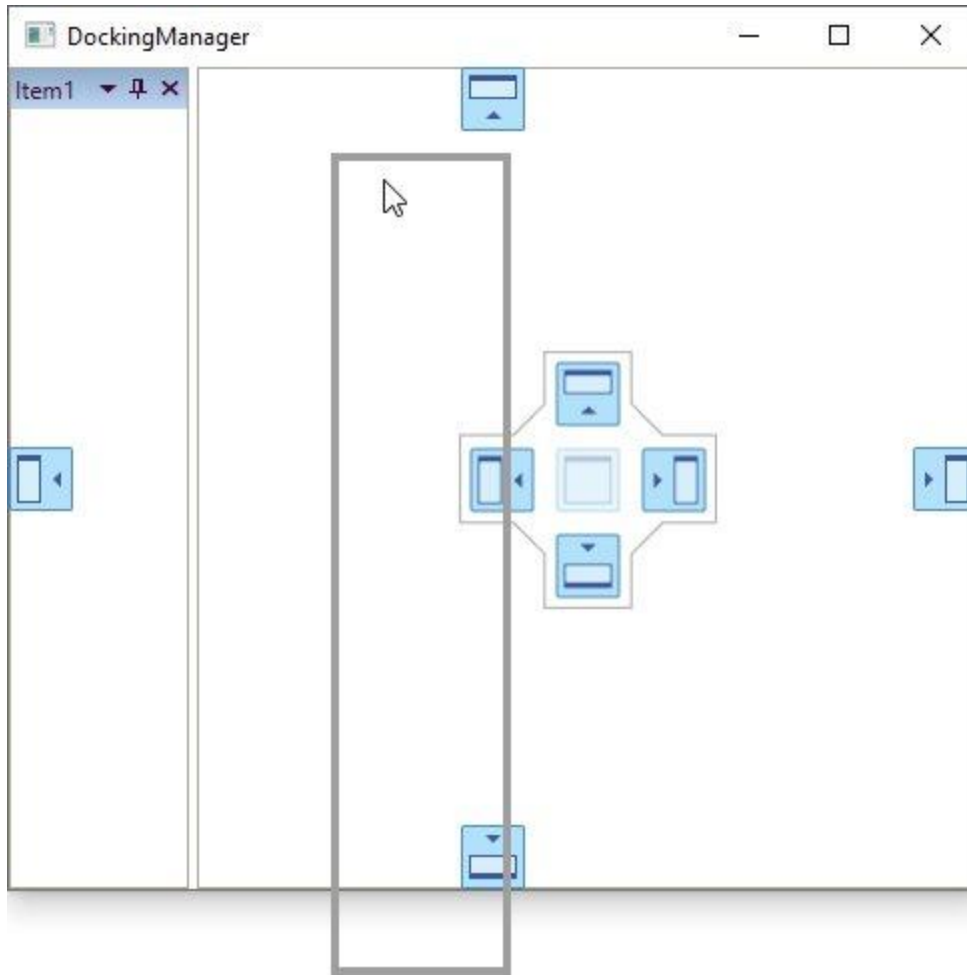
**XML**

```
<syncfusion:DockingManager x:Name="DockingManager1"
    DraggingType="BorderDragging">
    <ContentControl syncfusion:DockingManager.Header="Item1"/>
</syncfusion:DockingManager>
```

**C#**

```
SyncDockingManager.DraggingType = DraggingType.BorderDragging;
```





### Customizing a resizing behaviors

DockingManager allows to resize the dock and float windows by default. To restrict resizing the dock and float windows respectively, set the [CanResizeInDockedState](#) and [CanResizeInFloatState](#) properties with it value as False.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1">
  <ContentControl syncfusion:DockingManager.Header="Dockwindow"
    x:Name="Content1"
    syncfusion:DockingManager.State="Dock"
    syncfusion:DockingManager.CanResizeInDockedState="False"/>
  <ContentControl syncfusion:DockingManager.Header="Floatwindow"
    x:Name="Content2"
    syncfusion:DockingManager.State="Float"
    syncfusion:DockingManager.CanResizeInFloatState="False"/>
</syncfusion:DockingManager>
```

### C#

```
DockingManager.SetCanResizeInDockedState(Content1, false);
DockingManager.SetCanResizeInFloatState(Content2, false);
```

*Width resizing restriction*

To restrict resizing width for the Dock windows set the property [CanResizeWidthInDockedMode](#) and [CanResizeWidthInFloatState](#) as False.

**XML**

```
<ContentControl syncfusion:DockingManager.Header="Dockwindow"
x:Name="Content1"
syncfusion:DockingManager.State="Dock"
syncfusion:DockingManager.CanResizeWidthInDockedState="False"/>
<ContentControl syncfusion:DockingManager.Header="Floatwindow"
x:Name="Content2"
syncfusion:DockingManager.State="Float"
syncfusion:DockingManager.CanResizeWidthInFloatState="False"/>
```

**C#**

```
DockingManager.SetCanResizeWidthInDockedState(Content1, false);
DockingManager.SetCanResizeWidthInFloatState(Content2, false);
```

*Height resizing restriction*

To restrict resizing the height for the float and dock window respectively, set the property [CanResizeHeightInFloatState](#) and [CanResizeHeightInDockedState](#) as False.

**XML**

```
<ContentControl syncfusion:DockingManager.Header="Dockwindow"
x:Name="Content1"
syncfusion:DockingManager.State="Dock"
syncfusion:DockingManager.CanResizeHeightInDockedState="False"/>
<ContentControl syncfusion:DockingManager.Header="Floatwindow"
x:Name="Content2"
syncfusion:DockingManager.State="Float"
syncfusion:DockingManager.CanResizeHeightInFloatState="False"/>
```

**C#**

```
DockingManager.SetCanResizeHeightInDockedState(Content1, false);
DockingManager.SetCanResizeHeightInFloatState(Content2, false);
```

*Setting MaxWidth and MaxHeight for Window*

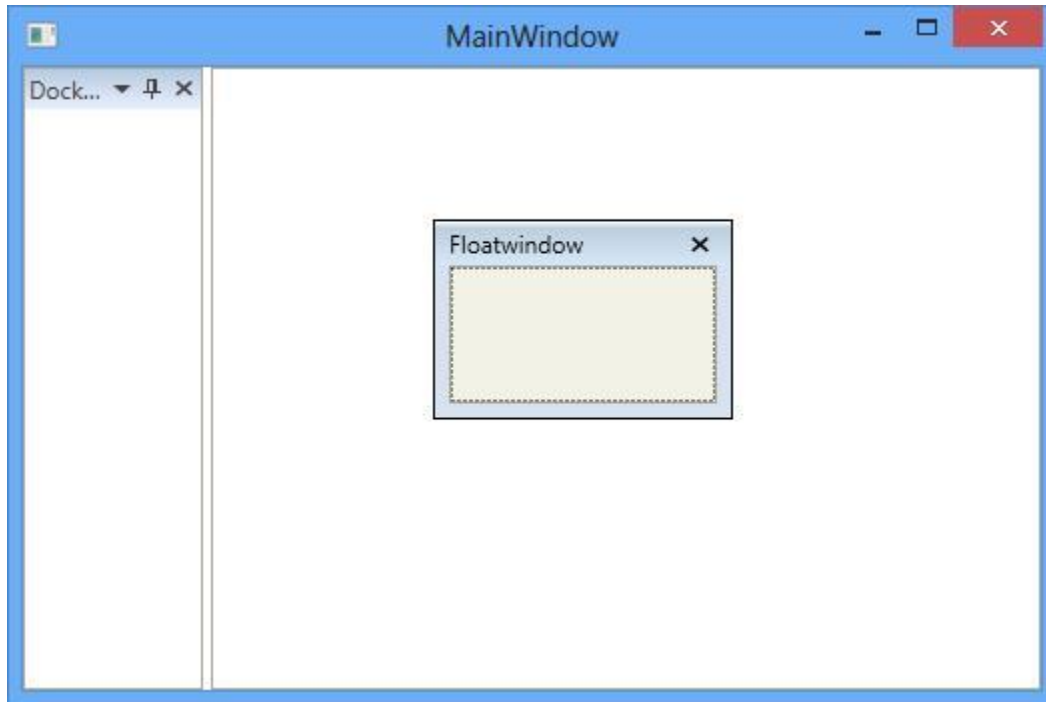
To set the desired maximum width for the float and dock windows respectively, set their properties [DesiredMaxWidthInDockedMode](#) and [DesiredMaxWidthInFloatingMode](#) with the desired values.

**XML**

```
<ContentControl syncfusion:DockingManager.Header="Dockwindow"
syncfusion:DockingManager.State="Dock"
syncfusion:DockingManager.DesiredMaxWidthInDockedMode="1000"/>
<ContentControl syncfusion:DockingManager.Header="Floatwindow"
syncfusion:DockingManager.State="Float"
syncfusion:DockingManager.DesiredMaxWidthInFloatingMode="600"/>
```

**C#**

```
DockingManager.SetDesiredMaxWidthInDockedMode(dockWindow1, 1000);  
DockingManager.SetDesiredMaxWidthInFloatingMode(floatWindow1, 600);
```



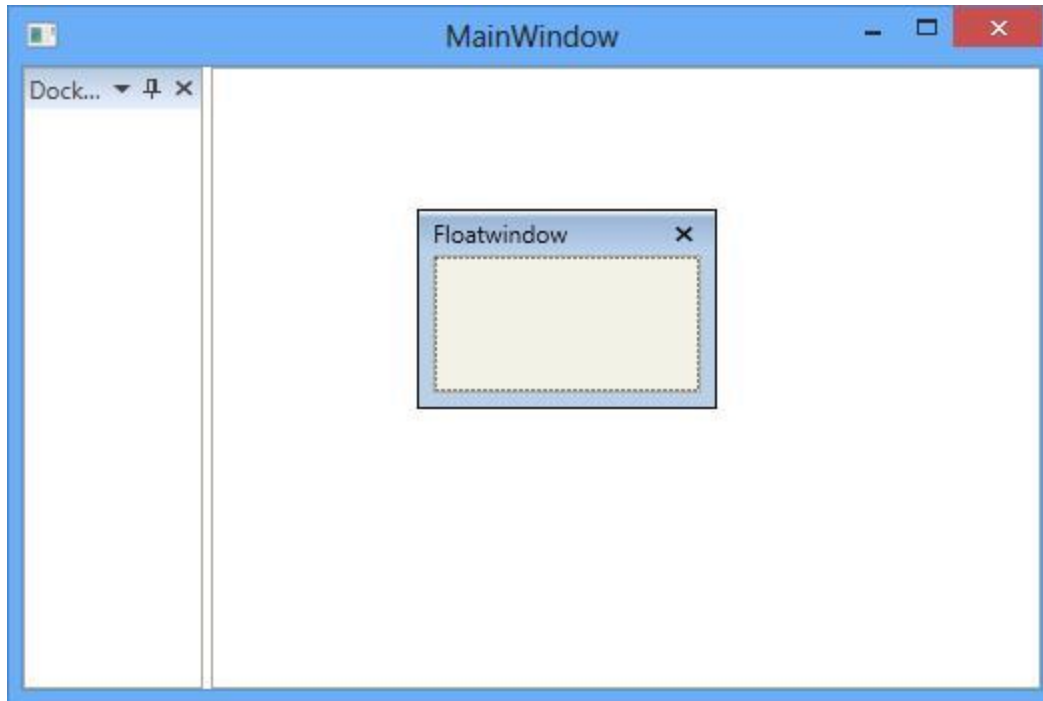
To set the maximum height for the float and dock windows respectively, set their properties [DesiredMaxHeightInDockedMode](#) and [DesiredMaxHeightInFloatingMode](#) with the desired values.

#### XML

```
<ContentControl syncfusion:DockingManager.Header="Dockwindow"  
syncfusion:DockingManager.State="Dock"  
syncfusion:DockingManager.DesiredMaxHeightInDockedMode="300"/>  
<ContentControl syncfusion:DockingManager.Header="Floatwindow"  
syncfusion:DockingManager.State="Float"  
syncfusion:DockingManager.DesiredMaxHeightInFloatingMode="200"/>
```

#### C#

```
DockingManager.SetDesiredMaxHeightInDockedMode(dockWindow1, 700);  
DockingManager.SetDesiredMaxHeightInFloatingMode(floatWindow1, 200);
```



### Configuring window sizing

DockingManager allows to set the desired width and height for the dock windows. The window **Width** and **Height** value set to "90" based on the container by default.

#### *Desire height and width*

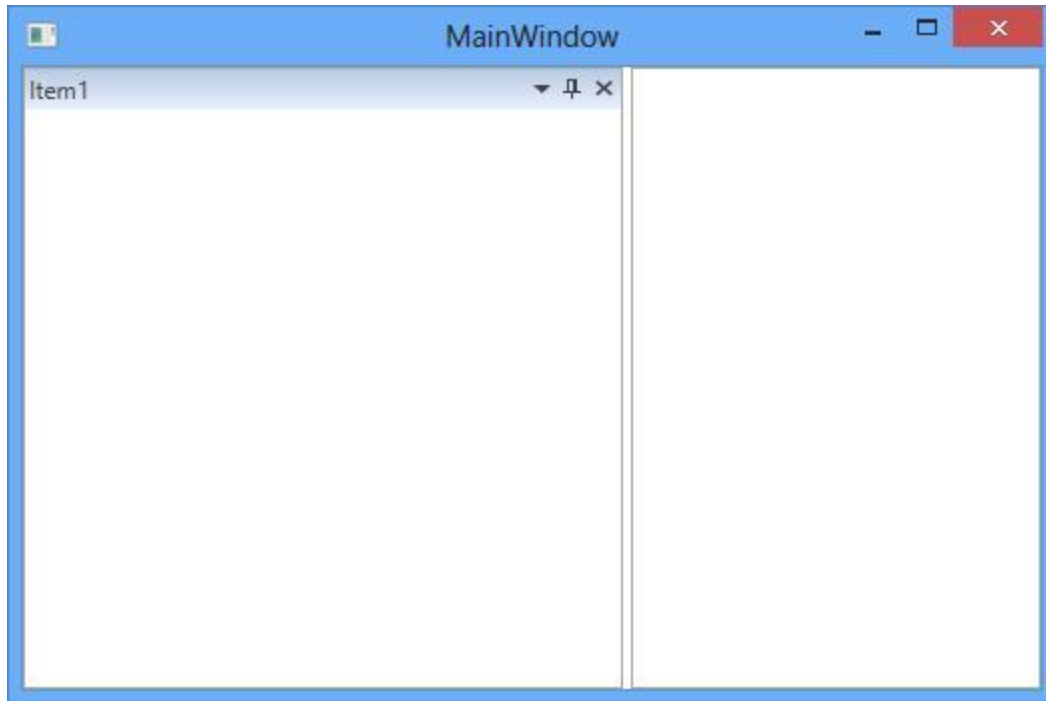
The desired height and width can be set for the Dock windows through the property [DesiredWidthInDockedMode](#) and [DesiredHeightInDockedMode](#) with the desired values.

### **XML**

```
<syncfusion:DockingManager x:Name="DockingManager1">
  <ContentControl syncfusion:DockingManager.Header="Item1"
    syncfusion:DockingManager.DesiredHeightInDockedMode="400"
    syncfusion:DockingManager.DesiredWidthInDockedMode="300"/>
</syncfusion:DockingManager>
```

### **C#**

```
DockingManager.SetDesiredHeightInDockedMode(Content1, 400);
DockingManager.SetDesiredWidthInDockedMode(Content1, 300);
```



### *Sizing Based on the Content*

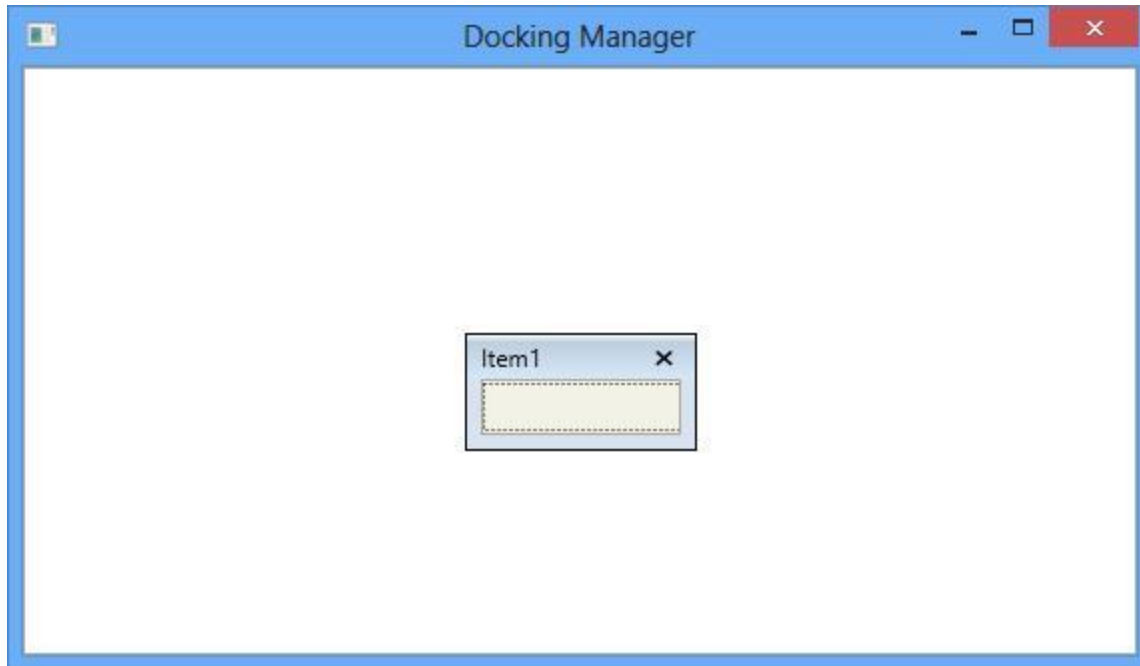
To size the float window based on the children window size, set the property [SizeToContentInFloat](#) property for the child window as True. By default, its value is False.

### **XML**

```
<ContentControl syncfusion:DockingManager.Header="Item1" x:Name="Content1"
syncfusion:DockingManager.SizeToContentInFloat="True" Width="100"
Height="24"/>
```

### **C#**

```
DockingManager.SetSizeToContentInFloat (Content1, true);
```



To size the Dock window based on the children window size, set the property [SizeToContentInDock](#) property for the child window as True. By default, its value is False.

#### XML

```
<syncfusion:DockingManager x:Name="DockingManager1" >  
  <ContentControl syncfusion:DockingManager.Header="Item1" x:Name="Content1"  
    syncfusion:DockingManager.SizeToContentInDock="True" Width="100"  
    Height="24"/>  
</syncfusion:DockingManager>
```

#### C#

```
DockingManager.SetSizeToContentInDock(Content1, true);
```



#### *Absolute Sizing on Dock to fill*

To load the child window initially with an absolute size, set the property [DockFillMode](#) as **Absolute**. By default, the child window loaded with the default size and it can be set through [DockFillMode](#) as **Default** also.

#### **XML**

```
<ContentControl syncfusion:DockingManager.Header="Item1"
syncfusion:DockingManager.DockFillMode="Absolute"/>
```

#### **C#**

```
DockingManager.SetDockFillMode(content1, DockFillModes.Absolute);
```



#### *Customizing the Splitter appearance*

The Splitter of the dock window can be customized using the [SplitterSize](#) and [SplitterBackground](#) properties depends upon its values respectively.

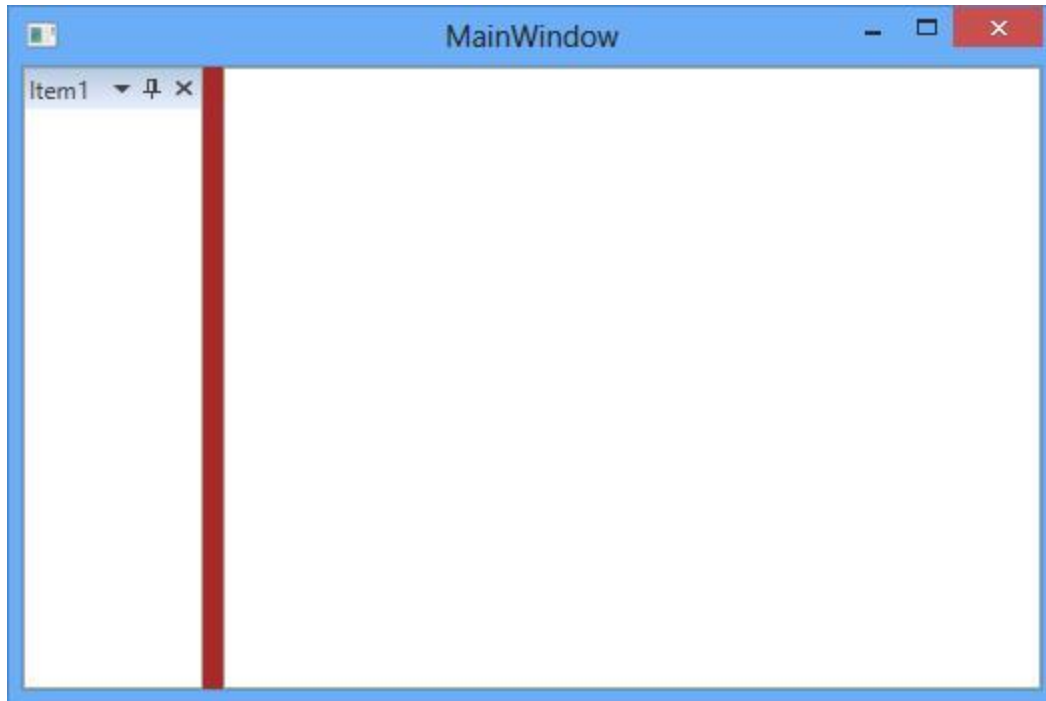
#### **XML**

```
<syncfusion:DockingManager UseDocumentContainer=" True"
SplitterBackground="Brown" SplitterSize="10" >
<ContentControl syncfusion:DockingManager.Header="Item1"></ContentControl>
</syncfusion:DockingManager>
```

#### **C#**

```
SyncDockingManager.SplitterBackground = new SolidColorBrush(Colors.Brown);
SyncDockingManager.SplitterSize = 10;
```





Occupy whole window

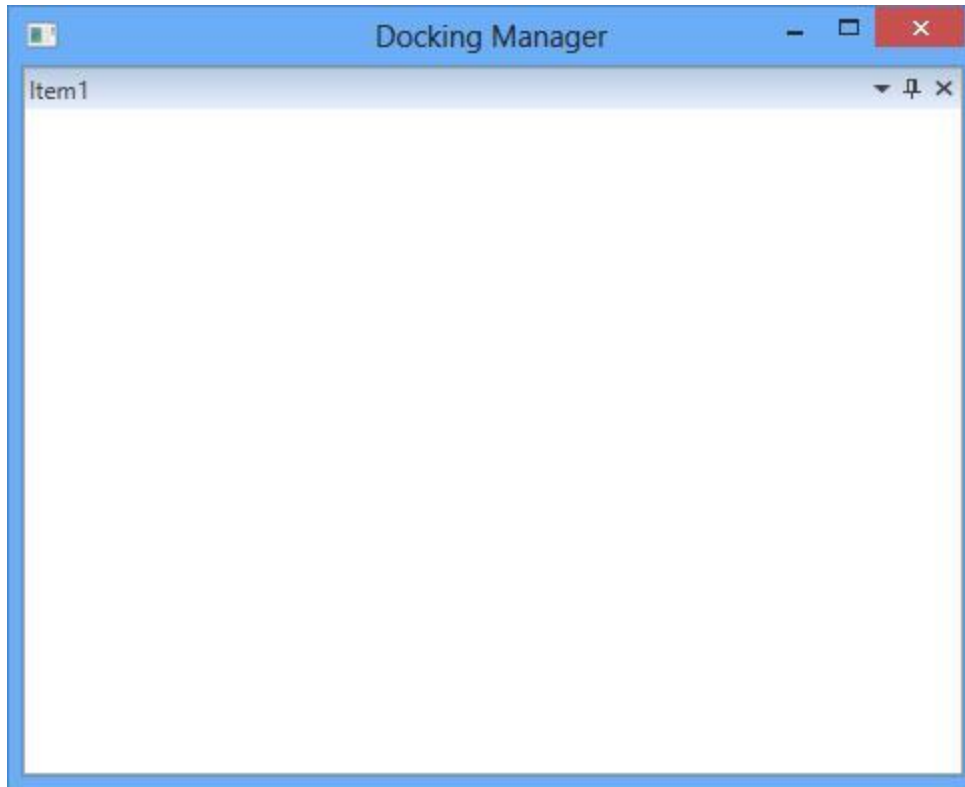
To arrange the dock windows to a whole available space in the DockingManager, set [DockFill](#) property of DockingManager as True.

#### XML

```
<syncfusion:DockingManager DockFill="True">  
  <ContentControl syncfusion:DockingManager.Header="Item1"/>  
</syncfusion:DockingManager>
```

#### C#

```
SyncDockingManager.DockFill = true;
```



And when DockFill functionality is enabled, DockingManager changes the DockWindow to AutoHidden state, if any Document state window is present .

#### XML

```
<syncfusion:DockingManager UseDocumentContainer="True" DockFill="True">
  <ContentControl syncfusion:DockingManager.Header="Item1"/>
  <ContentControl syncfusion:DockingManager.Header="Item1"
    syncfusion:DockingManager.State="Document"/>
</syncfusion:DockingManager>
```

#### C#

```
SyncDockingManager.UseDocumentContainer = true;
SyncDockingManager.DockFill = true;
DockingManager.SetState(dockWindow1, DockState.Document);
DockingManager.SetState(document1, DockState.Document);
```



*Restrict DockWindow to AutoHide while DockFill*

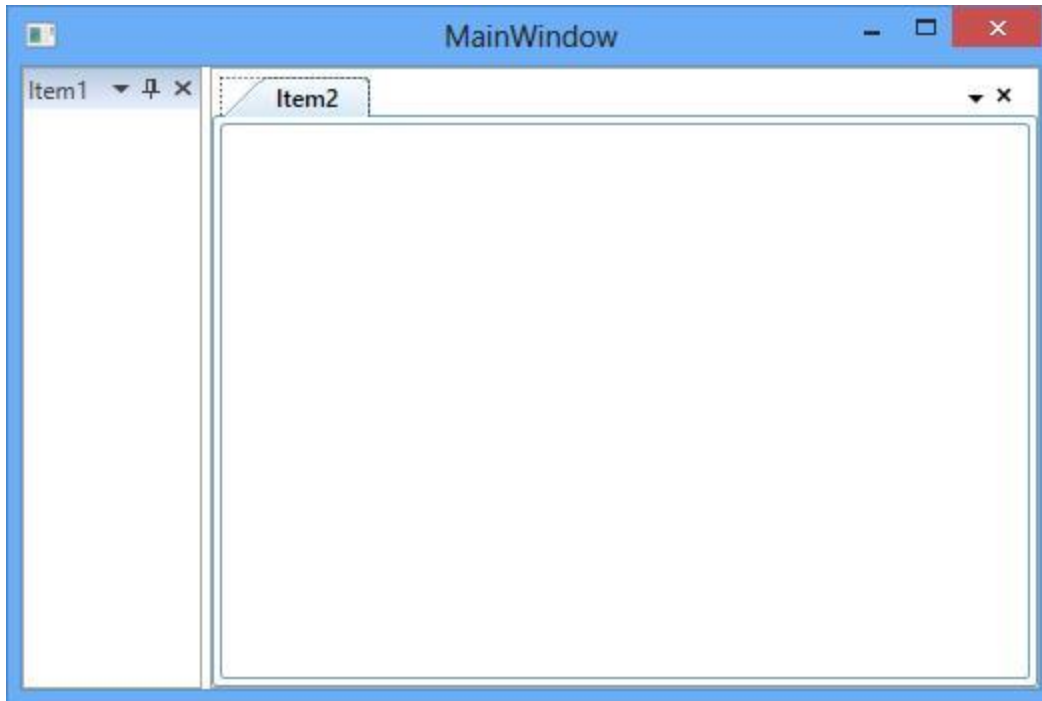
To restrict the behavior of changing the DockWindow to AutoHide when DockFill is True, set the property [DockFillDocumentMode](#) as Normal.

#### **XML**

```
<syncfusion:DockingManager x:Name="SyncDockingManager"
UseDocumentContainer="True" DockFill="True" DockFillDocumentMode="Normal">
  <ContentControl x:Name="Content1" syncfusion:DockingManager.Header="Item1"/>
  <ContentControl x:Name="Content2" syncfusion:DockingManager.Header="Item2"
syncfusion:DockingManager.State="Document"/>
</syncfusion:DockingManager>
```

#### **C#**

```
SyncDockingManager.DockFillDocumentMode = DockFillDocumentMode.Normal;
```



### Applying Context Menu

You can add custom context menu items for dock and float windows through an attached property [CustomMenuItems](#) property. You can also add sub menu items for custom context menu item by adding that sub [CustomMenuItem](#) to the parent [CustomMenuItem](#). You can check or uncheck the [CustomMenuItem](#) interactively or programmatically by using the [CustomMenuItem.IsChecked](#) property.

You can collapse the default context menu and show only the custom context menu items by setting the [CollapseDefaultContextMenuItemsInFloat](#) property to `true`. The default value of [CollapseDefaultContextMenuItemsInFloat](#) property is `false`.

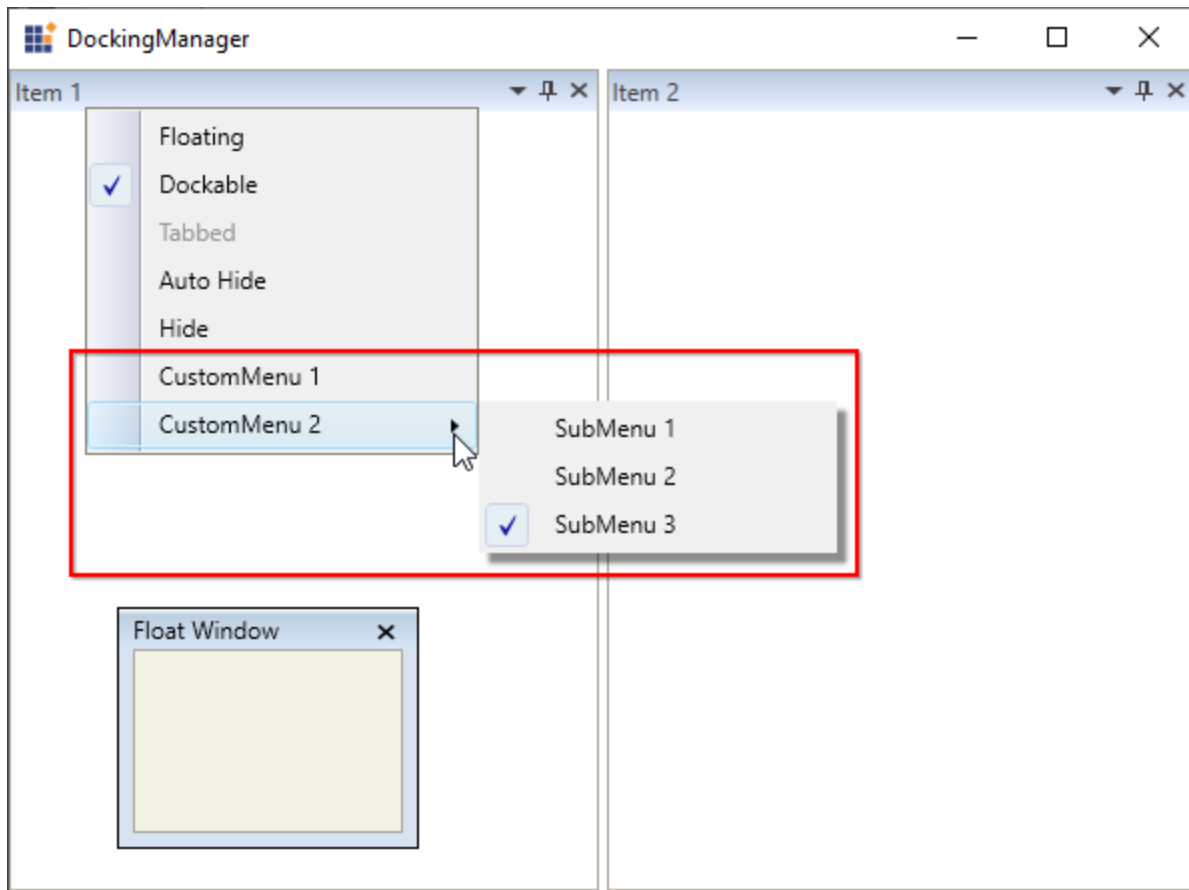
### XML

```
<syncfusion:DockingManager
    DockFill="True"
    Name="dockingManager" >
    <!--Adding custom context menu items for float windows-->
    <syncfusion:DockingManager.FloatWindowContextMenuItems>
    <syncfusion:CustomMenuItemCollection>
    <!--Adding custom context menu items-->
    <syncfusion:CustomMenuItem Header="Menu 1"/>
    <syncfusion:CustomMenuItem Header="Menu 2">
    <!--Adding sub custom context menu items-->
    <syncfusion:CustomMenuItem Header="SubMenu 1"/>
    <syncfusion:CustomMenuItem Header="SubMenu 2"/>
    <syncfusion:CustomMenuItem Header="SubMenu 3" IsChecked="True"/>
    </syncfusion:CustomMenuItem>
    </syncfusion:CustomMenuItemCollection>
    </syncfusion:DockingManager.FloatWindowContextMenuItems>
    <ContentControl syncfusion:DockingManager.Header="Item 1"
    syncfusion:DockingManager.State="Dock"/>
    <ContentControl syncfusion:DockingManager.Header="Item 2"
```

```
syncfusion:DockingManager.State="Dock"/>
<ContentControl syncfusion:DockingManager.Header="Float Window"
syncfusion:DockingManager.CollapseDefaultContextMenuItemsInFloat="True"
syncfusion:DockingManager.State="Float"/>
</syncfusion:DockingManager>
```

## C#

```
//Creating custom context menu items
CustomMenuItem menu1 = new CustomMenuItem();
menu1.Header = "Menu 1";
CustomMenuItem menu2 = new CustomMenuItem();
menu2.Header = "Menu 2";
//Creating custom sub context menu items
CustomMenuItem customMenuItem1 = new CustomMenuItem() { Header = "SubMenu 1"
};
CustomMenuItem customMenuItem2 = new CustomMenuItem() { Header = "SubMenu 2"
};
CustomMenuItem customMenuItem3 = new CustomMenuItem() { Header = "SubMenu
3", IsChecked = true };
menu2.Items.Add(customMenuItem1);
menu2.Items.Add(customMenuItem2);
menu2.Items.Add(customMenuItem3);
//Adding custom context menu items with sub menu items
dockingManager.FloatWindowContextMenuItems.Add(menu1);
dockingManager.FloatWindowContextMenuItems.Add(menu2);
```



**Note:** [View Sample in GitHub](#)

#### *Adding CustomContextMenuItems to Document window*

The custom context menu items can be added in addition to default ContextMenu items for the document window through an attached property [DocumentTabItemContextMenuItems](#).

#### **XML**

```
<syncfusion:DockingManager UseDocumentContainer="True"
x:Name="DockingManager1">
  <syncfusion:DockingManager.DocumentTabItemContextMenuItems>
    <syncfusion:DocumentTabItemMenuItemCollection>
      <syncfusion:CustomMenuItem Header="CustomItem1"/>
      <syncfusion:CustomMenuItem Header="CustomItem2"/>
    </syncfusion:DocumentTabItemMenuItemCollection>
  </syncfusion:DockingManager.DocumentTabItemContextMenuItems>
  <ContentControl x:Name="Content1" syncfusion:DockingManager.Header="Item1"
  syncfusion:DockingManager.State="Document"/>
</syncfusion:DockingManager>
```

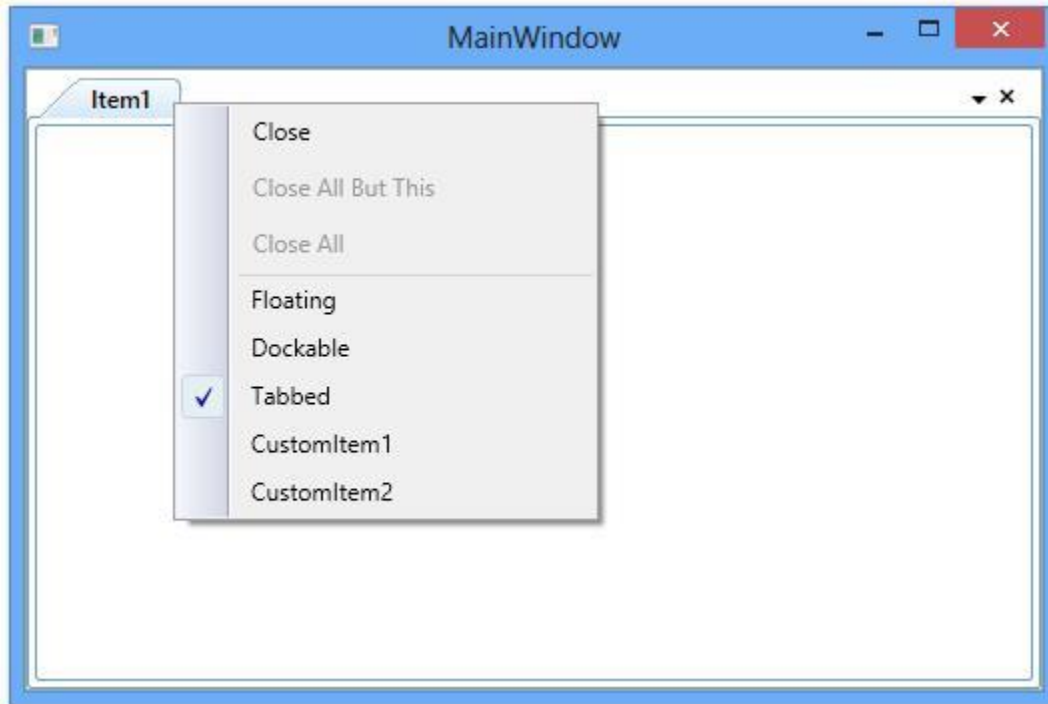
#### **C#**

```
DocumentTabItemMenuItemCollection collection = new
DocumentTabItemMenuItemCollection();
CustomMenuItem item1 = new CustomMenuItem();
CustomMenuItem item2 = new CustomMenuItem();
item1.Header = "CustomItem1";
```

```

item2.Header = "CustomItem2";
collection.Add(item1);
collection.Add(item2);
DockingManager.SetDocumentTabItemContextMenuItems(DockingManager1,
collection);

```



#### CustomMenuItem as Separator

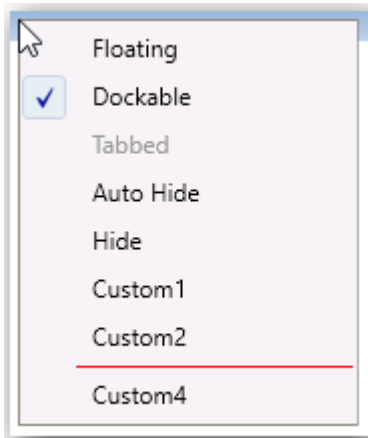
The [IsSeparator](#) property of [CustomMenuItem](#) is used to display separator between CustomMenuItems. If the property [IsSeparator](#) is set to true, the [CustomMenuItem](#) will act as Separator. The default value of this property is false.

#### C#

```

<syncfusion:DockingManager Grid.Row="1" DockFill="True" Name="Docking">
  <syncfusion:DockingManager.CustomMenuItems>
    <syncfusion:CustomMenuItemCollection>
      <syncfusion:CustomMenuItem Header="Custom1"/>
      <syncfusion:CustomMenuItem Header="Custom2"/>
      <syncfusion:CustomMenuItem BorderBrush="Red" IsSeparator="True"/>
      <syncfusion:CustomMenuItem Header="Custom4"/>
    </syncfusion:CustomMenuItemCollection>
  </syncfusion:DockingManager.CustomMenuItems>
  <ContentControl x:Name="Dock1" syncfusion:DockingManager.Header="Dock1"/>
  <ContentControl syncfusion:DockingManager.Header="Dock2"
    syncfusion:DockingManager.SideInDockedMode="Tabbed"
    syncfusion:DockingManager.TargetNameInDockedMode="Dock1"/>
  <ContentControl syncfusion:DockingManager.Header="Dock3"
    syncfusion:DockingManager.State="Dock"/>
  <ContentControl syncfusion:DockingManager.Header="Dock4"
    syncfusion:DockingManager.State="Dock"/>
</syncfusion:DockingManager>

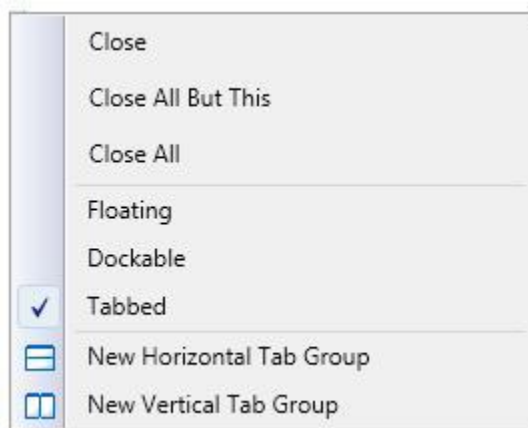
```



#### *Customizing ContextMenuItems Visibility*

The default context menu is visible on right-clicking the different state child windows and its visibility can be customized.

The default context menu items as shown in the following screenshot:



To collapse the default context menu, set the property [CollapseDefaultContextMenu](#) as True.

#### **XML**

```
<syncfusion:DockingManager x:Name="DockingManager1"
    UseDocumentContainer="True" CollapseDefaultContextMenuItems="True"/>
```

#### **C#**

```
DockingManager1.CollapseDefaultContextMenuItems = true;
```

To collapse the default context menu in Dock state window, set the property [CollapseDefaultContextMenuInDock](#) as True. By default, its value is False.

#### **XML**

```
<syncfusion:DockingManager x:Name="DockingManager1"
    UseDocumentContainer="True" >
```



```
<ContentControl syncfusion:DockingManager.Header="DockWindow"
x:Name="Content1"
syncfusion:DockingManager.State="Dock"
syncfusion:DockingManager.CollapseDefaultContextMenuItemsInDock="True"/>
</syncfusion:DockingManager>
```

### C#

```
DockingManager.SetCollapseDefaultContextMenuItemsInDock(Content1, true);
```

To collapse the default context menu in Document state window, set the property [CollapseDefaultContextMenuInDocument](#) as True. By default, its value is False.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True">
<ContentControl syncfusion:DockingManager.Header="DockWindow"
x:Name="Content1"
syncfusion:DockingManager.State="Document"
syncfusion:DockingManager.CollapseDefaultContextMenuItemsInDocumentTab="True"
/>
</syncfusion:DockingManager>
```

### C#

```
DockingManager.SetCollapseDefaultContextMenuItemsInDocumentTab(Content1,
true);
```

To collapse the default context menu in Float state window, set the property [CollapseDefaultContextMenuInFloat](#) as True. By default, its value is False.

### XML



```
<ContentControl syncfusion:DockingManager.Header="DockWindow"
syncfusion:DockingManager.State="Float"
syncfusion:DockingManager.CollapseDefaultContextMenuItemsInFloat="True"/>
```



### C#

```
DockingManager.SetCollapseDefaultContextMenuItemsInFloat(Content1, true);
```

### DockBehavior

[DockBehavior](#) has two options Default and VS2010, difference between the two modes is explained in the table below.

DockBehavior	Default	VS2010
Context menu options		

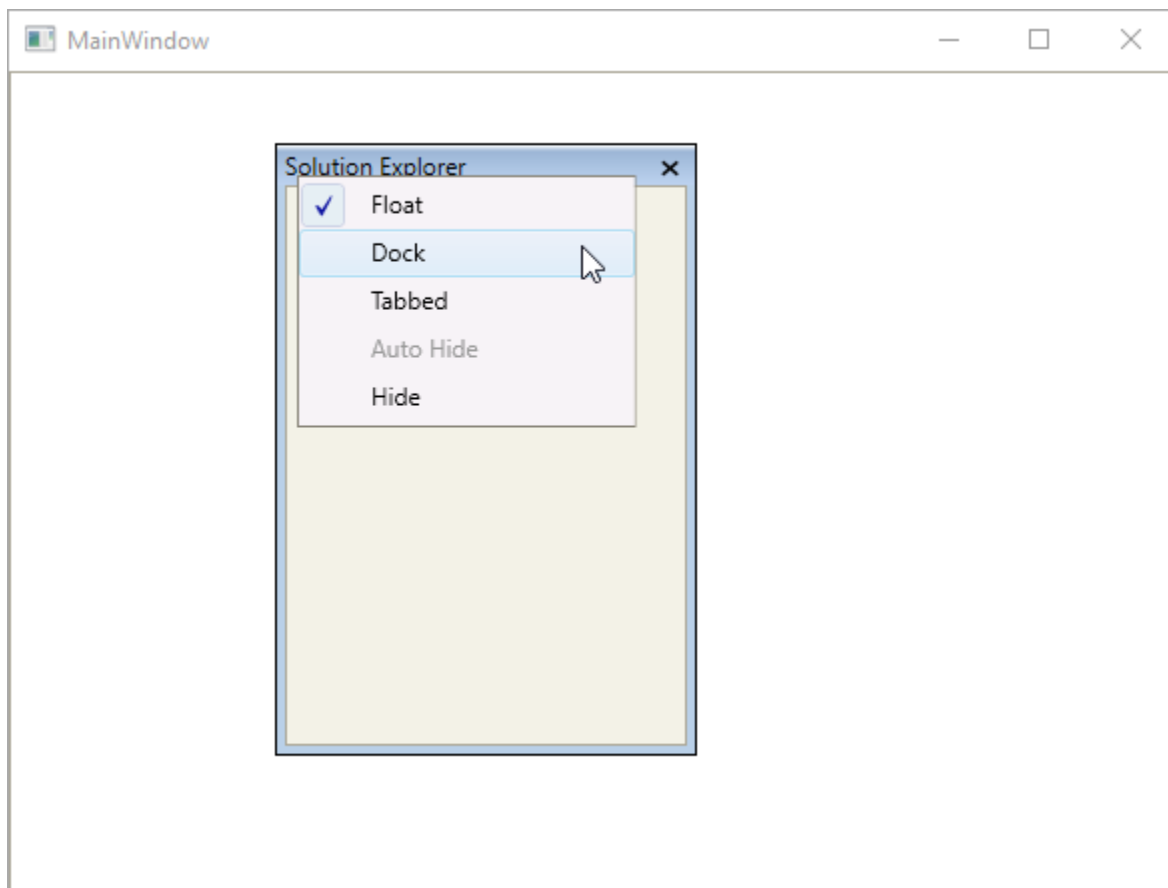
Different Behavior	Dockable: A window's <b>Dockable</b> menu item represents that it may or may not be docked.	Dock: You can dock a window from float or tabbed state by clicking <b>Dock</b> menu item.
Same Behavior	The checked status of all other menu items is the current status of the window.	The checked status of all other menu items is the current status of the window.
Context menu option when window is in <b>Auto-Hide</b> state		

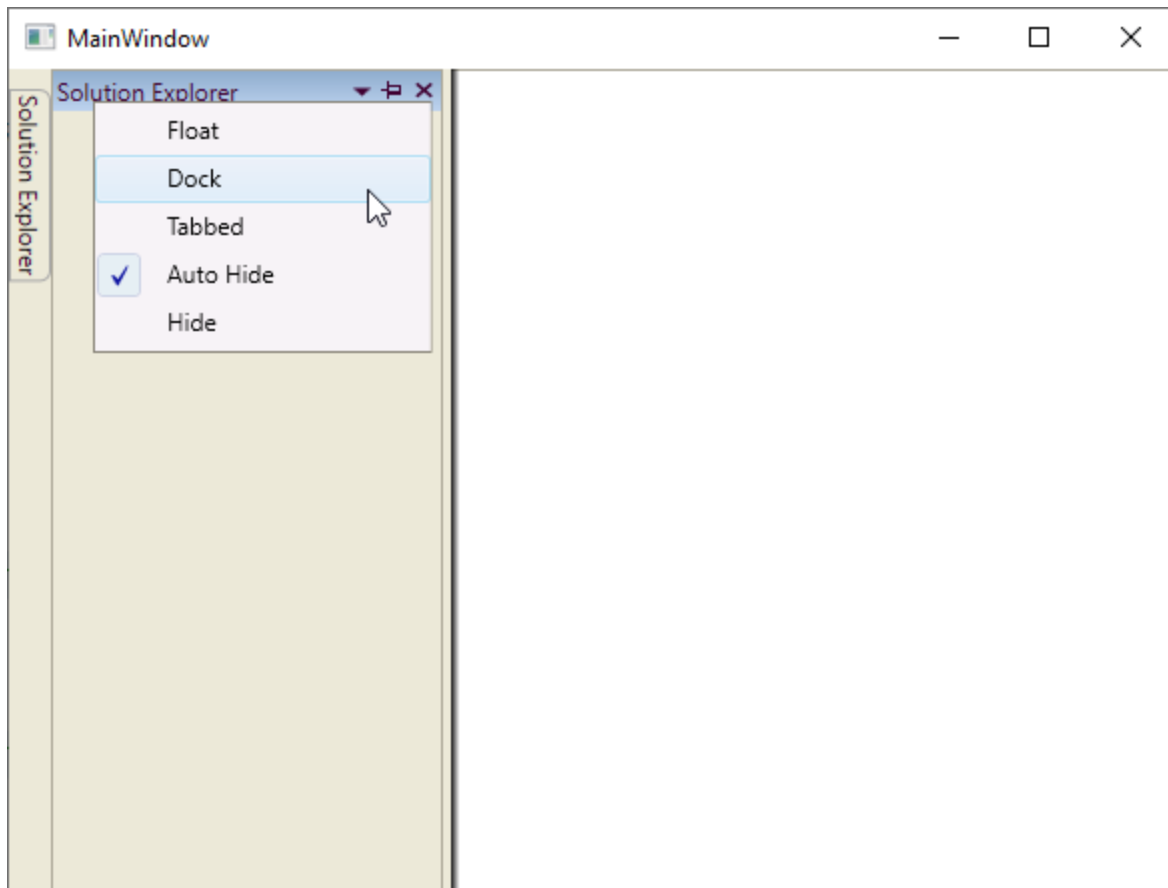
**XML**

```
<syncfusion:DockingManager x:Name="DockingManager1" DockBehavior="VS2010" >
  <ContentControl x:Name="SolutionExplorer"
    syncfusion:DockingManager.DesiredWidthInDockedMode="200"
    syncfusion:DockingManager.Header="Solution Explorer" />
</syncfusion:DockingManager>
```

**C#**

```
DockingManager1.DockBehavior = DockBehavior.VS2010;
```





Hosting a client control between windows

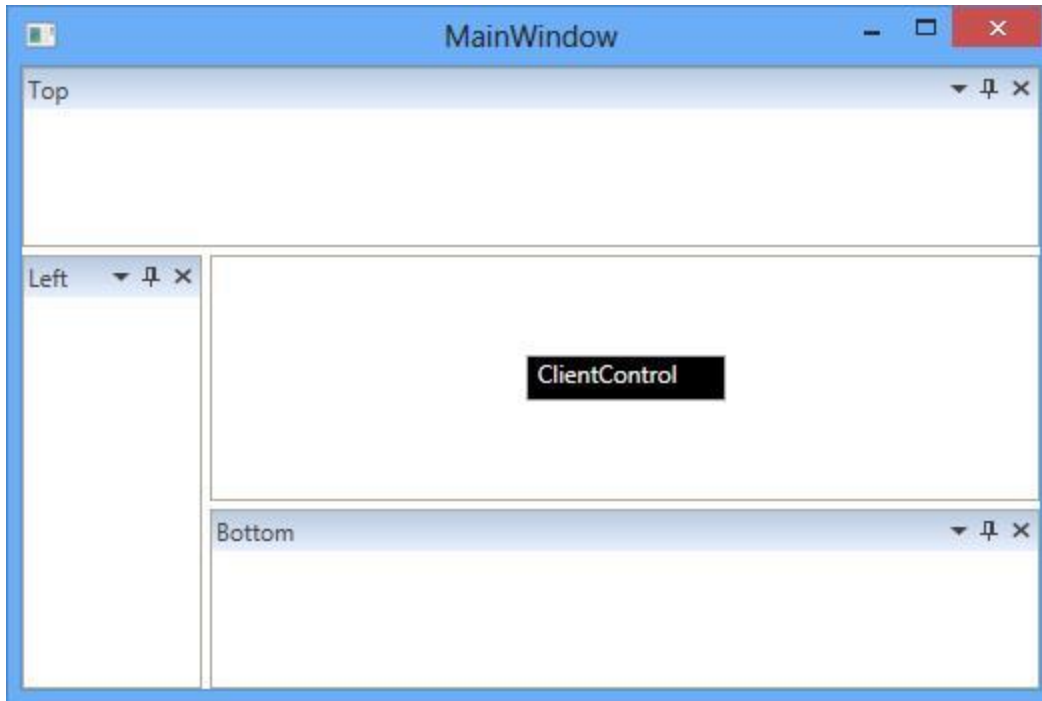
To add a client control in the DockingManager, set an attached property [ClientControl](#).

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="False">
// A client control TextBox has been added.
<syncfusion:DockingManager.ClientControl>
<TextBox x:Name="Text1" Width="100" Height="23" Text=" ClientControl"
Background="Black" Foreground="White"/>
</syncfusion:DockingManager.ClientControl>
<ContentControl syncfusion:DockingManager.Header="Top"
syncfusion:DockingManager.SideInDockedMode="Top"/>
<ContentControl syncfusion:DockingManager.Header="Left"
syncfusion:DockingManager.SideInDockedMode="Left"/>
<ContentControl syncfusion:DockingManager.Header="Bottom"
syncfusion:DockingManager.SideInDockedMode="Bottom"/>
</syncfusion:DockingManager>
```

### C#

```
DockingManager1.ClientControl = new TextBlock() { Text = "ClientControl",
Width = 100, Height = 23, Background = Brushes.Black, Foreground =
Brushes.White };
```



### DockingManager as FlatLayoutControl

The [EnableFlatLayout](#) property of DockingManager is used to disable all the docking functionalities such as drag-and-drop functionality in document windows, resizing the child elements, hiding the dock panel options, closing the child elements from view, and floating the dock windows. If the [EnableFlatLayout](#) property is true, the DockingManager control will act as LayoutControl. The default value of this property is false.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True" EnableFlatLayout="True">
  <ContentControl Name="Output"
syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Bottom"
syncfusion:DockingManager.DesiredHeightInDockedMode="200"
  >
  </ContentControl>
  <ContentControl Name="SolutionExplorer"
syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.DesiredWidthInDockedMode="300"
  >
  </ContentControl>
  <ContentControl Name="Toolbox"
syncfusion:DockingManager.Header="Toolbox"
syncfusion:DockingManager.State="Dock"
HorizontalContentAlignment="Left"
syncfusion:DockingManager.DesiredWidthInDockedMode="250"
  >
  </ContentControl>
  <ContentControl Name="Features"
syncfusion:DockingManager.Header="Features"
```

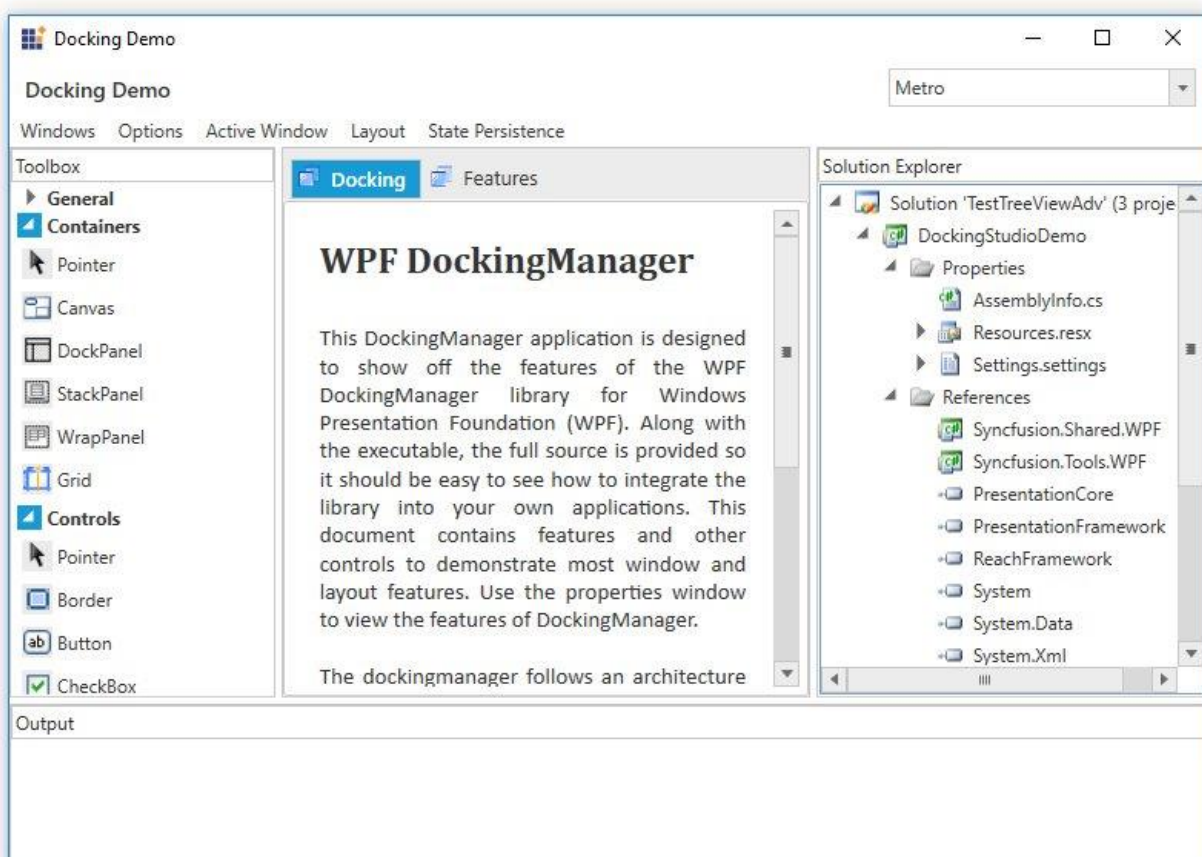
```

syncfusion:DockingManager.State="Document"
>
</ContentControl>
<!-- Docking dock window -->
<ContentControl Name="Docking"
syncfusion:DockingManager.Header="Docking"
syncfusion:DockingManager.State="Document"
>
</ContentControl>
</syncfusion:DockingManager>

```

**C#**

```
DockingManager.EnableFlatLayout = true;
```



## MDI/TDI functionalities in WPF Docking (DockingManager)

The MDI and TDI functionalities are applicable for the Document window in the DockingManager. So Document window can be displayed in both Multiple Document Interface and Tabbed Document Interface.

To change mode for the Document window, set the property [ContainerMode](#) with its respective values.

By default, the document state window is in TDI mode, that display child as tabbed document.

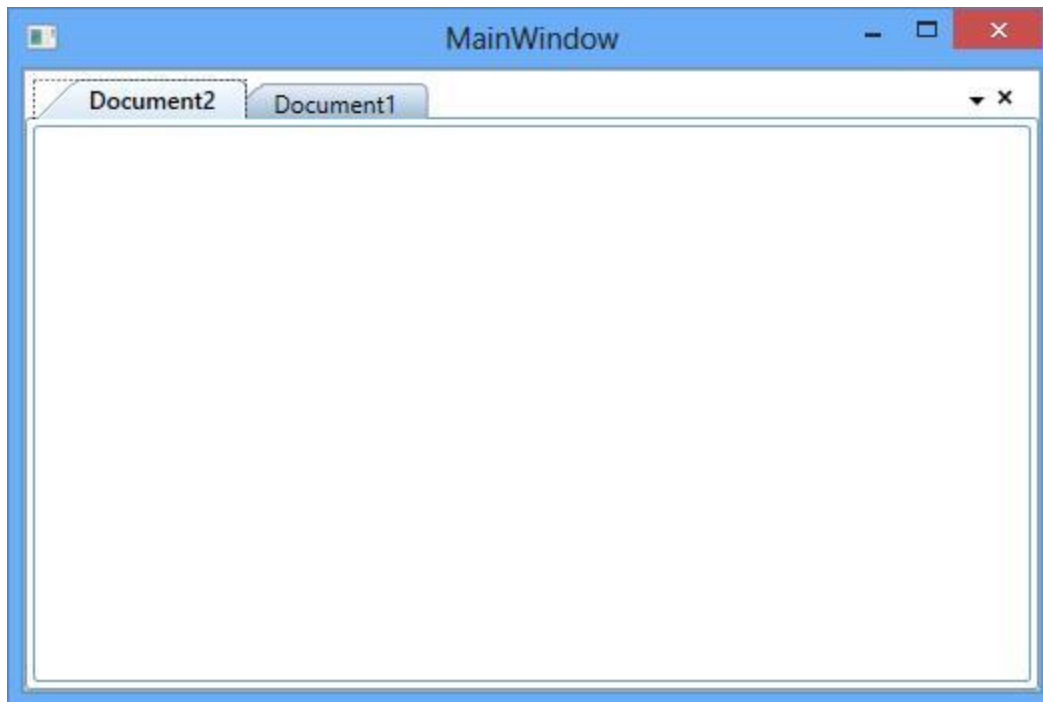
**Note:** When the window state is in MDI, child of docking manager will be hosted inside the DocumentContainer. So, user can customize child using DocumentContainer. Refer to this [documentation](#) for customizing DocumentContainer.

### XML

```
<syncfusion:DockingManager UseDocumentContainer="True" ContainerMode="TDI">
  <ContentControl x:Name="Content1"
    syncfusion:DockingManager.Header="Document1" />
  <ContentControl syncfusion:DockingManager.Header="Document2"
    syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>
```

### C#

```
SyncDockingManager.UseDocumentContainer = true;
SyncDockingManager.ContainerMode = DocumentContainerMode.TDI;
DockingManager.SetState(Document1, DockState.Document);
DockingManager.SetState(Document2, DockState.Document);
```



To make the document child window as MDI document, set the [ContainerMode](#) as MDI

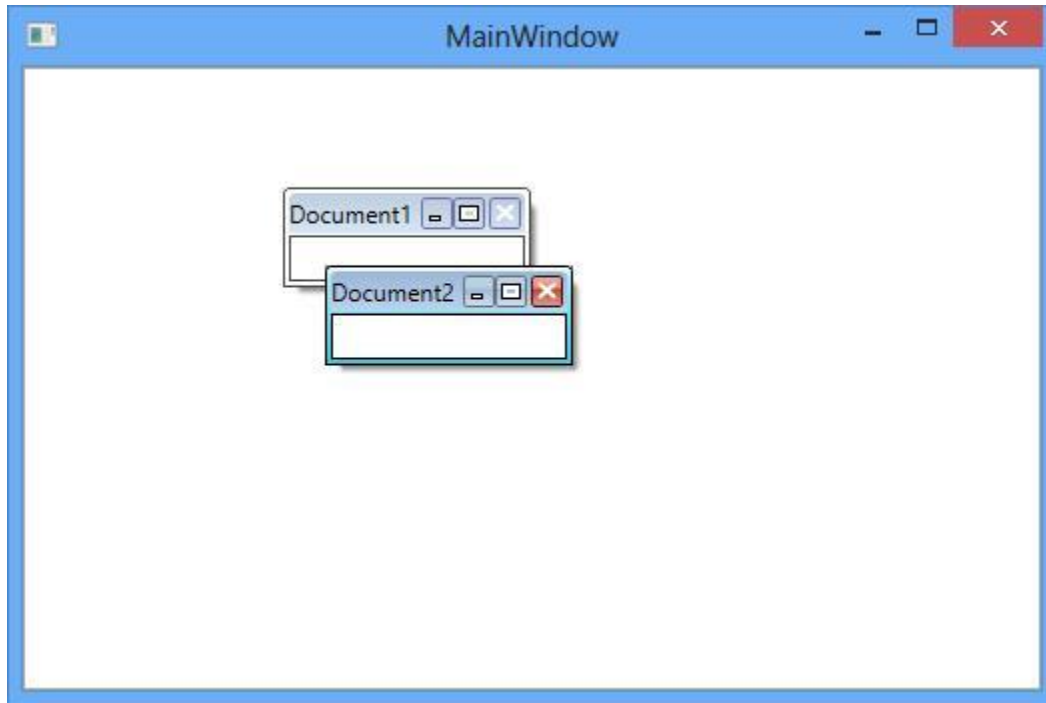
### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
  UseDocumentContainer="True" ContainerMode="MDI">
  <ContentControl x:Name="Content1"
    syncfusion:DockingManager.Header="Document1"
    syncfusion:DockingManager.State="Document" />
  <ContentControl x:Name="Content2"
    syncfusion:DockingManager.Header="Document2"
    syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>
```

```
</syncfusion:DockingManager>
```

**C#**

```
//Tabbed Document Interface.  
SyncDockingManager.ContainerMode = DocumentContainerMode.MDI;
```

**Setting MDI Window state**

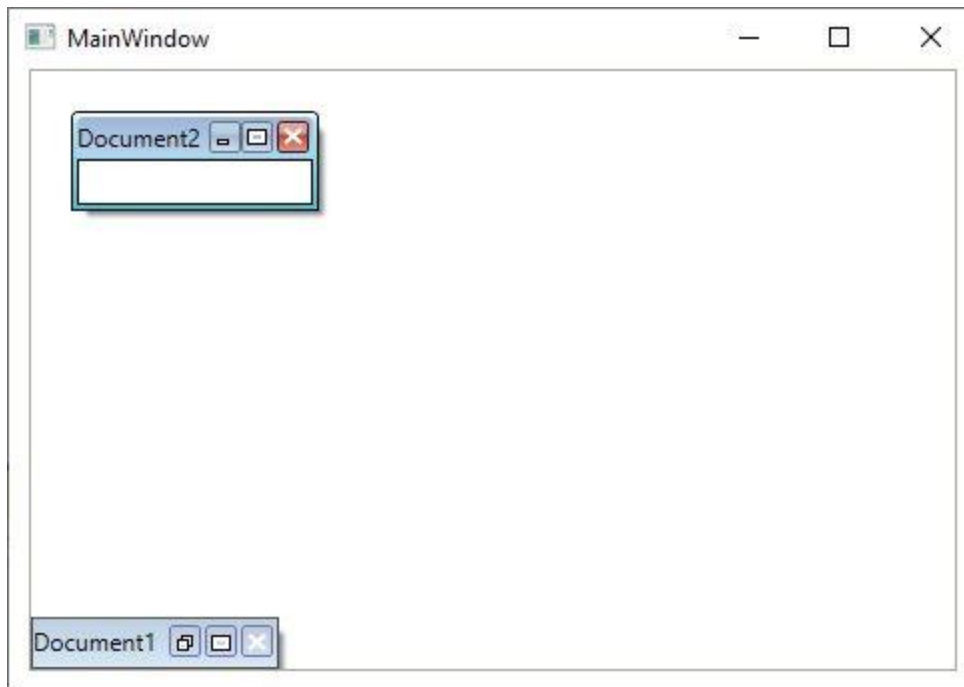
The state of the MDI Window can be set using the [SetMDIWindowState](#) method of DocumentContainer.

**Setting MDI WindowState as Minimized****C#**

```
DocumentContainer.SetMDIWindowState(Content1,MDIWindowState.Minimized);
```

**VB.NET**

```
DocumentContainer.SetMDIWindowState(Content1,MDIWindowState.Minimized)
```



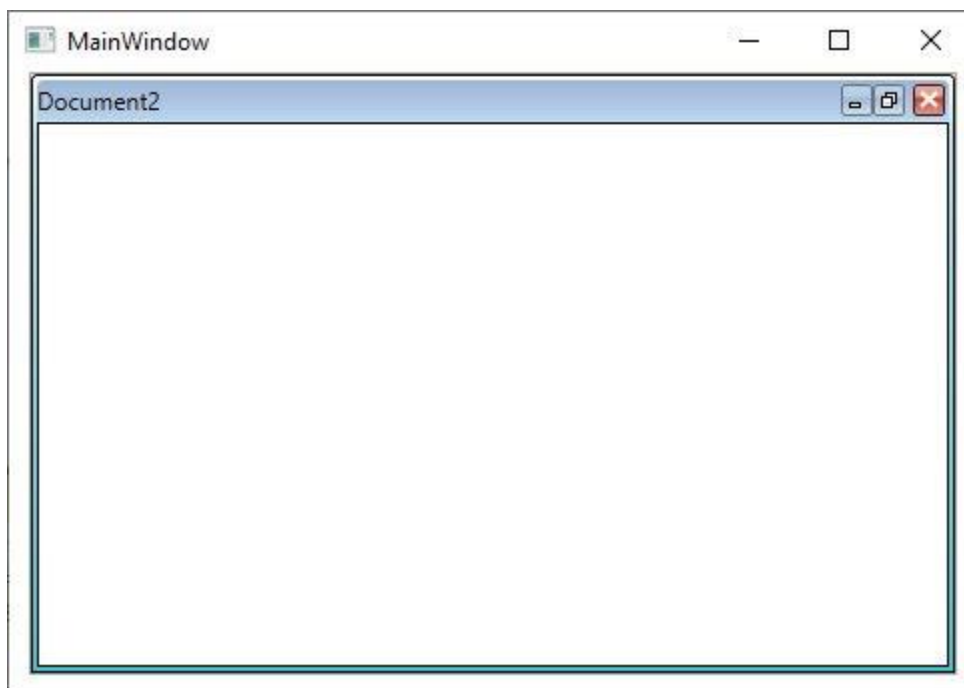
*Setting MDI WindowState as Maximized*

#### **C#**

```
DocumentContainer.SetMDIWindowState(Content1,MDIWindowState.Maximized);
```

#### **VB.NET**

```
DocumentContainer.SetMDIWindowState(Content1,MDIWindowState.Maximized)
```





### Getting state of the MDI window

The state of the MDI window can be detect using the [GetMDIWindowState](#) method of DocumentContainer.

#### C#

```
DocumentContainer.GetMDIWindowState(Content1);
```

#### VB.NET

```
DocumentContainer.GetMDIWindowState(Content1)
```

### Detecting the maximized state of the MDI window

Maximized state of the MDI Container can get by [IsInMDIMaximizedState](#) property of DocumentContainer. The container can be fetched from the DockingManager using the [DocContainer](#) property.

#### C#

```
(DockingManager1.DocContainer as DocumentContainer).IsInMDIMaximizedState = true;
```

#### VB.NET

```
TryCast(DockingManager1.DocContainer, DocumentContainer).IsInMDIMaximizedState = True
```

### Resizing MDI

MDI document window can be able to resize using the navigation arrows. To restrict resizing the MDI document windows, disable the Property [IsAllowMDIResize](#) of the [DocumentContainer](#) that can be get using the [DocContainer](#) property of the DockingManager. By default, its values is **True**.

#### C#

```
(DockingManager1.DocContainer as DocumentContainer).IsAllowMDIResize = false;
```

#### VB.NET

```
TryCast(DockingManager1.DocContainer, DocumentContainer).IsAllowMDIResize = False
```

### Closing TDI tab items on mouse middle click

You can close the document windows by clicking the mouse middle button on the document header. You can enable it by using the [CloseTabOnMiddleClick](#) property as true and the [ContainerMode](#) is set to TDI and [CanClose](#) property as true. The default value of [CloseTabOnMiddleClick](#) property is false.

#### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
    UseDocumentContainer="True"
    ContainerMode="TDI"
    CloseTabOnMiddleClick="True"
    CanClose="True">
```

```
CloseTabOnMiddleClick="True">
<ContentControl x:Name="Content1"
syncfusion:DockingManager.Header="Document1"
syncfusion:DockingManager.State="Document"
syncfusion:DockingManager.CanClose="True"/>
</syncfusion:DockingManager>
```

### C#

```
this.DockingManager1.CloseTabOnMiddleClick = true;
```

### Different Keyboard Navigation Modes

DockingManager allows to navigate between children (Both TDI and MDI) windows easily using the keyboard keys with combination of **CTRL + TAB** in five different modes by [SwitchMode](#) property of the DockingManager.

There are five switch modes.

- Immediate
- List
- QuickTabs
- VS2005
- Vista Flip

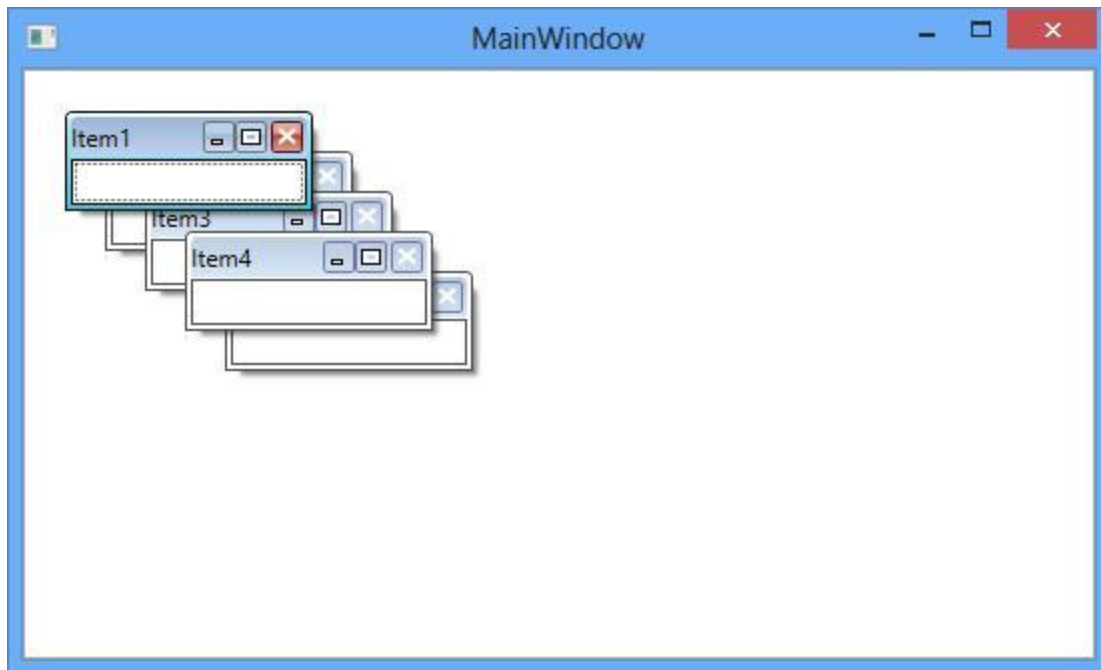
*Immediate – Switch the MDI document windows immediately.*

### C#

```
DockingManager1.SwitchMode =SwitchMode.Immediate;
```

### VB.NET

```
DockingManager1.SwitchMode =SwitchMode.Immediate
```



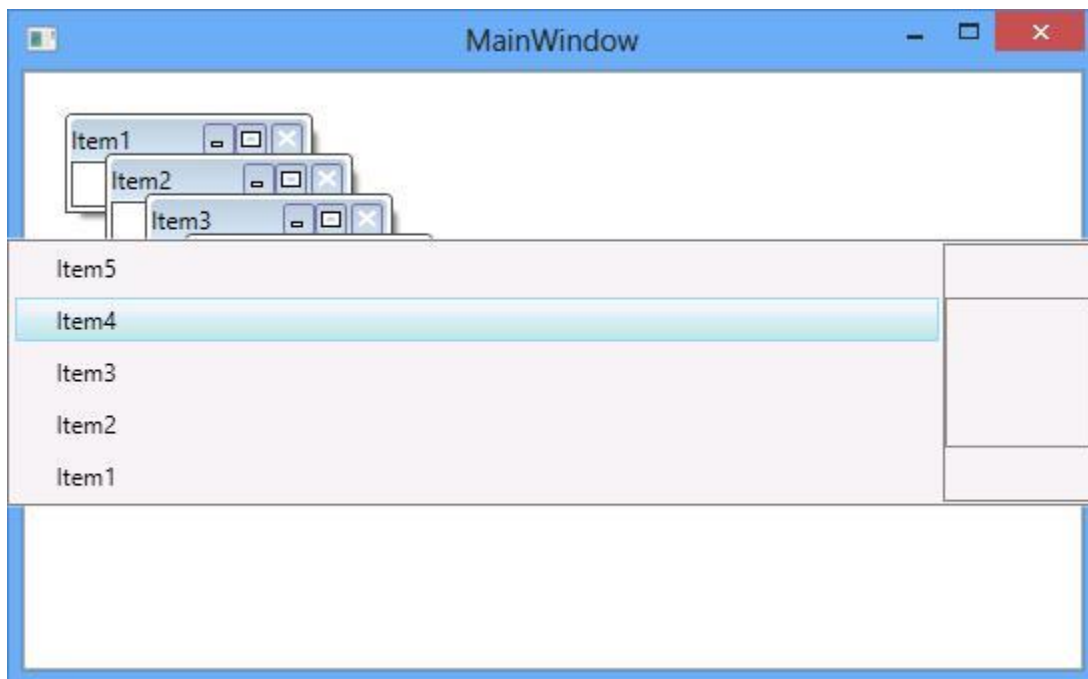
*List – Switch the MDI document windows in list format.*

#### **C#**

```
DockingManager1.SwitchMode = SwitchMode.List;
```

#### **VB.NET**

```
DockingManager1.SwitchMode = SwitchMode.List
```

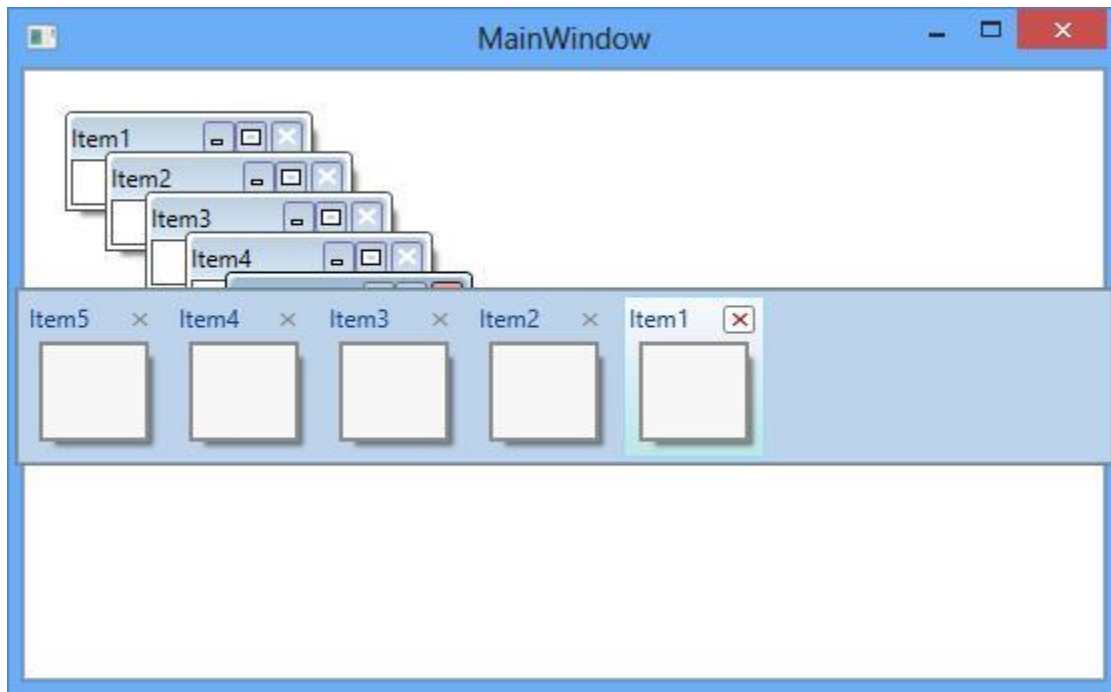


*QuickTabs***C#**

```
DockingManager1.SwitchMode = SwitchMode.QuickTabs;
```

**VB.NET**

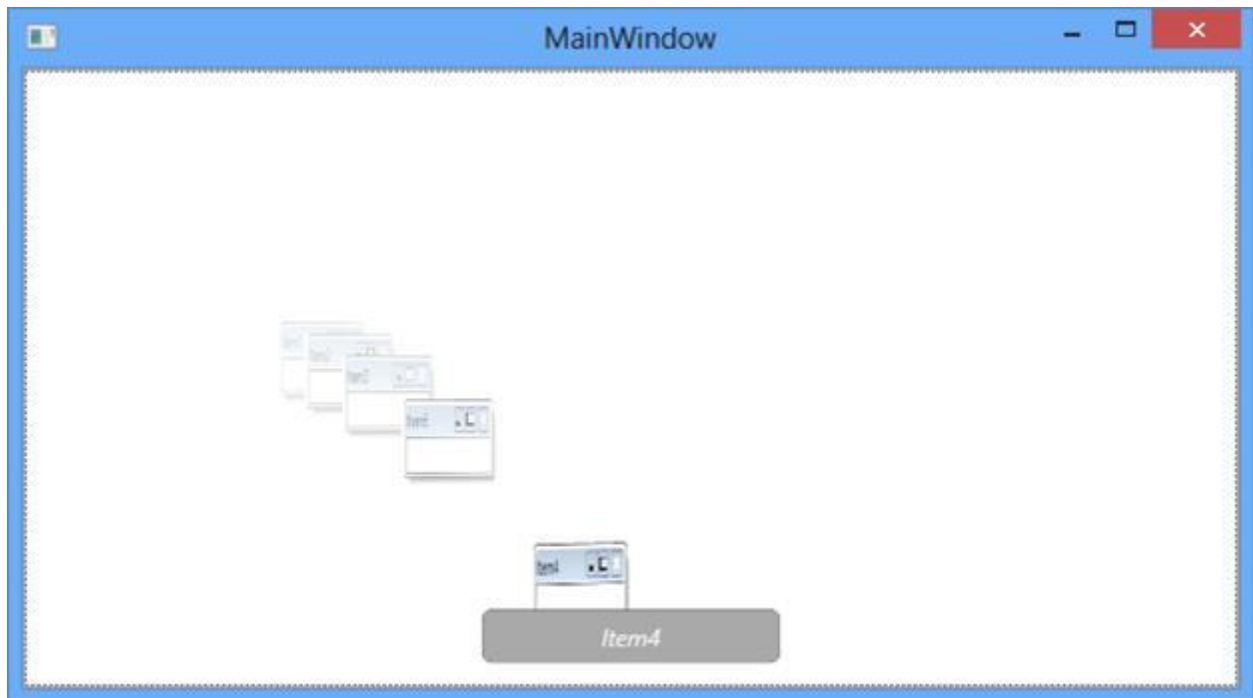
```
DockingManager1.SwitchMode =SwitchMode.QuickTabs
```

*VistaFlip***C#**

```
DockingManager1.SwitchMode = SwitchMode.VistaFlip;
```

**VB.NET**

```
DockingManager1.SwitchMode =SwitchMode.VistaFlip
```



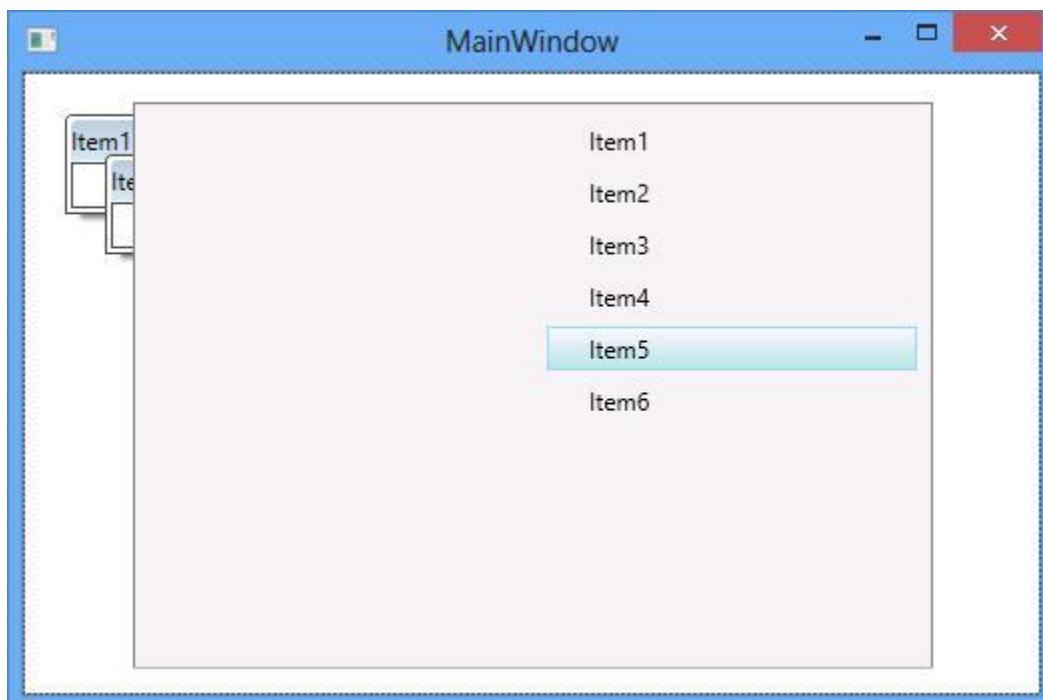
VS2005

### **C#**

```
DockingManager1.SwitchMode = SwitchMode.VS2005;
```

### **VB.NET**

```
DockingManager1.SwitchMode =SwitchMode.VS2005
```



### Setting MDI Layout

DockingManager allows to set the different layout for the MDI windows with the different [MDILayout](#) values such as [Horizontal](#), [Vertical](#) and [Cascade](#) layout through the [SetLayout](#) method of DocumentContainer.

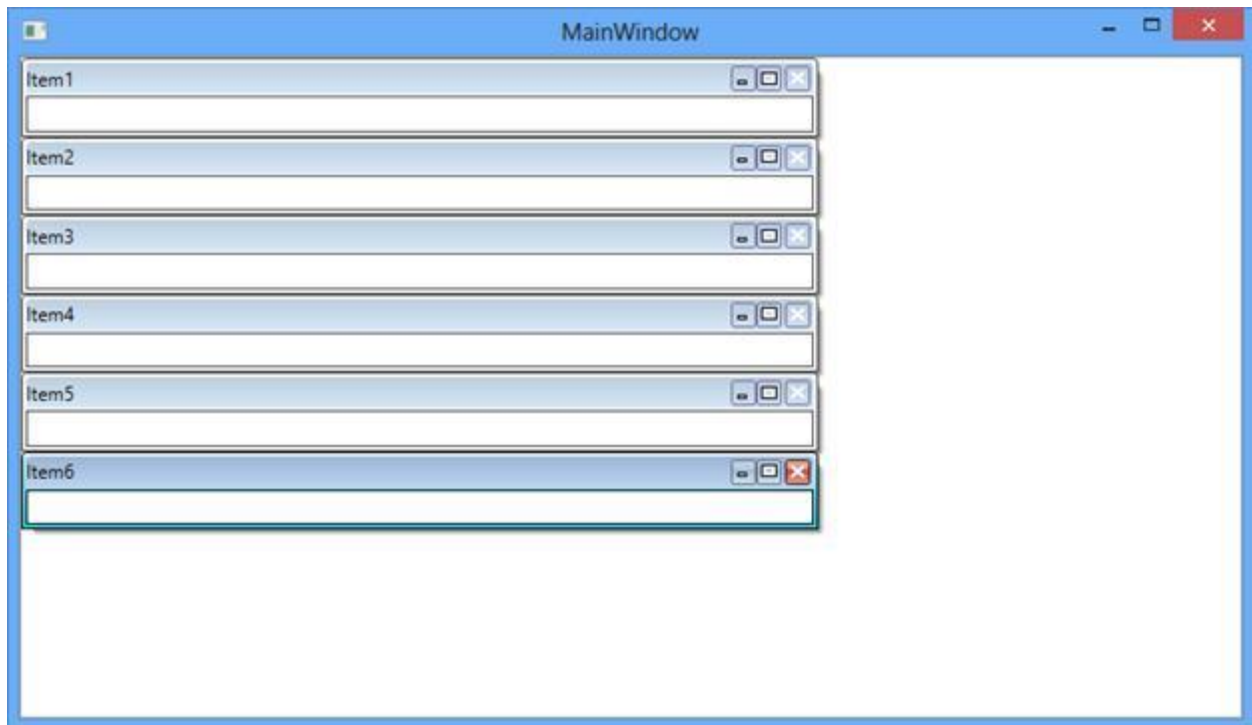
**Horizontal** - Arranges the MDI windows horizontally.

#### C#

```
void DocumentContainer_Loaded(object sender, RoutedEventArgs e)
{
    (DockingManager1.DocContainer as
    DocumentContainer).SetLayout(MDILayout.Horizontal);
}
```

#### VB.NET

```
Private Sub DocumentContainer_Loaded(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    TryCast(DockingManager1.DocContainer,
    DocumentContainer).SetLayout(MDILayout.Horizontal)
End Sub
```



**Vertical** – Arranges the MDI windows vertically.

#### C#

```
void DocumentContainer_Loaded(object sender, RoutedEventArgs e)
{
    (DockingManager1.DocContainer as
    DocumentContainer).SetLayout(MDILayout.Vertical);
}
```

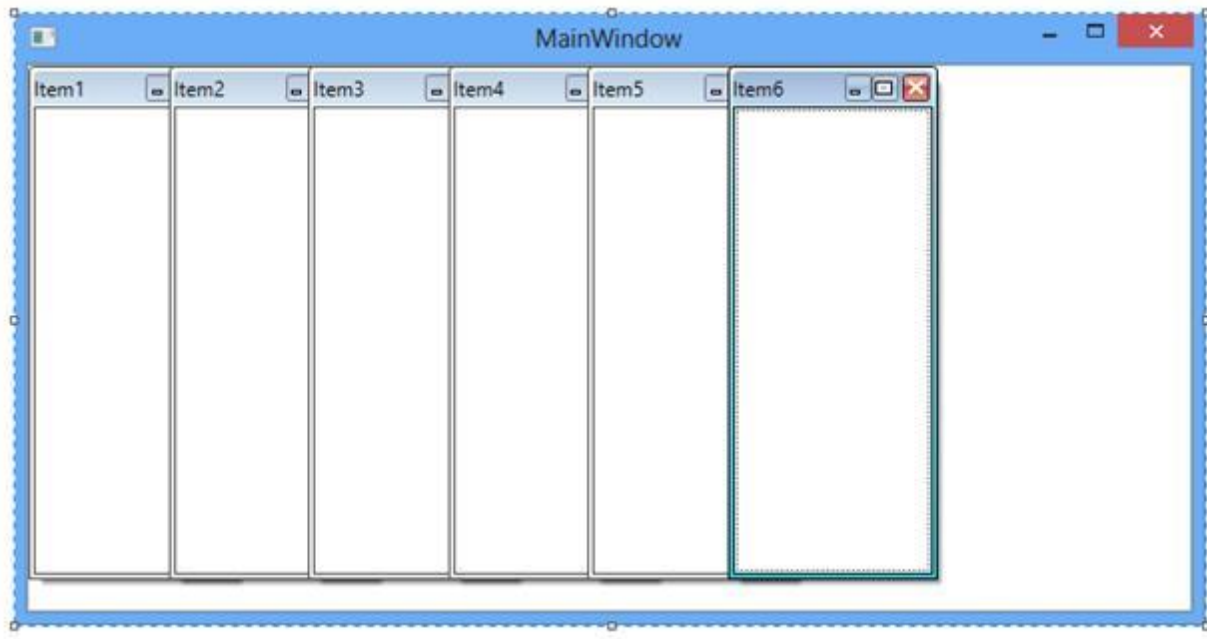
```
}

```

**VB.NET**

```
Private Sub DocumentContainer_Loaded(ByVal sender As Object, ByVal e As
RoutedEventArgs)
TryCast(DockingManager1.DocContainer,
DocumentContainer).SetLayout(MDILayout.Vertical)
End Sub

```



**Cascade** - Arranges the layout in a cascade manner.

**C#**

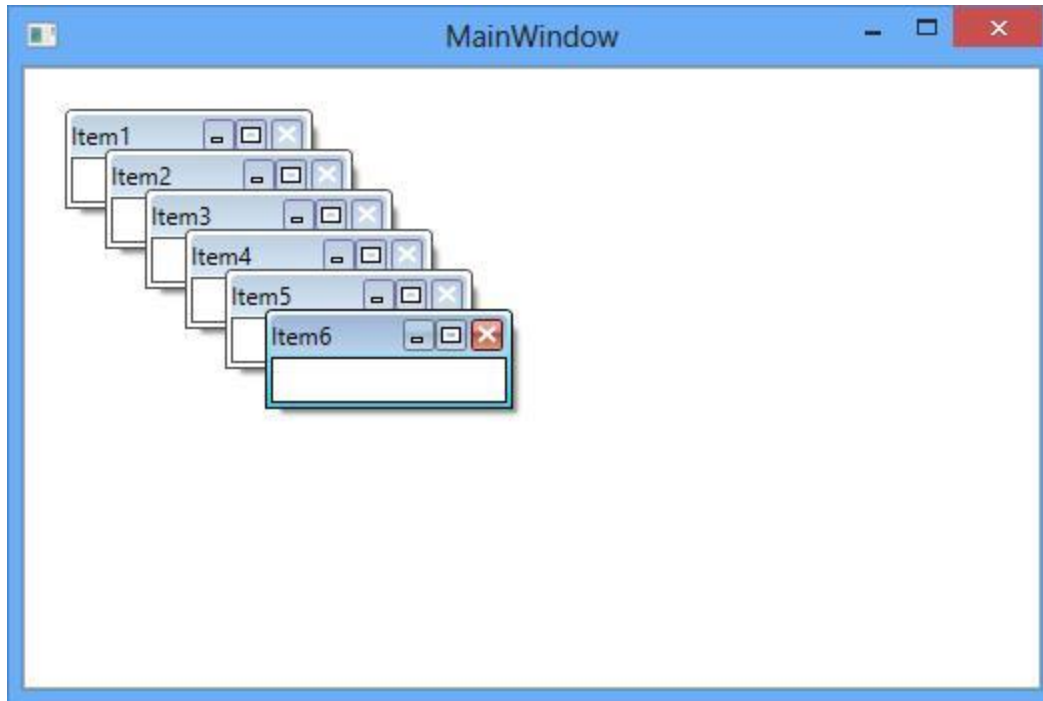
```
void DocumentContainer_Loaded(object sender, RoutedEventArgs e)
{
(DockingManager1.DocContainer as
DocumentContainer).SetLayout(MDILayout.Vertical);
}

```

**VB.NET**

```
Private Sub DocumentContainer_Loaded(ByVal sender As Object, ByVal e As
RoutedEventArgs)
TryCast(DockingManager1.DocContainer,
DocumentContainer).SetLayout(MDILayout.Vertical)
End Sub

```



### Closing a MDI Windows

To enable or disable closing functionality of the MDI windows, set [CanClose](#) an attached property of DockingManager with its respective values. By default, its value is `True`

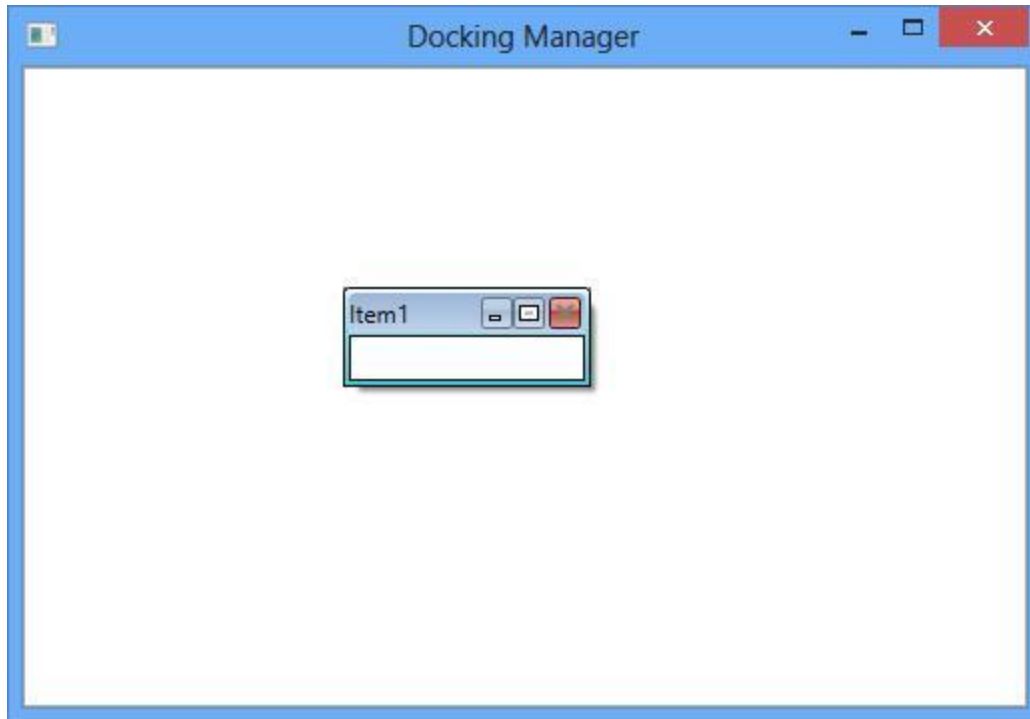
### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
    UseDocumentContainer="True" ContainerMode="MDI">
    <ContentControl x:Name="Content1" syncfusion:DockingManager.Header="Item1"
        syncfusion:DockingManager.State="Document"
        syncfusion:DockingManager.CanClose="False"/>
</syncfusion:DockingManager>
```

### C#

```
DockingManager.SetCanClose(Content1, false);
```





### Indexing an Item in TDI

A document window can be placed at different index position using the [SetTDIIndex](#) method of the `TDILayoutPanel`.

### XML

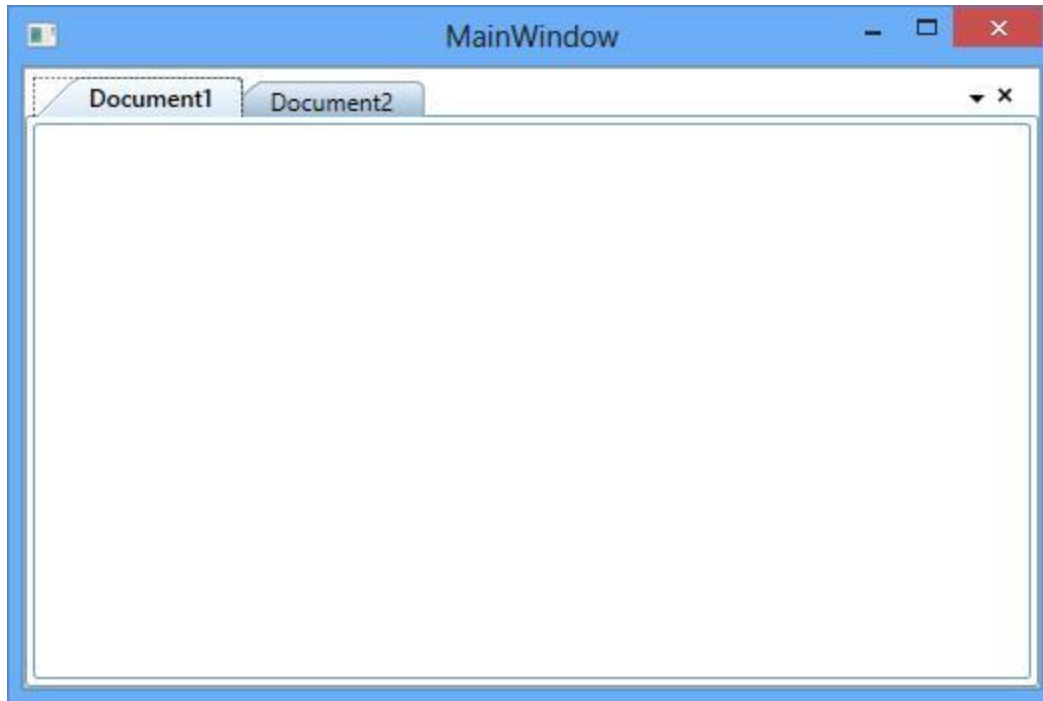
```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True">
  <ContentControl x:Name="Content1"
syncfusion:DockingManager.Header="Document1"
syncfusion:DockingManager.State="Document"/>
  <ContentControl x:Name="Content2"
syncfusion:DockingManager.Header="Document2"
syncfusion:DockingManager.State="Document"/>
</syncfusion:DockingManager>
```

### C#

```
TDILayoutPanel.SetTDIIndex(Content1, 0);
```

### VB.NET

```
TDILayoutPanel.SetTDIIndex(Content1, 0)
```



### Drag / Drop support in TDI

The TDI document index can be changed by dragging and dropping it like Visual Studio. This functionality can be enabled or disabled through the property [IsTDIDragDropEnabled](#) of DockingManager.

### XML

```
<syncfusion:DockingManager UseDocumentContainer="True"
IsTDIDragDropEnabled="True" >
<ContentControl syncfusion:DockingManager.Header="Document1"
syncfusion:DockingManager.State="Document"/>
<ContentControl syncfusion:DockingManager.Header="Document2"
syncfusion:DockingManager.State="Document"/>
<ContentControl syncfusion:DockingManager.Header="Document3"
syncfusion:DockingManager.State="Document"/>
</syncfusion:DockingManager>
```

### C#

```
SyncDockingManager.IsTDIDragDropEnabled = true;
```

### Rearrange position of document items with auto scrolling

You can easily move or rearrange TDI document items when there are several document items by setting the `EnableAutoScroll` property value as `true`. Drag the required document item to overflow button (with three dots) or tab scroll buttons to autoscroll.

The default value of `EnableAutoScroll` property is `false`.

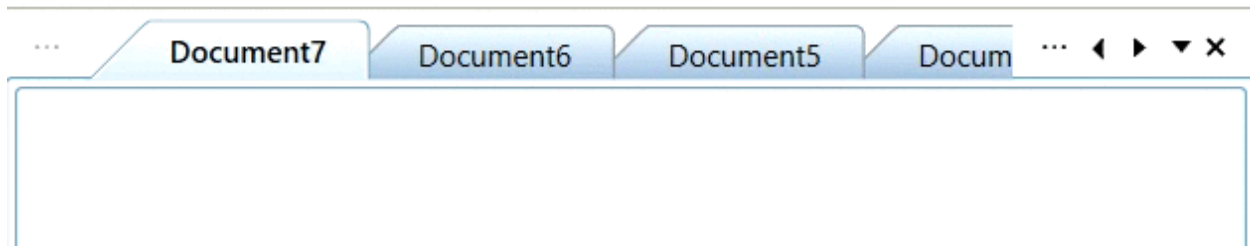
### XML

```
<syncfusion:DockingManager x:Name="dockingManager" EnableAutoScroll="True"
UseDocumentContainer="True" >
```

```
<ContentControl syncfusion:DockingManager.Header="Document1"
syncfusion:DockingManager.State="Document" />
<ContentControl syncfusion:DockingManager.Header="Document2"
syncfusion:DockingManager.State="Document" />
<ContentControl syncfusion:DockingManager.Header="Document3"
syncfusion:DockingManager.State="Document" />
<ContentControl syncfusion:DockingManager.Header="Document4"
syncfusion:DockingManager.State="Document" />
<ContentControl syncfusion:DockingManager.Header="Document5"
syncfusion:DockingManager.State="Document" />
<ContentControl syncfusion:DockingManager.Header="Document6"
syncfusion:DockingManager.State="Document" />
<ContentControl syncfusion:DockingManager.Header="Document7"
syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>
```

**C#**

```
DockingManager dockingManager = new DockingManager();
dockingManager.UseDocumentContainer = true;
dockingManager.EnableAutoScroll = true;
```

**TDI window's order changed notification**

You will be notified when the TDI item's order is changed by using the [DocumentTabOrderChanged](#) event. You can get the order changed TDI window with its old and new index values by using the [SourceTabItem](#), [OldIndex](#) and [NewIndex](#) properties. You can also get old and new tab group of the order changed item by using the [SourceTabGroup](#) and [TargetTabGroup](#) properties.

**Note:** You will also be notified when creating a horizontal or vertical tab groups using [ContextMenu](#) or move the tab document to previous or next tab groups and when tab the float window to Document Window or create new tabgroup using [DragProviders](#).

**XML**

```
<syncfusion:DockingManager
DocumentTabOrderChanged="DockingManager_DocumentTabOrderChanged"
UseDocumentContainer="UseDocumentContainer"
Name="dockingManager">
<ContentControl syncfusion:DockingManager.Header="Document1"
syncfusion:DockingManager.State="Document"/>
<ContentControl syncfusion:DockingManager.Header="Document2"
syncfusion:DockingManager.State="Document"/>
<ContentControl syncfusion:DockingManager.Header="Document3"
syncfusion:DockingManager.State="Document"/>
</syncfusion:DockingManager>
```

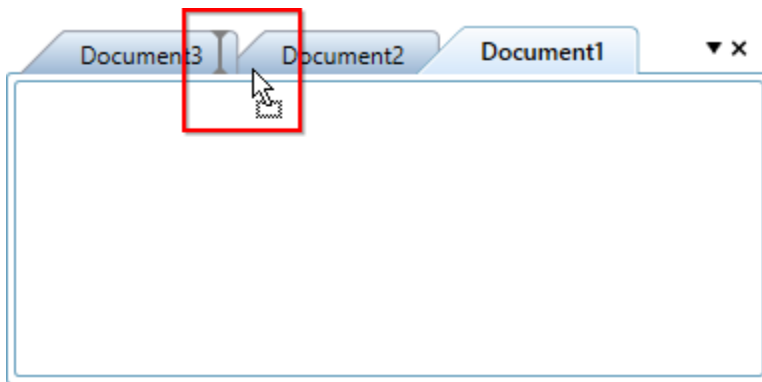
**C#**

```
documentcontainer1.UseDocumentContainer = true;  
dockingManager.DocumentTabOrderChanged +=  
    DockingManager.DocumentTabOrderChanged;
```

You can handle the event as follows,

**C#**

```
private void DockingManager_DocumentTabOrderChanged(object sender,  
    DocumentTabOrderChangedEventArgs e)  
{  
    var drag_Drop_Item = e.TargetTabGroup;  
    //Get the old and new index of the SourceTabItem  
    var oldIndex = e.OldIndex;  
    var newIndex = e.NewIndex;  
    //Get the old and new tab group of the SourceTabItem  
    var sourceTabGroup = e.SourceTabGroup;  
    var targetTabGroup = e.TargetTabGroup;  
}
```



**Note:** [View Sample in GitHub](#)

Restrict TDI window reordering

If you want to restrict the user to reordering the TDI window by drag and drop operation, use the [DocumentTabOrderChanging](#) event and set `Cancel` property value as `true`.

**Note:** You can handle the document window will be floated or not by `DockStateChangingEvent` instead of `DocumentTabOrderChanging`.

**XML**

```
<syncfusion:DockingManager  
    DocumentTabOrderChanging="DockingManager_DocumentTabOrderChanging"  
    UseDocumentContainer="True"  
    Name="dockingManager">  
    <ContentControl syncfusion:DockingManager.Header="Document1"  
        syncfusion:DockingManager.State="Document"/>  
    <ContentControl syncfusion:DockingManager.Header="Document2"  
        syncfusion:DockingManager.State="Document"/>  
    <ContentControl syncfusion:DockingManager.Header="Document3"  
        syncfusion:DockingManager.State="Document"/>  
</syncfusion:DockingManager>
```

```
</syncfusion:DockingManager>
```

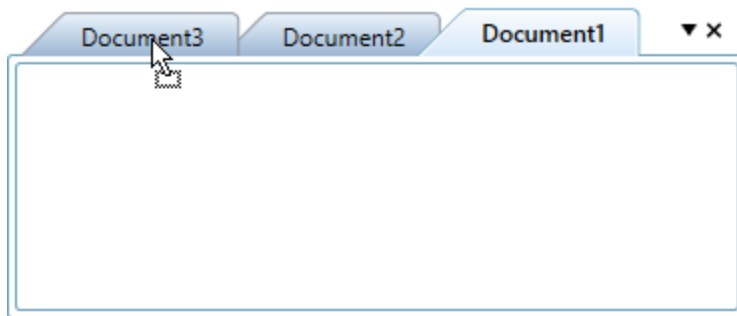
**C#**

```
dockingManager.UseDocumentContainer = true;
dockingManager.DocumentTabOrderChanging +=
    DockingManager_DocumentTabOrderChanging;
```

You can handle the event as follows,

**C#**

```
private void DockingManager_DocumentTabOrderChanging(object sender,
    DocumentTabOrderChangingEventArgs e)
{
    // Restrict the TDI window re-ordering
    e.Cancel = true;
}
```



**Note:** [View Sample in GitHub](#)

## Customizing Close Menu

Menu items like **Close**, **CloseAll** and **CloseAllButThis** are available for the document window when two or more documents used in the DockingManager. To collapse the visibility of these menu item, set the property [ShowCloseMenuItem](#), [ShowCloseAllMenuItem](#) and [ShowCloseAllButThisMenuItem](#) as **False**.

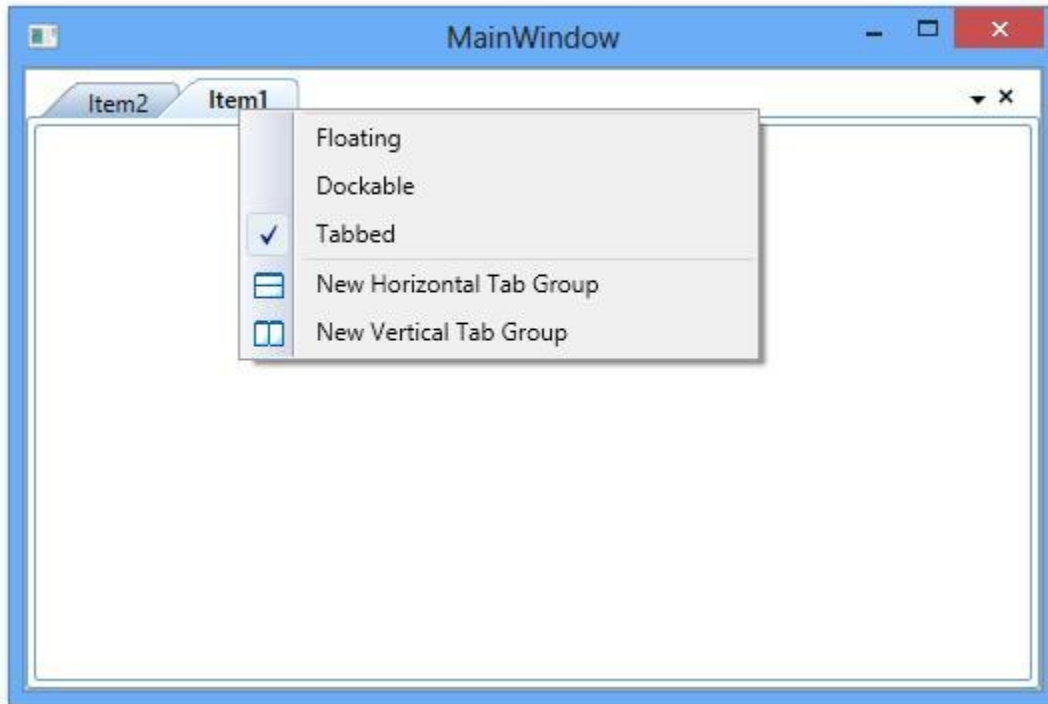
**XML**

```
<ContentControl syncfusion:DockingManager.Header="Item1"
syncfusion:DockingManager.State="Document"
syncfusion:DockingManager.ShowCloseMenuItem="False"
syncfusion:DockingManager.ShowCloseAllMenuItem="False"
syncfusion:DockingManager.ShowCloseAllButThisMenuItem="False"/>
<ContentControl syncfusion:DockingManager.Header="Item2"
syncfusion:DockingManager.State="Document"
syncfusion:DockingManager.ShowCloseMenuItem="False"
syncfusion:DockingManager.ShowCloseAllMenuItem="False"
syncfusion:DockingManager.ShowCloseAllButThisMenuItem="False"/>
```

**C#**

```
//Closing Customization
DockingManager.SetShowCloseMenuItem(Item1, false);
```

```
DockingManager.SetShowCloseAllMenuItem(Item1, false);  
DockingManager.SetShowCloseAllButThisMenuItem(Item1, false);  
DockingManager.SetShowCloseMenuItem(Item2, false);  
DockingManager.SetShowCloseAllMenuItem(Item2, false);  
DockingManager.SetShowCloseAllButThisMenuItem(Item2, false);
```



### Creating Document Tab Group

TDI document can be grouped like VisualStudio. It can be grouped by drag and Drop and also using the options in context menu items.

#### *TabGroup creation using ContextMenu option*

In DockingManager, new tab group can be created at horizontal or vertical side in the document area using **ContextMenu** option.

### Creating Vertical Tab Group

To create a vertical tab group in the Tabbed document, select the "New Vertical Tab Group" context menu item and also it can be created programmatically by calling the method [CreateVerticalTabGroup](#) of the DocumentContainer.

### **C#**

```
(DockingManager1.DocContainer as  
DocumentContainer).CreateVerticalTabGroup(Content1);
```

### **VB.NET**

```
TryCast(DockingManager1.DocContainer,  
DocumentContainer).CreateVerticalTabGroup(Content1)
```

### Creating Horizontal Tab Group

To create a horizontal tab group in the Tabbed document, select the "New Horizontal Tab Group" context menu item and also it can be created programmatically by calling the method [CreateHorizontalTabGroup](#) of the DocumentContainer.

#### C#

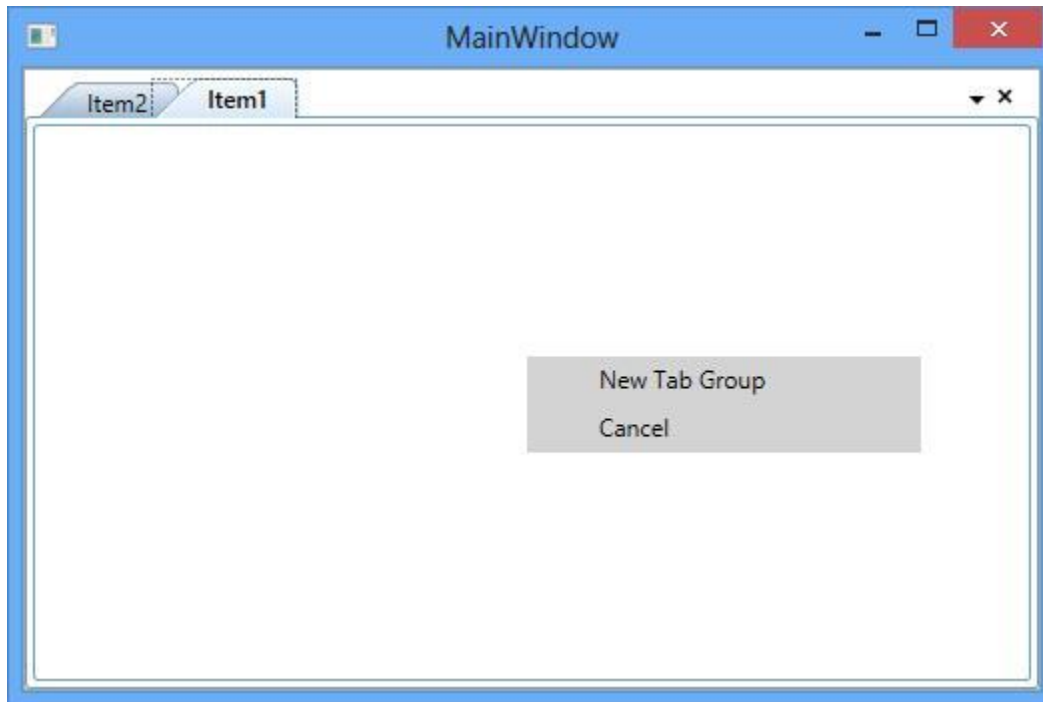
```
(DockingManager1.DocContainer as  
DocumentContainer).CreateHorizontalTabGroup(Content1);
```

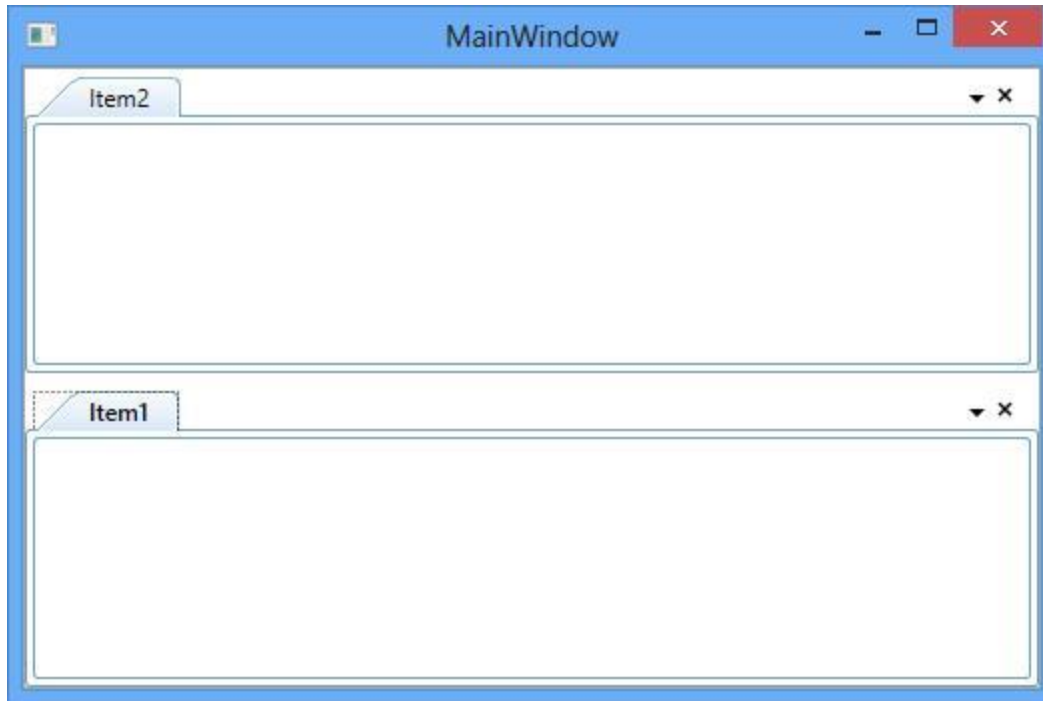
#### VB.NET

```
TryCast(DockingManager1.DocContainer,  
DocumentContainer).CreateHorizontalTabGroup(Content1)
```

### Adding Tab in a Group

In TDI document, new tab group can be created by dragging the tab item into the Document area and click the "New Tab Group" menu item from context menu item.





### Disable TabGroups

Vertical and Horizontal Tab Grouping feature can be enabled or disabled using the property [TabGroupEnabled](#) in DockingManager.

To disabling Tab Groups, set [TabGroupEnabled](#) as `False`. So it does not display "New Horizontal Tab Group" and "New Vertical Tab Group" context menu items even when [ShowHorizontalTabGroupMenuItem](#) is true. Drag and drop support to create new tab group is also restricted.

### XML

```
<syncfusion:DockingManager TabGroupEnabled="False" />
```

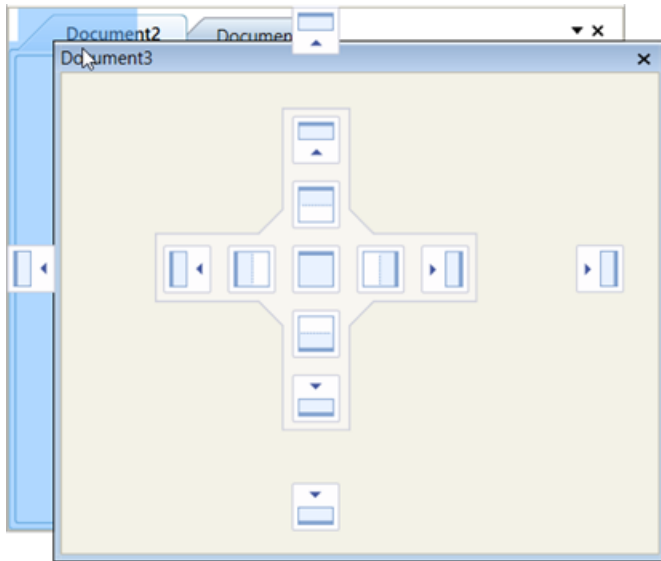
### C#

```
SyncDockingManager.TabGroupEnabled = false;
```

### VS2010 Behavior of TDI

TDI document of DockingManager can be changed to Float while dragging its TDI header. This functionality can be enabled or disabled using the property [IsVs2010DraggingEnabled](#). By default, its value is `True`.



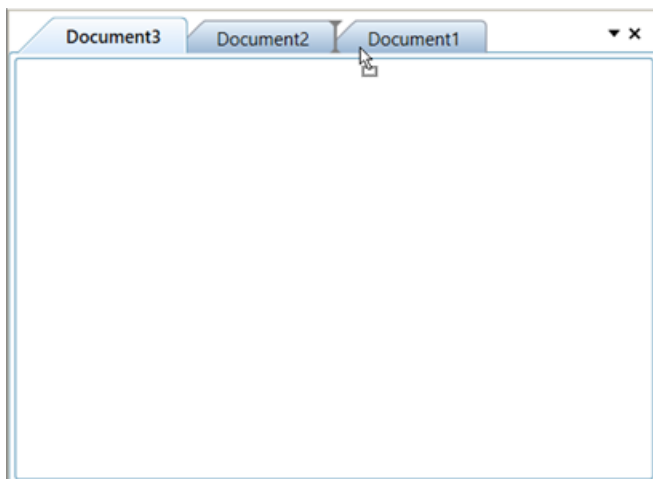


### XML

```
<syncfusion:DockingManager Name="SyncDockingManager"
    UseDocumentContainer="True" IsVS2010DraggingEnabled="False">
    <ContentControl syncfusion:DockingManager.Header="Document1"
        syncfusion:DockingManager.State="Document" />
    <ContentControl syncfusion:DockingManager.Header="Document2"
        syncfusion:DockingManager.State="Document" />
    <ContentControl syncfusion:DockingManager.Header="Document3"
        syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>
```

### C#

```
SyncDockingManager.IsVS2010DraggingEnabled = false;
```



### TDI Header Renaming Support

To enable the functionality of editing the TDI document header when double click on document header at runtime, set the property [EnableDocumentTabHeaderEdit](#) of the DockingManager as **True**. By default, its value is **False**.

#### XML

```
<syncfusion:DockingManager UseDocumentContainer="True"
EnableDocumentTabHeaderEdit="True">
```

#### C#

```
SyncDockingManager.EnableDocumentTabHeaderEdit = true;
```

### Hiding TDI Header

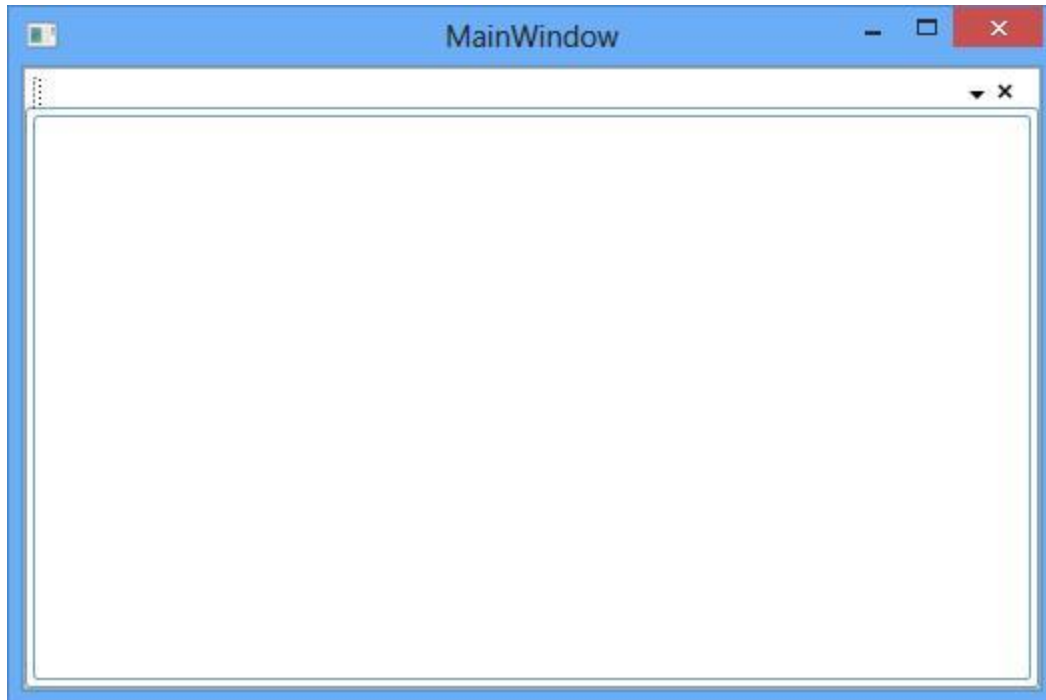
To hide the TDI document header when a single document child present in a DockingManager set the property [HideTDIHeaderOnSingleChild](#) as **True**. By default its value is **False**.

#### XML

```
<syncfusion:DockingManager UseDocumentContainer="True"
HideTDIHeaderOnSingleChild="True">
<ContentControl syncfusion:DockingManager.Header="Document1"
syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>
```

#### C#

```
SyncDockingManager.HideTDIHeaderOnSingleChild = true;
```



### Add New button in Header Panel

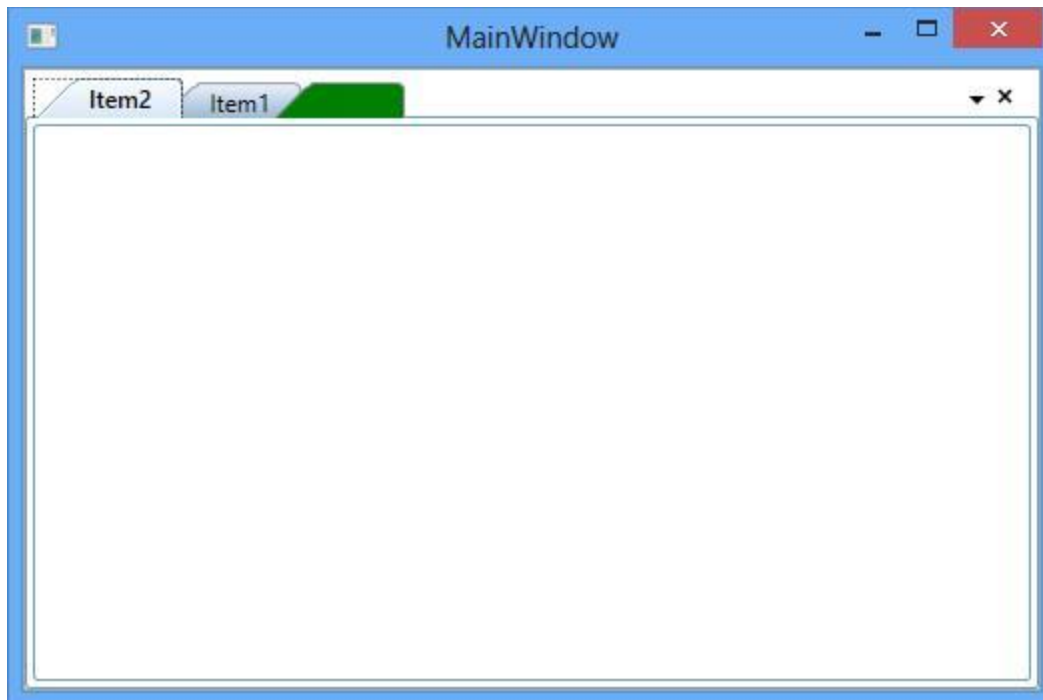
In DockingManager, the New button can be added in the Document state windows using the [IsNewButtonEnabled](#) property of the DocumentTabControl. To achieve this, DocumentTabControl must be fetched from the DockingManager.

#### C#

```
DocumentTabControl tab = VisualUtils.FindDescendant(DockingManager1, typeof
(DocumentTabControl)) as DocumentTabControl;
if (tab != null)
{
    tab.IsNewButtonEnabled = true;
    tab.NewButtonBackground = Brushes.Green;
}
```

#### VB.NET

```
Dim tab As DocumentTabControl =
TryCast(VisualUtils.FindDescendant(DockingManager1, GetType(DocumentTabContro
l)), DocumentTabControl)
If tab IsNot Nothing Then
    tab.IsNewButtonEnabled = True
    tab.NewButtonBackground = Brushes.Green
End If
```



### Pin and Unpin tab items

The following section explains the Pin and Unpin tab items in DockingManager.

#### Enabling/disabling pinning behavior

The [AllowPin](#) attached property of DocumentContainer decides whether the document tab item could be pinnable or not. The corresponding tab item will be pinned when the property [AllowPin](#) is true. When

this property is set to false, the pin and unpin behaviors of document tab item will be disabled. The default value of the [AllowPin](#) property is false.

### XML

```
<syncfusion:DockingManager Grid.Row="1" x:Name="DockingManager"
UseDocumentContainer="True">
  <!-- Output dock window -->
  <ContentControl Name="Output"
syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Bottom"
syncfusion:DockingManager.DesiredHeightInDockedMode="200"
  >
  </ContentControl>
  <!-- SolutionExplorer dock window -->
  <ContentControl Name="SolutionExplorer"
syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.DesiredWidthInDockedMode="300"
  >
  </ContentControl>
  <!-- ClassView dock window -->
  <!-- Toolbox dock window -->
  <ContentControl Name="Toolbox"
syncfusion:DockingManager.Header="Toolbox"
syncfusion:DockingManager.State="Dock"
HorizontalContentAlignment="Left"
syncfusion:DockingManager.DesiredWidthInDockedMode="250"
  >
  </ContentControl>
  <!-- Features dock window -->
  <ContentControl Name="Features"
syncfusion:DockingManager.Header="Features"
syncfusion:DockingManager.State="Document"
syncfusion:DocumentContainer.AllowPin="True"
  >
  </ContentControl>
  <!-- Docking dock window -->
  <ContentControl Name="Docking"
syncfusion:DockingManager.Header="Docking"
syncfusion:DockingManager.State="Document"
syncfusion:DocumentContainer.AllowPin="True"
  >
  </ContentControl>
</syncfusion:DockingManager>
```

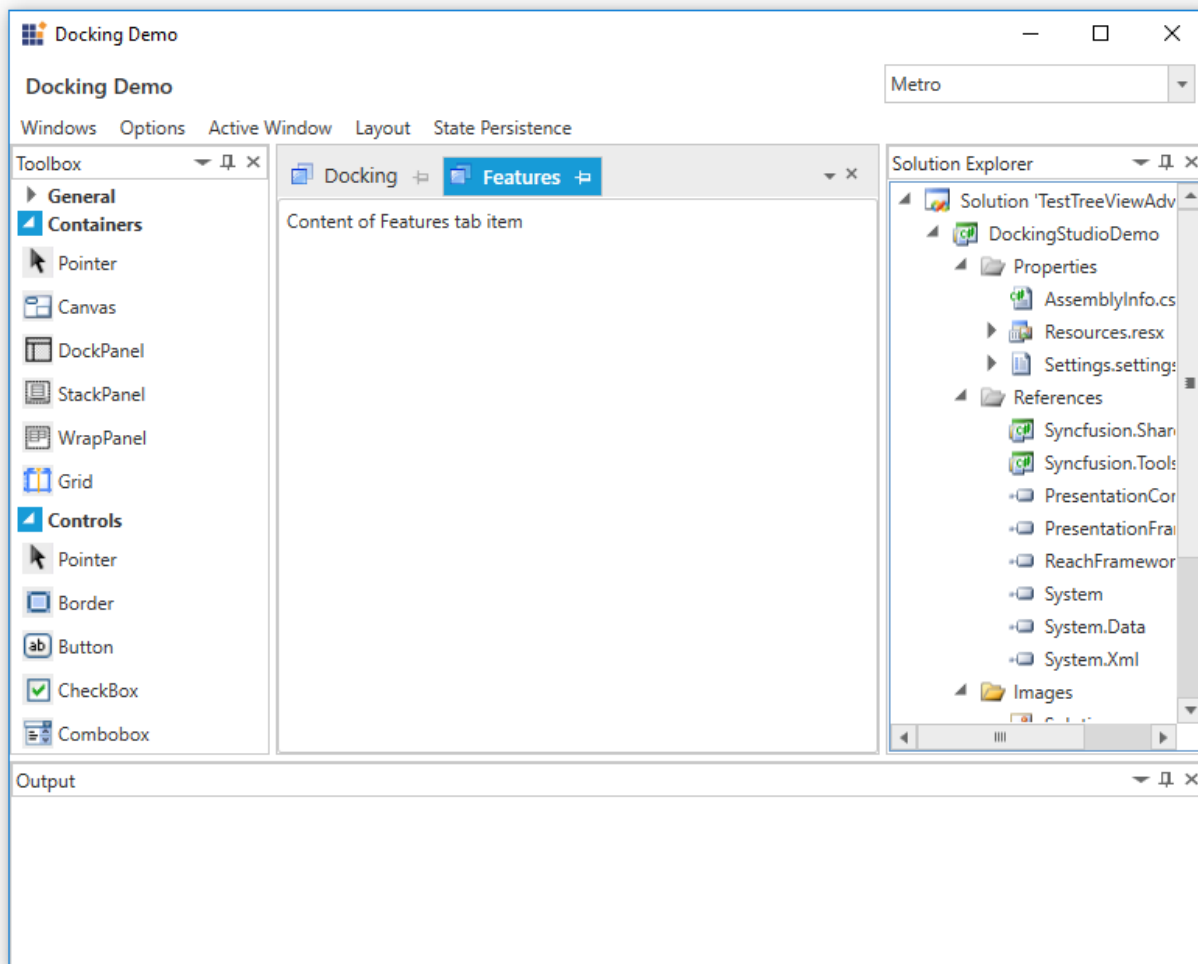
### Pin and Unpin tab items using PinButton

The PinButton will be visible in the document tabs only when the property [ShowPin](#) is true. The default value of the property is false, so the PinButton will be collapsed from the header panel of the document tab item.

### XML

```
<syncfusion:DockingManager Grid.Row="1" x:Name="DockingManager"
UseDocumentContainer="True">
  <!-- Output dock window -->
```

```
<ContentControl Name="Output"
syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Bottom"
syncfusion:DockingManager.DesiredHeightInDockedMode="200"
>
</ContentControl>
<!-- SolutionExplorer dock window -->
<ContentControl Name="SolutionExplorer"
syncfusion:DockingManager.Header="Solution Explorer"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.DesiredWidthInDockedMode="300"
>
</ContentControl>
<!-- ClassView dock window -->
<!-- Toolbox dock window -->
<ContentControl Name="Toolbox"
syncfusion:DockingManager.Header="Toolbox"
syncfusion:DockingManager.State="Dock"
HorizontalContentAlignment="Left"
syncfusion:DockingManager.DesiredWidthInDockedMode="250"
>
</ContentControl>
<!-- Features dock window -->
<ContentControl Name="Features"
syncfusion:DockingManager.Header="Features"
syncfusion:DockingManager.State="Document"
syncfusion:DocumentContainer.AllowPin="True"
syncfusion:DocumentContainer.ShowPin="True"
>
</ContentControl>
<!-- Docking dock window -->
<ContentControl Name="Docking"
syncfusion:DockingManager.Header="Docking"
syncfusion:DockingManager.State="Document"
syncfusion:DocumentContainer.AllowPin="False"
syncfusion:DocumentContainer.ShowPin="True"
>
</ContentControl>
</syncfusion:DockingManager>
```



If the property [AllowPin](#) is true, the pin button will be enabled and visible. If the property [ShowPin](#) is true and [AllowPin](#) is false, pin button will be displayed as disabled button.

#### Functionality of PinButton

When the pin button of the document tab item is visible, the corresponding document tab item can be pinned or unpinned from the DockingManager. When the corresponding document tab item is pinned, it will be inserted at first position of the document tab items collection(if the pinned tab item collection has zero count. Otherwise, the pinned tab item will be added to the existing collection). When the document tab item is unpinned, it will be removed from the pinned tab item collection and added to the first position of the unpinned tab item collection.

#### Pin and Unpin the tab items programmatically

document tab items can be pinned or unpinned from the DocumentContainer using [IsPinned](#) attached property of DocumentContainer. If the property [IsPinned](#) is set to true, the corresponding item will be added to respective index. Also, if the property [IsPinned](#) is set as false, the tab item will be removed from pinned collection and added to unpinned tab item collection. The default value of the property is False.

#### Pin and Unpin tab items through ContextMenu

The pin or unpin operations can be done through document tab item's ContextMenu also. If the property [AllowPin](#) is true, and the document tab item is not pinned, the "Pin Tab" option will be visible. If the document tab item is pinned already, "Unpin Tab" will be visible.

**XML**

```

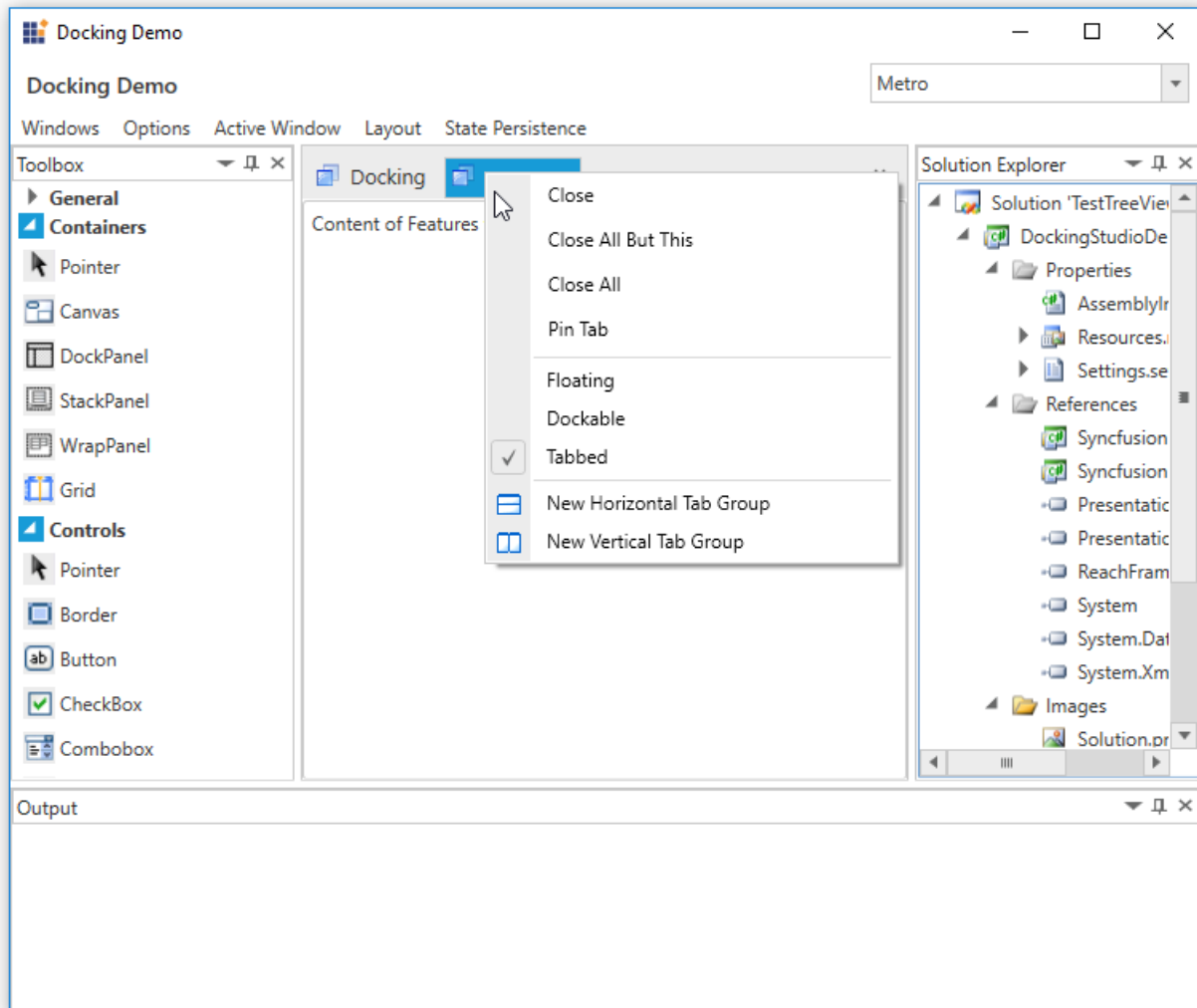
<syncfusion:DocumentContainer
Name="DocContainer"
Mode="TDI">
<syncfusion:DocumentContainer.Icon>
<ImageBrush ImageSource="document.png"/>
</syncfusion:DocumentContainer.Icon>
<!--TDI/MDI Children elements of the Document Container-->
<FlowDocumentScrollViewer
syncfusion:DocumentContainer.AllowPin="True"
syncfusion:DocumentContainer.Header="Document Container">
<FlowDocument FontFamily="Calibri" FontSize="13" TextAlignment="Left">
<Paragraph FontWeight="Bold" FontSize="15" TextAlignment="Center">
Syncfusion WPF Document Container</Paragraph>
<Paragraph>This sample exhibits the special features
of the Syncfusion Document Container Control for
Windows Presentation Foundation(WPF).
</Paragraph>
<Paragraph>View this document to experience the features of the
Document Container.Document Container supports both TDI and MDI.
</Paragraph>
<Paragraph>On certain occasions, users may need the data to be contained in
the traditional MDI(Multiple Document Interface) and others where constant
comparison needs to be made between the documents requires a
TDI(Tabbed Document Interface).</Paragraph>
</FlowDocument>
</FlowDocumentScrollViewer>
<FlowDocumentScrollViewer
syncfusion:DocumentContainer.AllowPin="True"
syncfusion:DocumentContainer.Header="Features" >
<FlowDocument FontFamily="Calibri" FontSize="13" TextAlignment="Left" >
<Paragraph FontWeight="Bold" FontSize="15" TextAlignment="Center">Document
Container-Features</Paragraph>
<Paragraph>Document container comes with a set of features. They include
</Paragraph>
<List>
<ListItem>
<Paragraph>Provides options for both MDI and TDI container Mode
</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Various window switching styles. Ctrl+tab could be used
to easily navigate through the windows</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Skins Support</Paragraph>
</ListItem>
<ListItem>
<Paragraph>State Persistence. Document container can be used to
load, save data in IS, BIN and in XML</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Huge set of properties, methods and events for easy
customization</Paragraph>
</ListItem>
</List>

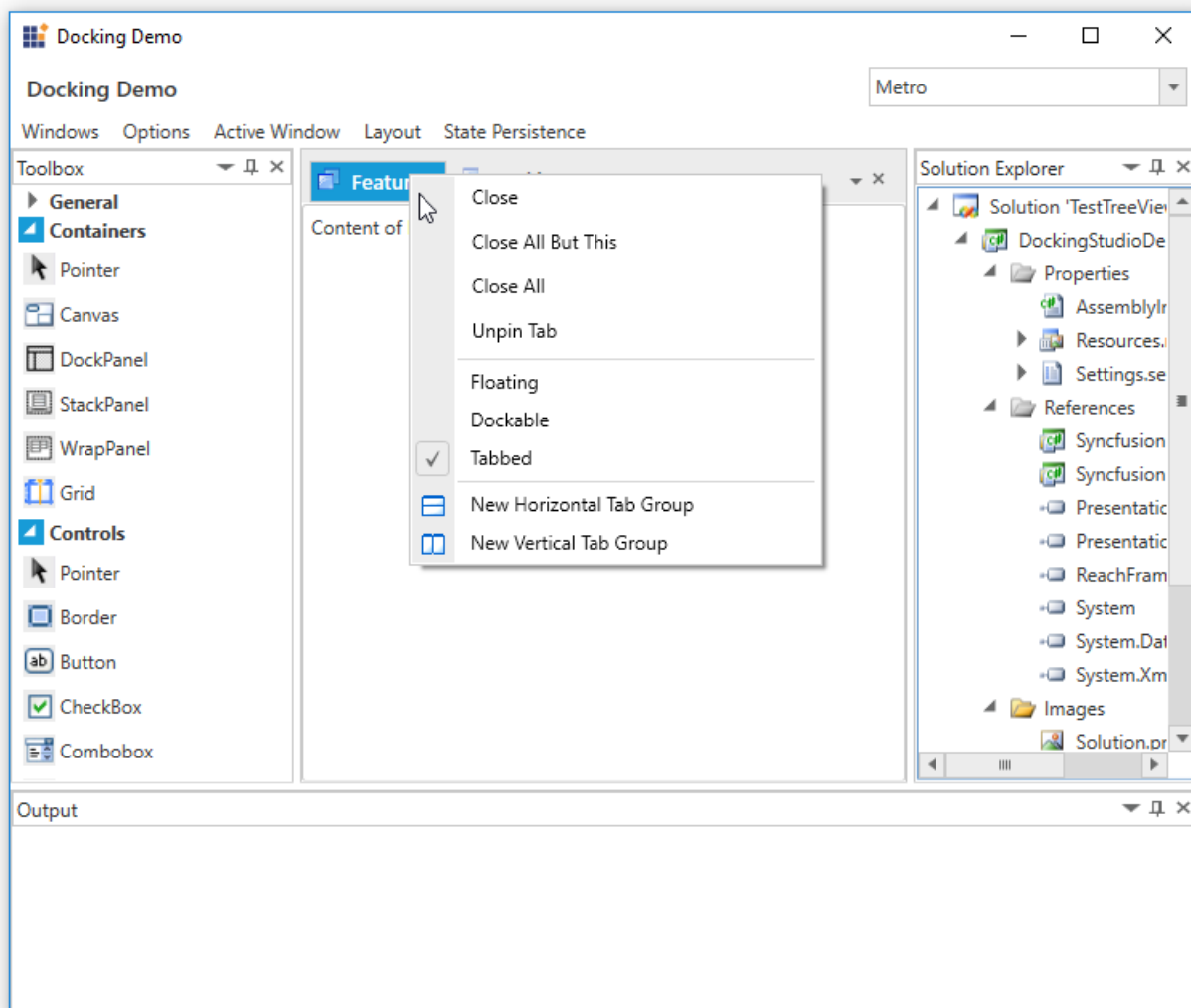
```

```
<Paragraph>Document container can be resized and moved using the keyboard.
</Paragraph>
</ListItem>
</List>
</FlowDocument>
</FlowDocumentScrollViewer >
<FlowDocumentScrollViewer
syncfusion:DocumentContainer.AllowPin="True"
syncfusion:DocumentContainer.Header="Window 1" >
<FlowDocument FontFamily="Calibri" FontSize="13" TextAlignment="Left">
<Paragraph>For ease of navigation, users can switch between the MDI and TDI
modes.
</Paragraph>
<Paragraph>Most users prefers keyboard to mouse navigation. Thus, to
satisfy all users, syncfusion Document Container is boosted by
Window Switcher(CTRL+TAB keys) for smooth, fine and flexible navigation
between interfaced windows. syncfusion ships five different modes of window
switchers.
</Paragraph>
<Paragraph>
Hold down the CTRL key and use the TAB keystroke repeatedly to
experience active switchers in effect.
</Paragraph>
</FlowDocument>
</FlowDocumentScrollViewer>
</syncfusion:DocumentContainer>
```

The following images illustrates the same,







### Custom context menu items for tab item

You can add the custom context menu items for tab item by using the [DocumentTabItemContextMenuItems](#) property. You can also add any level of sub menu items for custom context menu item by adding that sub `CustomMenuItem` to the parent [CustomMenuItem](#). You can check or uncheck the `CustomMenuItem` interactively or programmatically by using the `CustomMenuItem.IsChecked` property.

You can collapse the default tab item context menu and show only the custom context menu items by setting the [CollapseDefaultContextMenuItemsInDocumentTab](#) property to `true`. The default value of `CollapseDefaultContextMenuItemsInDocumentTab` property is `false`.

### XML

```
<syncfusion:DockingManager
  UseDocumentContainer="True"
  ContainerMode="TDI"
  Name="dockingManager" >
  <!--Adding custom context menu items for tab items-->
  <syncfusion:DockingManager.DocumentTabItemContextMenuItems>
  <!--Adding custom context menu items-->
```

```

<syncfusion:CustomMenuItem Header="Menu 1"/>
<syncfusion:CustomMenuItem Header="Menu 2">
<!--Adding sub custom context menu items-->
<syncfusion:CustomMenuItem Header="SubMenu 1"/>
<syncfusion:CustomMenuItem Header="SubMenu 2" IsChecked="True"/>
<syncfusion:CustomMenuItem Header="SubMenu 3">
<!--Adding sub custom context menu items for 'SubMenu 3'-->
<syncfusion:CustomMenuItem Header="Level 2"/>
</syncfusion:CustomMenuItem>
</syncfusion:CustomMenuItem>
</syncfusion:DockingManager.DocumentTabItemContextMenuItems>
<ContentControl syncfusion:DockingManager.Header="Document.xml"
syncfusion:DockingManager.State="Document"/>
<ContentControl syncfusion:DockingManager.Header="Document.xml.cs"
syncfusion:DockingManager.State="Document"
syncfusion:DockingManager.ShowCloseAllButThisMenuItem="False"
syncfusion:DockingManager.ShowCloseAllMenuItem="False"
syncfusion:DockingManager.ShowCloseMenuItem="False"
syncfusion:DockingManager.CollapseDefaultContextMenuItemsInDocumentTab="True"
"/>
<ContentControl syncfusion:DockingManager.Header="Toolbox"
syncfusion:DockingManager.State="Dock"/>
</syncfusion:DockingManager>

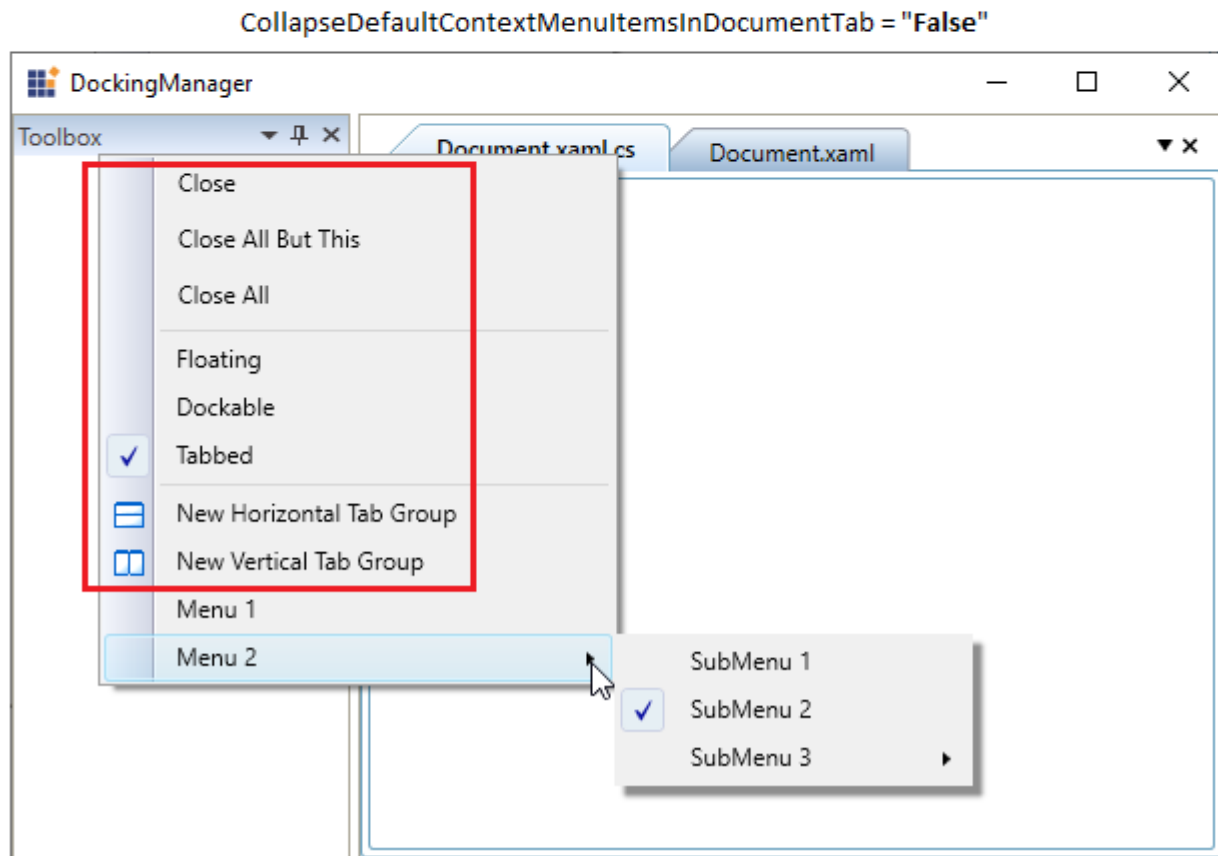
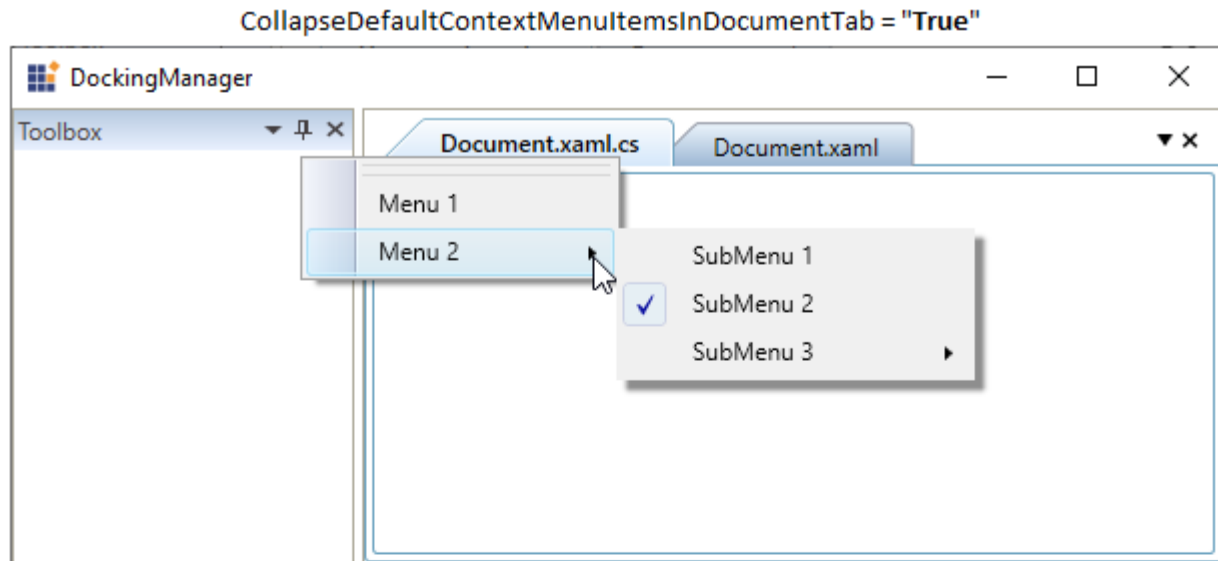
```

**C#**

```

//Creating custom context menu items
CustomMenuItem menu1 = new CustomMenuItem();
menu1.Header = "Menu 1";
CustomMenuItem menu2 = new CustomMenuItem();
menu2.Header = "Menu 2";
//Creating custom sub context menu items
CustomMenuItem customMenuItem1 = new CustomMenuItem() { Header = "SubMenu 1"
};
CustomMenuItem customMenuItem2 = new CustomMenuItem() { Header = "SubMenu
2", IsChecked = true };
CustomMenuItem customMenuItem3 = new CustomMenuItem() { Header = "SubMenu 3"
};
//Adding sub menu items for 'SubMenu 3' custom context menu item
CustomMenuItem level2_customMenuItem = new CustomMenuItem() { Header =
"Level 2" };
customMenuItem3.Items.Add(level2_customMenuItem);
menu2.Items.Add(customMenuItem1);
menu2.Items.Add(customMenuItem2);
menu2.Items.Add(customMenuItem3);
//Adding custom context menu items with sub menu items
dockingManager.DocumentTabItemContextMenuItems.Add(menu1);
dockingManager.DocumentTabItemContextMenuItems.Add(menu2);

```



**Note:** [View Sample in GitHub](#)

#### Custom tab list context menu item

You can add the custom tab list context menu items by using the [TabListContextMenuItems](#) property. You can also add any level of sub menu items for custom tab list context menu item by adding that sub `CustomMenuItem` to the parent `CustomMenuItem`. You can check or uncheck the

`CustomMenuItem` interactively or programmatically by using the `CustomMenuItem.IsChecked` property.

You can collapse the default tab list context menu items and show only the custom tab list context menu items by setting the [CollapseDefaultTabListContextMenuItems](#) property to `true`. The default value of `CollapseDefaultTabListContextMenuItems` property is `false`.

### XML

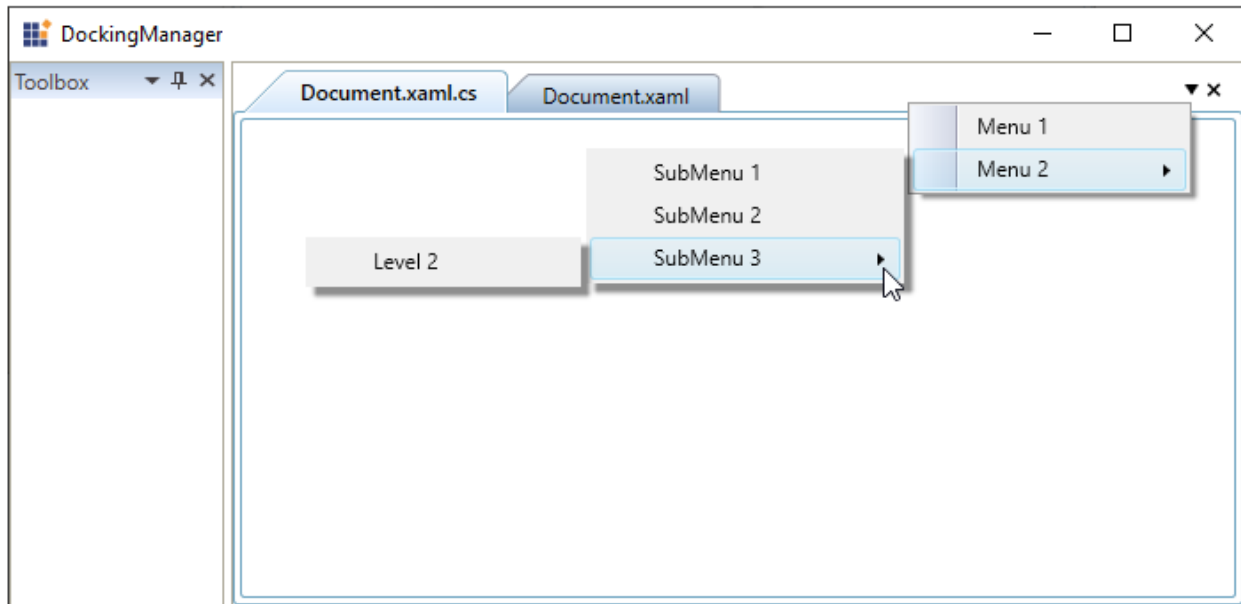
```
<syncfusion:DockingManager
  UseDocumentContainer="True"
  ContainerMode="TDI"
  CollapseDefaultTabListContextMenuItems="True"
  Name="dockingManager" >
  <!--Adding custom tab list context menu items-->
  <syncfusion:DockingManager.TabListContextMenuItems>
  <!--Adding custom context menu items-->
  <syncfusion:CustomMenuItem Header="Menu 1"/>
  <syncfusion:CustomMenuItem Header="Menu 2"/>
  <!--Adding sub custom context menu items-->
  <syncfusion:CustomMenuItem Header="SubMenu 1"/>
  <syncfusion:CustomMenuItem Header="SubMenu 2"/>
  <syncfusion:CustomMenuItem Header="SubMenu 3"/>
  <!--Adding sub custom context menu items for 'SubMenu 3'-->
  <syncfusion:CustomMenuItem Header="Level 2"/>
  </syncfusion:CustomMenuItem>
  </syncfusion:CustomMenuItem>
  </syncfusion:DockingManager.TabListContextMenuItems>
  <ContentControl syncfusion:DockingManager.Header="Document.xml"
  syncfusion:DockingManager.State="Document"/>
  <ContentControl syncfusion:DockingManager.Header="Document.xml.cs"
  syncfusion:DockingManager.State="Document"/>
  <ContentControl syncfusion:DockingManager.Header="Toolbox"
  syncfusion:DockingManager.State="Dock"/>
</syncfusion:DockingManager>
```

### C#

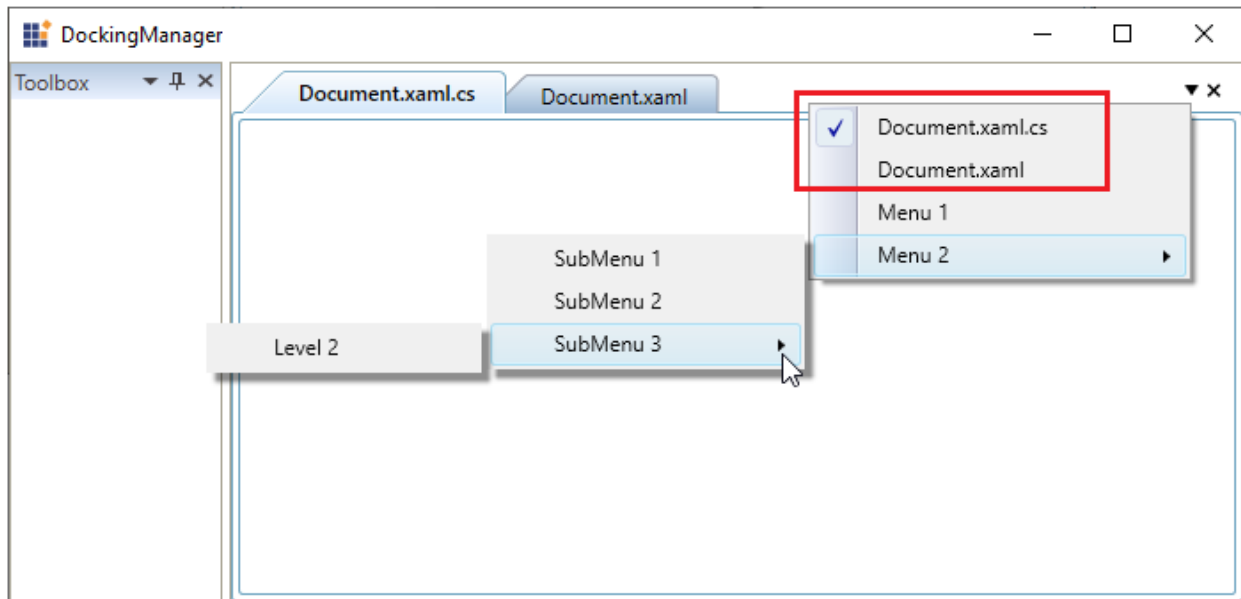
```
//Display only the custom tab list context menu
dockingManager.CollapseDefaultTabListContextMenuItems = true;
//Creating custom tab list context menu items
CustomMenuItem menu1 = new CustomMenuItem();
menu1.Header = "Menu 1";
CustomMenuItem menu2 = new CustomMenuItem();
menu2.Header = "Menu 2";
//Creating custom sub menu tab list context menu items
CustomMenuItem customMenuItem1 = new CustomMenuItem() { Header = "SubMenu 1"
};
CustomMenuItem customMenuItem2 = new CustomMenuItem() { Header = "SubMenu 2"
};
CustomMenuItem customMenuItem3 = new CustomMenuItem() { Header = "SubMenu 3"
};
//Adding sub menu items for 'SubMenu 3' custom tablist menu item
CustomMenuItem level2_customMenuItem = new CustomMenuItem() { Header =
"Level 2" };
customMenuItem3.Items.Add(level2_customMenuItem);
```

```
menu2.Items.Add(customMenuItem1);  
menu2.Items.Add(customMenuItem2);  
menu2.Items.Add(customMenuItem3);  
//Adding custom sub menu tab list context menu items  
dockingManager.TabListContextMenuItems.Add(menu1);  
dockingManager.TabListContextMenuItems.Add(menu2);
```

CollapseDefaultTabListContextMenuItems = "True"



CollapseDefaultTabListContextMenuItems = "False"



**Note:** [View Sample in GitHub](#)

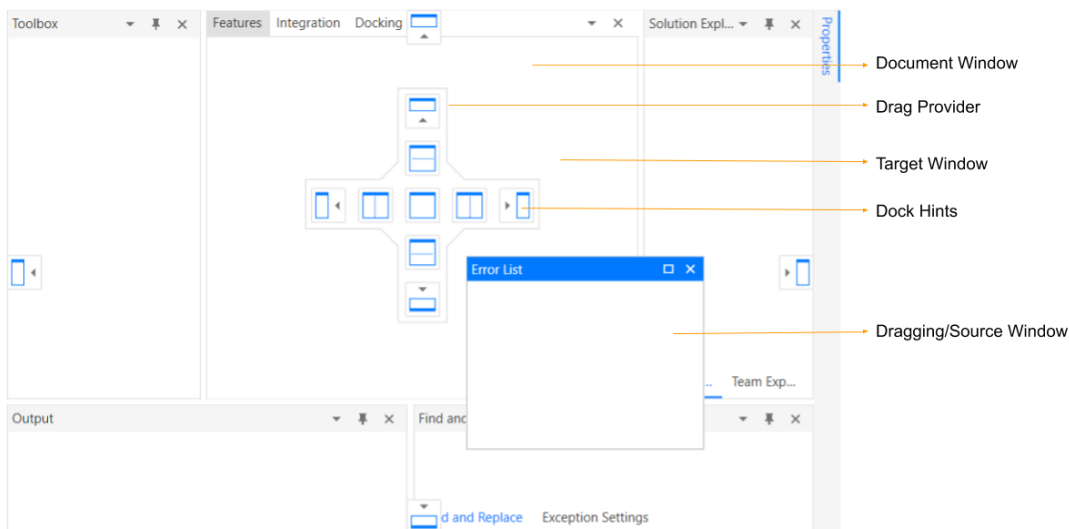
## Interaction with DragProvider in WPF Docking (DockingManager)

State of the dock children can be changed through the dock hints. This section explains how the dock item can be docked / undocked with the help of Dock hints and how the dock hints can be disabled/enabled at run time.

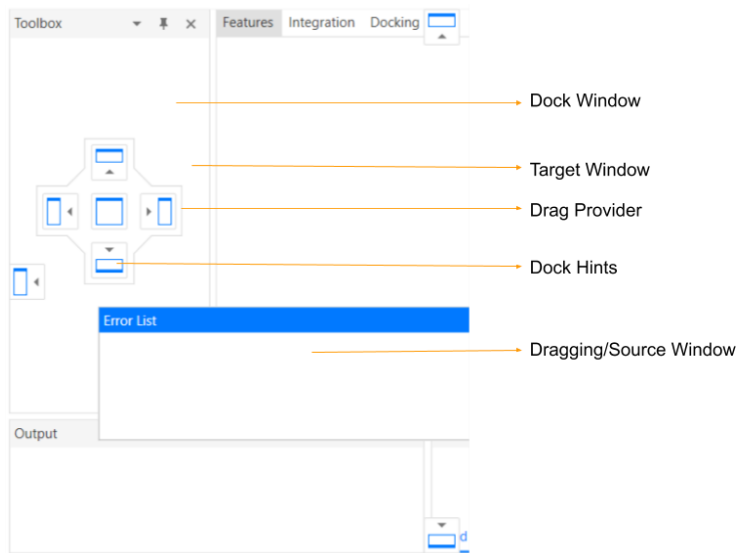
Terminologies:

1. Source Window - Window that is being dragged
2. Target Window - Window on which source window is going to be dropped
3. Document Window - A special window which is usually used to host document for editing
4. Dock Window - A normal window used to dock its child window in all four directions
5. Drag Provider - A prompt indicating dockable regions for docking
6. Dock Hints - Iconic indication of a region

Dock to document window



Dock to normal window



### Dock a window to desired direction

When user drag a window over another window, drag provider appears with dock hints. Based on dock hints, the window can be placed(docked or documented) to its desired place. If the window is dragged over the document area drag provider will be displayed with 9 hints,



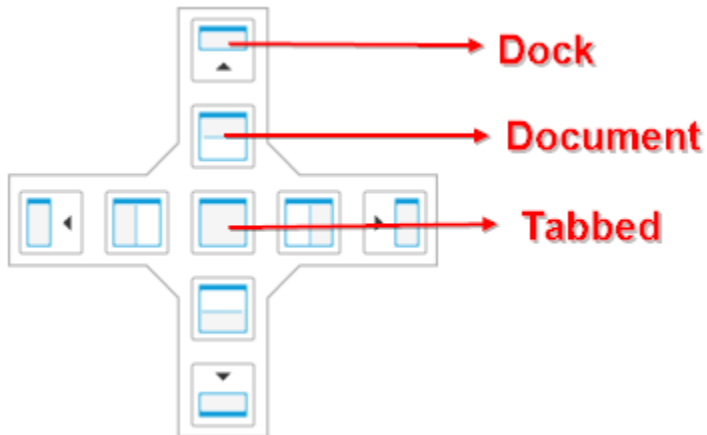
Otherwise the provider will be displayed with 5 hints.



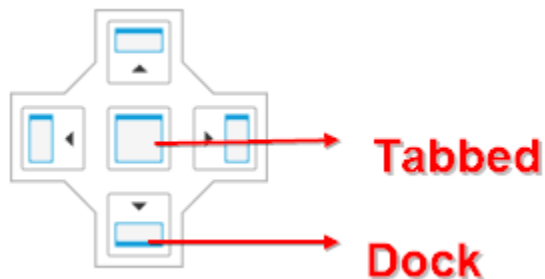


*Dock to document window*

1. User can document the dragged window with the existing tab group through the center dock hint.
2. New Tab group can be created at any sides(left, top, right and bottom) through the hints which is adjacent to the center dock hint.
3. Dragged window can be docked at any side using the hints which is outer side of the drag provider.

*Dock to normal window*

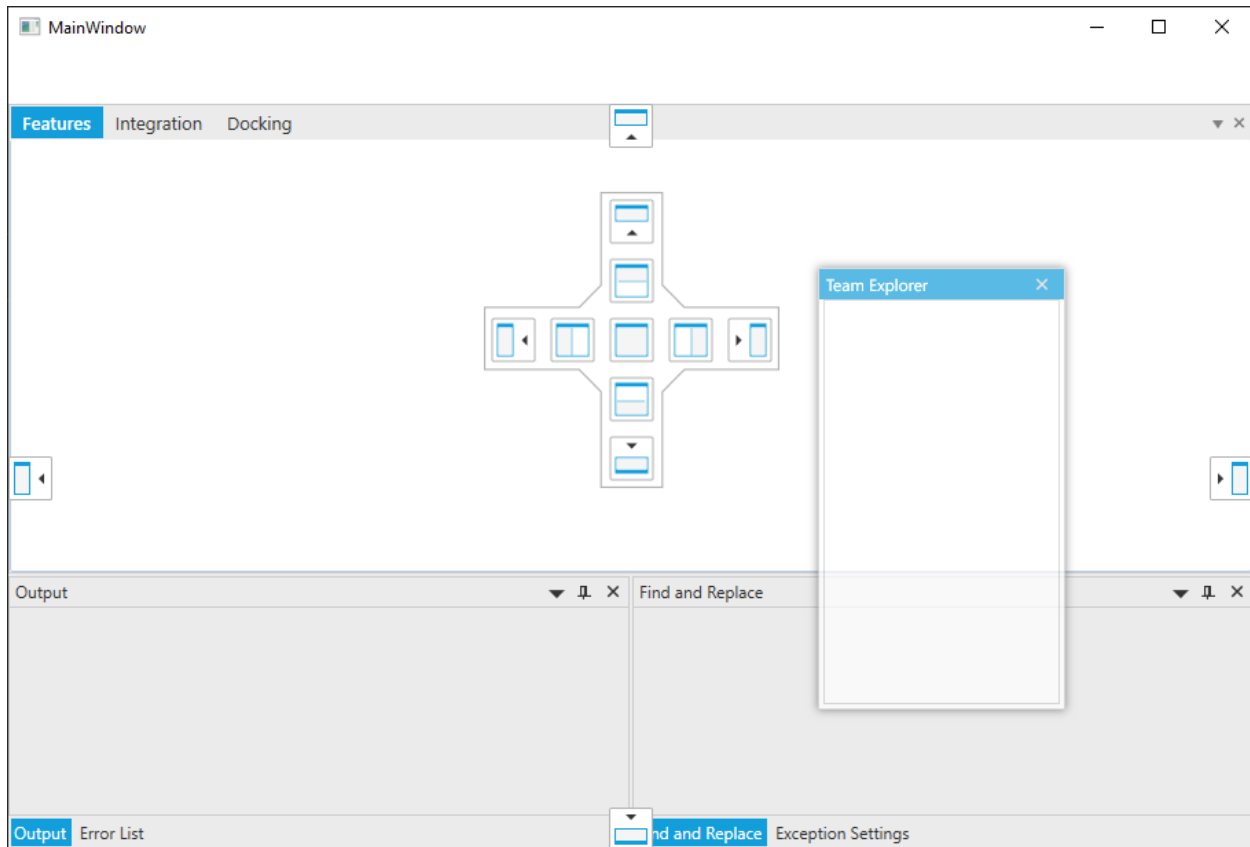
1. Using center dock hint, dragged window can be added as a tab to existing dock window.
2. Also the window can be docked to any side of the targeted window through the outer side of the drag provider.

*Restrict docking by disabling inner and outer dock hints*

Dock hints of the DragProvider can be disabled / enabled through attached property [DockAbility](#) of the DockingManager.

*All*

When the DockAbility of the window is set as **All**, all the dock hints of the DockingManager will be enabled in the DockingManager's DragProvider. Default value of the **DockAbility** property is **All**.



*DockLeft, DockRight, DockTop, DockBottom, DockTabbed*

The dock hints which allows the window to make the dragged window to Dock state will be enabled here. Based on the specified side **DockHints** will be enabled. The center dock hint will be enabled, if the DockAbility is **DockTabbed**.

### XML

```
<syncfusion:DockingManager
x:Name="dockingManager"
IsVS2010DraggingEnabled="True"
UseDocumentContainer="True">
  <ContentControl
x:Name="outputWindow"
syncfusion:DockingManager.DockAbility="DockTabbed"
syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Bottom" />
  <ContentControl
x:Name="errorlistWindow"
syncfusion:DockingManager.DockAbility="DockTop"
syncfusion:DockingManager.Header="Error List"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
  <ContentControl
x:Name="findWindow"
syncfusion:DockingManager.DockAbility="DockBottom"
syncfusion:DockingManager.Header="Find and Replace"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
</syncfusion:DockingManager>
```

```

<ContentControl
x:Name="exceptionWindow"
syncfusion:DockingManager.DockAbility="DockRight"
syncfusion:DockingManager.Header="Exception Settings"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="findWindow" />
<ContentControl
x:Name="teamExp"
syncfusion:DockingManager.DockAbility="DockLeft"
syncfusion:DockingManager.Header="Team Explorer"
syncfusion:DockingManager.SideInDockedMode="Left" />
<ContentControl
Name="dockingwindow"
syncfusion:DockingManager.Header="Docking"
syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>

```

**C#**

```

DockingManager.SetDockAbility(outputWindow, DockAbility.DockTabbed);
DockingManager.SetDockAbility(errorlistWindow, DockAbility.DockTop);
DockingManager.SetDockAbility(findWindow, DockAbility.DockBottom);
DockingManager.SetDockAbility(exceptionWindow, DockAbility.DockRight);
DockingManager.SetDockAbility(teamExp, DockAbility.DockLeft);

```

*DockAll*

When the DockAbility is **DockAll**, all the dock hints which allows the window to be Docked is enabled.

**XML**

```

<syncfusion:DockingManager
x:Name="dockingManager"
IsVS2010DraggingEnabled="True"
UseDocumentContainer="True">
<ContentControl
x:Name="outputWindow"
syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Bottom" />
<ContentControl
x:Name="errorlistWindow"
syncfusion:DockingManager.Header="Error List"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
<ContentControl
x:Name="findWindow"
syncfusion:DockingManager.Header="Find and Replace"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
<ContentControl
x:Name="exceptionWindow"
syncfusion:DockingManager.Header="Exception Settings"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="findWindow" />
<ContentControl
x:Name="teamExp"
syncfusion:DockingManager.DockAbility="DockAll"

```

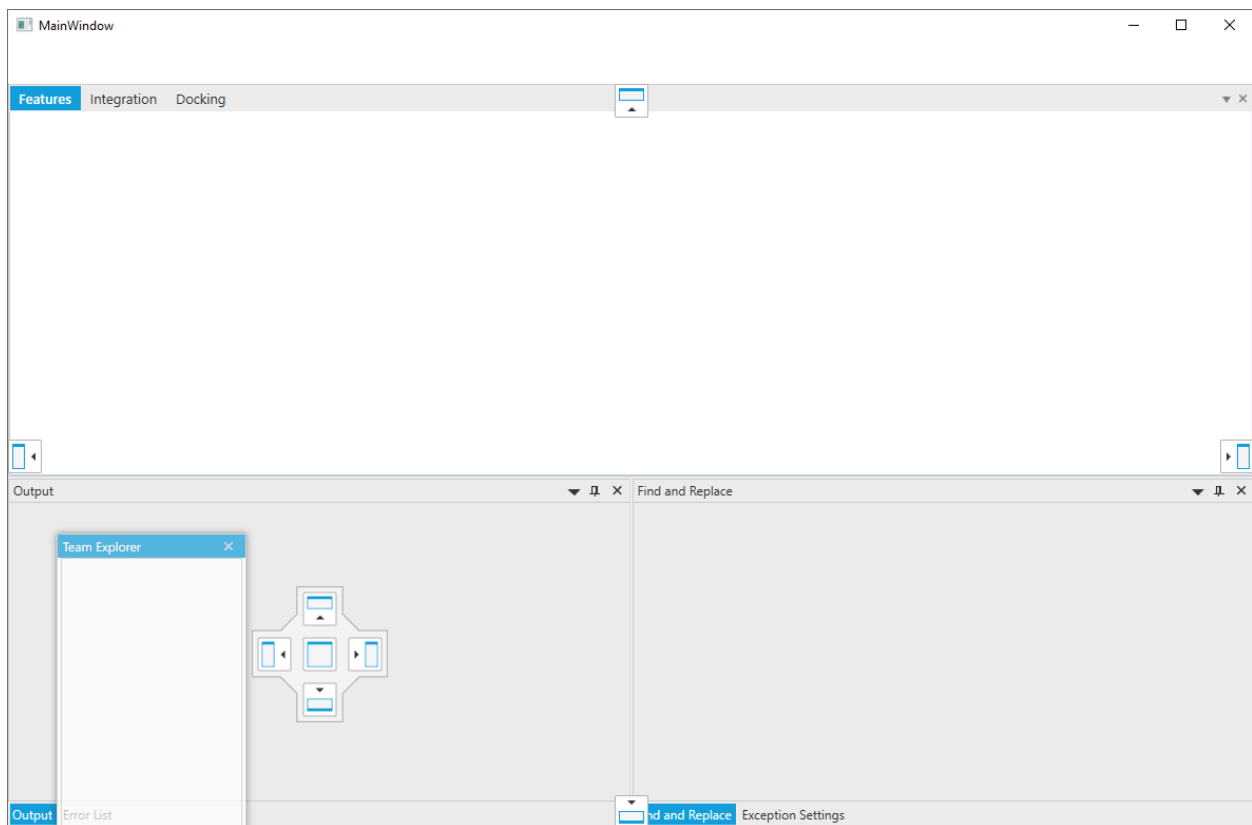
```

syncfusion:DockingManager.Header="Team Explorer"
syncfusion:DockingManager.SideInDockedMode="Left" />
<ContentControl
Name="dockingwindow"
syncfusion:DockingManager.Header="Docking"
syncfusion:DockingManager.State="Document" />
<ContentControl
x:Name="integration"
syncfusion:DockingManager.Header="Integration"
syncfusion:DockingManager.State="Document" />
<ContentControl
x:Name="features"
syncfusion:DockingManager.Header="Features"
syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>

```

## C#

```
DockingManager.SetDockAbility(teamExp, DockAbility.DockAll);
```



*DocumentLeft, DocumentRight, DocumentTop, DocumentBottom, DocumentTabbed*

The dock hints which allows the window to make the dragged window to Document state will be enabled here. Based on the specified side DockHints will be enabled. The center dock hint will be enabled, if the DockAbility is DocumentTabbed.

## XML

```

<syncfusion:DockingManager
x:Name="dockingManager"
IsVS2010DraggingEnabled="True"
UseDocumentContainer="True">
  <ContentControl
x:Name="outputWindow"
syncfusion:DockingManager.DockAbility="DocumentTabbed"
syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Bottom" />
  <ContentControl
x:Name="errorlistWindow"
syncfusion:DockingManager.DockAbility="DocumentTop"
syncfusion:DockingManager.Header="Error List"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
  <ContentControl
x:Name="findWindow"
syncfusion:DockingManager.DockAbility="DocumentBottom"
syncfusion:DockingManager.Header="Find and Replace"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
  <ContentControl
x:Name="exceptionWindow"
syncfusion:DockingManager.DockAbility="DocumentRight"
syncfusion:DockingManager.Header="Exception Settings"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="findWindow" />
  <ContentControl
x:Name="teamExp"
syncfusion:DockingManager.DockAbility="DocumentLeft"
syncfusion:DockingManager.Header="Team Explorer"
syncfusion:DockingManager.SideInDockedMode="Left" />
  <ContentControl
Name="dockingwindow"
syncfusion:DockingManager.Header="Docking"
syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>

```

**C#**

```

DockingManager.SetDockAbility(outputWindow, DockAbility.DocumentTabbed);
DockingManager.SetDockAbility(errorlistWindow, DockAbility.DocumentTop);
DockingManager.SetDockAbility(findWindow, DockAbility.DocumentBottom);
DockingManager.SetDockAbility(exceptionWindow, DockAbility.DocumentRight);
DockingManager.SetDockAbility(teamExp, DockAbility.DocumentLeft);

```

*DocumentAll*

If the DockAbility is **DocumentAll**, all the dock hints which allows the window to be documented is enabled in the DragProvider.

**XML**

```

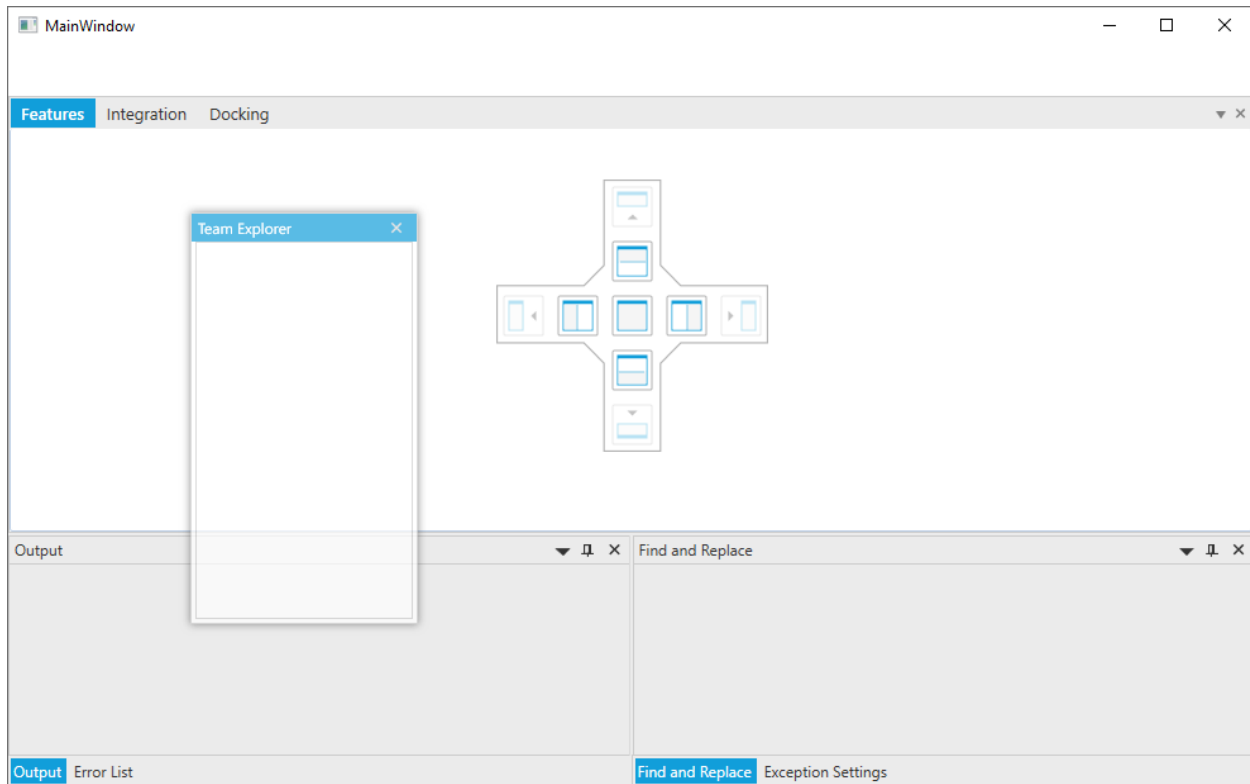
<syncfusion:DockingManager
x:Name="dockingManager"
IsVS2010DraggingEnabled="True"

```

```
UseDocumentContainer="True">
<ContentControl
x:Name="outputWindow"
syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Bottom" />
<ContentControl
x:Name="errorlistWindow"
syncfusion:DockingManager.Header="Error List"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
<ContentControl
x:Name="findWindow"
syncfusion:DockingManager.Header="Find and Replace"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
<ContentControl
x:Name="exceptionWindow"
syncfusion:DockingManager.Header="Exception Settings"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="findWindow" />
<ContentControl
x:Name="teamExp"
syncfusion:DockingManager.DockAbility="DocumentAll"
syncfusion:DockingManager.Header="Team Explorer"
syncfusion:DockingManager.SideInDockedMode="Left" />
<ContentControl
Name="dockingwindow"
syncfusion:DockingManager.Header="Docking"
syncfusion:DockingManager.State="Document" />
<ContentControl
x:Name="integration"
syncfusion:DockingManager.Header="Integration"
syncfusion:DockingManager.State="Document" />
<ContentControl
x:Name="features"
syncfusion:DockingManager.Header="Features"
syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>
```

## C#

```
DockingManager.SetDockAbility(teamExp, DockAbility.DocumentAll);
```



*Left, Right, Top, Bottom, Tabbed*

The dock hints which allows the window to make the dragged window to **Document** or **Dock** state will be enabled here. Based on the specified side **DockHints** will be enabled. Center dock hint which allows the dragged window to be **Dock** or **Document** will be enabled, if the **DockAbility** is **Tabbed**.

### XML

```
<syncfusion:DockingManager
x:Name="dockingManager"
IsVS2010DraggingEnabled="True"
UseDocumentContainer="True">
  <ContentControl
x:Name="outputWindow"
syncfusion:DockingManager.DockAbility="Tabbed"
syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Bottom" />
  <ContentControl
x:Name="errorlistWindow"
syncfusion:DockingManager.DockAbility="Top"
syncfusion:DockingManager.Header="Error List"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
  <ContentControl
x:Name="findWindow"
syncfusion:DockingManager.DockAbility="Bottom"
syncfusion:DockingManager.Header="Find and Replace"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
  <ContentControl
x:Name="exceptionWindow"
```

```

syncfusion:DockingManager.DockAbility="Right"
syncfusion:DockingManager.Header="Exception Settings"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="findWindow" />
<ContentControl
x:Name="teamExp"
syncfusion:DockingManager.DockAbility="Left"
syncfusion:DockingManager.Header="Team Explorer"
syncfusion:DockingManager.SideInDockedMode="Left" />
<ContentControl
Name="dockingwindow"
syncfusion:DockingManager.Header="Docking"
syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>

```

**C#**

```

DockingManager.SetDockAbility(outputWindow, DockAbility.Tabbed);
DockingManager.SetDockAbility(errorlistWindow, DockAbility.Top);
DockingManager.SetDockAbility(findWindow, DockAbility.Bottom);
DockingManager.SetDockAbility(exceptionWindow, DockAbility.Right);
DockingManager.SetDockAbility(teamExp, DockAbility.Left);

```

*Vertical*

The dock hints which are arranged vertically in the DragProvider will be enabled here if the DockAbility is **Vertical**. This will allow the dragged window's state to Dock or Document.

**XML**

```

<syncfusion:DockingManager
x:Name="dockingManager"
IsVS2010DraggingEnabled="True"
UseDocumentContainer="True">
<ContentControl
x:Name="outputWindow"
syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Bottom" />
<ContentControl
x:Name="errorlistWindow"
syncfusion:DockingManager.Header="Error List"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
<ContentControl
x:Name="findWindow"
syncfusion:DockingManager.Header="Find and Replace"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
<ContentControl
x:Name="exceptionWindow"
syncfusion:DockingManager.Header="Exception Settings"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="findWindow" />
<ContentControl
x:Name="teamExp"
syncfusion:DockingManager.DockAbility="Vertical"

```



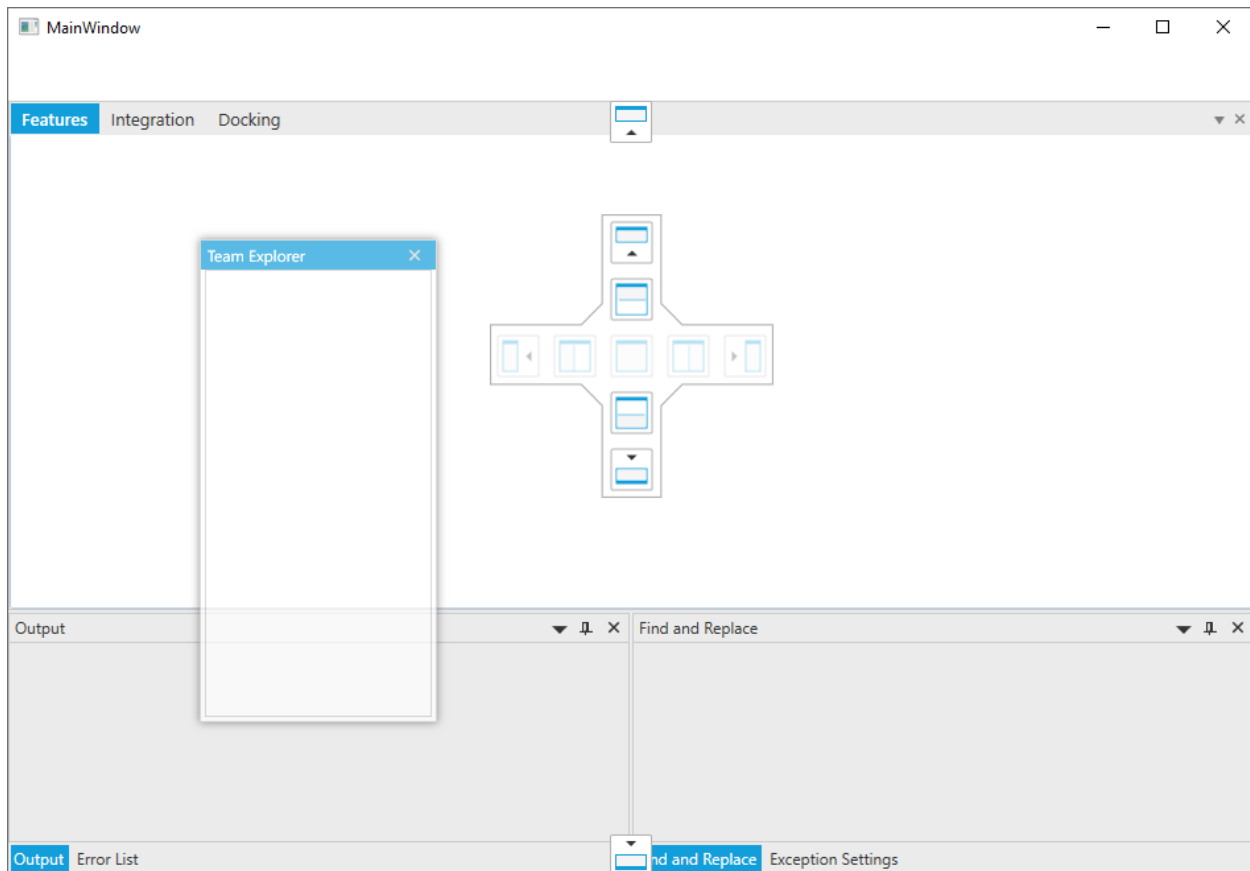
```

syncfusion:DockingManager.Header="Team Explorer"
syncfusion:DockingManager.SideInDockedMode="Left" />
<ContentControl
Name="dockingwindow"
syncfusion:DockingManager.Header="Docking"
syncfusion:DockingManager.State="Document" />
<ContentControl
x:Name="integration"
syncfusion:DockingManager.Header="Integration"
syncfusion:DockingManager.State="Document" />
<ContentControl
x:Name="features"
syncfusion:DockingManager.Header="Features"
syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>

```

**C#**

```
DockingManager.SetDockAbility(teamExp, DockAbility.Vertical);
```

*Horizontal*

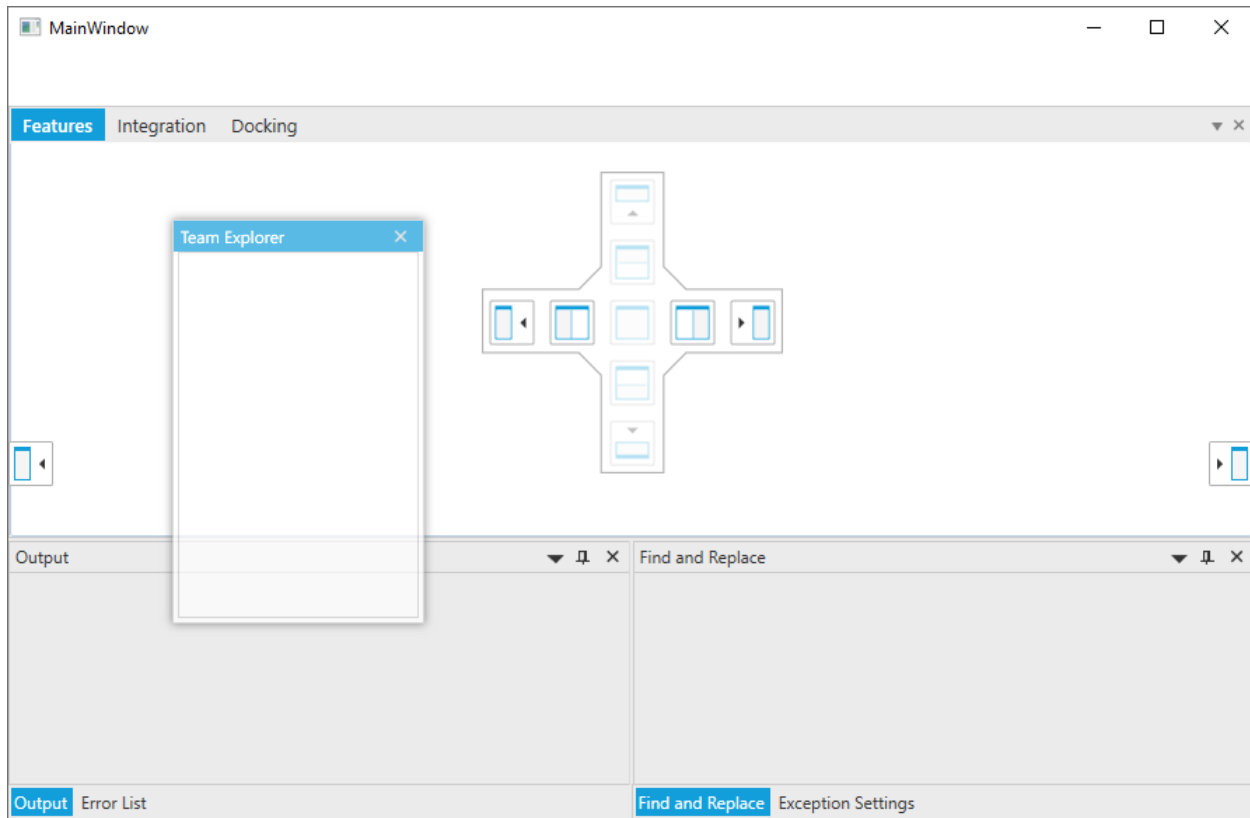
The dock hints which are arranged horizontally in the DragProvider will be enabled here if the DockAbility is **Horizontal**. This will allow the dragged window's state to Dock or Document.

**XML**

```
<syncfusion:DockingManager
x:Name="dockingManager"
IsVS2010DraggingEnabled="True"
UseDocumentContainer="True">
<ContentControl
x:Name="outputWindow"
syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Bottom" />
<ContentControl
x:Name="errorlistWindow"
syncfusion:DockingManager.Header="Error List"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
<ContentControl
x:Name="findWindow"
syncfusion:DockingManager.Header="Find and Replace"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
<ContentControl
x:Name="exceptionWindow"
syncfusion:DockingManager.Header="Exception Settings"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="findWindow" />
<ContentControl
x:Name="teamExp"
syncfusion:DockingManager.DockAbility="Horizontal"
syncfusion:DockingManager.Header="Team Explorer"
syncfusion:DockingManager.SideInDockedMode="Left" />
<ContentControl
Name="dockingwindow"
syncfusion:DockingManager.Header="Docking"
syncfusion:DockingManager.State="Document" />
<ContentControl
x:Name="integration"
syncfusion:DockingManager.Header="Integration"
syncfusion:DockingManager.State="Document" />
<ContentControl
x:Name="features"
syncfusion:DockingManager.Header="Features"
syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>
```

**C#**

```
DockingManager.SetDockAbility(teamExp, DockAbility.Horizontal);
```



### None

None of the dock hints will be enabled, if the DockAbility is **None**.

**Note:** View [Sample](#) in GitHub

### Restrict outer dockability

By setting **DockAbility** property to the dragged window, both inner and outer dockability has been handled. To disabled of the outer dockability alone, **OuterDockAbility** property is used. The values which is assinged to the **OuterDockAbility** will be effective only if the **UseOuterDockAbility** property is true.

### Restrict docking at run-time

You can disabled the dock hints at run-time by handling **PreviewDockHints** event in the **DockingManager**. It helps to handle before displaying the dock hints when drag the windows in **DockingManager** based on mouse hovered window. This event will be triggered for both inner dockability and outer dockability while drag the windows. It receives an argument of type **PreviewDockHintsEventArgs** containing the following information about the event.

Member	Description
DraggingSource	Gets or sets the dragging element of DockingManager that raises the PreviewDockHints event.
DraggingTarget	Gets or sets the target element in which the dragging window of DockingManager to be docked.
DockAbility	Gets or sets the DockAbility, to restrict docking on target window.

OuterDockAbility	Gets or sets the OuterDockability, to restrict docking on edges of DockingManager.
------------------	--

**XML**

```

<syncfusion:DockingManager
x:Name="dockingManager"
IsVS2010DraggingEnabled="True"
PreviewDockHints="DockingManager_PreviewDockHints"
UseDocumentContainer="True"
UseOuterDockAbility="True">
  <ContentControl
x:Name="outputWindow"
syncfusion:DockingManager.Header="Output"
syncfusion:DockingManager.SideInDockedMode="Bottom" />
  <ContentControl
x:Name="errorlistWindow"
syncfusion:DockingManager.Header="Error List"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
  <ContentControl
x:Name="findWindow"
syncfusion:DockingManager.Header="Find and Replace"
syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.TargetNameInDockedMode="outputWindow" />
  <ContentControl
x:Name="exceptionWindow"
syncfusion:DockingManager.Header="Exception Settings"
syncfusion:DockingManager.SideInDockedMode="Tabbed"
syncfusion:DockingManager.TargetNameInDockedMode="findWindow" />
  <ContentControl
x:Name="teamExp"
syncfusion:DockingManager.Header="Team Explorer"
syncfusion:DockingManager.SideInDockedMode="Left" />
  <ContentControl
Name="dockingwindow"
syncfusion:DockingManager.Header="Docking"
syncfusion:DockingManager.State="Document" />
  <ContentControl
x:Name="integration"
syncfusion:DockingManager.Header="Integration"
syncfusion:DockingManager.State="Document" />
  <ContentControl
x:Name="features"
syncfusion:DockingManager.Header="Features"
syncfusion:DockingManager.State="Document" />
</syncfusion:DockingManager>

```

You can handle the event as follows,

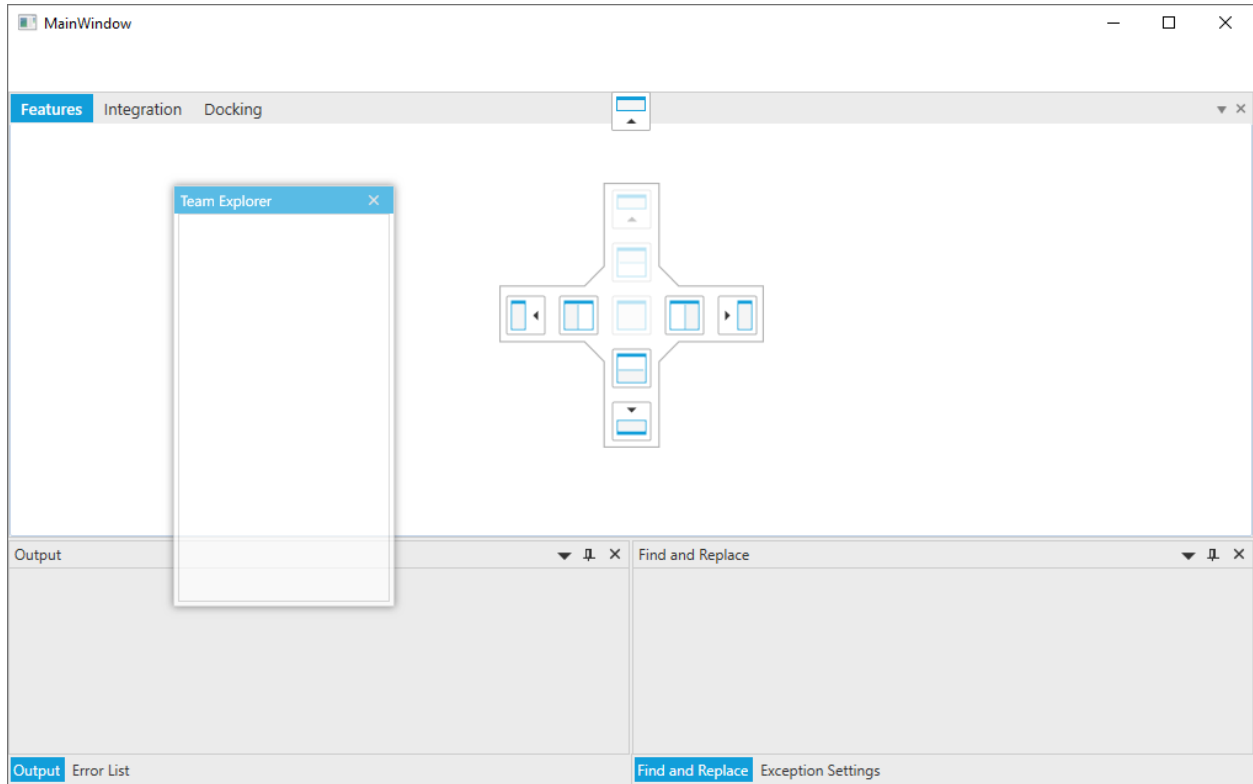
**C#**

```

private void DockingManager_PreviewDockHints(object sender,
PreviewDockHintsEventArgs e) {
if(e.DraggingSource == teamExp)
{

```

```
e.DockAbility = DockAbility.Horizontal | DockAbility.Bottom;
e.OuterDockAbility = OuterDockAbility.Top;
}
}
```



**Note:** View [Sample](#) in GitHub

## Linked Manager and Nested Docking in WPF Docking

### Linked Manager

The windows from one DockingManager cannot be dragged and dropped to another DockingManager by default. But Linked Manager support allows to drag and drop the windows from one DockingManager to another by setting `TargetDockingManager` list.

Source Docking Manager

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True">
  <ContentControl syncfusion:DockingManager.State="Document"
syncfusion:DockingManager.Header="Window1" />
  <ContentControl syncfusion:DockingManager.State="Dock"
syncfusion:DockingManager.Header="Window2" />
  <ContentControl syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.Header="Window3" />
</syncfusion:DockingManager>
```

Target Docking Manager

### XML

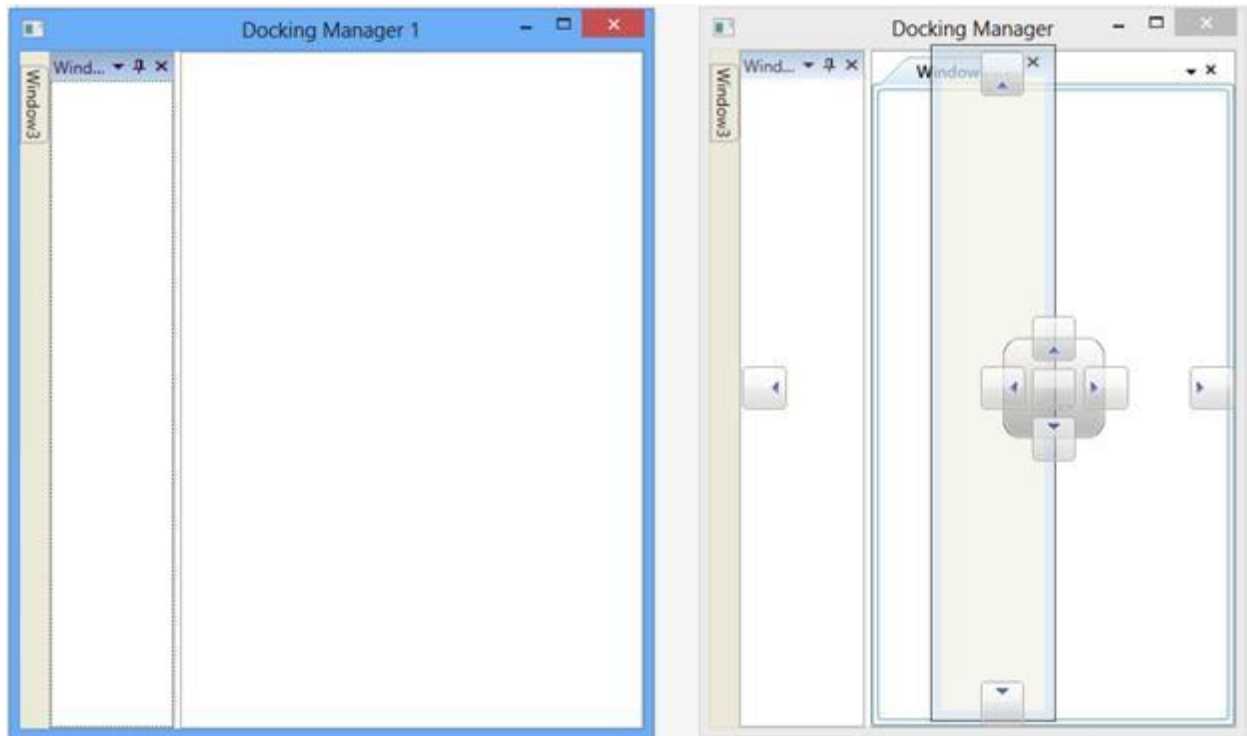
```
<syncfusion:DockingManager x:Name="DockingManager2"
UseDocumentContainer="True">
<ContentControl syncfusion:DockingManager.State="Dock"
syncfusion:DockingManager.Header="Window1"></ContentControl>
<ContentControl syncfusion:DockingManager.State="Dock"
syncfusion:DockingManager.Header="Window2"></ContentControl>
<ContentControl syncfusion:DockingManager.State="AutoHidden"
syncfusion:DockingManager.Header="Window3"></ContentControl>
</syncfusion:DockingManager>
```

### C#

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        count++;
        MainWindow1 MainWindow = new MainWindow1();
        MainWindow.Title = "Docking Manager 1";
        MainWindow.Show();
        this.DockingManager1.AddToTargetManagersList(MainWindow.DockingManager2);
        MainWindow.DockingManager2.AddToTargetManagersList(this.DockingManager1);
    }
}
```

### VB.NET

```
Partial Public Class MainWindow
Inherits Window
Public Sub New()
InitializeComponent()
count += 1
Dim MainWindow As New MainWindow1()
MainWindow.Title = "Docking Manager 1"
MainWindow.Show()
Me.DockingManager1.AddToTargetManagersList(MainWindow.DockingManager2)
MainWindow.DockingManager2.AddToTargetManagersList(Me.DockingManager1)
End Sub
End Class
```



#### *Adding TargetManager list of DockingManager*

To add TargetManager list in the DockingManager, call [AddToTargetManagersList](#) method of the DockingManager with the valid DockingManager instance as argument.

When only one DockingManager has TargetManagerList, the window drop to TargetManager cannot drag back to Owner DockingManager. For example DockingManager1 and DockingManager2 are the DockingManager instance and the DockingManager2 is added to TargetManagerList of DockingManager1, but the DockingManager2 is not aware of its TargetManager.

Here, the windows from DockingManager1 are only allowed to be dragged and dropped in DockingManager2,

#### **C#**

```
this.DockingManager1.AddToTargetManagersList(MainWindow.DockingManager2);
```

#### **VB.NET**

```
Me.DockingManager1.AddToTargetManagersList(MainWindow.DockingManager2)
```

To drag and drop the window from DockingManager2 to DockingManager1, DockingManager1 must be added to TargetManagerList of DockingManager2.

#### **C#**

```
this.DockingManager1.AddToTargetManagersList(MainWindow.DockingManager2);
MainWindow.DockingManager2.AddToTargetManagersList(this.DockingManager1);
```

#### **VB.NET**

```
Me.DockingManager1.AddToTargetManagersList(MainWindow.DockingManager2)
MainWindow.DockingManager2.AddToTargetManagersList(Me.DockingManager1)
```

### Removing Target Manager list

To remove DockingManager from the TargetManagerList, call [RemoveFromTargetManagerList](#) of DockingManager with the valid DockingManager instance argument. For example, to remove the DockingManager1 from the TargetManagersList of DockingManager2, follow the below code snippets:

#### C#

```
MainWindow.DockingManager2.RemoveFromTargetManagersList(this.DockingManager1);
```

#### VB.NET

```
MainWindow.DockingManager2.RemoveFromTargetManagersList(Me.DockingManager1)
```

### Nested Docking

DockingManager provides the NestedDockingManager support, that allows to add DockingManager as a child window to another DockingManager.

In Nested DockingManager, the whole DockingManager can be dragged and dropped inside the Parent DockingManager and DockWindows inside the DockingManager cannot be dragged and dropped on the owner DockingManager.

### Adding DockingManager as Child in DockingManager

#### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True" syncfusion:DockingManager.Header="Dock1">
<ContentControl x:Name="Content1" syncfusion:DockingManager.Header="Dock1"/>
<syncfusion:DockingManager x:Name="DockingManager2"
UseDocumentContainer="True"
SideInDockedMode="Left" syncfusion:DockingManager.Header="Dock2" >
<ContentControl syncfusion:DockingManager.Header="Dock2"
syncfusion:DockingManager.DesiredWidthInDockedMode="600" />
</syncfusion:DockingManager>
<syncfusion:DockingManager x:Name="DockingManager3"
UseDocumentContainer="True"
SideInDockedMode="Bottom" syncfusion:DockingManager.Header="Dock3">
<ContentControl syncfusion:DockingManager.Header="Dock3"
syncfusion:DockingManager.DesiredWidthInDockedMode="600"/>
</syncfusion:DockingManager>
</syncfusion:DockingManager>
```

#### C#

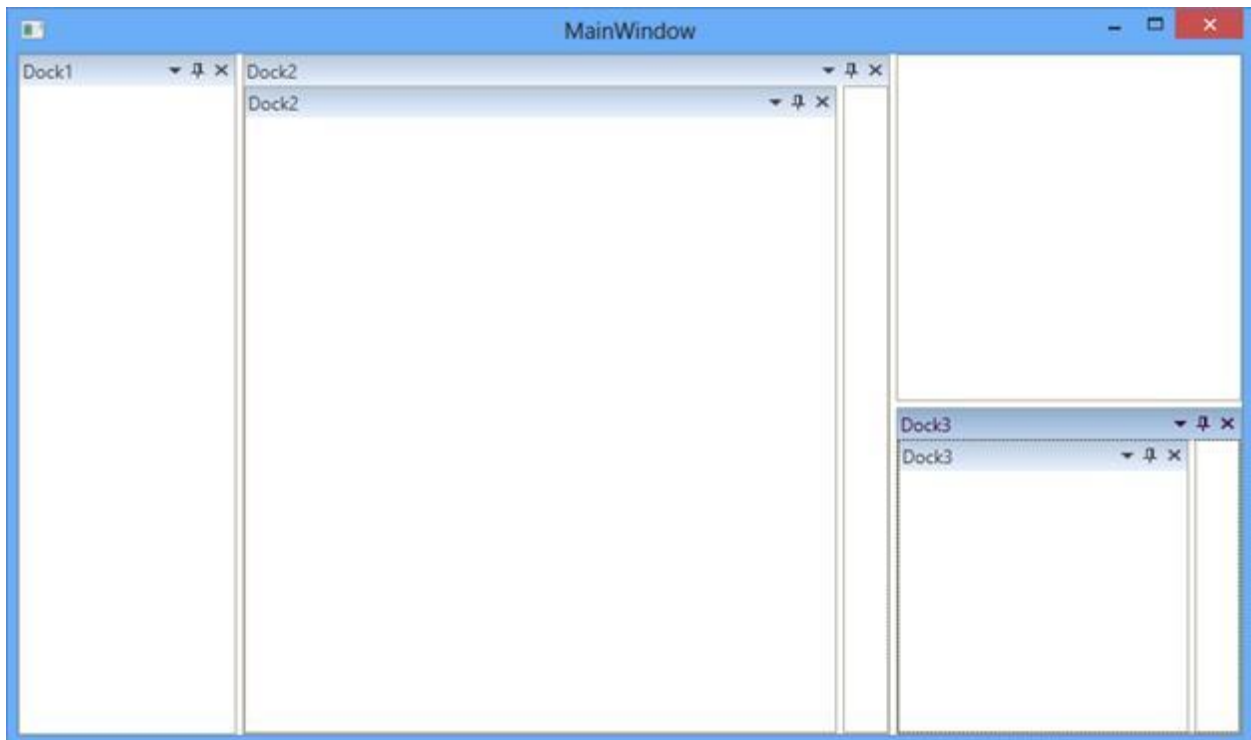
```
DockingManager DockingManager1 = new DockingManager() { UseDocumentContainer = true };
ContentControl content1 = new ContentControl() { Name = "Content1" };
DockingManager.SetHeader(content1, "Dock1");
DockingManager DockingManager2 = new DockingManager() { UseDocumentContainer = true };
DockingManager.SetHeader(DockingManager2, "Dock2");
```



```

DockingManager.SetSideInDockedMode(DockingManager2, DockSide.Left);
ContentControl content2 = new ContentControl() { Name = "Content2" };
DockingManager.SetHeader(content2, "Dock2");
DockingManager.SetDesiredWidthInDockedMode(content2, 600);
DockingManager2.Children.Add(content2);
DockingManager DockingManager3 = new DockingManager() { UseDocumentContainer
= true };
DockingManager.SetHeader(DockingManager3, "Dock3");
DockingManager.SetSideInDockedMode(DockingManager3, DockSide.Bottom);
ContentControl content3 = new ContentControl() { Name = "Content3" };
DockingManager.SetHeader(content3, "Dock3");
DockingManager.SetDesiredWidthInDockedMode(content3, 600);
DockingManager3.Children.Add(content3);
DockingManager1.Children.Add(content1);
DockingManager1.Children.Add(DockingManager2);
DockingManager1.Children.Add(DockingManager3);

```



## Hosting in WPF Docking (DockingManager)

### Hosting a Windows Form control

DockingManager allows to host a WindowsForm control as a Docking Child. Here a Windows Forms WebBrowser control added as Child Window

### XML

```

<syncfusion:DockingManager x:Name="DockingManager1" DockFill="True" >
  <WebBrowser Name="Web1" />
</syncfusion:DockingManager>

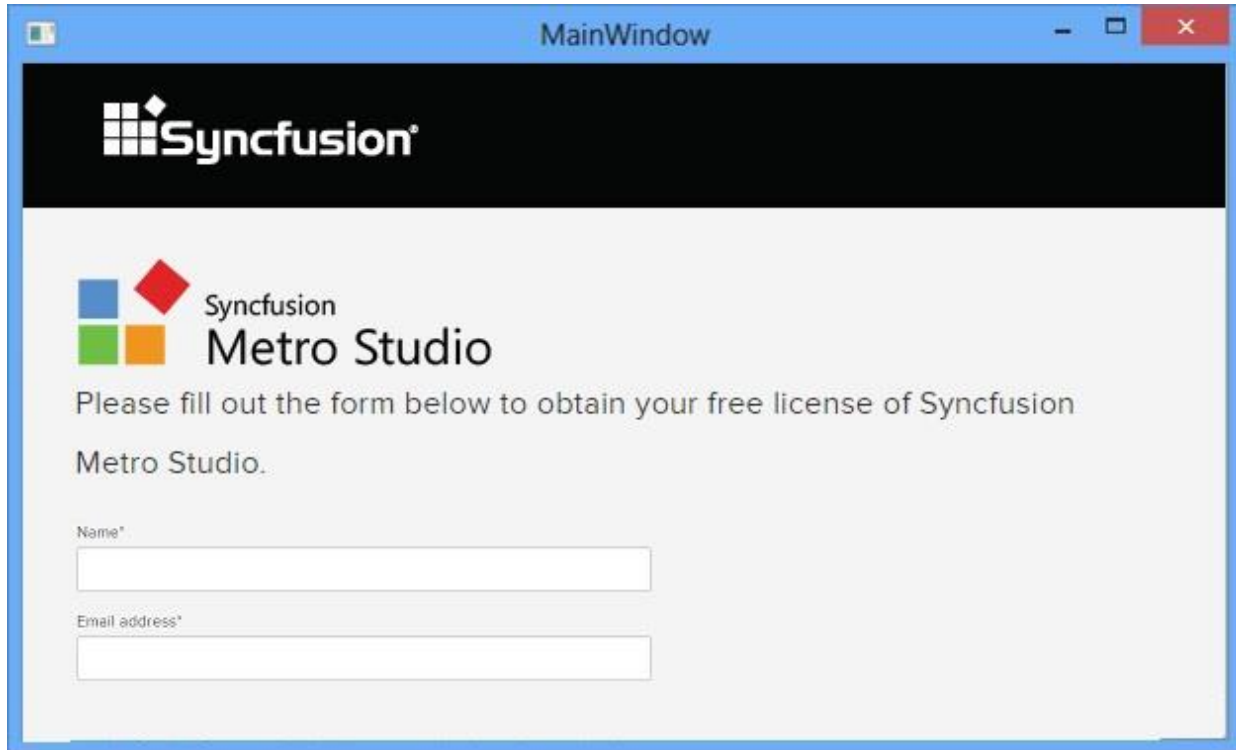
```

### C#

```

WebBrowser web1 = new WebBrowser();
web1.Name = "Web1";
dock.Children.Add(web1);
Web1.Navigate("http://www.syncfusion.com/downloads/metrostudio");

```



#### Interaction with control hosted by Win32 Host

- While interacting the WindowsForm control with WPF controls, it leads to inaccessibility of WPF controls behind WindowsForm control, since the WindowsForm control is on top.
- While floating the WindowsForm control in DockingManager, it is invisible. For these cases, set [UseInteropCompatibilityMode](#) property as `True` for the DockingManager.

#### XML

```

<syncfusion:DockingManager x:Name="DockingManager1" DockFill="True"
UseInteropCompatibilityMode="True">
<WebBrowser Name="Web1" />
</syncfusion:DockingManager>

```

#### C#

```

DockingManager1.UseInteropCompatibilityMode = true;

```

#### State Persistence in WPF Docking (DockingManager)

State persistence is the combined process of serialization and deserialization.

DockingManager provides built-in state persistence functionality to save and load at different states and sides. It also provides [DeleteDockState](#) and [ResetState](#) Method to work on state functionality.

To reset the DockingManager state, call [ResetState](#) method of DockingManager instance.

**Note:** DockingManager serializes and de-serializes the controls using [Name](#) property. So, ensure to set [Name](#) property for all child controls. The name of children in saved layout should be same as the name of children in DockingManager to load the saved layout. [LoadDockState](#) returns [true](#) or [false](#) to notify whether de-serialization process has been successful.

#### C#

```
DockingManager1.ResetState();
```

#### VB.NET

```
DockingManager1.ResetState()
```

To delete the DockState of the DockingManager, call [DeleteDockState](#) of DockingManager instance

#### C#

```
DockingManager1.DeleteDockState();
```

#### VB.NET

```
DockingManager1.DeleteDockState()
```

#### Auto Save / Load functionalities

DockingManager supports AutoSave support, that allows to persist its state automatically. To enable this functionality, set the [PersistState](#) property as [True](#). The default value of the [PersistState](#) property is [False](#). It saves the state of the DockingManager in an isolated storage format while [WindowClosing](#).

#### C#

```
DockingManager1.PersistState = true;
```

#### VB.NET

```
DockingManager1.PersistState = True
```

To load the AutoPersist state of the DockingManager, call the [LoadDockState](#) method of the DockingManager in its loaded event.

#### C#

```
void DockingManager1_Loaded(object sender, RoutedEventArgs e)
{
    DockingManager1.LoadDockState();
}
```

#### VB.NET

```
Private Sub DockingManager1_Loaded(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    DockingManager1.LoadDockState()
End Sub
```

### Manipulating Save / Load functionalities

#### Serialize a complex layout

DockingManager allows to save a complex layout also. For example, it also saves the complex layout like Nested DockingManager.

#### Serialize the dynamically added children

By default, DockingManager cannot de-serialize its Saved Layout properly, when its child collection is modified after DockState is saved.

Since the DockingManager state persistence feature implemented in such a way that the DockingManager matches the child collection of saved layout with current DockingManager layout internally and loads properly when DockingManager children collection remains same, so when any child collection changes dynamically, it results in an improper layout.

#### Notification for load DockState

When the children collection of DockingManager is changed dynamically after persisting the layout, incorrect layout may load. Success of loading of persisted state can be decided by return value of [LoadDockState](#) method. When the child count is same and then DockingManager checks the Name of the child in the collection. if the Name of the child in loaded state is different from the persisted child in the collection, DockingManager fails to load the saved layout. In such cases, **false** value will be returned by the [LoadDockState](#) method of DockingManager.

#### Various formats to Save / Load states

DockingManager allows to save and load the DockStates of windows in DockingManager in different format.

Some of the formats are:

- IsolatedStorage
- BinaryFormat
- XML file
- XmlWriter

#### Load and save the DockState using Isolated Storage:

DockingManager allows to save and load the dock state from isolated storage.

#### C#

```
// Shows the Isolated storage format.
DockingManager1.LoadDockState();
DockingManager1.SaveDockState();
DockingManager1.ResetState();
```

#### VB.NET

```
' Shows the Isolated storage format.
DockingManager1.LoadDockState()
```

```
DockingManager1.SaveDockState()  
DockingManager1.ResetState()
```

### *Save and Load using XML file*

DockingManager allows to save and load the XML file. It is done using binary formatter and soap formatter. The code example is below:

#### **C#**

```
//Shows serialization methods using XML file.  
BinaryFormatter formatter = new BinaryFormatter();  
DockingManager1.LoadDockState(formatter, StorageFormat.Xml,  
    "\\docking_xml.xml");  
BinaryFormatter formatter = new BinaryFormatter();  
DockingManager1.SaveDockState(formatter, StorageFormat.Xml,  
    "\\docking_xml.xml");  
SoapFormatter formatter1 = new SoapFormatter();  
DocContainer.SaveDockState(formatter1, StorageFormat.Xml,  
    "\\docking_xml_soap.xml");  
SoapFormatter formatter1 = new SoapFormatter();  
DocContainer.SaveDockState(formatter1, StorageFormat.Xml,  
    "\\docking_xml_soap.xml");
```

#### **VB.NET**

```
'Shows serialization methods using XML file.  
Dim formatter As New BinaryFormatter()  
DockingManager1.LoadDockState(formatter, StorageFormat.Xml,  
    "\\docking_xml.xml")  
Dim formatter As New BinaryFormatter()  
DockingManager1.SaveDockState(formatter, StorageFormat.Xml,  
    "\\docking_xml.xml")  
Dim formatter1 As New SoapFormatter()  
DocContainer.SaveDockState(formatter1, StorageFormat.Xml,  
    "\\docking_xml_soap.xml")  
Dim formatter1 As New SoapFormatter()  
DocContainer.SaveDockState(formatter1, StorageFormat.Xml,  
    "\\docking_xml_soap.xml")
```

### *Save and Load using Binary*

DockingManager allows to load and save the dock state in the binary format file.

#### **C#**

```
//Shows the load and save dock state in binary formatter.  
BinaryFormatter format = new BinaryFormatter();  
DockingManager1.LoadDockState(format, StorageFormat.Binary,  
    "\\docking_bin.bin");  
BinaryFormatter format = new BinaryFormatter();  
DockingManager1.SaveDockState(format, StorageFormat.Binary,  
    "\\docking_bin.bin");
```

#### **VB.NET**

```
'Shows the load and save dock state in binary formatter.
```

```
BinaryFormatter format = new BinaryFormatter()
DockingManager1.LoadDockState(format, StorageFormat.Binary, "\\docking_bin.bin")
BinaryFormatter format = new BinaryFormatter()
DockingManager1.SaveDockState(format, StorageFormat.Binary, "\\docking_bin.bin")
```

### Save and Load using XmlWriter

DockingManager allows to load and save the DockState using XMLWriter.

#### C#

```
//Shows the SaveDockState method using XmlWriter.
XmlWriter writer = XmlWriter.Create("DockStates.xml");
DockingManager.SaveDockState(writer);
writer.Close()
//Shows the LoadDockState method using XmlWriter
XmlReader reader = XmlReader.Create("DockStates.xml");
DockingManager.LoadDockState(reader);
reader.Close();
```

#### VB.NET

```
'Shows the SaveDockState method using XmlWriter.
Dim writer As XmlWriter = XmlWriter.Create("DockStates.xml")
DockingManager.SaveDockState(writer)
writer.Close() XmlReader reader = XmlReader.Create("DockStates.xml")
'Shows the LoadDockState method using XmlWriter
DockingManager.LoadDockState(reader)
reader.Close()
```

### Restrict state persistence for specific child

The [CanSerialize](#) attached property of DockingManager decides whether the child can be serialized or not. The default value of the [CanSerialize](#) property is true. When the property is false, while performing deserialization the non-serialized child will move to its default state. This can also be done programmatically by using the [SetCanSerialize](#) function of DockingManager.

#### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True">
<ContentControl syncfusion:DockingManager.Header="Solution Explorer"
x:Name="solutionExplorer" syncfusion:DockingManager.SideInDockedMode="Right"
syncfusion:DockingManager.CanSerialize="False"/>
<ContentControl syncfusion:DockingManager.Header="Start Page"
x:Name="startPage" syncfusion:DockingManager.State="Document" />
<ContentControl syncfusion:DockingManager.Header="Toolbox" x:Name="toolBox"
syncfusion:DockingManager.State="AutoHidden"/>
</syncfusion:DockingManager>
```

#### C#

```
DockingManager.SetCanSerialize(solutionExplorer, false);
```

**Note:** Restrict state persistence does not support to children that were added at run time in DockingManager when performing serialization and de-serialization using [XmlWriter](#).

**Note:** Docking State persistence will be applied to active Docking Children. So it must to load dynamically added controls into DockingManager before applying Deserialization process.

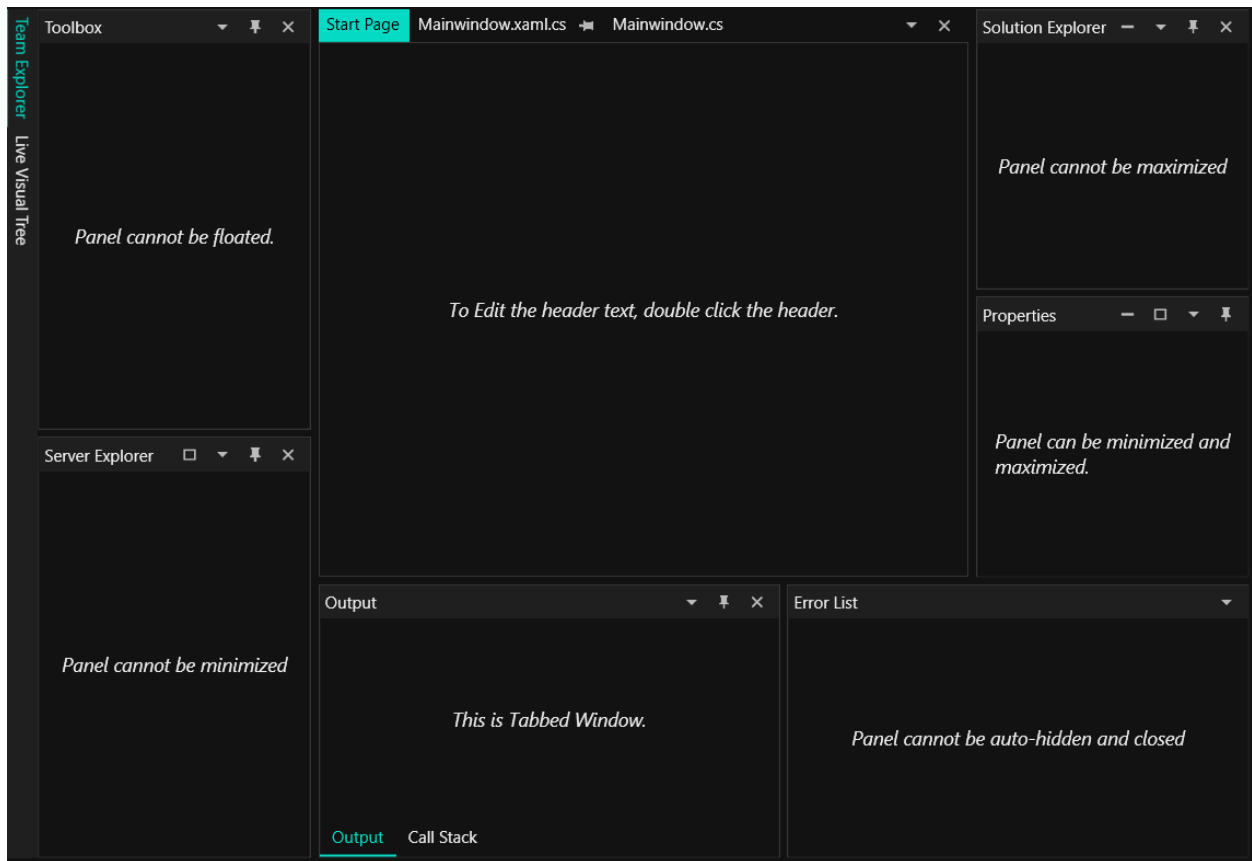
## Styling and Templates in WPF Docking (DockingManager)

The UI for Dock, Float and Document windows of DockingManager can be changed using different Styles and Templates.

### Theme

DockingManager supports various built-in themes. Refer to the below links to apply themes for the DockingManager,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



### Dock Window Style

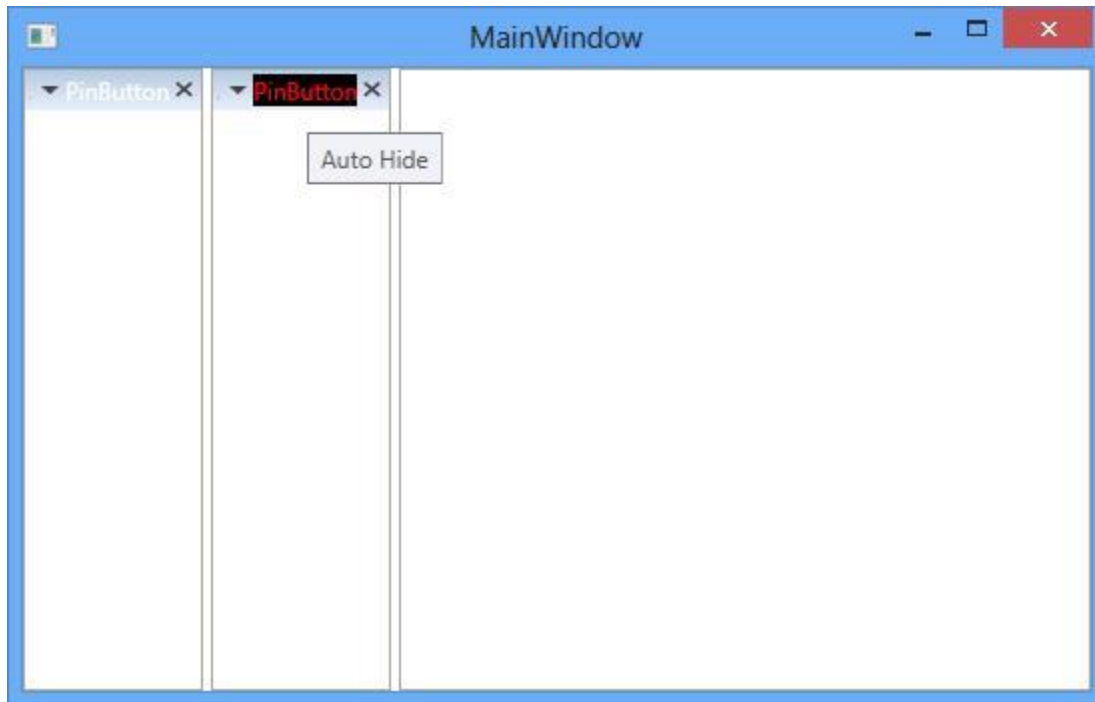
DockingManager allows to set style for some of the Docking controls such as DockedHeaderPresenter, DockedElementTabbedHost. This is explained in detail under the DockHeaderStyle and DockedElementTabbedHostStyle section.

*AwlButtonTemplate*

The style of the pin button of the Dock window can be customized using the `AwlButtonTemplate` property of `DockingManager` with the `TargetType` as `ToggleButton` to have a customized look and feel for the Pin Button.

**XML**

```
<syncfusion:DockingManager x:Name="DockingManager1">
  <syncfusion:DockingManager.AwlButtonTemplate>
    <ControlTemplate TargetType="{x:Type ToggleButton}">
      <Border x:Name="Border1">
        <TextBlock x:Name="Text1" Text="PinButton" Foreground="White"></TextBlock>
      </Border>
      <ControlTemplate.Triggers>
        <Trigger Property="IsMouseOver" Value="True">
          <Setter TargetName="Border1" Property="Background" Value="Black"></Setter>
          <Setter TargetName="Text1" Property="Foreground" Value="Red"></Setter>
        </Trigger>
      </ControlTemplate.Triggers>
    </ControlTemplate>
  </syncfusion:DockingManager.AwlButtonTemplate>
  <ContentControl syncfusion:DockingManager.Header="Child1"/>
  <ContentControl syncfusion:DockingManager.Header="Child2"/>
</syncfusion:DockingManager>
```

*CloseButtonTemplate*

The close button for the Docked window can be customized using the `CloseButtonTemplate` and can be used to get or set the control template for the close button for the windows of `DockingManager` with `TargetType` as `ToggleButton`.

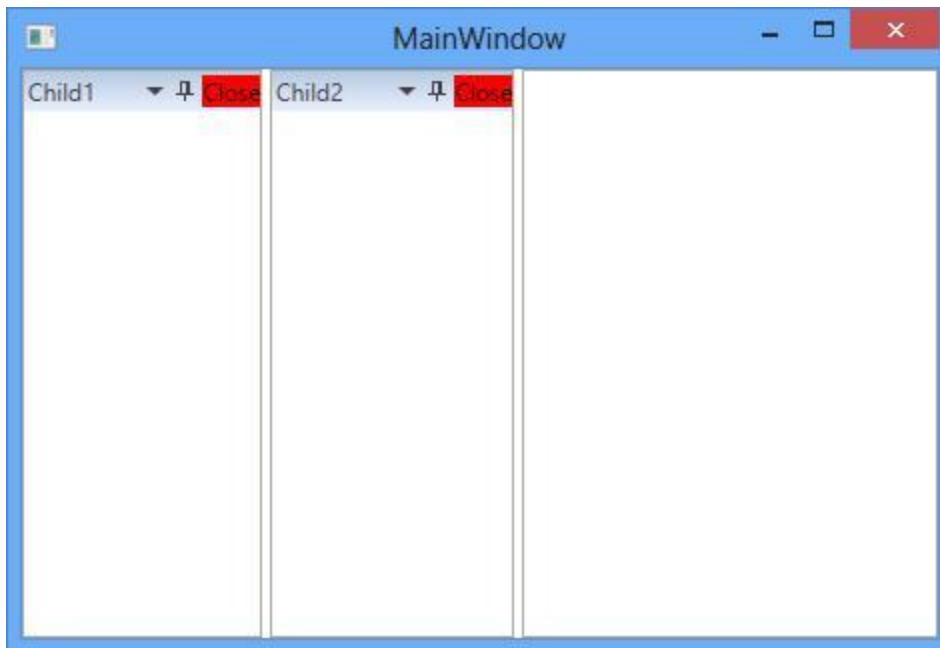
**XML**



```

<syncfusion:DockingManager x:Name="DockingManager1">
  <syncfusion:DockingManager.CloseButtonTemplate>
    <ControlTemplate TargetType="{x:Type ToggleButton}">
      <StackPanel>
        <Border x:Name="Border1">
          <TextBlock x:Name="Block" Background="Red" Text="Close"
            Foreground="Black"></TextBlock>
        </Border>
      </StackPanel>
    </ControlTemplate>
  </syncfusion:DockingManager.CloseButtonTemplate>
  <ContentControl
    syncfusion:DockingManager.Header="Child1"/>
  <ContentControl
    syncfusion:DockingManager.Header="Child2"/>
</syncfusion:DockingManager>

```



### MenuButtonTemplate

The context menu button appearance in the Dock window header can be customized using `MenuButtonTemplate` property by setting with the Target Type as `ToggleButton`.

### XML

```

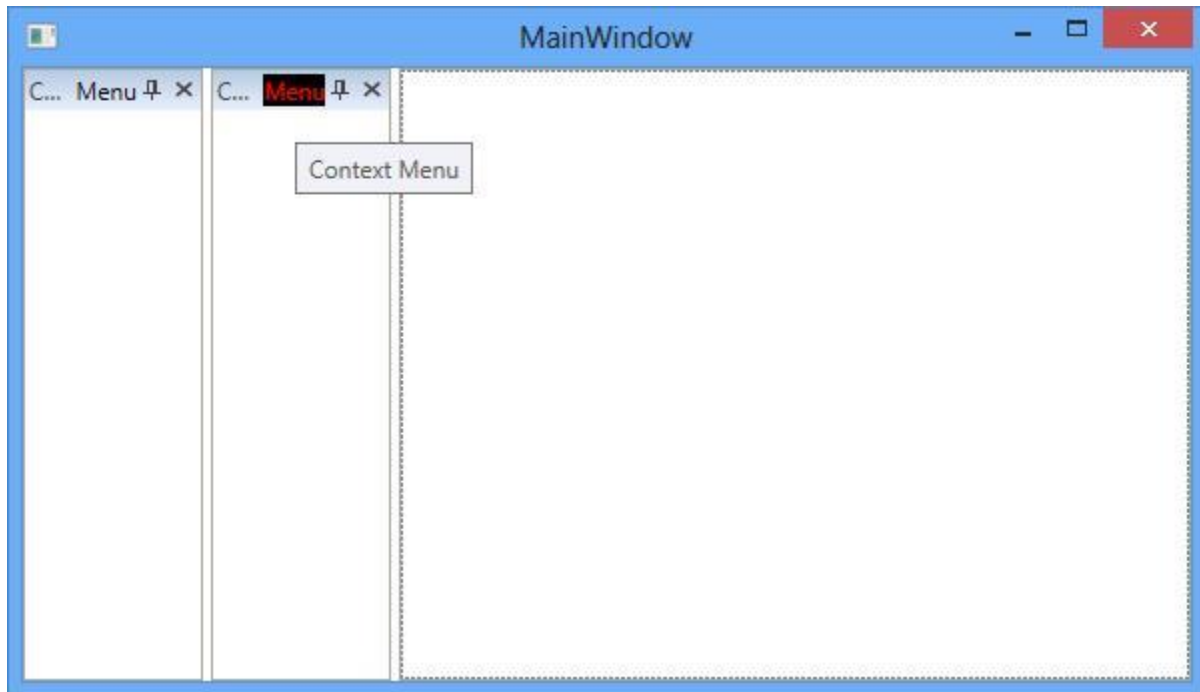
<syncfusion:DockingManager x:Name="DockingManager1">
  <syncfusion:DockingManager.MenuButtonTemplate>
    <ControlTemplate TargetType="{x:Type ToggleButton}">
      <StackPanel>
        <Border x:Name="Border1">
          <TextBlock x:Name="TextBlock" Text="Menu" Foreground="Black"></TextBlock>
        </Border>
      </StackPanel>
      <ControlTemplate.Triggers>
        <Trigger Property="IsMouseOver" Value="True">
          <Setter TargetName="Border1" Property="Background" Value="Black"></Setter>
        </Trigger>
      </ControlTemplate.Triggers>
    </ControlTemplate>
  </syncfusion:DockingManager.MenuButtonTemplate>
  <ContentControl
    syncfusion:DockingManager.Header="Child1"/>
  <ContentControl
    syncfusion:DockingManager.Header="Child2"/>
</syncfusion:DockingManager>

```

```

<Setter TargetName="TextBlock" Property="Foreground" Value="Red"/>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</syncfusion:DockingManager.MenuButtonTemplate>
<ContentControl syncfusion:DockingManager.Header="Child1"/>
<ContentControl syncfusion:DockingManager.Header="Child2"/>
</syncfusion:DockingManager>

```



### MinimizeButton Template

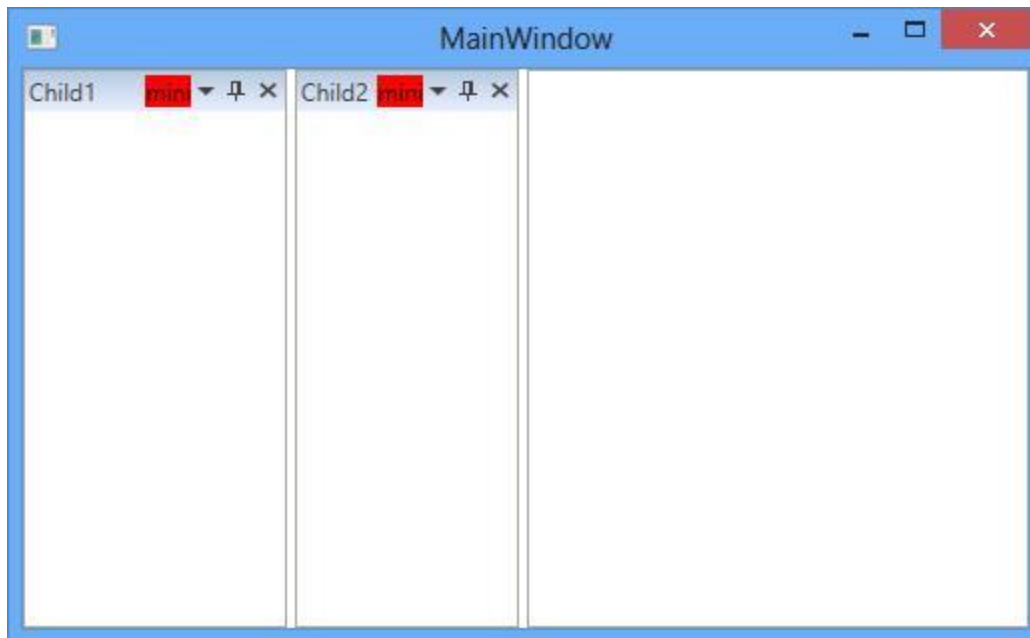
The minimize button for the dock window can be customized using the `MinimizeButtonTemplate` by setting with the `TargetType` as `ToggleButton`.

### XML

```

<syncfusion:DockingManager x:Name="DockingManager1"
MinimizeButtonEnabled="True" >
<syncfusion:DockingManager.MinimizeButtonTemplate>
<ControlTemplate TargetType="{x:Type ToggleButton}">
<Border x:Name="Border1">
<TextBlock x:Name="Block" Background="Red" Text="mini" </TextBlock>
</Border>
<ControlTemplate.Triggers>
<Trigger Property="IsMouseOver" Value="True">
<Setter TargetName="Border1" Property="Background" Value="Black"</Setter>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</syncfusion:DockingManager.MinimizeButtonTemplate>
<ContentControl syncfusion:DockingManager.Header="Child1"/>
<ContentControl syncfusion:DockingManager.Header="Child2"/>
</syncfusion:DockingManager>

```

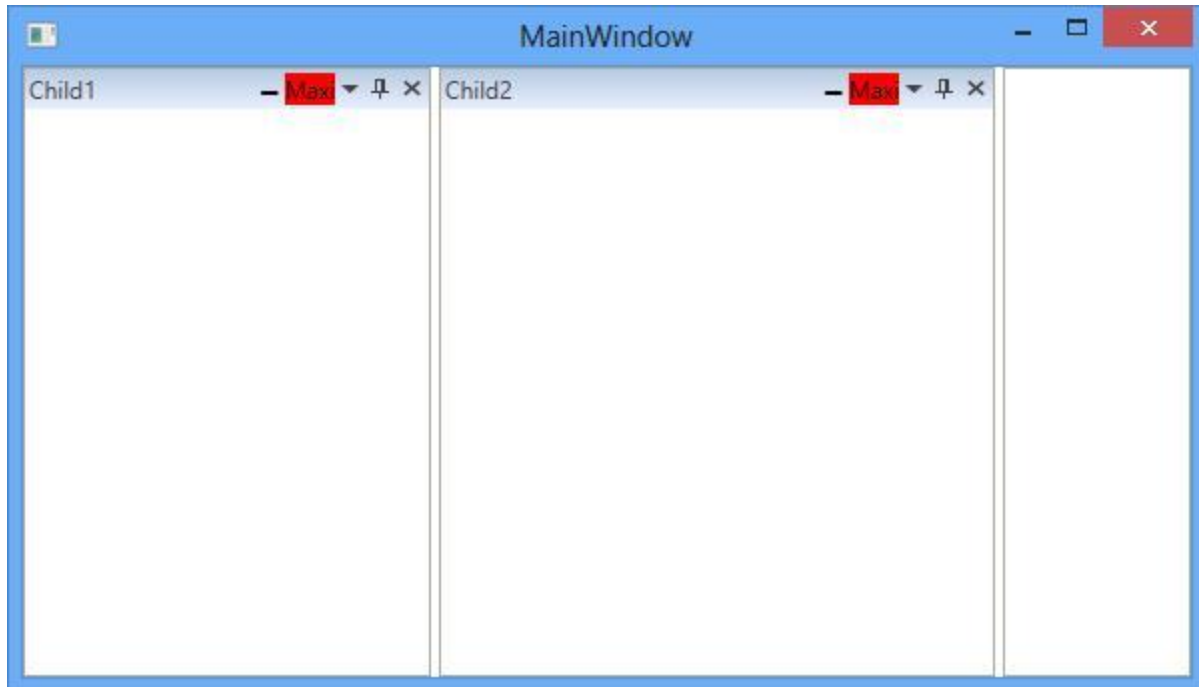


#### MaximizeButtonTemplate

The maximize button for the dock window can be customized using the `MaximizeButtonTemplate` property with the `TargetType` as `ToggleButton`.

#### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
MaximizeButtonEnabled="True" MinimizeButtonEnabled="True" >
  <syncfusion:DockingManager.MaximizeButtonTemplate>
    <ControlTemplate TargetType="{x:Type ToggleButton}">
      <Border x:Name="Border1">
        <TextBlock x:Name="Block" Background="Red" Text="Maxi"/>
      </Border>
      <ControlTemplate.Triggers>
        <Trigger Property="IsMouseOver" Value="True">
          <Setter TargetName="Border1" Property="Background" Value="Black"/>
        </Trigger>
      </ControlTemplate.Triggers>
    </ControlTemplate>
  </syncfusion:DockingManager.MaximizeButtonTemplate>
  <ContentControl syncfusion:DockingManager.Header="Child1"/>
  <ContentControl syncfusion:DockingManager.Header="Child2"/>
</syncfusion:DockingManager>
```

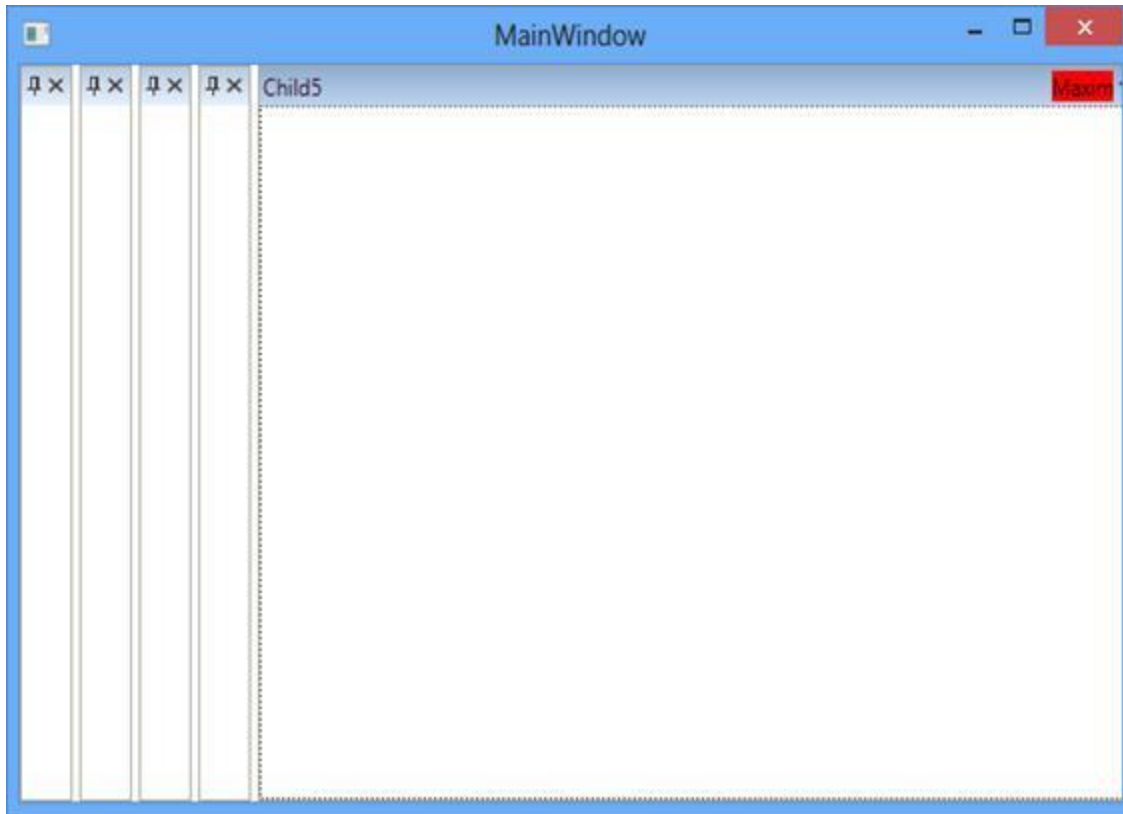


### RestoreButtonTemplate

The Restore button for the dock window can be customized using the `RestoreButtonTemplate` property with the `TargetType` as `ToggleButton`.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
MaximizeButtonEnabled="True">
  <syncfusion:DockingManager.RestoreButtonTemplate>
    <ControlTemplate TargetType="{x:Type ToggleButton}">
      <Border x:Name="Border1">
        <TextBlock x:Name="text1" Background="Red" Text="Maxim"/>
      </Border>
      <ControlTemplate.Triggers>
        <Trigger Property="IsMouseOver" Value="True">
          <Setter TargetName="Border1" Property="Background" Value="Black"></Setter>
        </Trigger>
      </ControlTemplate.Triggers>
    </ControlTemplate>
  </syncfusion:DockingManager.RestoreButtonTemplate>
  <ContentControl syncfusion:DockingManager.Header="Child1"
  syncfusion:DockingManager.State="Dock"/>
  <ContentControl syncfusion:DockingManager.Header="Child2"
  syncfusion:DockingManager.State="Dock"/>
  <ContentControl syncfusion:DockingManager.Header="Child3"
  syncfusion:DockingManager.State="Dock"/>
  <ContentControl syncfusion:DockingManager.Header="Child4"
  syncfusion:DockingManager.State="Dock"/>
  <ContentControl syncfusion:DockingManager.Header="Child5"
  syncfusion:DockingManager.State="Dock"/>
</syncfusion:DockingManager>
```



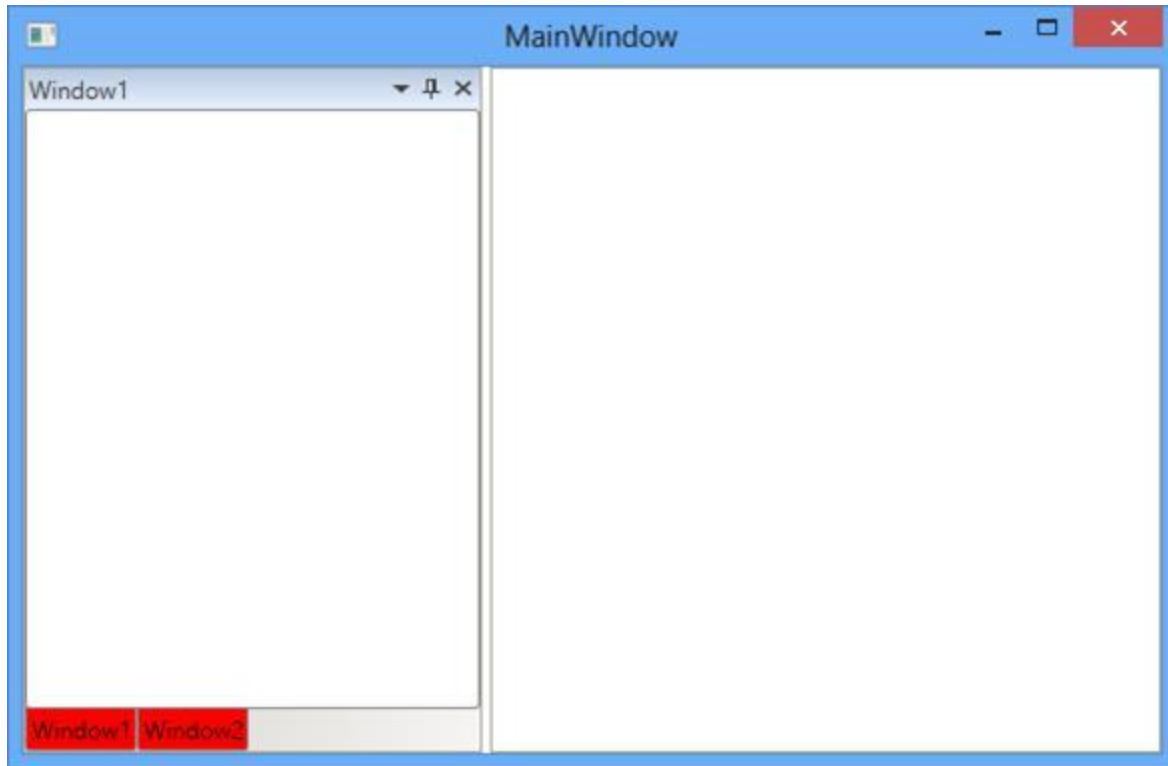
### TabItemTemplate

DockedElementTabbedHost have internal TabControl for Tabbed Windows in DockingManager and its tab item template can be customized using **TabItemTemplate** with the TargetType as TabItem.

### XML

```
<Window.Resources>
<Style x:Key="TabItemStyle1" TargetType="{x:Type TabItem}" >
<Setter Property="Header" Value="{Binding
Path=(Syncfusion:DockingManager.Header)}" />
<Setter Property="Syncfusion:DockingManager.ListenTabItemEvents"
Value="True" />
<Setter Property="MinWidth" Value="21" />
<Setter Property="MinHeight" Value="21" />
<Setter Property="SnapsToDevicePixels" Value="True" />
<Setter Property="Tag" Value="IsInternalTabItem" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type TabItem}">
<Grid Name="Transform">
<Syncfusion:ContextMenuBorder Name="Border" Background="{TemplateBinding
Background}"
BorderBrush="{TemplateBinding BorderBrush}" BorderThickness="1">
<Border.ContextMenu>
<Syncfusion:CustomContextMenu Name="PART_ContextMenu" Focusable="false" />
</Border.ContextMenu>
<Border.LayoutTransform>
<RotateTransform Angle="0" />
</Border.LayoutTransform>
```

```
<DockPanel LastChildFill="True" Background="Red">
<Border Name="Icon" DockPanel.Dock="Left" Margin="1" Width="16"
Background="{Binding
Path=(TabItem.Content).(Syncfusion:DockingManager.Icon),
RelativeSource={RelativeSource TemplatedParent}}" />
<ContentPresenter x:Name="ContentSite" VerticalAlignment="Center"
HorizontalAlignment="Center"
ContentSource="Header" Margin="2,2,2,2" RecognizesAccessKey="True"
ContentTemplate="{Binding
Path=(TabItem.Content).(Syncfusion:DockingManager.HeaderTemplate),
RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
TabItem}}}" />
</DockPanel>
</Syncfusion:ContextMenuBorder>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>
<Grid x:Name="Grid1">
<Syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True" TabItemStyle="{StaticResource TabItemStyle1}">
<ContentControl Syncfusion:DockingManager.Header="Window1" x:Name="Content1"
Syncfusion:DockingManager.State="Dock" />
<ContentControl Syncfusion:DockingManager.Header="Window2"
Syncfusion:DockingManager.State="Dock" x:Name="Content2"
Syncfusion:DockingManager.TargetNameInDockedMode="Content1"
Syncfusion:DockingManager.SideInDockedMode="Tabbed" />
</Syncfusion:DockingManager>
</Grid>
</Window>
```



### *DockedElementTabbedHostStyle*

The DockedElementTabbedHost can be customized using the `DockedElementTabbedHostStyle` with the TargetType as DockedElementTabbedHost

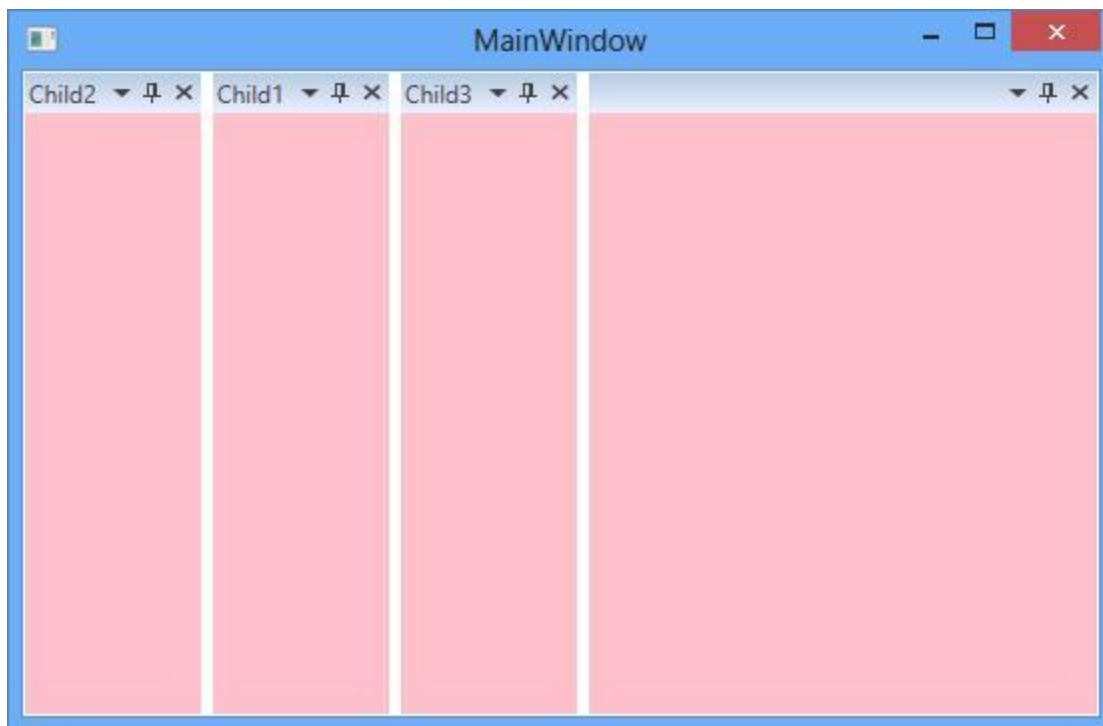
### XML

```
<Syncfusion:DockingManager.DockedElementTabbedHostStyle>
<Style TargetType="{x:Type Syncfusion:DockedElementTabbedHost}">
<Setter Property="Syncfusion:DockingManager.InternalDataContext"
Value="{Binding Path=(Syncfusion:DockedElementTabbedHost.HostedElement),
RelativeSource={RelativeSource Self}}"/>
<Setter Property="FocusVisualStyle" Value="{Binding
Path=(Syncfusion:DockedElementTabbedHost.DockingManager).
(Syncfusion:DockingManager.FocusVisualStyle),RelativeSource={RelativeSource
TemplatedParent}}"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Syncfusion:DockedElementTabbedHost}">
<Border x:Name="BorderWrap" Width="Auto" FocusVisualStyle="{Binding
Path=(Syncfusion:DockedElementTabbedHost.DockingManager).
(Syncfusion:DockingManager.FocusVisualStyle), RelativeSource={RelativeSource
TemplatedParent}}"
SnapsToDevicePixels="True" BorderBrush="{TemplateBinding BorderBrush}"
Background="Pink" BorderThickness="1">
<DockPanel x:Name="DockPanel" Width="Auto" LastChildFill="True"
Focusable="False">
<Syncfusion:DockHeaderPresenter x:Name="header" DockPanel.Dock="Top"
RenderTransformOrigin="0.5,0.5"
Style="{Binding
Path=(Syncfusion:DockedElementTabbedHost.DockingManager). (Syncfusion:Docking
Manager.DockHeaderStyle),
```

```

RelativeSource={RelativeSource AncestorType={x:Type
Syncfusion:DockedElementTabbedHost}}}}"
IsTemplateParenKeyboardFocusWithin="{TemplateBinding
IsKeyboardFocusWithin}"/>
<Grid>
<ContentPresenter Name="HostedElement" ClipToBounds="True"
ContentSource="HostedElement"
ContentTemplate="{TemplateBinding ContentControl.ContentTemplate}"
ContentTemplateSelector="{TemplateBinding
ContentControl.ContentTemplateSelector}"/>
<Border Name="PART_CoverletControl" Visibility="Collapsed"
Background="Transparent" />
</Grid>
</DockPanel>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Syncfusion:DockingManager.DockedElementTabbedHostStyle>

```



#### Custom header for all child

A common header style for all dock window can be customized using the property `DockHeaderStyle` with the `TargetType` as `DockHeaderPresenter`.

#### XML

```

<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True" >
<syncfusion:DockingManager.DockHeaderStyle>
<Style TargetType="{x:Type syncfusion:DockHeaderPresenter}">

```



```
<Setter Property="MinHeight" Value="60"></Setter>
</Style>
</syncfusion:DockingManager.DockHeaderStyle>
<ContentControl x:Name="Content1" syncfusion:DockingManager.State="Dock"
syncfusion:DockingManager.Header="Child1" />
<ContentControl x:Name="Content2" syncfusion:DockingManager.Header="Child2"
/>
</syncfusion:DockingManager>
```



#### Custom header for individual child

Header of individual dock child can be customized through the [HeaderStyle](#) property of DockingManager. The customized style will apply to dock, auto hidden and tabbed windows. The following code illustrate the dock windows with customized header style,

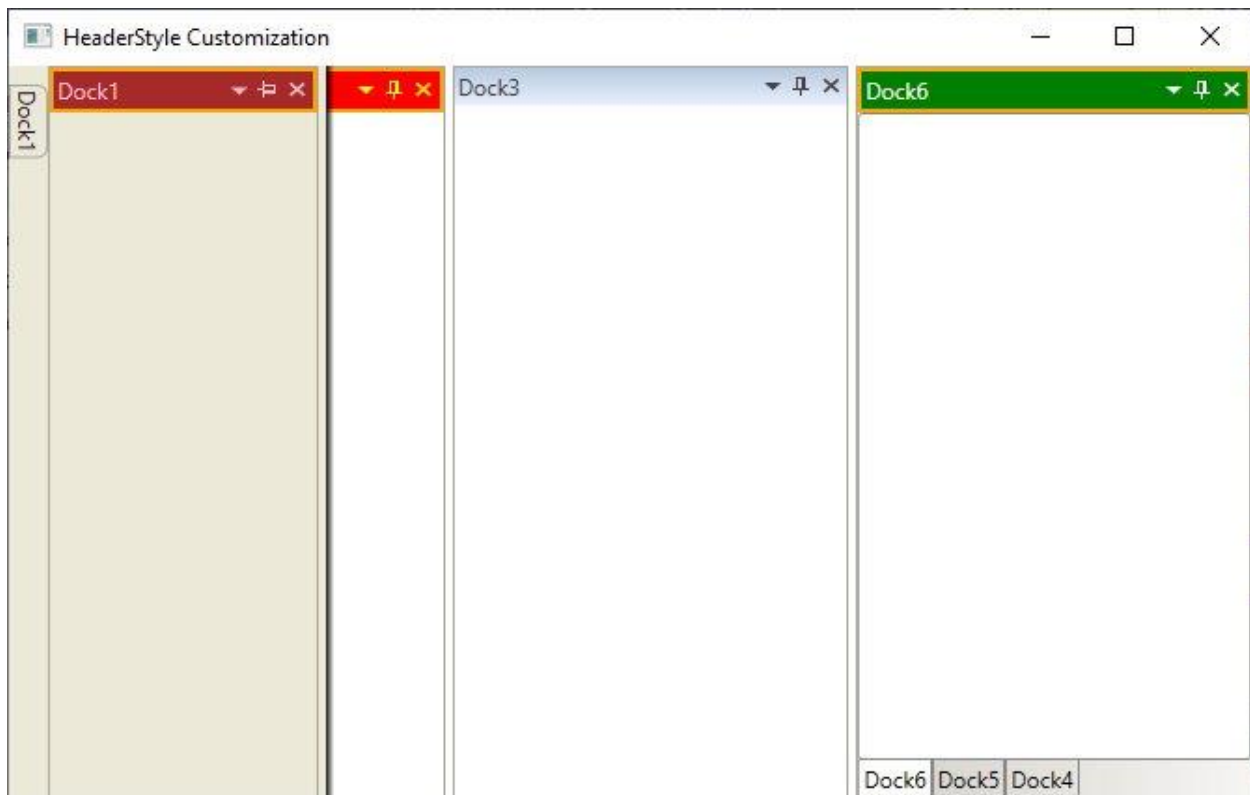
#### XML

```
<Window.Resources>
<Style TargetType="Syncfusion:DockHeaderPresenter" x:Key="headerStyle1" >
<Setter Property="Background" Value="Red"/>
<Setter Property="Foreground" Value="Yellow"/>
<Setter Property="BorderBrush" Value="Orange"/>
<Setter Property="BorderThickness" Value="2"/>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=IsTemplateParentKeyboardFocusWithin,
RelativeSource={RelativeSource Self}}"
Value="True">
<Setter Property="Foreground"
Value="White" />
<Setter Property="Background"
Value="Green" />
</DataTrigger>
<MultiDataTrigger>
<MultiDataTrigger.Conditions>
<Condition Binding="{Binding Path=IsMouseOver
, RelativeSource={RelativeSource Self}}"
Value="True" />
</MultiDataTrigger.Conditions>
```

```

<Setter Property="Foreground"
Value="Pink"/>
<Setter Property="Background"
Value="Brown"/>
</MultiDataTrigger>
</Style.Triggers>
</Style>
</Window.Resources>
<Syncfusion:DockingManager Grid.Row="1" x:Name="dockingManager"
DockFill="True">
<ContentControl Syncfusion:DockingManager.Header="Dock1"
Syncfusion:DockingManager.HeaderStyle="{StaticResource headerStyle1}"/>
<ContentControl Syncfusion:DockingManager.Header="Dock2"
Syncfusion:DockingManager.HeaderStyle="{StaticResource headerStyle1}"/>
<ContentControl Syncfusion:DockingManager.Header="Dock3"/>
<ContentControl Syncfusion:DockingManager.Header="Dock4"
Syncfusion:DockingManager.HeaderStyle="{StaticResource headerStyle1}"/>
<ContentControl Syncfusion:DockingManager.Header="Dock5"
Syncfusion:DockingManager.HeaderStyle="{StaticResource headerStyle1}"/>
<ContentControl Syncfusion:DockingManager.Header="Dock6"
Syncfusion:DockingManager.HeaderStyle="{StaticResource headerStyle1}"/>
</Syncfusion:DockingManager>

```



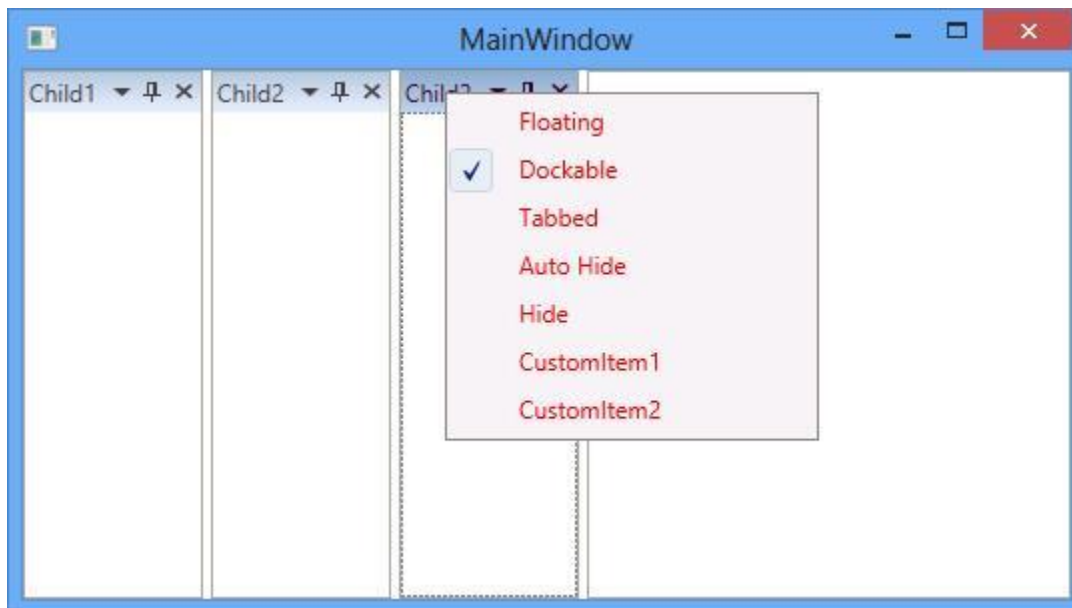
Refer to this [sample](#) to know how the header of the dock windows are customized through style of the Header.

*DockWindowContextMenuStyle*

The context menu of DockingManager can be customized using the `DockWindowContextMenuStyle` by setting its Target Type as CustomMenuItem.

**XML**

```
<Syncfusion:DockingManager UseDocumentContainer="True" ContainerMode="TDI"
UseNativeFloatWindow="True">
  <Syncfusion:DockingManager.DockWindowContextMenuStyle>
    <Style TargetType="{x:Type Syncfusion:CustomMenuItem}">
      <Setter Property="Foreground" Value="Red"></Setter>
    </Style>
  </Syncfusion:DockingManager.DockWindowContextMenuStyle>
  <Syncfusion:DockingManager.CustomMenuItems>
    <Syncfusion:CustomMenuItemCollection>
      <Syncfusion:CustomMenuItem Header="CustomItem1"/>
      <Syncfusion:CustomMenuItem Header="CustomItem2"/>
    </Syncfusion:CustomMenuItemCollection>
  </Syncfusion:DockingManager.CustomMenuItems>
  <ContentControl Syncfusion:DockingManager.Header="Child1"
Syncfusion:DockingManager.State="Dock"/>
  <ContentControl Syncfusion:DockingManager.Header="Child2"
Syncfusion:DockingManager.State="Dock"/>
  <ContentControl Syncfusion:DockingManager.Header="Child3"
Syncfusion:DockingManager.State="Dock"/>
</Syncfusion:DockingManager>
```

*Float Window Style*

The float window can be customized using its template and style.

*FloatWindowTemplate*

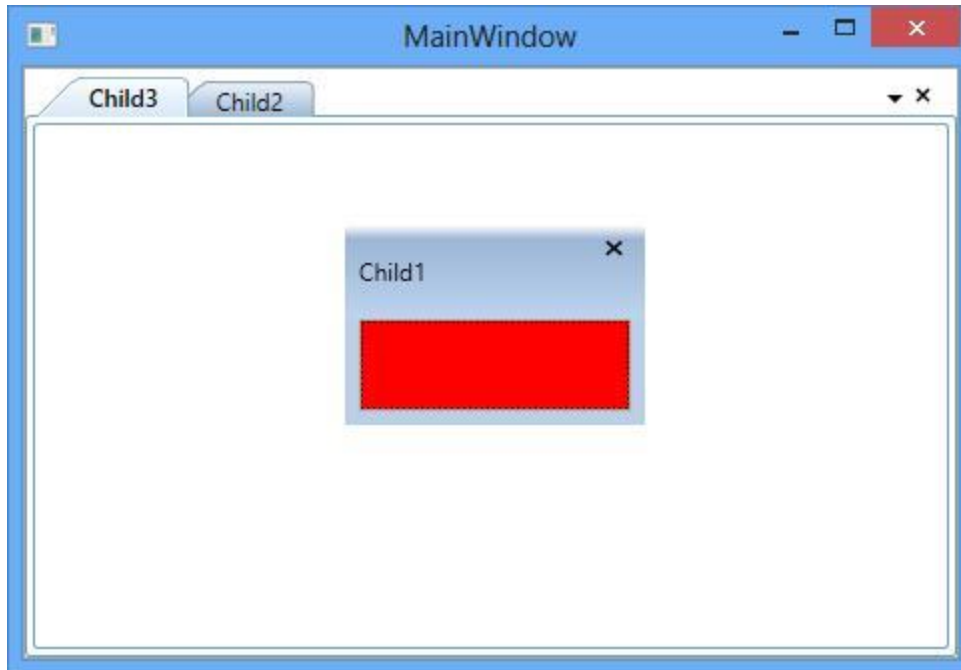
Template of FloatWindow can be customized using the `FloatWindowTemplate` with the TargetType as ContentControl.

**XML**

```

<Syncfusion:DockingManager UseDocumentContainer="True" ContainerMode="TDI"
EnableDocumentTabHeaderEdit="True">
<Syncfusion:DockingManager.FloatWindowTemplate>
<ControlTemplate TargetType="{x:Type ContentControl}">
<AdornerDecorator>
<DockPanel Focusable="False" LastChildFill="True" Opacity="{Binding
Path=Opacity,
RelativeSource={RelativeSource AncestorType={x:Type Syncfusion:IWindow}}}">
<Border Name="FloatWindowOutBorder" Focusable="False"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}" Background="Red" >
<Grid Focusable="False">
<Grid.RowDefinitions>
<RowDefinition Name="TopRow"/>
<RowDefinition Height="*" />
<RowDefinition Name="BottomRow" Height="7" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Name="LeftCol" Width="7" />
<ColumnDefinition Width="*" />
<ColumnDefinition Name="RightCol" Width="7" />
</Grid.ColumnDefinitions>
<Syncfusion:FloatWindowBorder BorderMode="LeftTop" Name="BorderLeftTop"
Grid.Column="0" Grid.Row="0" />
<Syncfusion:FloatWindowBorder BorderMode="Header" Name="BorderHeader"
Grid.Column="1" Grid.Row="0" />
<Syncfusion:FloatWindowBorder BorderMode="RightTop" Name="BorderRightTop"
Grid.Column="2" Grid.Row="0" />
<Syncfusion:FloatWindowBorder BorderMode="Left" Name="BorderLeft"
Grid.Column="0" Grid.Row="1" />
<ContentPresenter Name="ContentPresenter" Grid.Column="1" Grid.Row="1"
ContentTemplate="{TemplateBinding ContentControl.ContentTemplate}"
Content="{TemplateBinding ContentControl.Content}" />
<Syncfusion:FloatWindowBorder BorderMode="Right" Name="BorderRight"
Grid.Column="2" Grid.Row="1" />
<Syncfusion:FloatWindowBorder BorderMode="LeftBottom"
Name="BorderLeftBottom" Grid.Column="0" Grid.Row="2" />
<Syncfusion:FloatWindowBorder BorderMode="Bottom" Name="BorderBottom"
Grid.Column="1" Grid.Row="2" />
<Syncfusion:FloatWindowBorder BorderMode="RightBottom"
Name="BorderRightBottom" Grid.Column="2" Grid.Row="2" />
</Grid>
</Border>
</DockPanel>
</AdornerDecorator>
</ControlTemplate>
</Syncfusion:DockingManager.FloatWindowTemplate>
<ContentControl Syncfusion:DockingManager.Header="Child1"
Syncfusion:DockingManager.State="Float"/>
<ContentControl Syncfusion:DockingManager.Header="Child2"
Syncfusion:DockingManager.State="Document"/>
<ContentControl Syncfusion:DockingManager.Header="Child3"
Syncfusion:DockingManager.State="Document"/>
</Syncfusion:DockingManager>

```



### *FloatWindowStyle*

The style for the float window can be applied using the `FloatWindowStyle` property of the `DockingManager` by setting its `TargetType` as `AutoTemplatedContentControl`. It gets or sets the style for the `FloatWindow` when it is rendered.

### **XML**

```
<syncfusion:DockingManager x:Name="DockingManager1">
  <syncfusion:DockingManager.FloatWindowStyle>
    <Style TargetType="{x:Type syncfusion:AutoTemplatedContentControl}">
      <Setter Property="Background" Value="Red"></Setter>
    </Style>
  </syncfusion:DockingManager.FloatWindowStyle>
  <ContentControl syncfusion:DockingManager.Header="Child"
    syncfusion:DockingManager.State="Float"
    syncfusion:DockingManager.SideInDockedMode="Left"/>
  <ContentControl syncfusion:DockingManager.Header="Child2"></ContentControl>
</syncfusion:DockingManager>
```

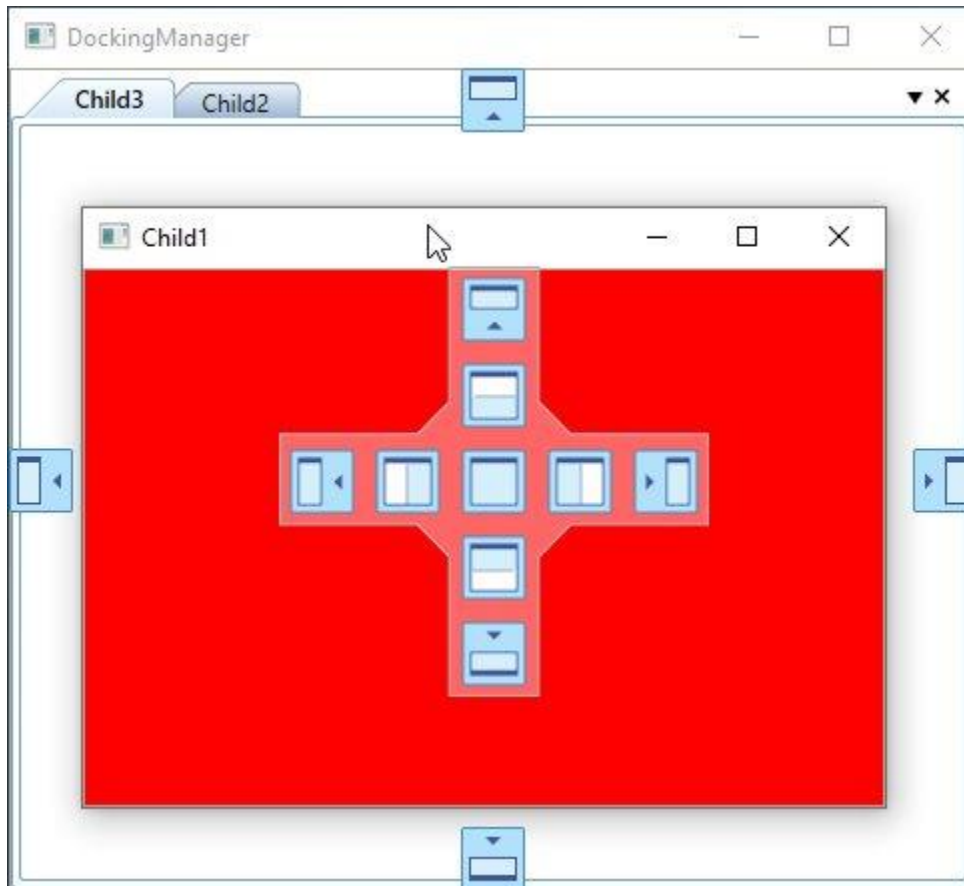
### *NativeWindowStyle*

The `NativeFloatWindow` of `DockingManager` can be customized using the `NativeWindowStyle` property of the `DockingManager` with the `TargetType` as `NativeFloatWindow`.

### **XML**

```
<Syncfusion:DockingManager UseDocumentContainer="True" ContainerMode="TDI"
  UseNativeFloatWindow="True">
  <Syncfusion:DockingManager.NativeWindowStyle>
    <Style TargetType="{x:Type Syncfusion:NativeFloatWindow}">
      <Setter Property="Background" Value="Red"></Setter>
    </Style>
  </Syncfusion:DockingManager.NativeWindowStyle>
```

```
<ContentControl Syncfusion:DockingManager.Header="Child1"
Syncfusion:DockingManager.State="Float"/>
<ContentControl Syncfusion:DockingManager.Header="Child2"
Syncfusion:DockingManager.State="Document"/>
<ContentControl Syncfusion:DockingManager.Header="Child3"
Syncfusion:DockingManager.State="Document"/>
</Syncfusion:DockingManager>
```



### Auto Hide Window Style

AutoHidden window is constructed by SidePanels and SideTabItems. To customize the AutoHidden window, the following style and Templates are used as follows:

#### SideTabItemTemplate

The template of the SideTabItem of the DockingManager can be customized using the SideTabItemTemplate with the TargetType as TabItem.

#### XML

```
<ControlTemplate x:Key="SideTabItemTemplate" TargetType="{x:Type TabItem}">
<Border Name="Border" CornerRadius="5,5,0,0"
BorderBrush="{StaticResource TabItemBorderBrush}" BorderThickness="1">
<Border.LayoutTransform>
<RotateTransform Angle="90" />
</Border.LayoutTransform>
<Grid>
<DockPanel LastChildFill="True" Background="Orange">
```

```

<Rectangle Fill="Transparent" Height="Auto" Width="Auto" RadiusX="5"
RadiusY="5" />
<Border Name="Icon" DockPanel.Dock="Left" Margin="1" Width="16"
Background="{Binding Path=(TabItem.Content) .
(Syncfusion:DockingManager.Icon), RelativeSource={RelativeSource
TemplatedParent}}" />
<ContentPresenter x:Name="ContentSite" VerticalAlignment="Center"
HorizontalAlignment="Center"
ContentSource="Header" Margin="2,2,2,2" RecognizesAccessKey="True"
ContentTemplate="{Binding
Path=(Border.DataContext) . (Syncfusion:DockingManager.HeaderTemplate) ,
ElementName=Border}" ContentTemplateSelector="{StaticResource
TabItemTrimmingTemplate}"
TextBlock.Foreground={StaticResource Default.TabForeground}"/>
</DockPanel>
</Grid>
</Border>
</ControlTemplate>

```



### *SideItemStyle*

The style for the sideTabItem can be changed using `SideItemStyle` by settings its TargetType as TabItem

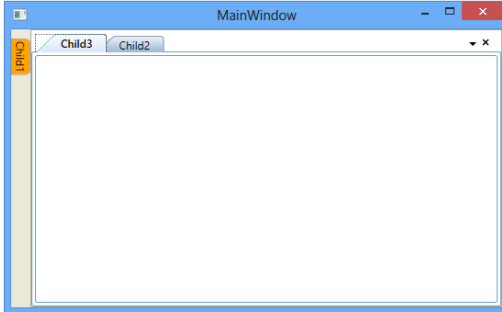
### XML

```

<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True" ContainerMode="TDI" >
<syncfusion:DockingManager.SideItemStyle>
<Style TargetType="{x:Type TabItem}">
<Setter Property="Background" Value="Red"></Setter>
<Setter Property="MinHeight" Value="500"></Setter>
<Setter Property="MinWidth" Value="200"></Setter>
</Style>
</syncfusion:DockingManager.SideItemStyle>

```

```
<ContentControl syncfusion:DockingManager.Header="Child1"
syncfusion:DockingManager.State="Dock"/>
<ContentControl syncfusion:DockingManager.Header="Child2"
syncfusion:DockingManager.State="AutoHidden"/>
<ContentControl syncfusion:DockingManager.Header="Child3"
syncfusion:DockingManager.State="Dock"/>
</syncfusion:DockingManager>
```



### SidePanelTemplate

SidePanel template in AutoHidden window can be changed using **SidePanelTemplate** with the TargetType SidePanel.

### XML

```
<ControlTemplate x:Key="SidePanelTemplate" TargetType="{x:Type
Syncfusion:SidePanel}" >
<Canvas KeyboardNavigation.TabNavigation="Local" ClipToBounds="False" >
<Border Name="PART_BorderName" Height="{TemplateBinding ActualHeight}"
Width="{TemplateBinding ActualWidth}"
Margin="0" ClipToBounds="True" Background="Orange" Panel.ZIndex="1"
KeyboardNavigation.TabIndex="1"
BorderThickness="{Binding Path=SidePanelBorderThickness,
RelativeSource={RelativeSource FindAncestor,
AncestorType={x:Type Syncfusion:DockingManager}}}"
BorderBrush="{StaticResource TabItemBorderBrush}">
<Border.ContextMenu>
<ContextMenu Style="{StaticResource SideContextMenuStyle}"
ItemsSource="{TemplateBinding TabChildren}"/>
</Border.ContextMenu>
<Syncfusion:DirectTabPanel Name="PART_PanelName"
KeyboardNavigation.TabNavigation="Local" IsItemsHost="True"/>
</Border>
<Syncfusion:OpacityDockPanel LastChildFill="True"
Opacity="{Binding Path=ContentOpacity,
RelativeSource={RelativeSource TemplatedParent}}"/>
</ Canvas >
</ControlTemplate>
```



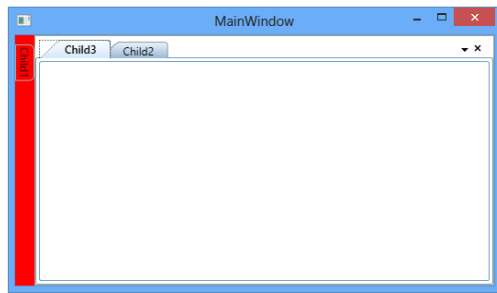


### *SidePanelStyle*

The style for the SidePanel of the DockingManager can be customized using the `SidePanelStyle` property with the TargetType as SidePanel.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
    UseDocumentContainer="True" ContainerMode="TDI">
    <syncfusion:DockingManager.SidePanelStyle>
    <Style TargetType="{x:Type syncfusion:SidePanel}">
    <Setter Property="Background" Value="Red"></Setter>
    </Style>
    </syncfusion:DockingManager.SidePanelStyle>
    <ContentControl syncfusion:DockingManager.Header="Child1"
    syncfusion:DockingManager.State="AutoHidden"/>
    <ContentControl
    syncfusion:DockingManager.Header="Child2" syncfusion:DockingManager.State="Do
    cument"/>
    <ContentControl
    syncfusion:DockingManager.Header="Child3" syncfusion:DockingManager.State="Do
    cument"/>
    </syncfusion:DockingManager>
```



### Drag Provider Style

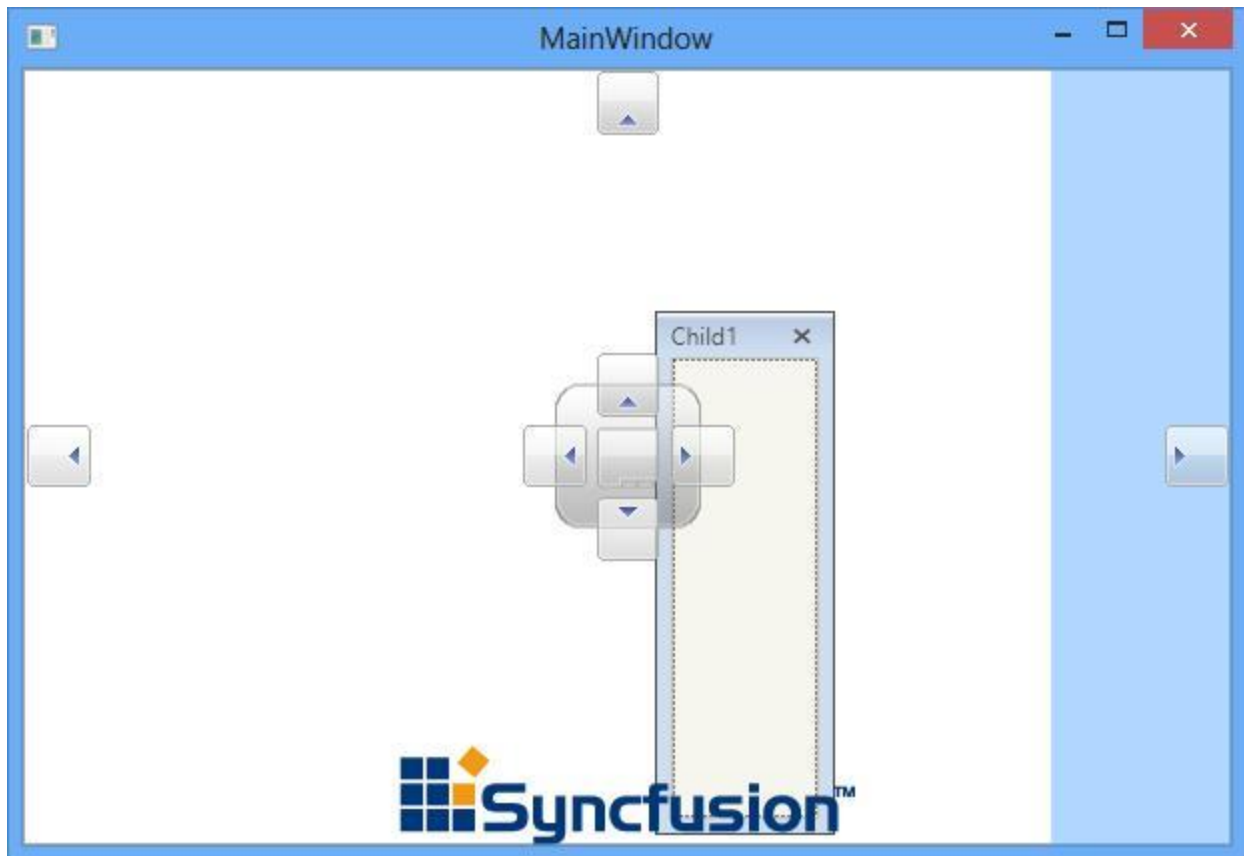
DragProviders appear while dragging a window. DragProviders are displayed for providing options to dock the FloatWindow while dragging and this drag provider button templates can be changed by the following templates.

#### BottomDragProvider

The BottomDragProvider is used to dock the children of the Docking in the bottom side and its template can be customized using the `BottomDragProviderTemplate` by setting its `TargetType` as `ContentControl`.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True" EnableDocumentTabHeaderEdit="True" >
<syncfusion:DockingManager.BottomDragProvider>
<ControlTemplate TargetType="{x:Type ContentControl}">
<Image Name="Img"
syncfusion:DockPreviewManagerVS2005.ProviderAction="GlobalLeft"
Source="Images\synclogo.png" />
<ControlTemplate.Triggers>
<DataTrigger Binding="{Binding Path=IsSideButtonActive,
RelativeSource={RelativeSource FindAncestor,
AncestorType={x:Type syncfusion:DockPreviewMainButtonVS2005}}}"
Value="true">
<Setter TargetName="Img" Property="Source" Value="Images\synclogo.png"/>
</DataTrigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</syncfusion:DockingManager.BottomDragProvider>
<ContentControlsyncfusion:DockingManager.Header="Child1"
syncfusion:DockingManager.State="Dock"/>
</syncfusion:DockingManager>
```

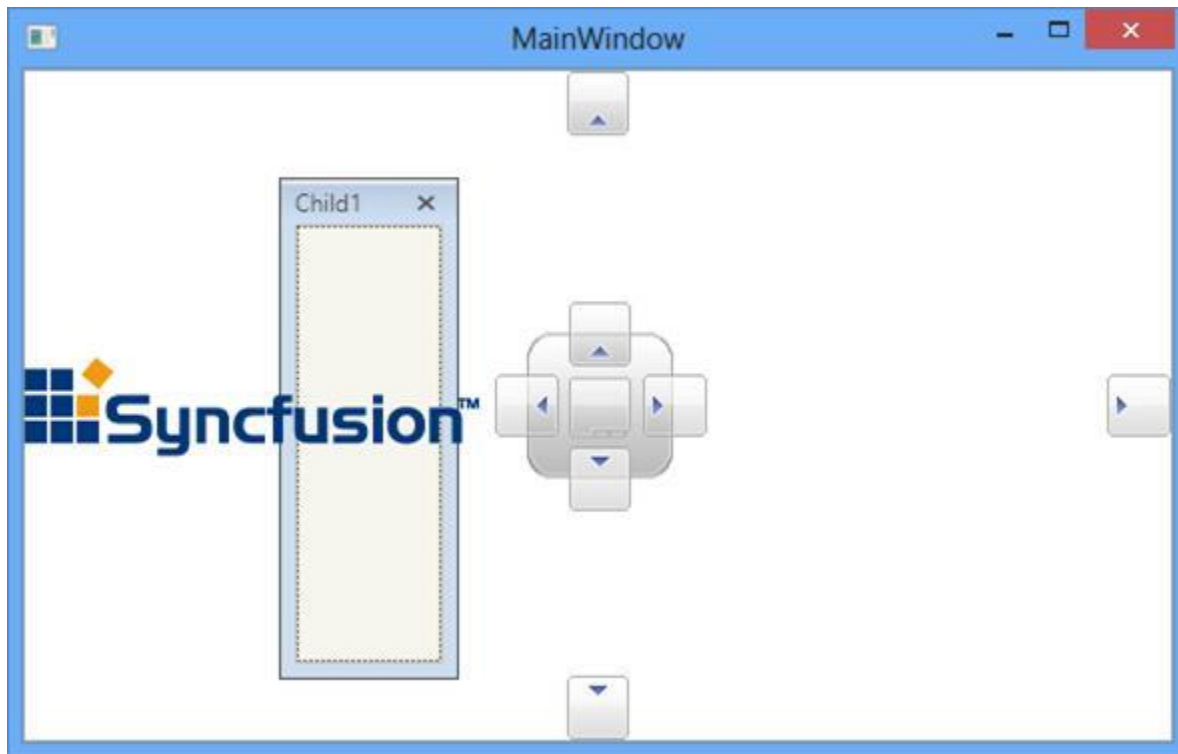


### *LeftDragProvider*

The `LeftDragProvider` is used to dock the dock window to the left side and it can be customized using the `LeftDragProvider` property that helps to customize the template for the Left DragProvider by setting its `Target` as `ContentControl`. The same has been explained below:

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True" EnableDocumentTabHeaderEdit="True" >
  <syncfusion:DockingManager.LeftDragProvider>
    <ControlTemplate TargetType="{x:Type ContentControl}">
      <Image Name="Img"
syncfusion:DockPreviewManagerVS2005.ProviderAction="GlobalLeft"
Source="Images\sinclogo.png" />
      <ControlTemplate.Triggers>
        <DataTrigger Binding="{Binding Path=IsSideButtonActive,
RelativeSource={RelativeSource FindAncestor,
AncestorType={x:Type syncfusion:DockPreviewMainButtonVS2005}}}"
Value="true">
          <Setter TargetName="Img" Property="Source" Value="Images\sinclogo.png"/>
        </DataTrigger>
      </ControlTemplate.Triggers>
    </ControlTemplate>
  </syncfusion:DockingManager.LeftDragProvider>
  <ContentControl syncfusion:DockingManager.Header="Child1"
syncfusion:DockingManager.State="Dock"/>
</syncfusion:DockingManager>
```

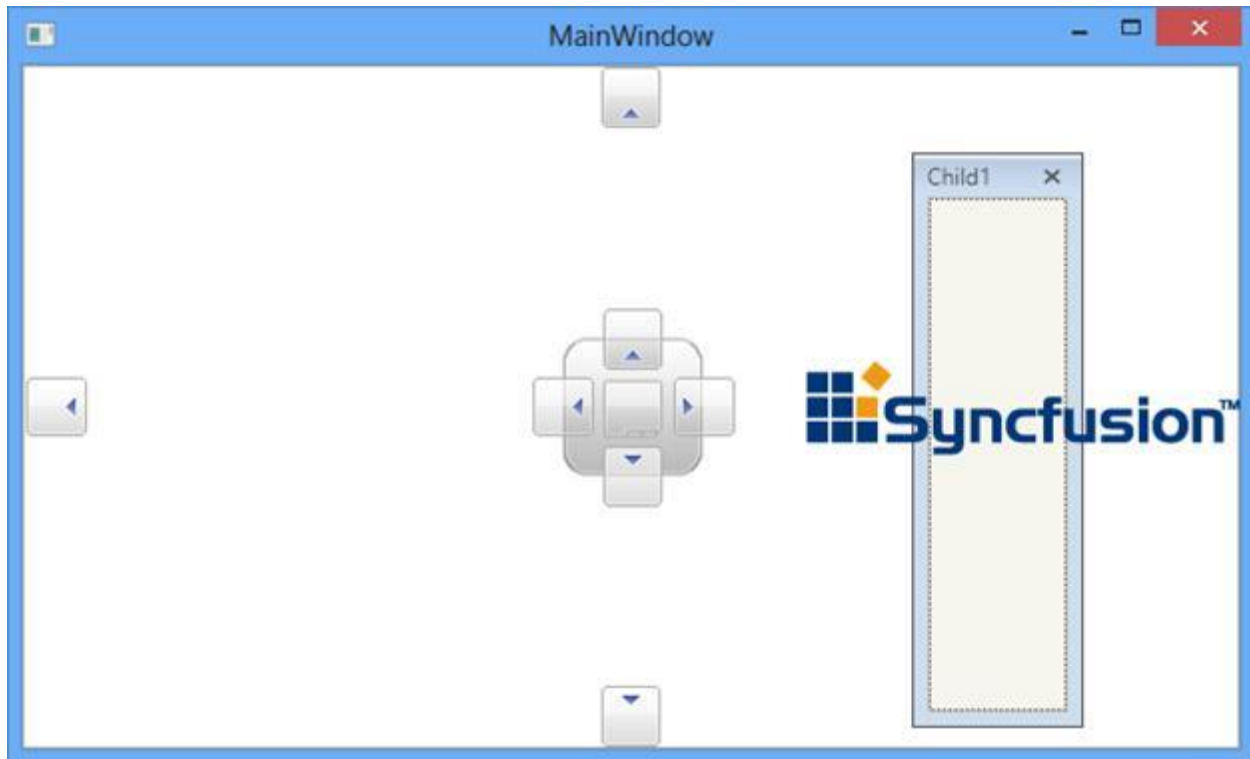


### *RightDragProvider*

To dock the DockWindow to the right side, *RightDragProvider* is used. It can be customized using the *RightDragProviderTemplate* with the *TargetType* as *ContentControl*.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True" EnableDocumentTabHeaderEdit="True" >
  <syncfusion:DockingManager.RightDragProvider>
    <ControlTemplate TargetType="{x:Type ContentControl}">
      <Image Name="Img"
syncfusion:DockPreviewManagerVS2005.ProviderAction="GlobalLeft"
Source="Images\synclogo.png" />
      <ControlTemplate.Triggers>
        <DataTrigger Binding="{Binding Path=IsSideButtonActive,
RelativeSource={RelativeSource FindAncestor,
AncestorType={x:Type syncfusion:DockPreviewMainButtonVS2005}}}"
Value="true">
          <Setter TargetName="Img" Property="Source" Value="Images\synclogo.png"/>
        </DataTrigger>
      </ControlTemplate.Triggers>
    </ControlTemplate>
  </syncfusion:DockingManager.RightDragProvider>
  <ContentControl syncfusion:DockingManager.Header="Child1"
syncfusion:DockingManager.State="Dock"/>
</syncfusion:DockingManager>
```

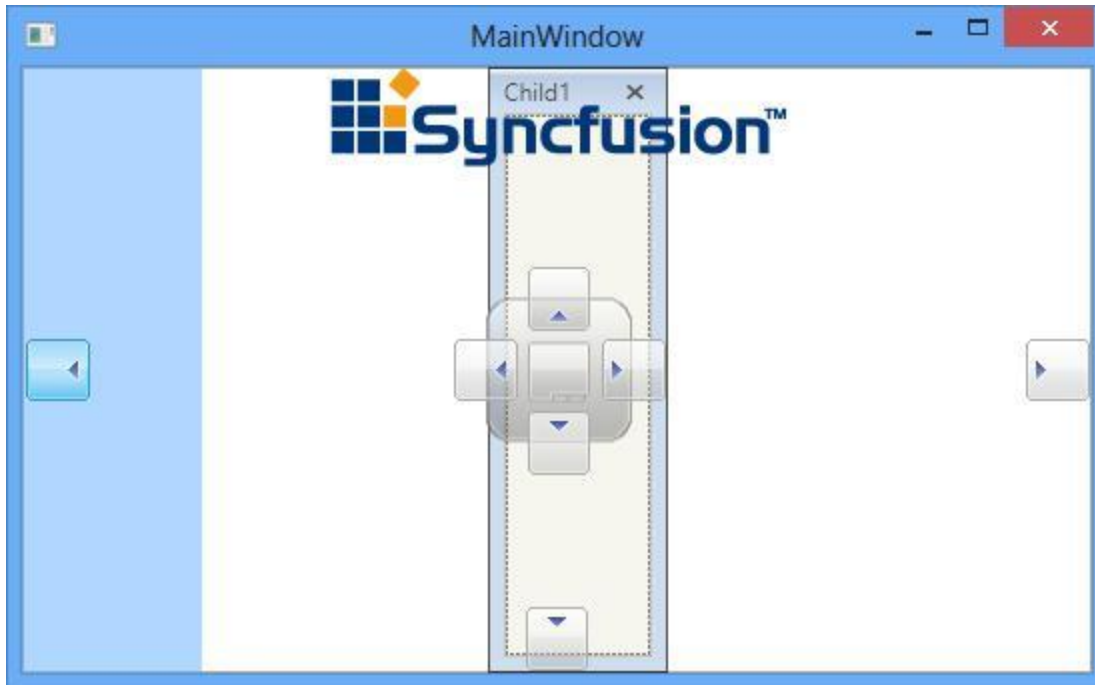


### *TopDragProvider*

To dock the DockWindow at top, the TopDragProvider is used. It can be customized using the `TopDragProviderTemplate` with the TargetType as ContentControl.

### **XML**

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True" EnableDocumentTabHeaderEdit="True" >
<syncfusion:DockingManager.LeftDragProvider>
<ControlTemplate TargetType="{x:Type ContentControl}">
<Image Name="Img"
syncfusion:DockPreviewManagerVS2005.ProviderAction="GlobalLeft"
Source="Images\synclogo.png" />
<ControlTemplate.Triggers>
<DataTrigger Binding="{Binding Path=IsSideButtonActive,
RelativeSource={RelativeSource FindAncestor,
AncestorType={x:Type syncfusion:DockPreviewMainButtonVS2005}}}"
Value="true">
<Setter TargetName="Img" Property="Source" Value="Images\synclogo.png"/>
</DataTrigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</syncfusion:DockingManager.LeftDragProvider>
<ContentControl syncfusion:DockingManager.Header="Child1"
syncfusion:DockingManager.State="Dock"/>
</syncfusion:DockingManager>
```

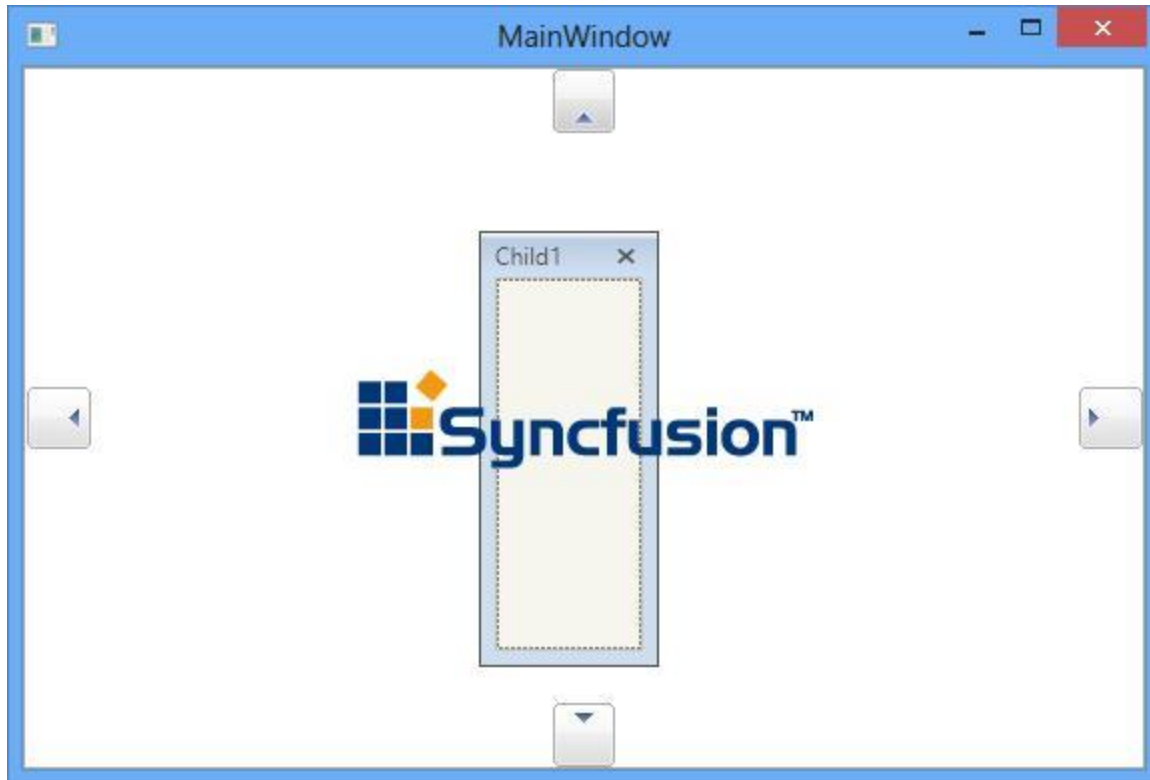


### CenterDragProvider

CenterDragProvider is used to dock the children of the DockingManager at the center Position and its template can be customized using the property `CenterDragProvider`.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
    UseDocumentContainer="True">
    <syncfusion:DockingManager.CenterDragProvider>
    <ControlTemplate TargetType="{x:Type ContentControl}">
    <Image Name="Img"
        syncfusion:DockPreviewManagerVS2005.ProviderAction="GlobalLeft"
        Source="Images\synclogo.png" />
    <ControlTemplate.Triggers>
    <DataTrigger Binding="{Binding Path=IsSideButtonActive,
        RelativeSource={RelativeSource FindAncestor,
        AncestorType={x:Type syncfusion:DockPreviewMainButtonVS2005}}}"
        Value="true">
    <Setter TargetName="Img" Property="Source" Value="Images\synclogo.png"/>
    </DataTrigger>
    </ControlTemplate.Triggers>
    </ControlTemplate>
    </syncfusion:DockingManager.CenterDragProvider>
    <ContentControl syncfusion:DockingManager.Header="Child1"
        syncfusion:DockingManager.State="Dock" />
    </syncfusion:DockingManager>
```

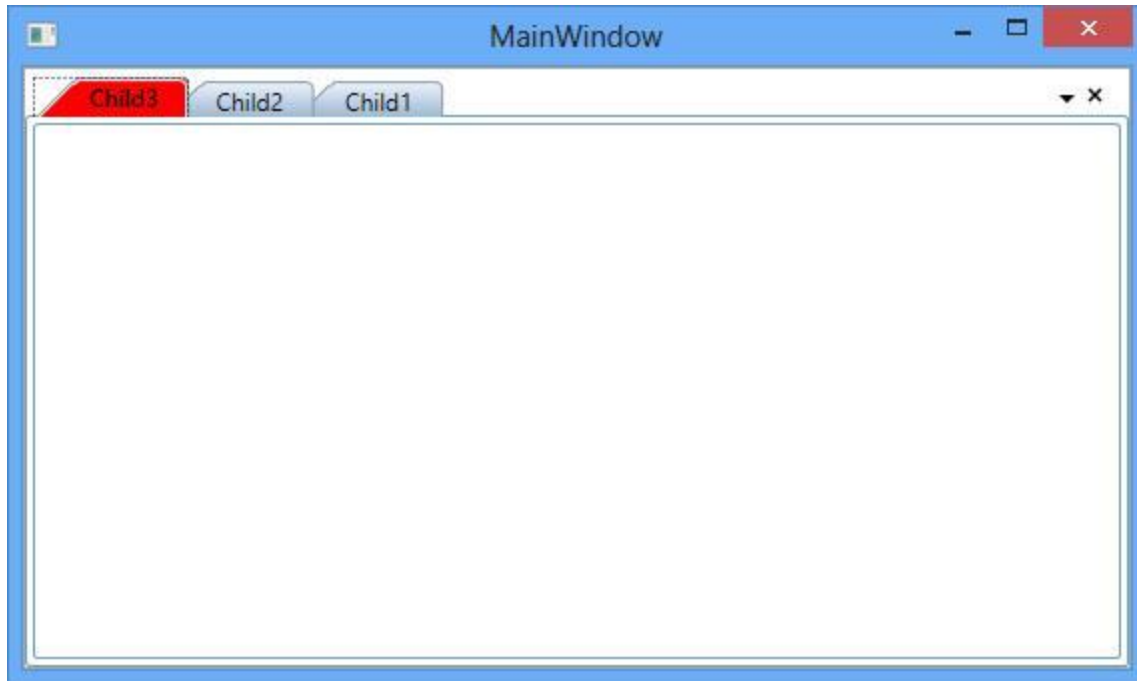


### DocumentTabControlStyle

Document state in DockingManager used the DocumentTabControl and its style can be customized using `DocumentTabControlStyle` with the TargetType as DocumentTabControl.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True">
<syncfusion:DockingManager.DocumentTabControlStyle>
<Style TargetType="{x:Type syncfusion:DocumentTabControl}">
<Setter Property="TabItemSelectedBackground" Value="Red" />
</Style>
</syncfusion:DockingManager.DocumentTabControlStyle>
<ContentControl syncfusion:DockingManager.Header="Child1"
syncfusion:DockingManager.State="Document"/>
<ContentControl syncfusion:DockingManager.Header="Child2"
syncfusion:DockingManager.State="Document"/>
<ContentControl syncfusion:DockingManager.Header="Child3"
syncfusion:DockingManager.State="Document"/>
</syncfusion:DockingManager>
```



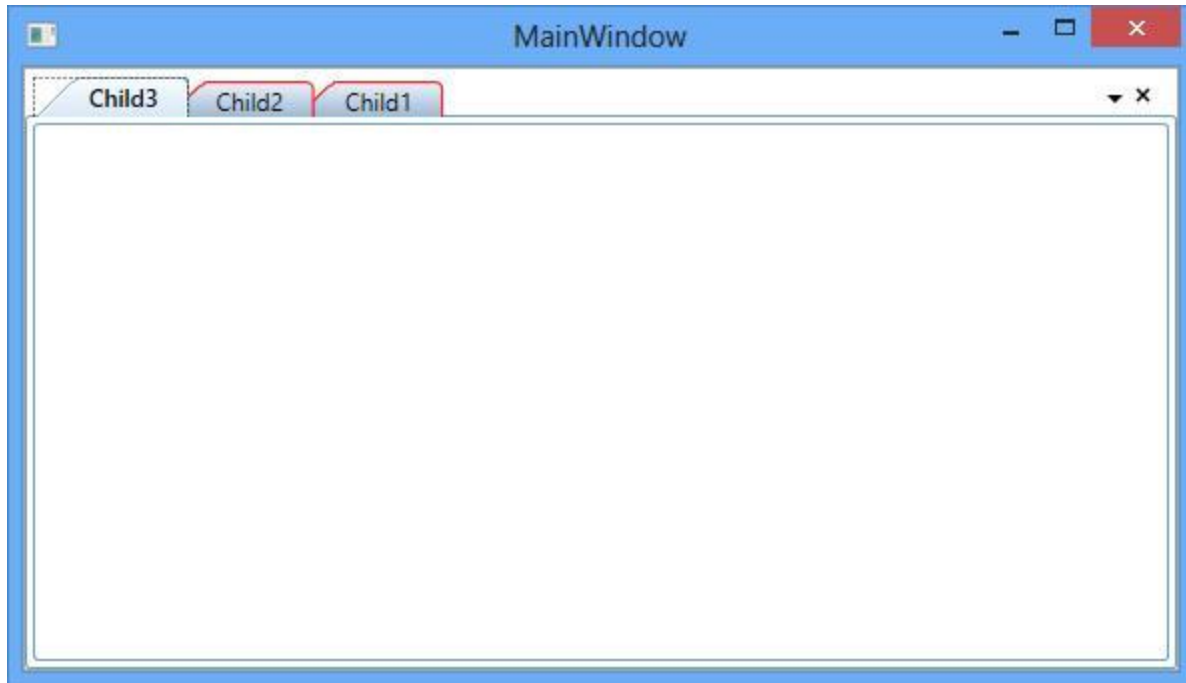
### DocumentTabItemStyle

Each Document TabItem in DockingManager constructed by the TabItemExt and its style can be customized using `DocumentTabItemStyle` with the TargetType as TabItemExt.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True" >
  <syncfusion:DockingManager.DocumentTabItemStyle>
    <Style TargetType="{x:Type syncfusion:TabItemExt}">
      <Setter Property="BorderBrush" Value="Red" />
    </Style>
  </syncfusion:DockingManager.DocumentTabItemStyle>
  <ContentControl syncfusion:DockingManager.Header="Child1"
syncfusion:DockingManager.State="Document"/>
  <ContentControl syncfusion:DockingManager.Header="Child2"
syncfusion:DockingManager.State="Document"/>
  <ContentControl syncfusion:DockingManager.Header="Child3"
syncfusion:DockingManager.State="Document"/>
</syncfusion:DockingManager>
```



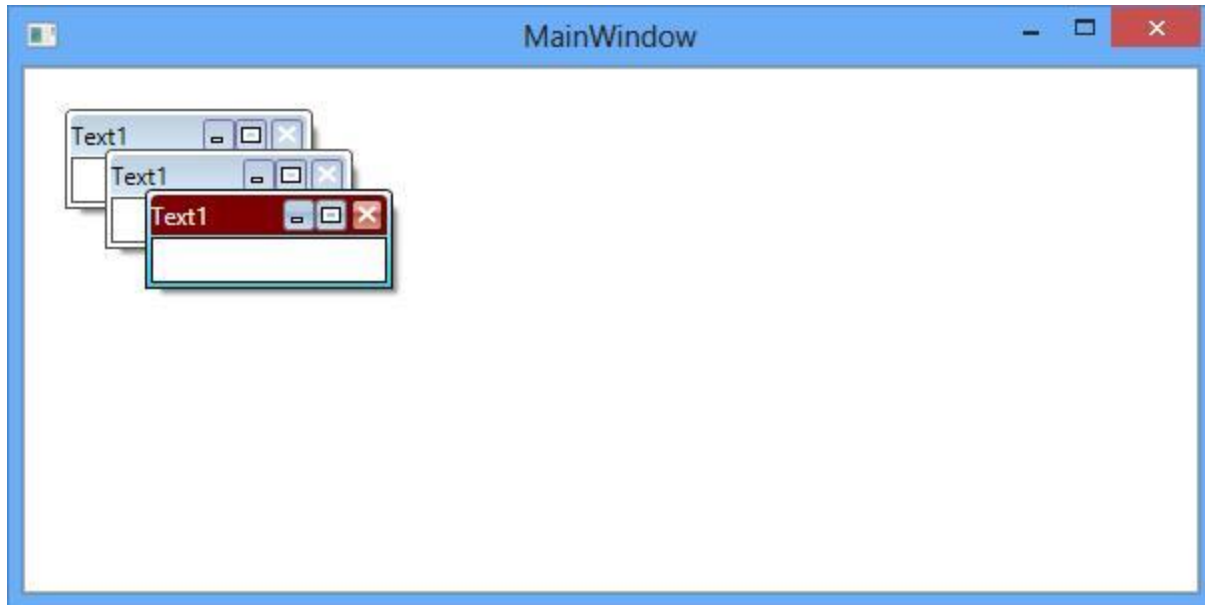


### DocumentMDIHeaderStyle

The header style for the MDI Document can be changed using the `DocumentMDIHeaderStyle` with the `TargetType` as `DocumentHeader`.

### XML

```
<syncfusion:DockingManager x:Name="DockingManager1"
    UseDocumentContainer="True" ContainerMode="MDI">
    <syncfusion:DockingManager.DocumentMDIHeaderStyle>
        <Style TargetType="{x:Type syncfusion:DocumentHeader}">
            <Setter Property="Background" Value="Maroon"/>
            <Setter Property="Foreground" Value="White"/>
            <Setter Property="HeaderTemplate">
                <Setter.Value>
                    <DataTemplate >
                        <TextBlock x:Name="Block" Text="Text1" />
                    </DataTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </syncfusion:DockingManager.DocumentMDIHeaderStyle>
    <ContentControl syncfusion:DockingManager.Header="Child1"
        syncfusion:DockingManager.State="Document"/>
    <ContentControl syncfusion:DockingManager.Header="Child2"
        syncfusion:DockingManager.State="Document"/>
    <ContentControl syncfusion:DockingManager.Header="Child3"
        syncfusion:DockingManager.State="Document"/>
</syncfusion:DockingManager>
```



### TabControl style

TabControl inside the Tabbed DockWindow can be customized using `TabControlStyle` with the TargetType TabControl in the DockingManager

### XML

```
<Syncfusion:DockingManager x:Name="DockingManager1"
    UseDocumentContainer="True" >
    <Syncfusion:DockingManager.TabControlStyle>
    <Style TargetType="{x:Type TabControl}">
    <Setter Property="Background" Value="Orange" />
    <Setter Property="ItemContainerStyle"
    Value="{Binding Path=(Syncfusion:DockedElementTabbedHost.DockingManager).
    (Syncfusion:DockingManager.TabItemStyle), RelativeSource={RelativeSource
    AncestorType={x:Type Syncfusion:DockedElementTabbedHost}}}" />
    </Style>
    </Syncfusion:DockingManager.TabControlStyle>
    <ContentControl Syncfusion:DockingManager.State="Dock"
    Syncfusion:DockingManager.Header="Child1" />
    <ContentControl Syncfusion:DockingManager.SideInDockedMode="Tabbed"
    Syncfusion:DockingManager.Header="Child2"
    Syncfusion:DockingManager.TargetNameInDockedMode="Content1">
    </ContentControl>
    </Syncfusion:DockingManager>
```

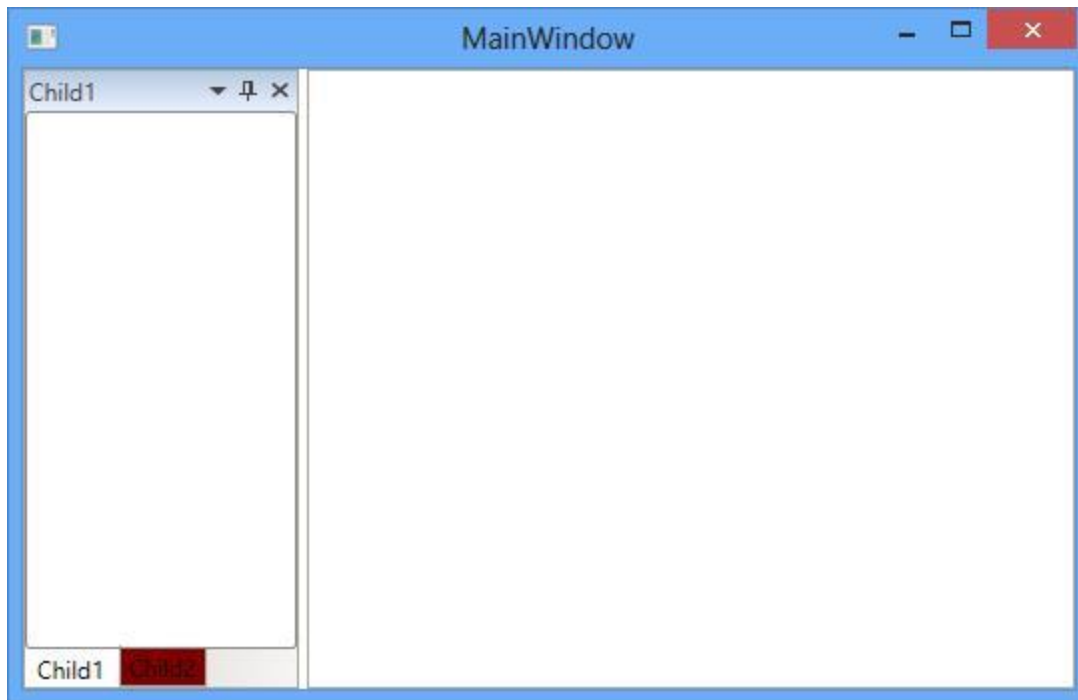


### TabItemStyle

The style for the Tabbed children of DockingManager can be customized using the `TabItemStyle` property with the Target Type as `TabItem`.

### XML

```
<Syncfusion:DockingManager x:Name="DockingManager1"
UseDocumentContainer="True">
  <Syncfusion:DockingManager.TabItemStyle>
    <Style TargetType="{x:Type TabItem}">
      <Setter Property="Background" Value="Maroon"/>
      <Setter Property="Header" Value="{Binding
Path=(Syncfusion:DockingManager.Header)}" />
    </Style>
  </Syncfusion:DockingManager.TabItemStyle>
  <ContentControl Syncfusion:DockingManager.State="Dock" x:Name="Content1"
Syncfusion:DockingManager.Header="Child1"/>
  <ContentControl Syncfusion:DockingManager.Header="Child2"
Syncfusion:DockingManager.SideInDockedMode="Tabbed"
Syncfusion:DockingManager.TargetNameInDockedMode="Content1"/>
</Syncfusion:DockingManager>
```



## Patterns and Practices in WPF Docking (DockingManager)

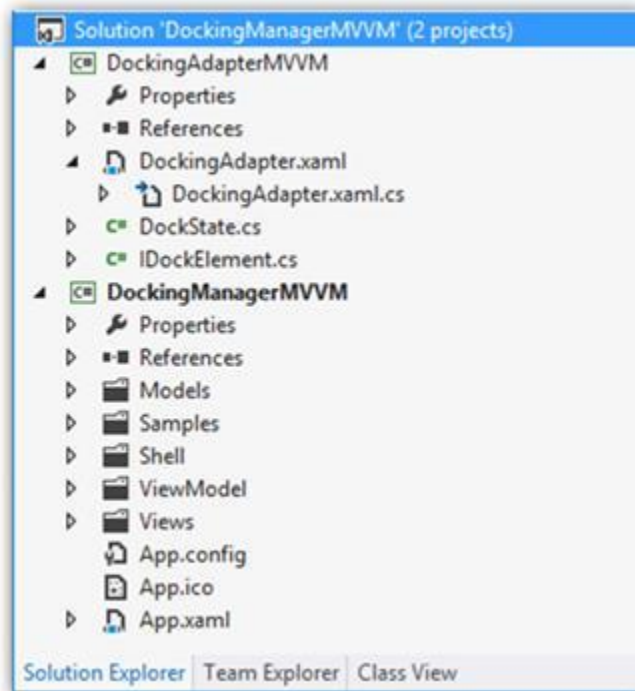
### MVVM

This section explains how to adapt the Syncfusion docking manager to an MVVM application. Since the WPF DockingManager is not an Items Control, it is not possible to have a traditional [ItemsSource](#) binding to a collection of objects in the view model. However this can be achieved by creating a wrapper or adapter for the DockingManager.

Here a simple text-reader application is used to demonstrate this approach.



1. **DocumentsView** - The pane that lists all the available documents and tooltip display the path of the document.
2. **PropertiesView** - The pane that shows the properties of a document. Our PropertyGrid control is used here.
3. **DocumentView** - The pane that uses the WPF flow-document reader to display the content of a file.
4. **CommandView** - The view has two commands: **Open** Document and **Exit**. Executing an Open Document action opens the Open File Dialog. The document that opened, added to the existing documents list. Other commands like Close Document and New Document can also be implemented the same way.
5. The project structure looks like this:

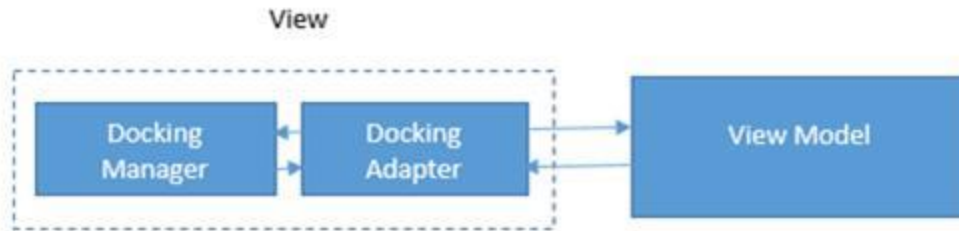


### *Docking Adapter*

The adapter is simply a user control that contains DockingManager as its content. The adapter has two properties — `ItemsSource` and `ActiveDocument`. Binding a collection of objects to the [ItemsSource](#) property triggers a collection change where the adapter creates a corresponding Framework element, example: ContentControl in the DockingManager, setting the underlying data context of the control to the business model.

### **XML**

```
<mvvm:dockingadapter itemssource="{Binding Workspaces}"
  activedocument="{Binding ActiveDocument, Mode=TwoWay}">
</mvvm:dockingadapter>
```



The text-reader application maintains a collection of workspaces. A workspace can be a normal dock pane or a document pane. The adapter also maintains an interface called `IDockElement` that maintains basic attributes needed for every dock element.

The text-reader application is just for the sample and contains very basic operations.

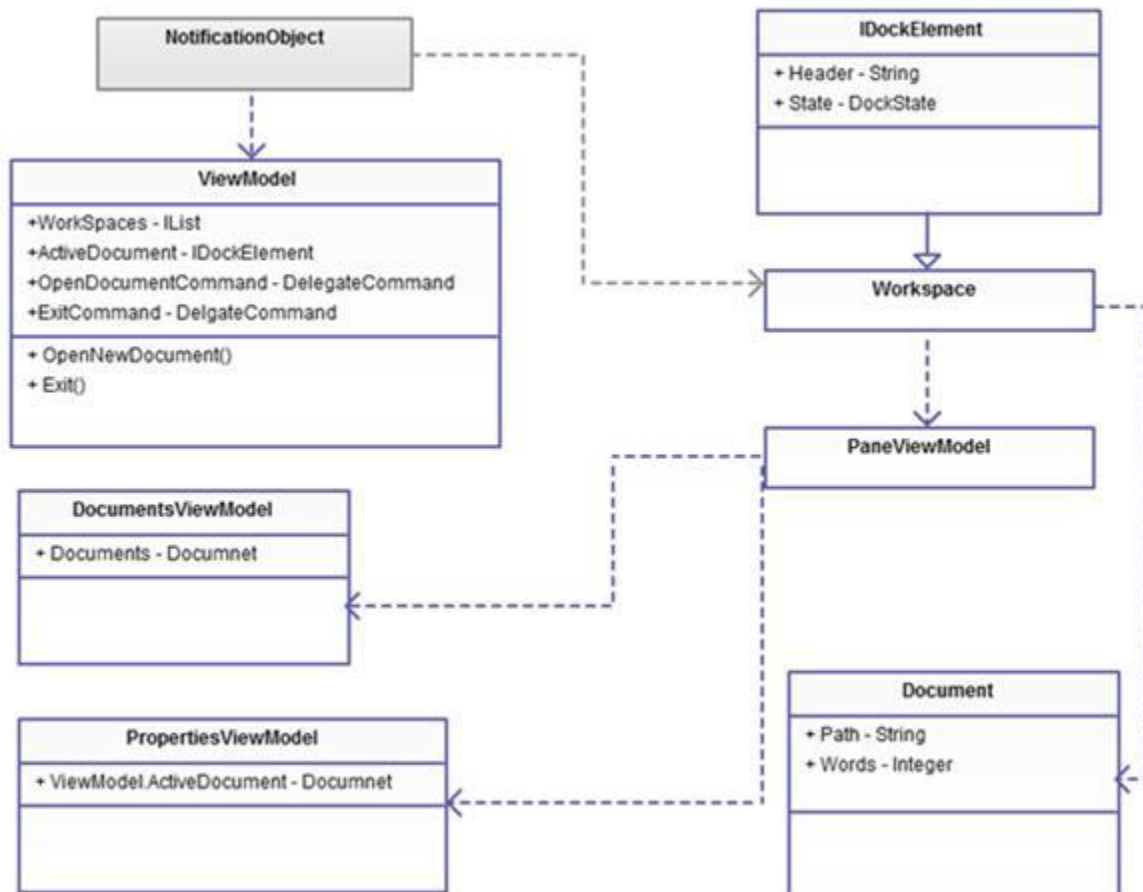
This article and sample intend to showcase the MVVM support for the docking manager.

The adapter user control also determines the state of the element, whether it should be added to the DockingManager as a dock element or document tab.

The adapter can be further customized to add elements as floating or auto-hidden.

The DockingManager provides an [ActiveWindowChanged](#) event. Using this, the [ActiveDocument](#) property in the adapter needs to be updated every time focus changes to other panes.

#### Application structure



The view model has a collection of workspaces that is data-bound to the [ItemsSource](#) property of the docking adapter. The adapter transforms the particular view model or business object into a corresponding dock element in the DockingManager.

Every dock element in the application is a workspace. There are three kinds of workspaces: the All Documents view, the Properties view, and the Document view. The docking adapter hooks up the “active window changed” event of the docking manager; the view model receives the message whenever the active document is changed.

#### Data Template

Since WPF has an implicit template approach, it is easy to apply visuals to the view models. In this application, the data templates are defined in App.xaml with only the DataType attribute mentioned and not key-specified. The WPF template engine can traverse the tree and find the appropriate model type and apply the templates.

#### XML

```
<application.resources>
<datatemplate datatype="{x:Type local:Document}">
<grid>
<local:documentview>
</local:documentview>
</grid>
</datatemplate>
<datatemplate datatype="{x:Type local:DocumentsViewModel}">
<grid>
<local:documentsview>
</local:documentsview>
</grid>
</datatemplate>
<datatemplate datatype="{x:Type local:PropertiesViewModel}">
<grid>
<local:propertiesview>
</local:propertiesview>
</grid>
</datatemplate>
</application.resources>
```

Following this approach, the docking adapter can also be treated as a normal item control and can be used in any MVVM application.

---

**Note:** [Download sample from GitHub](#)

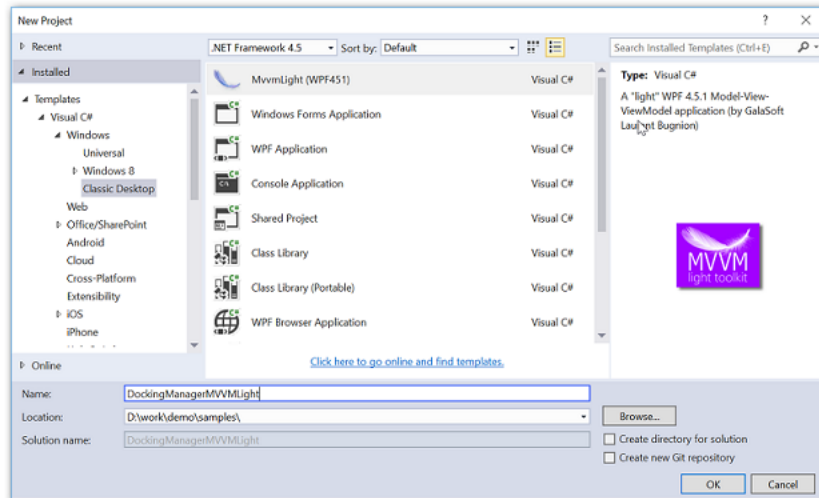
---

#### MVVMLight

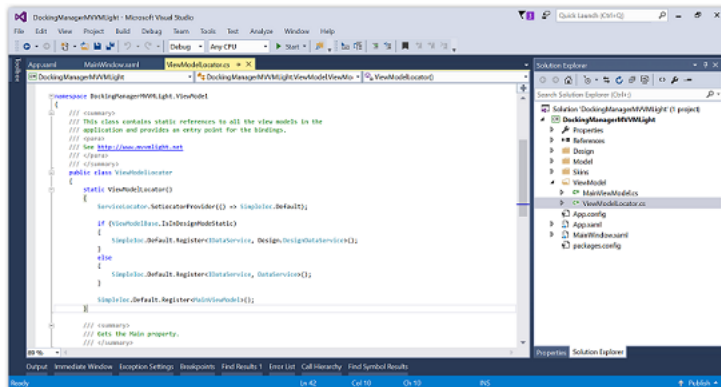
This section explains how to create MVVMLight sample with DockingManager. Since DockingManager is not an Items Control, it is not possible to have a traditional ItemsSource binding to a collection of objects in the view model. It can be achieved using DockingAdapter from the above MVVM sample creation section.

The following steps explain how to create sample project with MVVMLight templates.

1. Download MVVMLight toolkit and install it to avail the predefined MVVMLight templates for all platforms. MVVMLight extension can be installed from Tools and Extension.
2. Create a new WPF project and select MVVMLight WPF template.



3.Template for MVVMLight sample will be created with required assemblies, Simple IOC container and ViewModelLocator



4.Attach DockingAdapter project to the DockingManagerMVVMLight project from MVVM sample. Create necessary ViewModels and Views with perfect naming conventions. Once the ItemsSource has been set to DockingAdapter children will be populate.

DockingManager MVVMLight sample has been created following the above steps.

**Note:** [Download sample from GitHub](#)

## Practice with PRISM

The following steps helps to create sample project in the PRISM 5.0.

- 1.Create a New WPF project and add the following references to the solution project.  
*Microsoft.Practices.Composite.dll* *Microsoft.Practices.Composite.Presentation.dll*  
*Microsoft.Practices.Composite.UnityExtensions.dll* *Microsoft.Practices.ServiceLocation.dll* \*  
*Microsoft.Practices.Unity.dll*
- 2.Rename MainWindow to Shell in the Project.
- 3.Add new class called Bootstrapper.cs to initialize the prism application.

Here MainWindow is treated as shell, so returning the MainWindow in the CreateShell method.

**C#**

```
public class BootStrapper : UnityBootstrapper
{
```



```
protected override System.Windows.DependencyObject CreateShell()  
{  
    return new MainWindow();  
}  
protected override void InitializeModules()  
{  
    base.InitializeModules();  
    App.Current.MainWindow = (Window) this.Shell;  
    App.Current.MainWindow.Show();  
}  
}
```

## VB.NET

```
Public Class BootStrapper  
    Inherits UnityBootstrapper  
    Protected Overrides Function CreateShell() As  
        System.Windows.DependencyObject  
    Return New MainWindow()  
    End Function  
    Protected Overrides Sub InitializeModules()  
        MyBase.InitializeModules()  
        App.Current.MainWindow = CType(Me.Shell, Window)  
        App.Current.MainWindow.Show()  
    End Sub  
End Class
```

4. Override OnStartup method in the App.xaml.cs to execute Bootstrapper when the application starts

## C#

```
public partial class App : Application  
{  
    protected override void OnStartup(StartupEventArgs e)  
    {  
        base.OnStartup(e);  
        Bootstrapper bootstrapper = new Bootstrapper();  
        bootstrapper.Run();  
    }  
}
```

## VB.NET

```
Partial Public Class App  
    Inherits Application  
    Protected Overrides Sub OnStartup(ByVal e As StartupEventArgs)  
        MyBase.OnStartup(e)  
        Dim bootstrapper As New Bootstrapper()  
        bootstrapper.Run()  
    End Sub  
End Class
```

5. Next, create regions in the shell. To do this, first add the following namespace in the shell Window.

## XML

```
xmlns:prsm="http://www.codeplex.com/prism"
```

In the following code example, a region called “MainRegion” has been created to load DockingManager Module views.

#### XML

```
<Window x:Class="DockingManagerPrism.App.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:prsm="http://www.codeplex.com/prism"
Title="MainWindow" Height="350" Width="525">
<Grid>
<syncfusion:DockingManager prsm:RegionManager.RegionName="MainRegion"
DockFill="True">
</syncfusion:DockingManager>
</Grid>
</Window>
```

6.Add Module to the project.

Right-click the Solution project, point to “Add” and then click “NewProject”. Then a new window called AddNewProject opens. Select “ClassLibrary” from Visual C#, then rename the project with desired name and click OK. Now a New Module is created in the Solution Project.

Now add following assemblies to the Module project:

- PresentationCore.dll
- PresentationFramework.dll
- WindowsBase.dll

Also add the following Prism assemblies:

- Microsoft.Practices.Composite.dll
- Microsoft.Practices.Composite.Presentation.dll
- Microsoft.Practices.Composite.UnityExtensions.dll
- Microsoft.Practices.ServiceLocation.dll
- Microsoft.Practices.Unity.dll 7.In the Shell project, add the reference to the type of DockingManager module by registering with ModuleCatalog instance in the ConfigureModuleCatalog method.

#### C#

```
protected override void ConfigureModuleCatalog()
{
    base.ConfigureModuleCatalog();
    ModuleCatalog moduleCatalog = (ModuleCatalog) this.ModuleCatalog;
    moduleCatalog.AddModule(typeof(DockingModule));
}
```

#### VB.NET

```

Protected Overrides Sub ConfigureModuleCatalog()
MyBase.ConfigureModuleCatalog()
Dim moduleCatalog As ModuleCatalog = CType(Me.ModuleCatalog, ModuleCatalog)
moduleCatalog.AddModule(GetType(DockingModule))
End Sub

```

8. Adding Views to the Module, shown here is BottomLeftModule, similarly the view for the module can be added according to number of modules.

#### XML

```

<UserControl x:Class="DockingManagerPrism.Modules.BottomLeftModule"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
syncfusion:DockingManager.Header="BottomLeftModule"
syncfusion:DockingManager.State="Dock"
syncfusion:DockingManager.SideInDockedMode="Left" d:DesignHeight="300"
d:DesignWidth="300">
  <Grid>
</Grid>
</UserControl>

```

9. Add a region to the shell.

After creating View for the Module, register the view as Module using the following code example.

#### C#

```

public class DockingModule : IModule
{
  private readonly IRegionManager regionManager;
  public DockingModule(IRegionManager regionManager)
  {
    this.regionManager = regionManager;
  }
  public void Initialize()
  {
    regionManager.RegisterViewWithRegion("MainRegion",
      typeof(BottomLeftModule));
    regionManager.RegisterViewWithRegion("MainRegion",
      typeof(BottomRightModule));
    regionManager.RegisterViewWithRegion("MainRegion", typeof(TopModule));
  }
}

```

#### VB.NET

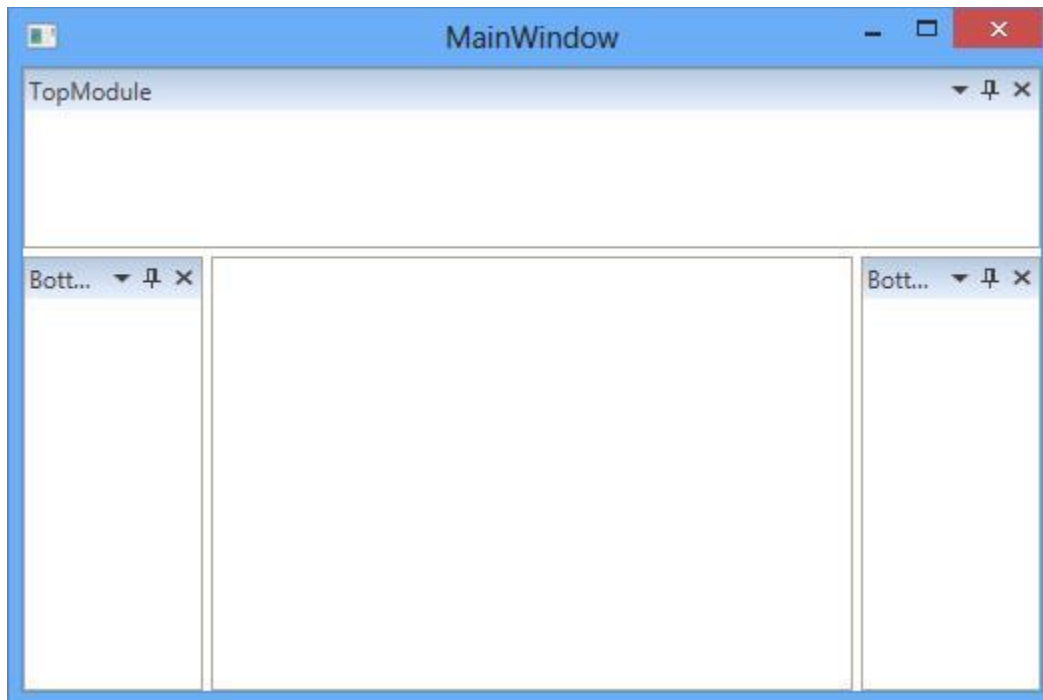
```

Public Class DockingModule
Implements IModule
  Private ReadOnly regionManager As IRegionManager
  Public Sub New(ByVal regionManager As IRegionManager)
    Me.regionManager = regionManager
  End Sub

```

```
End Sub
Public Sub Initialize()
    regionManager.RegisterViewWithRegion("MainRegion",
    GetType(BottomLeftModule))
    regionManager.RegisterViewWithRegion("MainRegion",
    GetType(BottomRightModule))
    regionManager.RegisterViewWithRegion("MainRegion", GetType(TopModule))
End Sub
End Class
```

Then when run the project, it is added as three of the Module in the Shell. The number of modules can be add based on the complexity of the project.



### Configuring DockingManager with Prism 6.1

Prism is a practice of building loosely coupled applications in WPF. It is intended to provide flexibility for testing and maintaining applications that are maintained in long term.

#### PRISM 6.1

Essential WPF controls are flexible with all the Prism versions. This section explains about creating a simple application using DockingManager in PRISM 6.1 pattern.

#### Setting up WPF application

1.Create a WPF application and rename the file MainWindow.xaml as Shell.xaml and MainWindow.xaml.cs as Shell.xaml.cs. 2.Rename the class name MainWindow as Shell in all the occurrences. 3.Add reference to the following assemblies *Prism Prism.WPF Prism.Unity.WPF Microsoft.Practices.ServiceLocation Microsoft.Practices.Unity Microsoft.Practices.Unity.Configuration \** Microsoft.Practices.Unity.RegistrationByConvention 4.In the Shell.xaml file, add the namespace definition for Prism Library as given below:

#### XML

```
<Window x:Class="DockingManagerPrism.App.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:prism="http://prismlibrary.com/"
Title="MainWindow" Height="350" Width="525">
</Window>
```

5. Create an instance of the control in Shell.xaml file and set the attached property `RegionManager.RegionName` for it. Here we have used `DockingManager` control.

#### XML

```
<syncfusion:DockingManager prism:RegionManager.RegionName="MainRegion"
UseDocumentContainer="True" DockFill="True" DockFillDocumentMode="Normal"/>
```

When we create an instance for Shell, it will resolve the value of the `RegionManager.RegionName` attached property and create a region for connecting it with the `DockingManager`.

#### *Setting up the Bootstrapper*

1. Create and add a class file for the Bootstrapper to the project. 2. Add the following using statements to the class that are referred by Bootstrapper.

#### C#

```
using System.Windows;
using Microsoft.Practices.Unity;
using Prism.Unity;
using Prism.Modularity;
```

3. Create a class Bootstrapper and inherit it from `UnityBootstrapper`. 4. Override the methods `CreateShell`, `InitializeShell` and `ConfigureModuleCatalog` as given below.

#### C#

```
public class Bootstrapper: UnityBootstrapper
{
    protected override DependencyObject CreateShell()
    {
        return Container.Resolve<Shell>();
    }
    protected override void InitializeShell()
    {
        Application.Current.MainWindow.Show();
    }
    protected override void ConfigureModuleCatalog()
    {
        ModuleCatalog catalog = (ModuleCatalog)ModuleCatalog;
        // Need to add the module catalogs here
    }
}
```

5. Remove the `StartupUri` in App.xaml file and override the `OnStartup` method in App.xaml.cs file. In that instantiate the Bootstrapper and run it as given below:

**C#**

```
protected override void OnStartup(StartupEventArgs e)
{
    base.OnStartup(e);
    Bootstrapper bootstrapper = new Bootstrapper();
    bootstrapper.Run();
}
```

*Adding modules to the project*

1.Create as many ClassLibrary projects for the modules. Here three class libraries are created for three modules. 2.Design views for all the modules in their projects as required. We have created UserControl as views and configured the attached properties of DockingManager for it.

**XML**

```
<UserControl x:Class="Program.ProgramView"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
syncfusion:DockingManager.Header="Program.cs"
syncfusion:DockingManager.State="Document"
d:DesignHeight="300" d:DesignWidth="200">
    <Grid>
        <StackPanel Orientation="Vertical">
            <TextBlock Text="using System;"/>
            <TextBlock Text="using System.Windows;"/>
            <TextBlock Text="using System.Windows.Controls;"/>
            <TextBlock Text="using System.Windows.Data;"/>
            <TextBlock Text="using System.Text;"/>
            <TextBlock Text=""/>
            <TextBlock Text="namespace Program"/>
            <TextBlock Text="{"/>
            <TextBlock Text="public class MyProgram"/>
            <TextBlock Text="{"/>
            <TextBlock Text=""/>
            <TextBlock Text=" }"/>
            <TextBlock Text=" }"/>
        </StackPanel>
    </Grid>
</UserControl>
```

3.Create a class implementing IModule interface for all the modules in their project. Using Initialize method register the view to the region using region name("MainRegion").

**C#**

```
public class ProgramModule : IModule
{
    private readonly IRegionManager regionManager;
    public ProgramModule(IRegionManager regionManager)
    {
        this.regionManager = regionManager;
    }
}
```

```
public void Initialize()
{
    regionManager.RegisterViewWithRegion("MainRegion", typeof(ProgramView));
}
```

4. Add all the modules as reference projects in main application. Add all modules to module catalog in Bootstrapper.cs as given below

#### C#

```
protected override void ConfigureModuleCatalog()
{
    ModuleCatalog catalog = (ModuleCatalog)ModuleCatalog;
    catalog.AddModule(typeof(SolutionExplorer.SolutionExplorerModule));
    catalog.AddModule(typeof(Toolbox.ToolboxModule));
    catalog.AddModule(typeof(Program.ProgramModule));
}
```

#### Create RegionAdapter for DockingManager

So far implementations will work as expected for an ItemsControl. Since DockingManager is not an ItemsControl, we need a region adapter to notify that regions should be mapped into Children property.

Create a ClassLibrary project for defining the region adapter and the class should inherit from RegionAdapterBase class. In that override the methods, Adapt and CreateRegion. In Adapt method, add the regions to DockingManager.Children whenever the regions collection is changed as shown below:

#### C#

```
public class DockingManagerRegionAdapter : RegionAdapterBase<DockingManager>
{
    public DockingManagerRegionAdapter(IRegionBehaviorFactory
    regionBehaviorFactory) : base(regionBehaviorFactory)
    {
    }
    protected override void Adapt(IRegion region, DockingManager regionTarget)
    {
        region.Views.CollectionChanged += delegate
        {
            foreach (var child in region.Views.Cast<UserControl>())
            {
                if (!regionTarget.Children.Contains(child))
                {
                    regionTarget.BeginInit();
                    regionTarget.Children.Add(child);
                    regionTarget.EndInit();
                }
            }
        };
    }
    protected override IRegion CreateRegion()
    {
        return new AllActiveRegion();
    }
}
```

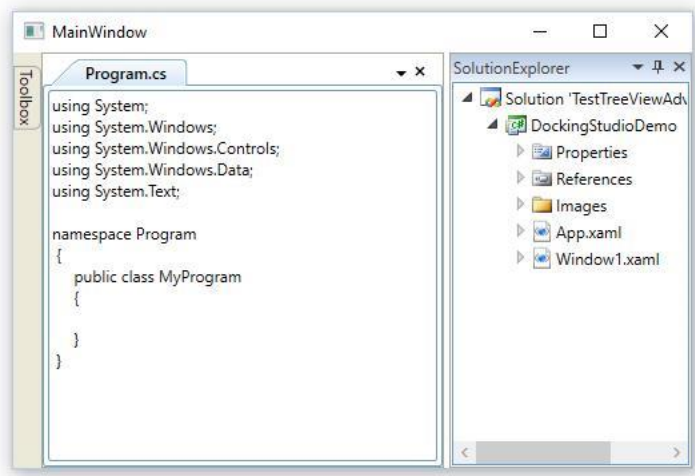
### Configure the region adapter mappings

Add reference to RegionAdapter project from the main application. Region adapter mapping have to be performed in Bootstrapper class. Override the method `ConfigureRegionAdapterMappings` and set the mapping as given below:

#### C#

```
protected override Prism.Regions.RegionAdapterMappings
ConfigureRegionAdapterMappings ()
{
    RegionAdapterMappings regionAdapterMappings =
    base.ConfigureRegionAdapterMappings ();
    if (regionAdapterMappings != null)
    {
        regionAdapterMappings.RegisterMapping (typeof (DockingManager) ,
        Container.Resolve<DockingManagerRegionAdapter.DockingManagerRegionAdapter> ()
        );
    }
    return regionAdapterMappings;
}
```

The final output of application is given below:



**Note:** [Download sample from GitHub](#)

### Configuring DockingManager with Prism 7.1

This section explains about creating a simple application using `DockingManager` in `PRISM 7.1` pattern.

#### Setting up WPF application

**Step 1:** Create a WPF application and rename the file `MainWindow.xaml` as `Shell.xaml` and `MainWindow.xaml.cs` as `Shell.xaml.cs`.

**Step 2:** Rename the class name `MainWindow` as `Shell` in all the occurrences.

**Step 3:** Add the following required assembly references to the project:

- Prism



- Prism.WPF
- Prism.Unity.WPF
- Microsoft.Practices.ServiceLocation
- Microsoft.Practices.Unity
- Microsoft.Practices.Unity.Configuration
- Microsoft.Practices.Unity.RegistrationByConvention

**Step 4:** In the Shell.xaml file, add the namespace definition for Prism Library as given below:

#### XML

```
<Window x:Class="DockingManagerPrism.App.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
xmlns:prism="http://prismlibrary.com/"
Title="MainWindow" Height="350" Width="525">
</Window>
```

**Step 5:** Create an instance of the control in Shell.xaml file and set the attached property `RegionManager.RegionName` for it. Here we have used `DockingManager` control.

#### XML

```
<syncfusion:DockingManager prism:RegionManager.RegionName="MainRegion"
UseDocumentContainer="True" DockFill="True" DockFillDocumentMode="Normal"/>
```

When we create an instance for Shell, it will resolve the value of the `RegionManager.RegionName` attached property and create a region for connecting it with the `DockingManager`.

**Step 6:** Add the following required assembly references in App.xaml.cs file.

#### C#

```
using Prism.Ioc;
using Prism.Modularity;
using Prism.Regions;
using Prism.Unity;
using System.Windows;
```

**Step 7:** Inherit the App class from `PrismApplication` in App.xaml.cs file.

#### C#

```
/// <summary>
/// Interaction logic for App.xaml
/// </summary>
public partial class App : PrismApplication
{
}
```

**Step 8:** Override the methods `CreateShell`, `RegisterTypes` and `CreateModuleCatalog` as given below.

#### C#

```

public partial class App : PrismApplication
{
    protected override Window CreateShell()
    {
        return Container.Resolve<Shell>();
    }
    protected override void RegisterTypes(IContainerRegistry containerRegistry)
    {
    }
    protected override IModuleCatalog CreateModuleCatalog()
    {
        ModuleCatalog catalog = new ModuleCatalog();
        // Need to add the module catalogs here
        return catalog;
    }
}

```

### *Adding modules to the project*

**Step 1:** Create as ClassLibrary projects for the modules. Here three class libraries are created for three modules.

**Step 2:** Design views for all the modules in their projects as required. We have created UserControl as views and configured the attached properties of DockingManager for it.

### XML

```

<UserControl x:Class="Program.ProgramView"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d" xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
syncfusion:DockingManager.Header="Program.cs"
syncfusion:DockingManager.State="Document"
d:DesignHeight="300" d:DesignWidth="200">
    <Grid>
        <StackPanel Orientation="Vertical">
            <TextBlock Text="using System;"/>
            <TextBlock Text="using System.Windows;"/>
            <TextBlock Text="using System.Windows.Controls;"/>
            <TextBlock Text="using System.Windows.Data;"/>
            <TextBlock Text="using System.Text;"/>
            <TextBlock Text=""/>
            <TextBlock Text="namespace Program"/>
            <TextBlock Text="{"/>
            <TextBlock Text="public class MyProgram"/>
            <TextBlock Text="{"/>
            <TextBlock Text=""/>
            <TextBlock Text=" }"/>
            <TextBlock Text="}"/>
        </StackPanel>
    </Grid>
</UserControl>

```

**Step 3:** Create a class implementing IModule interface for all the modules in your project. Using OnInitialized method register the view to the region using region name (MainRegion).

**C#**

```
public class ProgramModule : IModule
{
    public void OnInitialized(IContainerProvider containerProvider)
    {
        var regionManager = containerProvider.Resolve<IRegionManager>();
        regionManager.RegisterViewWithRegion("MainRegion", typeof(ProgramView));
    }
    public void RegisterTypes(IContainerRegistry containerRegistry)
    {
    }
}
```

**Step 4:** Add all the modules as reference projects in main application. Add all modules to module catalog and also register it in App.xaml.cs as given below:

**C#**

```
protected override void RegisterTypes(IContainerRegistry containerRegistry)
{
    containerRegistry.RegisterForNavigation<Toolbox.ToolboxModule>();
    containerRegistry.RegisterForNavigation<Program.ProgramModule>();
    containerRegistry.RegisterForNavigation<SolutionExplorer.SolutionExplorerModule>();
}
protected override IModuleCatalog CreateModuleCatalog()
{
    ModuleCatalog catalog = new ModuleCatalog();
    catalog.AddModule(typeof(Toolbox.ToolboxModule));
    catalog.AddModule(typeof(Program.ProgramModule));
    catalog.AddModule(typeof(SolutionExplorer.SolutionExplorerModule));
    return catalog;
}
```

#### Create RegionAdapter for DockingManager

So, far implementations will work as expected for an ItemsControl. Since DockingManager is not an ItemsControl, we need a region adapter to notify that regions should be mapped into Children property.

**Step 1:** Create a ClassLibrary project for defining the region adapter and the class should inherit from RegionAdapterBase class. In that override the methods, Adapt and CreateRegion. There are three types of region class to create a region based on the ContentControlRegionAdapter, ItemsControlRegionAdapter and SelectorRegionAdapter as follows:

- SingleActiveRegion - The Region that allows a maximum of one active view at a time.
- AllActiveRegion - The Region that keeps all the views in it as active and de-activation of views are not allowed.
- Region - The Region that allows multiple active views and de-activation of views are allowed.

In `Adapt` method, add the regions to `DockingManager.Children` whenever the regions collection is changed as shown below:

**C#**

```
public class DockingManagerRegionAdapter : RegionAdapterBase<DockingManager>
{
    public DockingManagerRegionAdapter(IRegionBehaviorFactory
regionBehaviorFactory)
: base(regionBehaviorFactory)
{
}
protected override void Adapt(IRegion region, DockingManager regionTarget)
{
    region.Views.CollectionChanged += (s, e) =>
    {
        if (e.Action == NotifyCollectionChangedAction.Add)
        {
            foreach (FrameworkElement element in e.NewItems)
            {
                if (!regionTarget.Children.Contains(element))
                {
                    regionTarget.BeginInit();
                    regionTarget.Children.Add(element);
                    regionTarget.EndInit();
                }
            }
        }
    };
}
protected override IRegion CreateRegion()
{
    return new SingleActiveRegion();
}
```

**Step 2:** Also, we can activate and deactivate views by creating a behavior class and the class should inherit from `RegionBehavior`, `IHostAwareRegionBehavior` classes. In below code example, shown how to activate and deactivate the document views using `ActiveDocumentChanged` event of `DocumentContainer`.

**C#**

```
public class DocumentRegionActiveAwareBehavior : RegionBehavior,
IHostAwareRegionBehavior
{
    public const string BehaviorKey = "DocumentRegionActiveAwareBehavior";
    DependencyObject _hostControl;
    public DependencyObject HostControl
    {
        get { return _hostControl; }
        set { _hostControl = value as DockingManager; }
    }
    protected override void OnAttach()
    {

```

```
((HostControl as DockingManager).DocContainer as
DocumentContainer).AddTabDocumentAtLast = true;
((HostControl as DockingManager).DocContainer as
DocumentContainer).ActiveDocumentChanged +=
DocumentRegionActiveAwareBehavior_ActiveDocumentChanged;
}
private void
DocumentRegionActiveAwareBehavior_ActiveDocumentChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    if (e.OldValue != null)
    {
        var item = e.OldValue;
        //are we dealing with a ContentPane directly
        if (Region.Views.Contains(item) && Region.ActiveViews.Contains(item))
        {
            Region.Deactivate(item);
        }
        else
        {
            //now check to see if we have any views that were injected
            var contentControl = item as ContentControl;
            if (contentControl != null)
            {
                var injectedView = contentControl.Content;
                if (Region.Views.Contains(injectedView) &&
                    Region.ActiveViews.Contains(injectedView))
                {
                    Region.Deactivate(injectedView);
                }
            }
        }
    }
    if (e.NewValue != null)
    {
        var item = e.NewValue;
        //are we dealing with a ContentPane directly
        if (Region.Views.Contains(item) && !this.Region.ActiveViews.Contains(item))
        {
            Region.Activate(item);
        }
        else
        {
            //now check to see if we have any views that were injected
            var contentControl = item as ContentControl;
            if (contentControl != null)
            {
                var injectedView = contentControl.Content;
                if (Region.Views.Contains(injectedView) &&
                    !this.Region.ActiveViews.Contains(injectedView))
                {
                    Region.Activate(injectedView);
                }
            }
        }
    }
}
```

**Step 3:** Override the method `AttachBehaviors` in `DockingManagerRegionAdapter` class and add the created behavior to the region as like below code snippet:

**C#**

```
protected override void AttachBehaviors(IRegion region, DockingManager
regionTarget)
{
    base.AttachBehaviors(region, regionTarget);
    if
    (!region.Behaviors.ContainsKey(DocumentRegionActiveAwareBehavior.BehaviorKey
    ))
    region.Behaviors.Add(DocumentRegionActiveAwareBehavior.BehaviorKey, new
    DocumentRegionActiveAwareBehavior { HostControl = regionTarget });
}
```

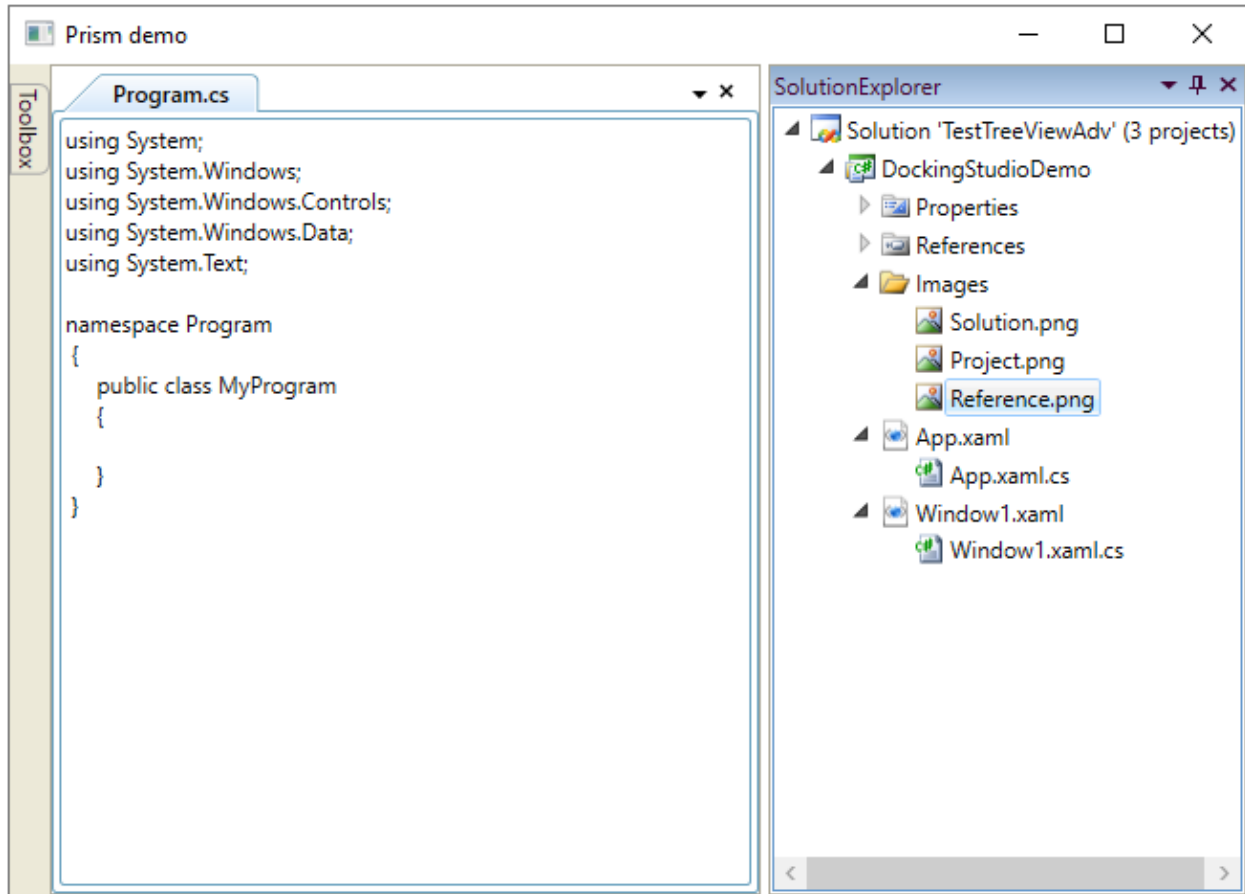
#### *Configure the region adapter mappings*

Add reference to `RegionAdapter` project from the main application. Region adapter mapping have to be performed in `App` class. Override the method `ConfigureRegionAdapterMappings` and set the mapping as given below:

**C#**

```
protected override void ConfigureRegionAdapterMappings(RegionAdapterMappings
regionAdapterMappings)
{
    base.ConfigureRegionAdapterMappings(regionAdapterMappings);
    regionAdapterMappings.RegisterMapping(typeof(DockingManager),
    Container.Resolve<DockingManagerRegionAdapter.DockingManagerRegionAdapter>()
    );
}
```

The final output of application is given below:



**Note:** [Download sample from GitHub](#)

## DocumentContainer

### WPF Tabbed MDI Form (DocumentContainer) Overview

A DocumentContainer is a control that is used for holding the documents, controls and panels inside it. Using a DocumentContainer you can create interfaces like MDI and TDI, which can make the end-user to create easily navigable applications. This help section will demonstrate all the important properties of the DocumentContainer that will help the end-user to know about all the available features.

#### Features

- Provides options for both MDI and TDI container mode.
- Various window switching styles. CTRL+TAB keyboard shortcut is used to easily navigate through the windows.
- Skins support.
- State persistence; the document container is used to load and save data in IS, BIN, and XML.
- A large set of properties, methods, and events for easy customization.
- Resizing and moving the DocumentContainer using the keyboard.

### Getting Started with WPF Tabbed MDI Form (DocumentContainer)

This section describes how to add [WPF Tabbed MDI Form](#) (DocumentContainer) control into wpf application and its basic functionalities.

### Assembly deployment

Refer to the [control dependencies](#) section to get the list of assemblies or NuGet package that needs to be added as a reference to use the control in any application.

You can find more details about installing the NuGet package in a WPF application in the following link:

### [How to install nuget packages](#)

### Create a simple application with DocumentContainer

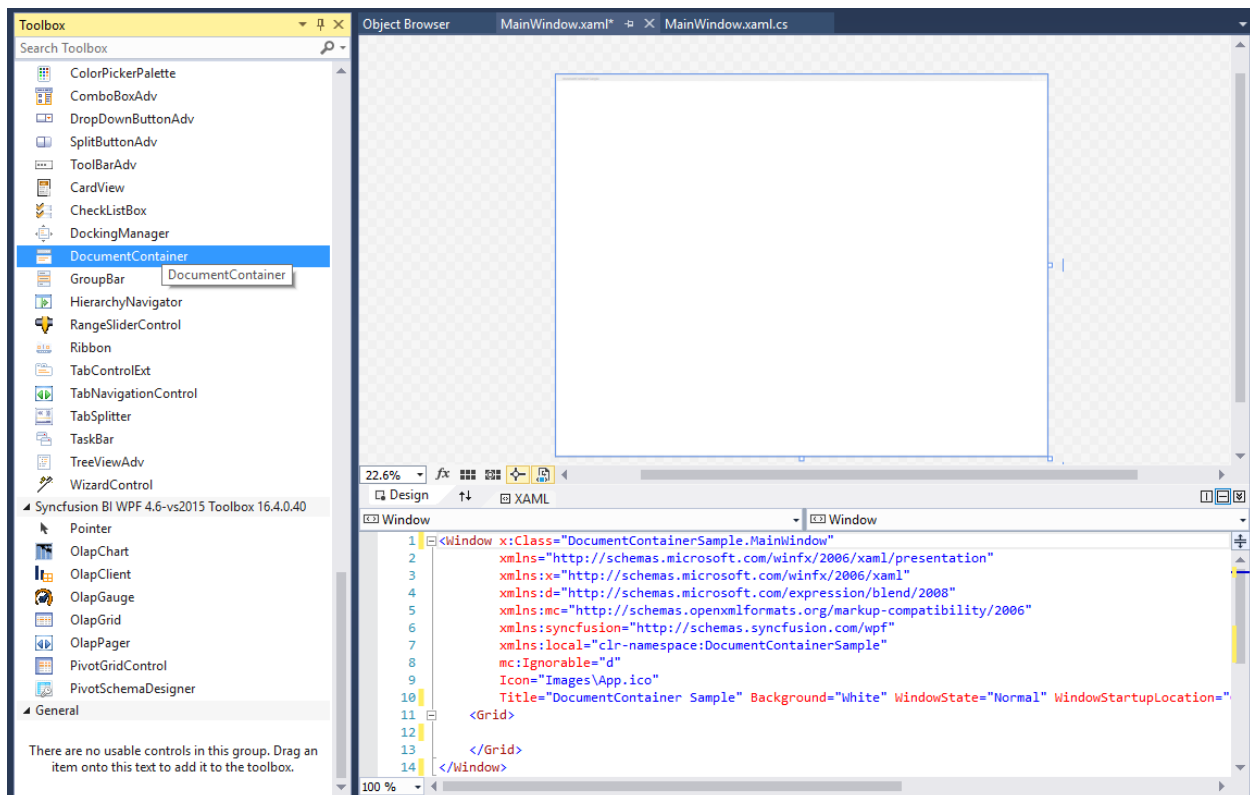
#### Create a project

Create a new WPF project in Visual Studio to display the DocumentContainer with functionalities.

#### Add control through designer

The DocumentContainer control can be added to an application by dragging it from the toolbox to a designer view. The following required assembly references will be added automatically:

- Syncfusion.Tools.WPF
- Syncfusion.Shared.WPF



#### Add control manually in XAML

To add the control manually in XAML, follow the given steps:

1. Add the following required assembly references to the project: *Syncfusion.Tools.WPF* and *Syncfusion.Shared.WPF*
2. Import Syncfusion WPF schema <http://schemas.syncfusion.com/wpf> in the XAML page.
3. Declare the DocumentContainer control in the XAML page.

### XML



```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="DocumentContainerSample.MainWindow"
Title="DocumentContainer Sample" Height="350" Width="525">
<Grid>
<!--Adding DocumentContainer control -->
<syncfusion:DocumentContainer x:Name="documentContainer" Width="100"
Height="100" VerticalAlignment="Center" HorizontalAlignment="Center"/>
</Grid>
</Window>
```

### Add control manually in C\#

To add the control manually in C#, follow the given steps:

1. Add the following required assembly references to the project: *Syncfusion.Tools.WPF* *Syncfusion.Shared.WPF*
2. Import the DocumentContainer namespace **using Syncfusion.Windows.Tools.Controls;**
3. Create a DocumentContainer instance, and add it to the window.

### C#

```
using Syncfusion.Windows.Tools.Controls;
namespace DocumentContainerSample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            //Creating an instance of DocumentContainer control
            DocumentContainer documentContainer = new DocumentContainer();
            //Adding DocumentContainer as window content
            this.Content = documentContainer;
        }
    }
}
```

### Add document windows

The document container allows users add new framework elements such as button and text block to its container using the [Items](#) property.

### XML

```
<syncfusion:DocumentContainer x:Name="documentContainer" Mode="TDI">
<Button></Button>
<Button/></Button>
<Button/></Button>
</syncfusion:DocumentContainer>
```

### C#

```
Button button1 = new Button();
Button button2 = new Button();
Button button3 = new Button();
//Adding buttons as document container window
documentContainer.Items.Add(button1);
documentContainer.Items.Add(button2);
documentContainer.Items.Add(button3);
```

### Set header to document

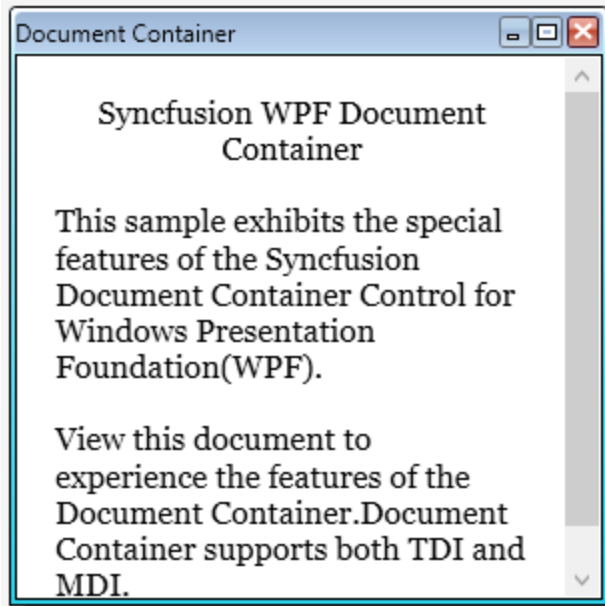
You can set header to the DocumentContainer elements by setting the [Header](#) property.

### XML

```
<syncfusion:DocumentContainer Name="documentContainer" Mode="MDI"
SwitchMode="VS2005">
  <!-- Setting header for window -->
  <FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Document
Container">
    <FlowDocument TextAlignment="Left">
      <Paragraph TextAlignment="Center">
        Syncfusion WPF Document Container</Paragraph>
      <Paragraph>
        This sample exhibits the special features of the Syncfusion Document
        Container Control for Windows Presentation Foundation(WPF) .
      </Paragraph>
      <Paragraph>
        View this document to experience the features of the Document
        Container.Document Container supports both TDI and MDI.
      </Paragraph>
    </FlowDocument>
  </FlowDocumentScrollViewer>
</syncfusion:DocumentContainer>
```

### C#

```
//Setting header for document container elements
DocumentContainer.SetHeader(flowScrollViewer, "Document Container");
```



#### Set TDI/MDI document mode

The DocumentContainer supports the following document modes :

- **TDI** - Tabbed Document Interface
- **MDI** - Multiple Document Interface

You can change the above modes using the [Mode](#) property of DocumentContainer.

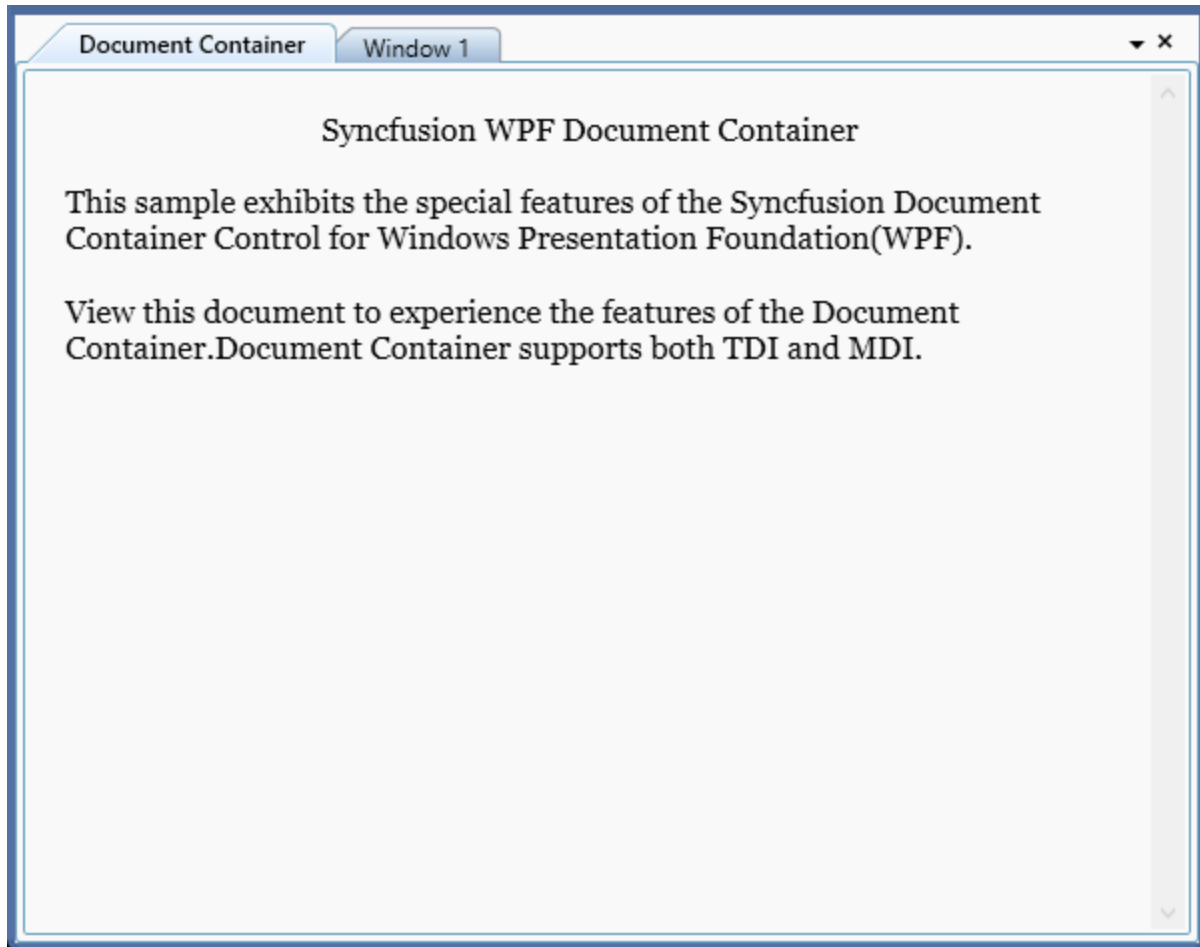
#### XML

```
<syncfusion:DocumentContainer Name="documentContainer" Mode="TDI" />
```

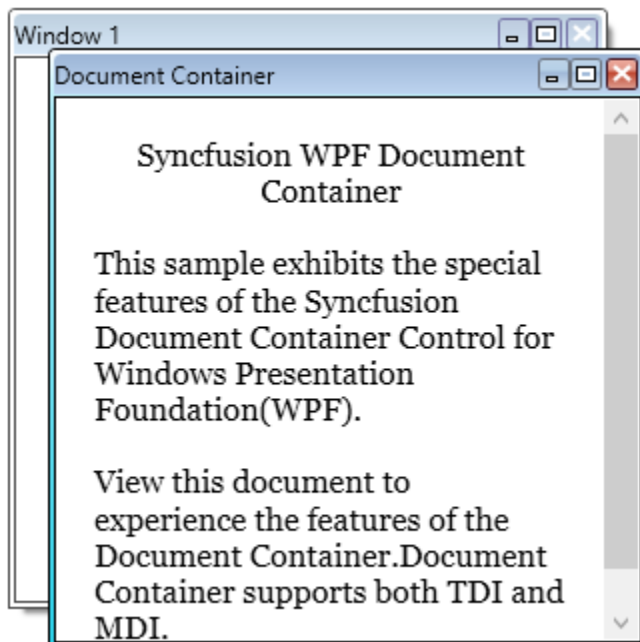
#### C#

```
documentContainer.Mode=DocumentContainerMode.TDI;
```

- **TDI**



- MDI

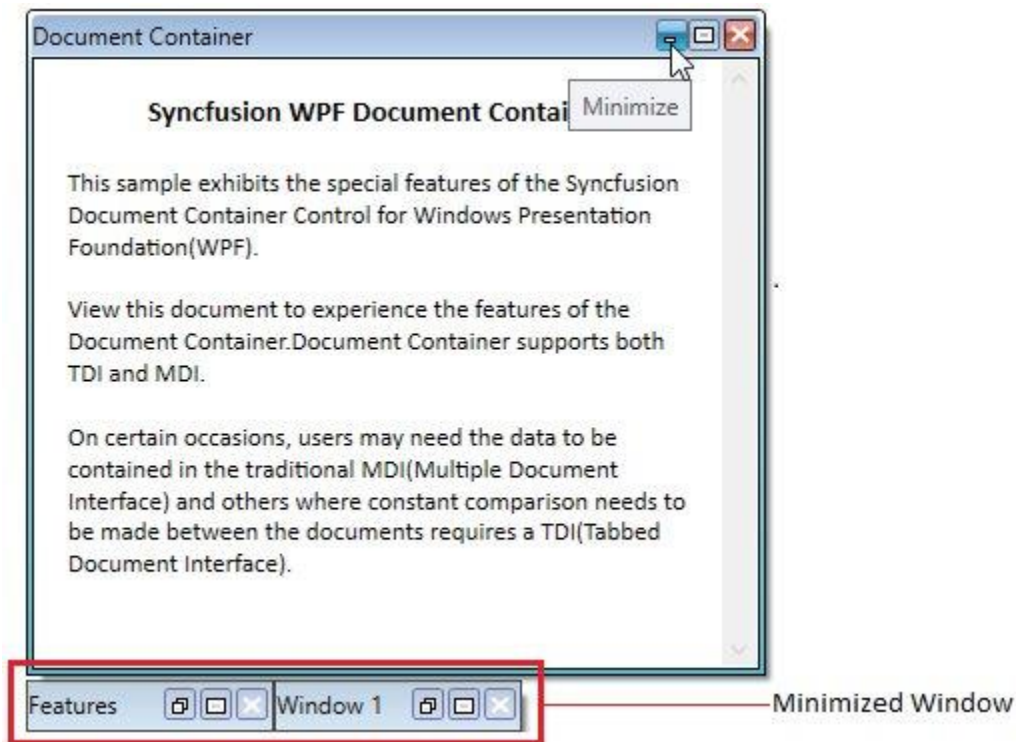


### Minimizing MDI window

You can minimize the MDI window by setting the [CanMDIMinimize](#) property as `true`. The default value of `CanMDIMinimize` property is `false`. The minimized MDI windows are arranged one by one in the bottom-left corner of the window.

### XML

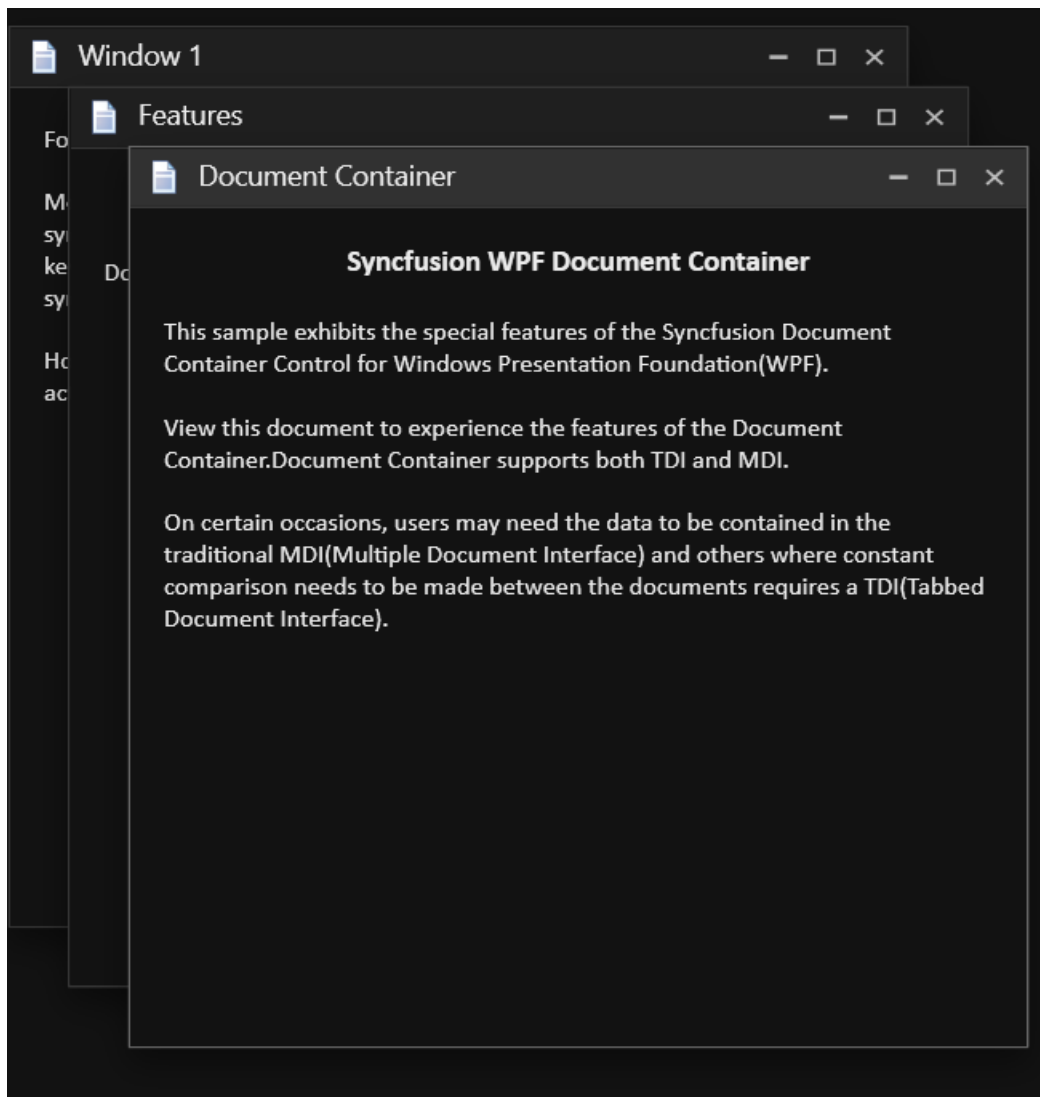
```
<syncfusion:DocumentContainer Name="DocContainer"
CanMDIMinimize="True"
Mode="MDI">
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Features"/>
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Window1"/>
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Document
Container"/>
</syncfusion:DocumentContainer>
```



### Theme

DocumentContainer supports various built-in themes. Refer to the below links to apply themes for the DocumentContainer,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)



## Adding and Removing Items from the WPF Tabbed MDI Form control

This topic illustrates how to add and remove items from Document Container control.

### Adding items

Document Container allows the user to add new elements to its container (such as button, text block), using Items.Add method. Use the following code snippet, for calling this method.

#### XML

```
<syncfusion:DocumentContainer Name="DocContainer">
  <Button ></Button></syncfusion:DocumentContainer>
```

#### C#

```
Button a = new Button();
DocContainer.Items.Add(a);
```

### Remove item

You can remove all the items in the Document Container using Items.Clear method. To remove all the items in the Document Container, use the following code snippet.

#### C#

```
DocContainer.Items.Clear();
```

## setting mode for document container in WPF Tabbed MDI Form

Document Container supports two important modes which are listed below.

- TDI - Tabbed Document Interface
- MDI - Multiple Document Interface

To set the Document Container in TDI mode, use the following code snippet.

#### XML

```
<!-- Adding Document Container -->
<syncfusion:DocumentContainer Name="DocContainer" Mode="TDI">.....
</syncfusion:DocumentContainer>
```

#### C#

```
//Creating instance of Document Container
DocumentContainer DocContainer =
new DocumentContainer();
//Set mode as TDI
DocContainer.Mode =
DocumentContainerMode.TDI;.....
//Adding control to window this. Content = DocContainer;
```

The following is the screen shot of a document container, which is in TDI mode.



To set the Document Container in MDI mode, use the following code snippet.

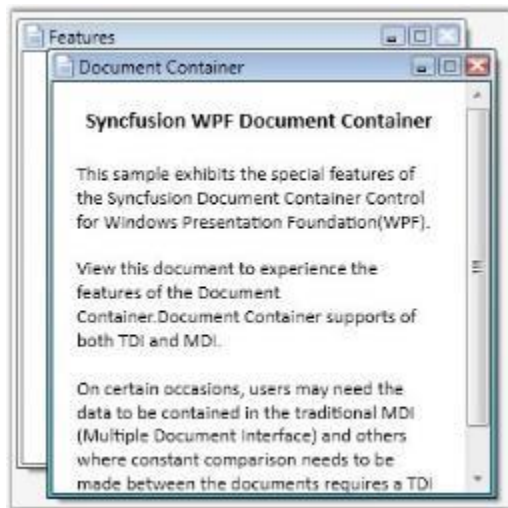
### XML

```
<!-- Adding Document Container -->
<syncfusion:DocumentContainer Name="DocContainer" Mode="MDI"> ..... 
</syncfusion:DocumentContainer>
```

### C#

```
//Creating instance of Document Container
DocumentContainer DocContainer = new DocumentContainer();
//Set mode as MDIDocContainer.
Mode = DocumentContainerMode.MDI;.....
//Adding control to window this.
Content = DocContainer;
```

The following screen shot shows the document container in MDI mode.



## Setting Window State in WPF Tabbed MDI Form

### Setting Window State

There are three possible window states of MDI windows for the Document Container control. They are as follows.

- Maximized

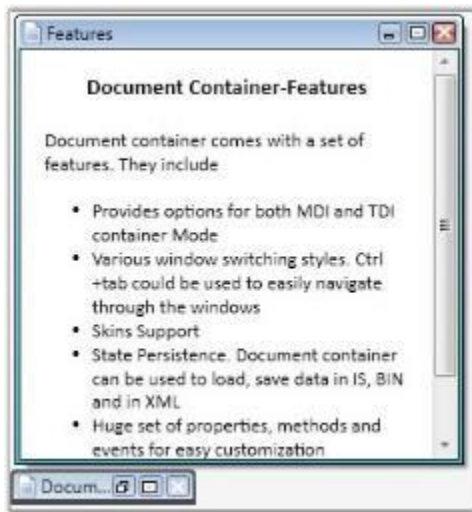


- Minimized
- Normal

To set the MDI window state to "minimized", use the below code snippet. FlowDocumentScrollViewer is considered as an element of the Document Container in the below mentioned example.

#### XML

```
<!-- Adding Document Container -->
<syncfusion:DocumentContainer Name="DocContainer" Mode="MDI">
  <FlowDocumentScrollViewer
    syncfusion:DocumentContainer.MDIWindowState="Minimized" >
  </FlowDocumentScrollViewer>
  ....
  ....
</syncfusion:DocumentContainer>
```



#### Notify event for MDIWindow State Changes

The [MDIWindowStateChanging](#) event occurs before the state of the MDIWindow is changed. The state changing of MDIWindow can be handled by setting e.Cancel to true.

#### XML

```
<syncfusion:DocumentContainer Name="DocContainer" Mode="MDI"
  MDIWindowStateChanging="DocContainer_MDIWindowStateChanging">
  <FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Window1" >
  </FlowDocumentScrollViewer>
  <FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Window2" >
  </FlowDocumentScrollViewer>
</syncfusion:DocumentContainer>
```

#### C#

```
private void DocContainer_MDIWindowStateChanging(object sender,
  MDIWindowStateChangingEventArgs e)
{
  if (e.NewState == MDIWindowState.Maximized)
    e.Cancel = true;
```

```
}
```

### Setting MDIBounds in WPF Tabbed MDI Form (DocumentContainer)

This property helps the Document Container control in properly placing its elements within the container.

The general syntax of the MDI bound property is given below.

Syncfusion:DocumentContainer.MDIBounds="a,b,c,d"

where,

- The first two values (a and b) stands for X and Y co-ordinates for the MDI bounds.
- The second two values(c and d) stands for width and height of the element in a Document Container.

To set the MDI Bounds, use the following code snippet.

#### XML

```
<!-- Adding Document Container -->
<syncfusion:DocumentContainer Name="DocContainer" Mode="MDI">
  <FlowDocumentScrollViewer
    Syncfusion:DocumentContainer.MDIBounds="0,0,200,300">
  </FlowDocumentScrollViewer>
  ....
  ....
</syncfusion:DocumentContainer>
```



### Setting Header of the Document container in WPF Tabbed MDI Form

Using the **Header** property, user can set the header for the DocumentContainer elements. Use the following code snippet, to set the header for the DocumentContainer element.

#### XML

```
<!-- Adding Document Container -->
<syncfusion:DocumentContainer Name="DocContainer" Mode="MDI">
```

```
<FlowDocumentScrollViewer x:Name="flow"
syncfusion:DocumentContainer.Header="Features">
</FlowDocumentScrollViewer>
.....
.....
</syncfusion:DocumentContainer>
```

### Setting header programmatically

Header of the DocumentContainer elements can be set by `SetHeader` method.

### C#

```
//Set the Header of the DocumentContainer
DocumentContainer.SetHeader(flow, "Features");
```

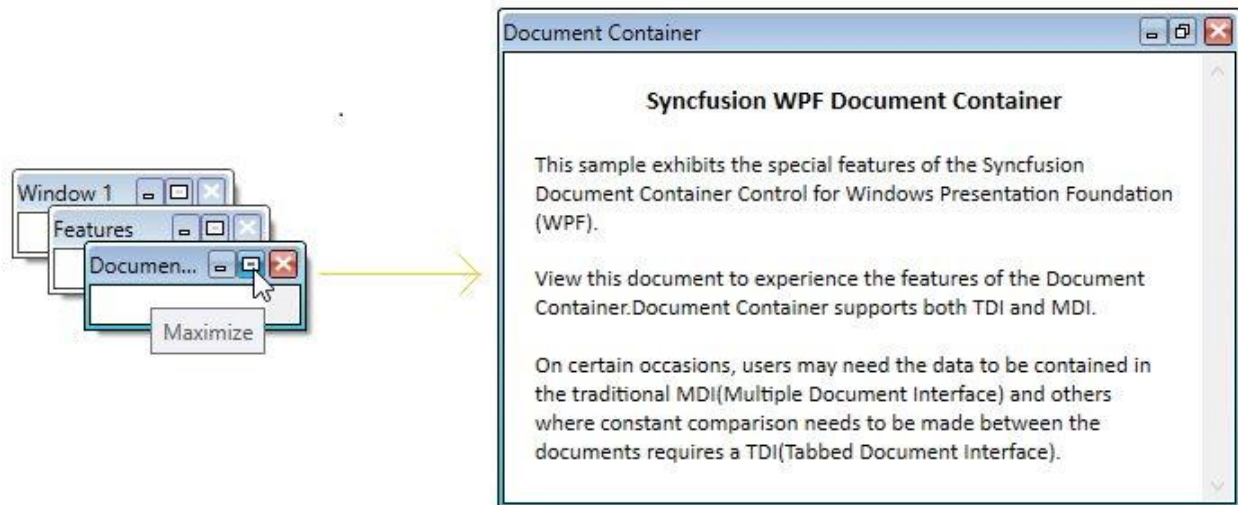


### Maximizing MDI window in WPF Tabbed MDI Form (DocumentContainer)

You can maximize the MDI window by setting the `CanMDIMaximize` property as `true`. The default value of `CanMDIMaximize` property is `false`.

### XML

```
<syncfusion:DocumentContainer Name="DocContainer"
CanMDIMaximize="True"
Mode="MDI">
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Features"/>
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Window1"/>
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Document
Container"/>
</syncfusion:DocumentContainer>
```

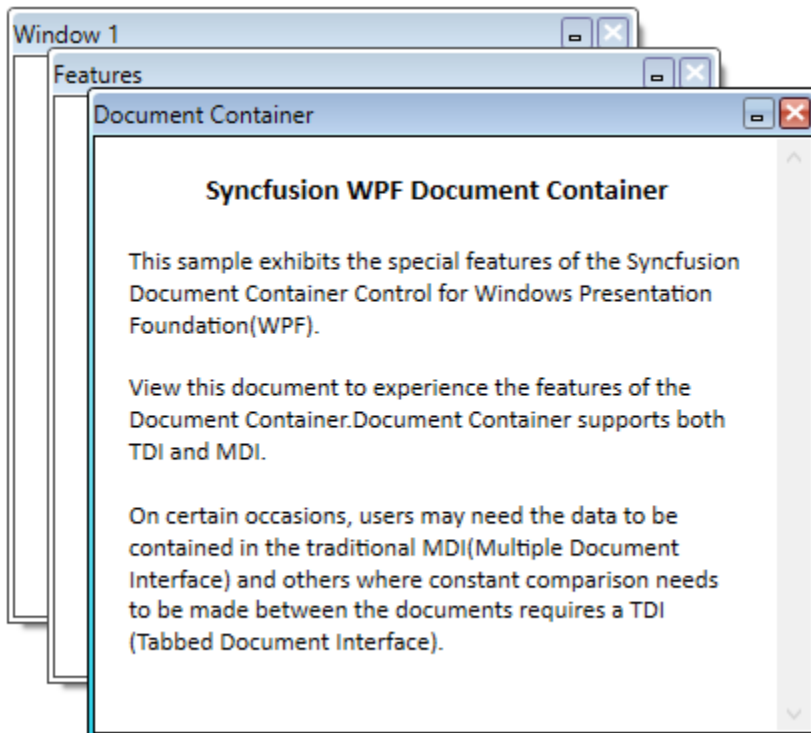


### Restrict Maximizing the MDI window

You can restrict maximizing the MDI window by setting the `CanMDIMaximize` property as `false`.

### XML

```
<syncfusion:DocumentContainer Name="DocContainer"
CanMDIMaximize="False"
Mode="MDI">
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Features"/>
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Window1"/>
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Document
Container"/>
</syncfusion:DocumentContainer>
```

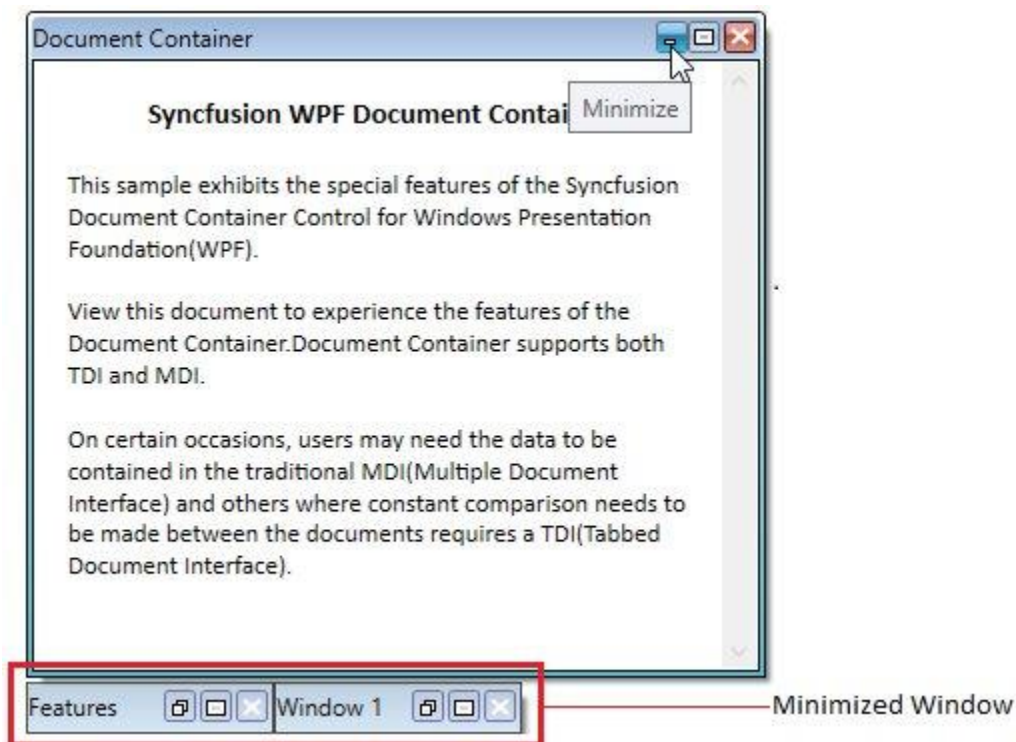


## Minimizing MDI window in WPF Tabbed MDI Form (DocumentContainer)

You can minimize the MDI window by setting the `CanMDIMinimize` property as `true`. The default value of `CanMDIMinimize` property is `false`. The minimized MDI windows are arranged one by one in the bottom-left corner of the window.

### XML

```
<syncfusion:DocumentContainer Name="DocContainer"
CanMDIMinimize="True"
Mode="MDI">
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Features"/>
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Window1"/>
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Document
Container"/>
</syncfusion:DocumentContainer>
```

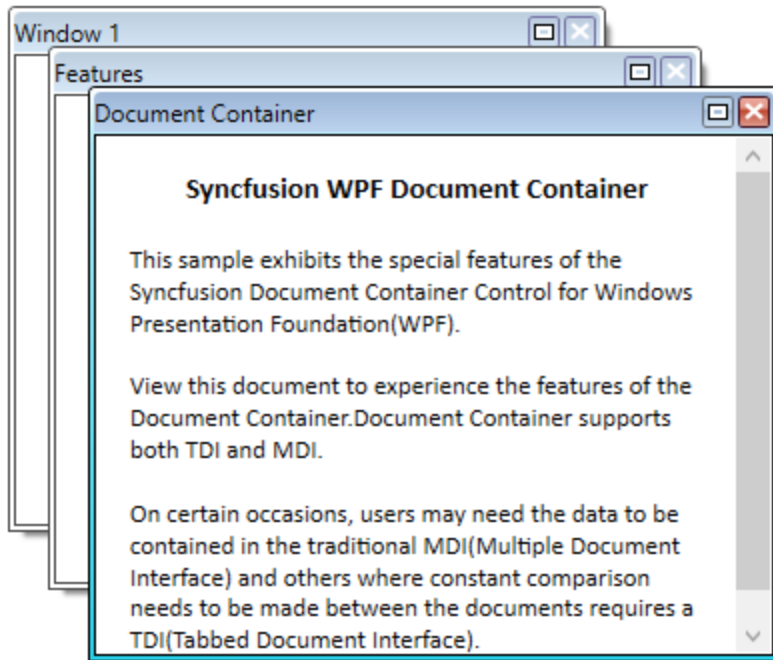


### Restrict Minimizing the MDI window

You can restrict minimizing the MDI window by setting the `CanMDIMinimize` property as `false`.

### XML

```
<syncfusion:DocumentContainer Name="DocContainer"
CanMDIMinimize="False"
Mode="MDI">
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Features"/>
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Window1"/>
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Document
Container"/>
</syncfusion:DocumentContainer>
```



### MDI Resize in WPF Tabbed MDI Form (DocumentContainer)

Document Container provides options to resize its elements. Setting `AllowMDIResize` property to `true`, will enable the end users to resize the container elements.

To set this property, use the below code.

#### XML

```
<!-- Adding Document Container -->
<syncfusion:DocumentContainer Name="DocContainer" IsAllowMDIResize="True"
Mode="MDI">
<FlowDocumentScrollViewer syncfusion:DocumentContainer.Header="Features">
</FlowDocumentScrollViewer>
...
</syncfusion:DocumentContainer>
```

### Setting Window Switchers in WPF Tabbed MDI Form

Document Container enables the users to switch between the windows using keyboard keys. This feature facilitates easy navigation between the documents. By using CTRL + TAB combination of keys in the keyboard, user can navigate between windows. Window switchers are available for the Document Container for this purpose.

Currently five modes of window switchers are supported. They are as follows.

- Immediate
- List
- QuickTabs
- VS2005
- VistaFlip

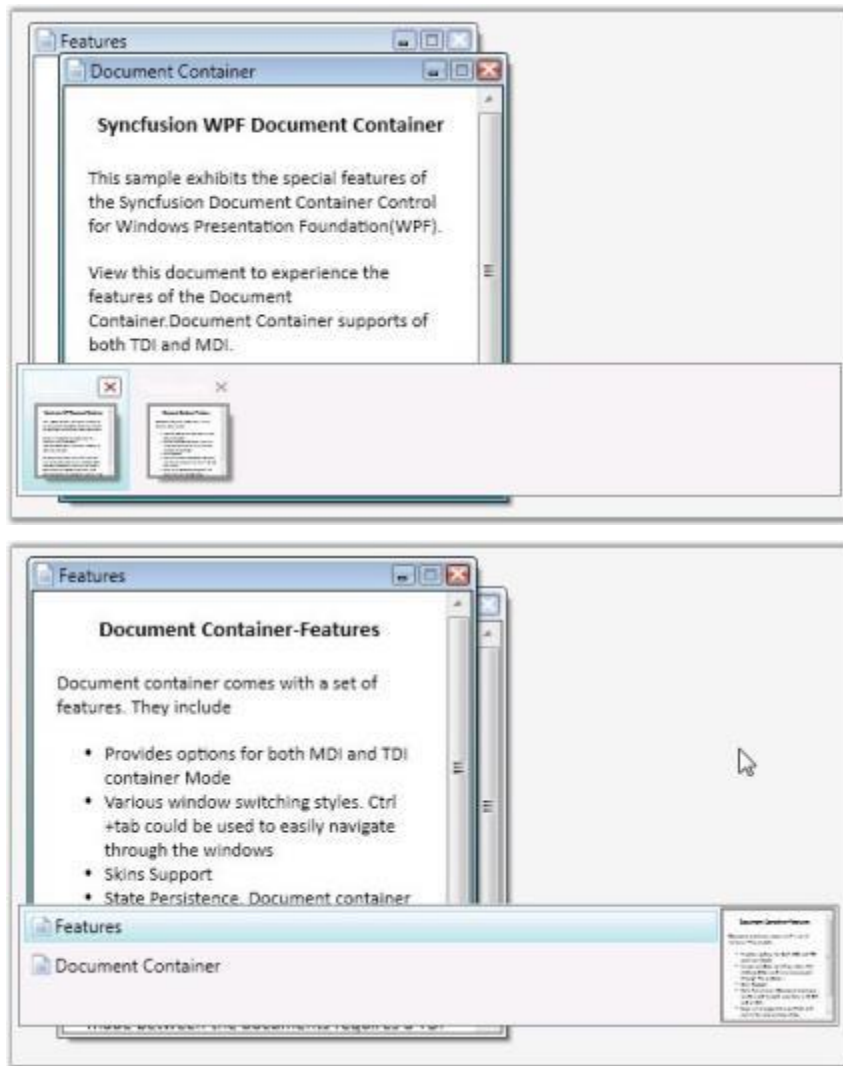
To set the Quick Tab Mode for the window switchers, use the following code.

### XML

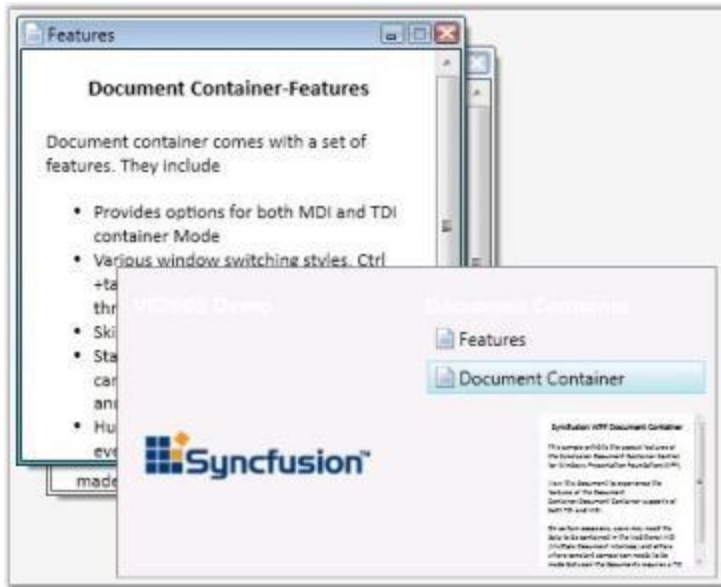
```
<!-- Adding Document Container -->
<syncfusion:DocumentContainer Name="DocContainer" SwitchMode="QuickTabs"
Mode="MDI"> ..... 
</syncfusion:DocumentContainer>
```

### C#

```
//Creating instance of Document Container
DocumentContainer DocContainer = new DocumentContainer();
//Set mode as MDIDocContainer.
Mode = DocumentContainerMode.MDI;
//Set switch modeDocContainer.SwitchMode = SwitchMode.QuickTabs;
.....
//Adding control to the window this.
Content = DocContainer;
```







### State Persistence in WPF Tabbed MDI Form (DocumentContainer)

This topic illustrates loading and saving the dock state in various places. It also gives information about resetting and deleting the states.

#### Load and Save in Registry

In the Document Container, you can save and load dock states into the Registry using the `SaveDockState` and `LoadDockState` methods. Refer to the following code snippet for setting these methods.

#### C#

```
// Code to Save State:
BinaryFormatter formatter1 = new BinaryFormatter();
DocContainer.SaveDockState(formatter1);
// Code to Load State:
BinaryFormatter formatter1 = new BinaryFormatter();
DocContainer.LoadDockState(formatter1);
```

#### Load and Save in Isolated Storage

In the Document Container you can Load/Save/Reset the dock state in the Isolated Storage. To Load, Save and Reset, use the following code snippet.

#### C#

```
// Code to Save:
DocContainer.SaveDockState();
// Code to Load:
DocContainer.LoadDockState();
// Code to Reset:
DocContainer.DeleteInternalIsolatedStorage();
```

#### Load and Save in XML

You can save and load the data for the Document Container in XML also. To Load and Save data in XML use the following code snippet. This is done by using both Binary Formatter as well as Soap Formatter as follows.



**C#**

```
// Code to Save in XML using Binary Formatter:
BinaryFormatter formatter1 = new BinaryFormatter();
DocContainer.SaveDockState(formatter1, StorageFormat.Xml,
@"d:\docum_xml.xml");
// Code save in XML using SOAP formatter:
SoapFormatter formatter1 = new SoapFormatter();
DocContainer.SaveDockState(formatter1, StorageFormat.Xml,
@"d:\docum_xml.xml");
// Code to Load XML using Binary formatter:
BinaryFormatter formatter1 = new BinaryFormatter();
DocContainer.LoadDockState(formatter1, StorageFormat.Xml,
@"d:\docum_xml.xml");
// Code to Load XML using Soap Formatter:
SoapFormatter formatter1 = new SoapFormatter();
DocContainer.LoadDockState(formatter1, StorageFormat.Xml,
@"d:\docum_xml.xml");
```

## Load and Save in Bin

Document Container enables the user to save and load the data in BIN. Refer to the following code snippet for saving or loading data in BIN. This is achieved by using Binary Formatter and Soap Formatter as well.

**C#**

```
// Code to Save Using Binary Formatter:
BinaryFormatter formatter1 = new BinaryFormatter();
DocContainer.SaveDockState(formatter1, StorageFormat.Binary,
@"d:\docum_bin.bin");
// Code to Save Using Soap Formatter:
SoapFormatter formatter1 = new SoapFormatter();
DocContainer.SaveDockState(formatter1, StorageFormat.Binary,
@"d:\docum_bin.bin");
// Code to Load using Binary Formatter:
BinaryFormatter formatter1 = new BinaryFormatter();
DocContainer.LoadDockState(formatter1, StorageFormat.Binary,
@"d:\docum_bin.bin");
// Code to Load using Soap Formatter:
SoapFormatter formatter1 = new SoapFormatter();
DocContainer.LoadDockState(formatter1, StorageFormat.Binary,
@"d:\docum_bin.bin");
```

## Reset the states in Document Container

You can easily reset the states in Document Container by using the ResetState method as in the below code snippet.

**C#**

```
DocumentContainer DocContainer = new DocumentContainer();
// Reset the states
DocContainer.ResetState();
```

### Delete the State

You can delete all the states you have created in the Document Container by using the DeleteDockState property. Refer to the below code example.

#### C#

```
DocContainer.DeleteDockState(@"d:\docum_xml.xml");  
DocContainer.DeleteDockState(@"d:\docum_bin.bin");  
DocContainer.DeleteDockState();
```

### Disabling TDI items Drag-Drop in WPF DockingManager & Tabbed MDI Form

You can disable the dragging and dropping of TDI items in DocumentContainer and DockingManager by setting the [IsTDIDragDropEnabled](#) property value as `false`. By default, [IsTDIDragDropEnabled](#) property value is `true`.

#### XML

```
<syncfusion:DockingManager Name="dockingmanager1"  
UseDocumentContainer="True"  
IsTDIDragDropEnabled="False">  
  <Grid syncfusion:DockingManager.Header="Tab1"  
    syncfusion:DockingManager.State="Document"/>  
  <Grid syncfusion:DockingManager.Header="Tab2"  
    syncfusion:DockingManager.State="Document"/>  
</syncfusion:DockingManager>
```

#### C#

```
dockingmanager1.IsTDIDragDropEnabled=false;
```

This property is also applicable to DocumentContainer, as shown in the following code.

#### XML

```
<syncfusion:DocumentContainer Name="documentcontainer1" Mode="TDI"  
IsTDIDragDropEnabled="False" >  
  <Grid syncfusion:DockingManager.Header="Tab1"  
    syncfusion:DockingManager.State="Document"/>  
  <Grid syncfusion:DockingManager.Header="Tab2"  
    syncfusion:DockingManager.State="Document"/>  
</syncfusion:DocumentContainer>
```

#### C#

```
documentcontainer1.IsTDIDragDropEnabled=false;
```

### Rearrange position of document items with auto scrolling

You can easily move or rearrange document items when there are several document items by setting the EnableAutoScroll property value as `true`. Drag the required item over the overflow button (with three dots) or tab scroll buttons to autoscroll.

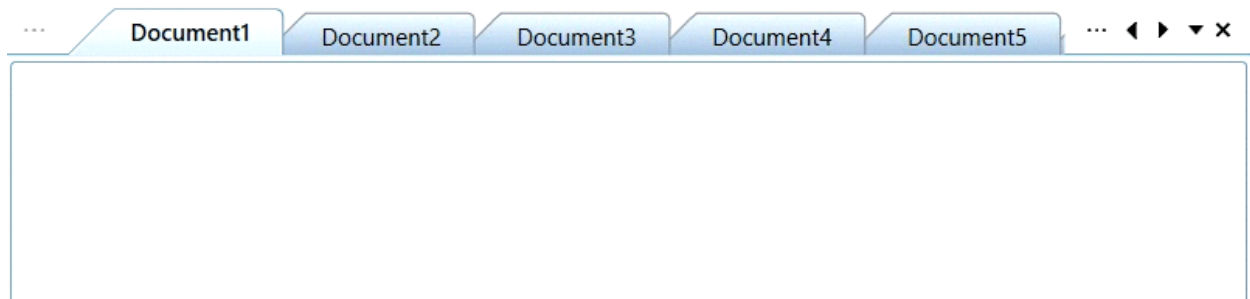
The default value of `EnableAutoScroll` property is `false`.

**XML**

```
<syncfusion:DocumentContainer x:Name="documentContainer"
EnableAutoScroll="True" Mode="TDI" >
<ContentControl x:Name="Content1"
syncfusion:DockingManager.Header="Document1" />
<ContentControl x:Name="Content2"
syncfusion:DockingManager.Header="Document2" />
<ContentControl x:Name="Content3"
syncfusion:DockingManager.Header="Document3" />
<ContentControl x:Name="Content4"
syncfusion:DockingManager.Header="Document4" />
<ContentControl x:Name="Content5"
syncfusion:DockingManager.Header="Document5" />
<ContentControl x:Name="Content6"
syncfusion:DockingManager.Header="Document6" />
<ContentControl x:Name="Content7"
syncfusion:DockingManager.Header="Document7" />
</syncfusion:DocumentContainer>
```

**C#**

```
DocumentContainer documentContainer = new DocumentContainer();
documentContainer.EnableAutoScroll = true;
documentContainer.Mode = DocumentContainerMode.TDI;
```

**TDI item's order changed notification in DocumentContainer**

You will be notified when the TDI item's order is changed by using the [DocumentTabOrderChanged](#) event. You can get the order changed TDI item with its old and new index values by using the [SourceTabItem](#), [OldIndex](#) and [NewIndex](#) properties. You can also get old and new tab group of the order changed item by using the [SourceTabGroup](#) and [TargetTabGroup](#) properties.

**Note:** The [DocumentTabOrderChanged](#) event also invoked when create a horizontal or vertical tab groups using context menu or move the tab document to previous or next tab groups.

**XML**

```
<syncfusion:DocumentContainer
DocumentTabOrderChanging="Documentcontainer1_DocumentTabOrderChanging"
Name="documentcontainer1"
Mode="TDI">
<Grid syncfusion:DockingManager.Header="Tab1"
syncfusion:DockingManager.State="Document"/>
```

```
<Grid syncfusion:DockingManager.Header="Tab2"
syncfusion:DockingManager.State="Document"/>
<Grid syncfusion:DockingManager.Header="Tab3"
syncfusion:DockingManager.State="Document"/>
</syncfusion:DocumentContainer>
```

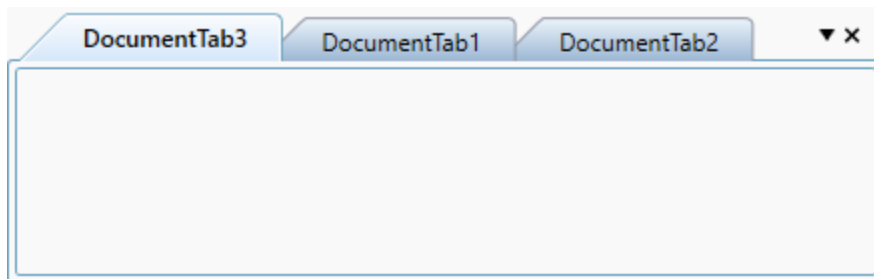
### C#

```
documentcontainer1.Mode = DocumentContainerMode.TDI;
documentcontainer1.DocumentTabOrderChanging +=
Documentcontainer1_DocumentTabOrderChanging;
```

You can handle the event as follows,

### C#

```
private void Documentcontainer1_DocumentTabOrderChanging(object sender,
Syncfusion.Windows.Tools.Controls.DocumentTabOrderChangedEventArgs e)
{
    var drag_Drop_Item = e.TargetTabGroup;
    //Get the old and new index of the SourceTabItem
    var oldIndex = e.OldIndex;
    var newIndex = e.NewIndex;
    //Get the old and new tab group of the SourceTabItem
    var sourceTabGroup = e.SourceTabGroup;
    var targetTabGroup = e.TargetTabGroup;
}
```



**Note:** [View Sample in GitHub](#)

### Restrict TDI item reordering in DocumentContainer

If you want to restrict the user to reordering the TDI items by drag and drop operation, use the `DocumentTabOrderChanging` event and set `Cancel` property value as `true`.

### XML

```
<syncfusion:DocumentContainer
DocumentTabOrderChanging="Documentcontainer1_DocumentTabOrderChanging"
Name="documentcontainer1"
Mode="TDI">
    <Grid syncfusion:DockingManager.Header="Tab1"
syncfusion:DockingManager.State="Document"/>
    <Grid syncfusion:DockingManager.Header="Tab2"
syncfusion:DockingManager.State="Document"/>
    <Grid syncfusion:DockingManager.Header="Tab3"
syncfusion:DockingManager.State="Document"/>
</syncfusion:DocumentContainer>
```

```
</syncfusion:DocumentContainer>
```

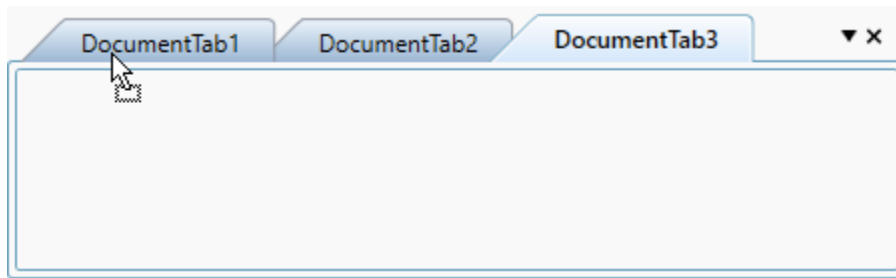
### C#

```
documentcontainer1.Mode = DocumentContainerMode.TDI;  
documentcontainer1.DocumentTabOrderChanging +=  
Documentcontainer1_DocumentTabOrderChanging;
```

You can handle the event as follows,

### C#

```
private void Documentcontainer1_DocumentTabOrderChanging(object sender,  
Syncfusion.Windows.Tools.Controls.DocumentTabOrderChangingEventArgs e)  
{  
    // Restrict the TDI item re-ordering  
    e.Cancel = true;  
}
```



**Note:** [View Sample in GitHub](#)

## Creating Tab Groups in WPF Tabbed MDI Form (DocumentContainer)

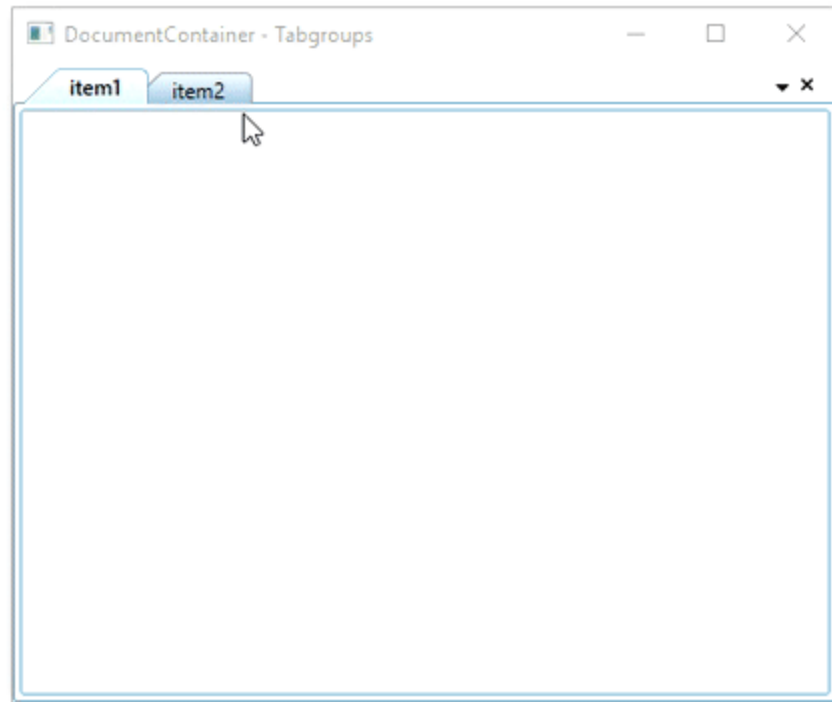
you can create tab groups for the tabitem of TDI mode [DocumentContainer](#) by dragging and also using the options in context menu items.

### Creating tab groups using context menu item

You can create a new horizontal or vertical tab groups for the TDI [DocumentContainer](#) by clicking the [New Horizontal Tab Group](#) or [New Vertical Tab Group](#) options available in the tabitem context menu.

### XML

```
<syncfusion:DocumentContainer Mode="TDI"  
x:Name="documentContainer" >  
    <ContentControl syncfusion:DocumentContainer.Header="item1"  
Name="item1" />  
    <ContentControl syncfusion:DocumentContainer.Header="item2"  
Name="item2" />  
</syncfusion:DocumentContainer>
```



---

**Note:** Tab groups created only in TDI mode of the `DocumentContainer`. You can enable it by setting the `Mode` property value as TDI.

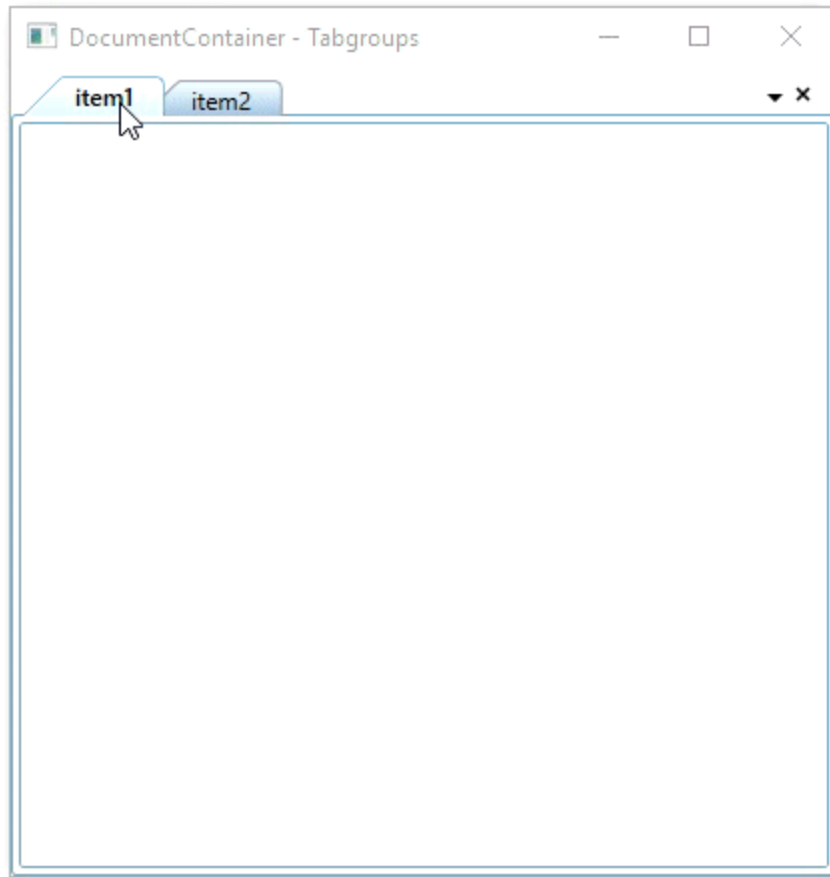
---

#### Creating tab groups using tabitem dragging

You can create new horizontal tab group for the TDI `DocumentContainer` tab item by dragging the tabitem into the Document area and click the `New Tab Group` menu item from context menu item. You can cancel this tab group creation by clicking the `Cancel` menu item from context menu item.

#### XML

```
<syncfusion:DocumentContainer Mode="TDI"
x:Name="documentContainer" >
  <ContentControl syncfusion:DocumentContainer.Header="item1"
Name="item1" />
  <ContentControl syncfusion:DocumentContainer.Header="item2"
Name="item2" />
</syncfusion:DocumentContainer>
```



Creating tab groups programmatically

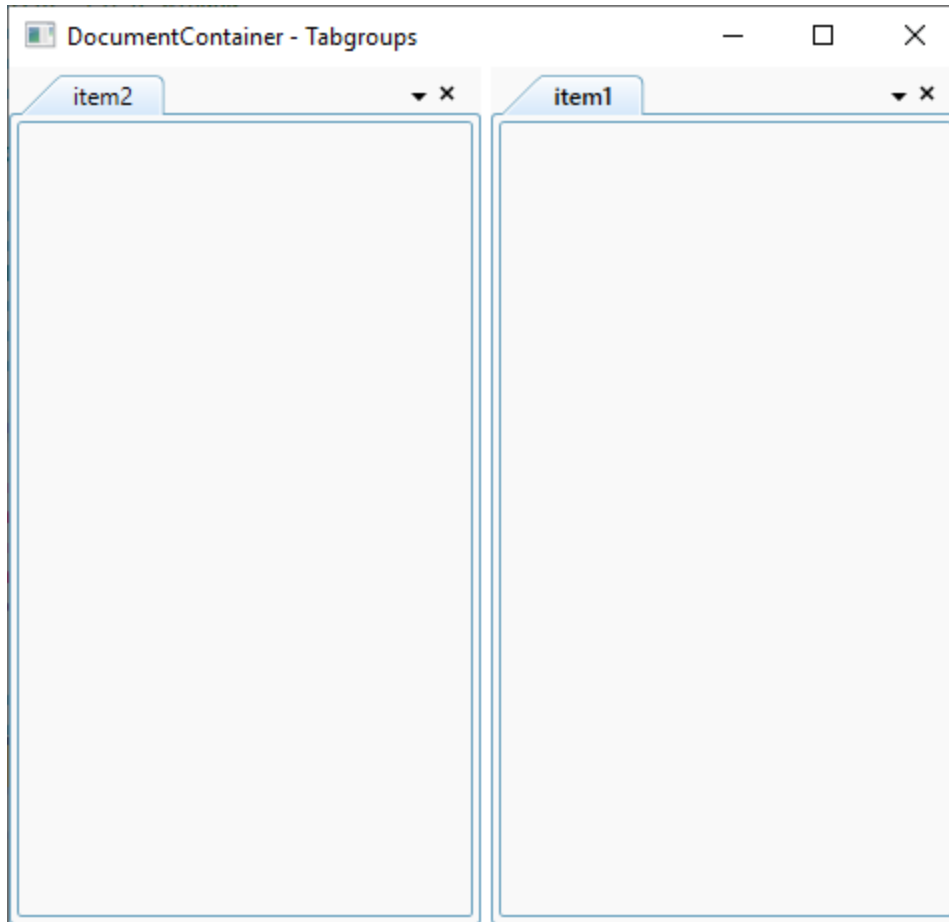
You can programmatically create a tab group on horizontally or vertically by using the [CreateHorizontalTabGroup](#) and [CreateVerticalTabGroup](#) methods.

#### **XML**

```
<syncfusion:DocumentContainer Mode="TDI"
Loaded="DocumentContainer_Loaded"
x:Name="documentContainer" >
<ContentControl syncfusion:DocumentContainer.Header="item1"
Name="item1" />
<ContentControl syncfusion:DocumentContainer.Header="item2"
Name="item2" />
</syncfusion:DocumentContainer>
```

#### **C#**

```
private void DocumentContainer_Loaded(object sender, RoutedEventArgs e) {
documentContainer.CreateVerticalTabGroup(item1 as UIElement);
}
```



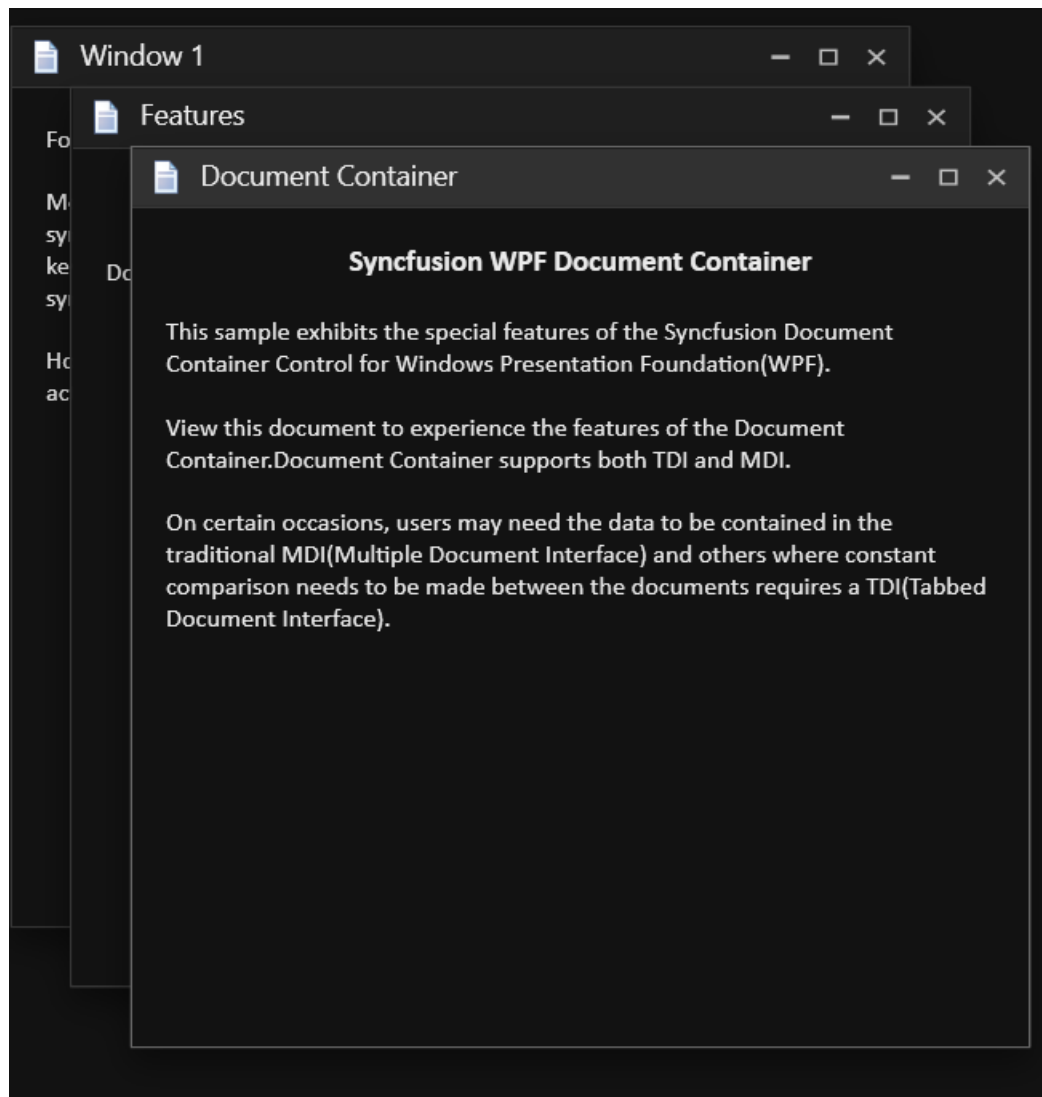
### Layout Related Features in WPF Tabbed MDI Form (DocumentContainer)

#### Theme

DocumentContainer supports various built-in themes. Refer to the below links to apply themes for the DocumentContainer,

- [Apply theme using SfSkinManager](#)
- [Create a custom theme using ThemeStudio](#)





## Pin and Unpin TabItems in WPF Tabbed MDI Form (DocumentContainer)

This section explains the pin and unpin tab items support in DocumentContainer.

### Enabling/disabling pinning behavior

The [AllowPin](#) attached property of DocumentContainer decides whether the tab item can be pinnable or not. The corresponding tab item will be pinned only if the property [AllowPin](#) is true. When the property is false, pin and unpin behavior of tab item will be disabled. The default value of the [AllowPin](#) property is false.

### XML

```
<syncfusion:DocumentContainer
Name="DocContainer"
Mode="TDI">
<syncfusion:DocumentContainer.Icon>
<ImageBrush ImageSource="document.png"/>
</syncfusion:DocumentContainer.Icon>
<!--TDI/MDI Children elements of the Document Container-->
<FlowDocumentScrollViewer
syncfusion:DocumentContainer.AllowPin="True"
syncfusion:DocumentContainer.Header="Document Container">
<FlowDocument FontFamily="Calibri" FontSize="13" TextAlignment="Left">
<Paragraph FontWeight="Bold" FontSize="15" TextAlignment="Center">
Syncfusion WPF Document Container</Paragraph>
<Paragraph>This sample exhibits the special features
of the Syncfusion Document Container Control for
Windows Presentation Foundation(WPF) .
</Paragraph>
<Paragraph>View this document to experience the features of the
Document Container.Document Container supports both TDI and MDI.
</Paragraph>
<Paragraph>On certain occasions, users may need the data to be contained in
the traditional MDI(Multiple Document Interface) and others where constant
comparison needs to be made between the documents requires a
TDI(Tabbed Document Interface).</Paragraph>
</FlowDocument>
</FlowDocumentScrollViewer>
<FlowDocumentScrollViewer
syncfusion:DocumentContainer.AllowPin="True"
syncfusion:DocumentContainer.Header="Features" >
<FlowDocument FontFamily="Calibri" FontSize="13" TextAlignment="Left" >
<Paragraph FontWeight="Bold" FontSize="15" TextAlignment="Center">Document
Container-Features</Paragraph>
<Paragraph>Document container comes with a set of features. They include
</Paragraph>
<List>
<ListItem>
<Paragraph>Provides options for both MDI and TDI container Mode
</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Various window switching styles. Ctrl+tab could be used
to easily navigate through the windows</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Skins Support</Paragraph>
</ListItem>
</List>
</FlowDocument>
</FlowDocumentScrollViewer>
```

```

</ListItem>
<ListItem>
<Paragraph>State Persistence. Document container can be used to
load, save data in IS, BIN and in XML</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Huge set of properties, methods and events for easy
customization</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Document container can be resized and moved using the keyboard.
</Paragraph>
</ListItem>
</List>
</FlowDocument>
</FlowDocumentScrollViewer >
<FlowDocumentScrollViewer
syncfusion:DocumentContainer.AllowPin="True"
syncfusion:DocumentContainer.Header="Window 1" >
<FlowDocument FontFamily="Calibri" FontSize="13" TextAlignment="Left">
<Paragraph>For ease of navigation, users can switch between the MDI and TDI
modes.
</Paragraph>
<Paragraph>Most users prefers keyboard to mouse navigation. Thus, to
satisfy all users, syncfusion Document Container is boosted by
Window Switcher(CTRL+TAB keys) for smooth, fine and flexible navigation
between interfaced windows. syncfusion ships five different modes of window
switchers.
</Paragraph>
<Paragraph>
Hold down the CTRL key and use the TAB keystroke repeatedly to
experience active switchers in effect.
</Paragraph>
</FlowDocument>
</FlowDocumentScrollViewer>
</syncfusion:DocumentContainer>

```

### Pin and Unpin tab items using PinButton

The PinButton will be visible in the tab items only when the property [ShowPin](#) is true. The default value of the property is false, so the pin button will be collapsed in the tab item.

### XML

```

<syncfusion:DocumentContainer
Name="DocContainer"
Mode="TDI">
<syncfusion:DocumentContainer.Icon>
<ImageBrush ImageSource="document.png"/>
</syncfusion:DocumentContainer.Icon>
<!--TDI/MDI Children elements of the Document Container-->
<FlowDocumentScrollViewer
syncfusion:DocumentContainer.AllowPin="True"
syncfusion:DocumentContainer.ShowPin="True"
syncfusion:DocumentContainer.Header="Document Container">
<FlowDocument FontFamily="Calibri" FontSize="13" TextAlignment="Left">
<Paragraph FontWeight="Bold" FontSize="15" TextAlignment="Center">

```

```

Syncfusion WPF Document Container</Paragraph>
<Paragraph>This sample exhibits the special features
of the Syncfusion Document Container Control for
Windows Presentation Foundation(WPF) .
</Paragraph>
<Paragraph>View this document to experience the features of the
Document Container.Document Container supports both TDI and MDI.
</Paragraph>
<Paragraph>On certain occasions, users may need the data to be contained in
the traditional MDI(Multiple Document Interface) and others where constant
comparison needs to be made between the documents requires a
TDI(Tabbed Document Interface) .</Paragraph>
</FlowDocument>
</FlowDocumentScrollViewer>
<FlowDocumentScrollViewer
syncfusion:DocumentContainer.AllowPin="False"
syncfusion:DocumentContainer.ShowPin="True"
syncfusion:DocumentContainer.Header="Features" >
<FlowDocument FontFamily="Calibri" FontSize="13" TextAlignment="Left" >
<Paragraph FontWeight="Bold" FontSize="15" TextAlignment="Center">Document
Container-Features</Paragraph>
<Paragraph>Document container comes with a set of features. They include
</Paragraph>
<List>
<ListItem>
<Paragraph>Provides options for both MDI and TDI container Mode
</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Various window switching styles. Ctrl+tab could be used
to easily navigate through the windows</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Skins Support</Paragraph>
</ListItem>
<ListItem>
<Paragraph>State Persistence. Document container can be used to
load, save data in IS, BIN and in XML</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Huge set of properties, methods and events for easy
customization</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Document container can be resized and moved using the keyboard.
</Paragraph>
</ListItem>
</List>
</FlowDocument>
</FlowDocumentScrollViewer >
<FlowDocumentScrollViewer
syncfusion:DocumentContainer.AllowPin="True"
syncfusion:DocumentContainer.Header="Window 1" >
<FlowDocument FontFamily="Calibri" FontSize="13" TextAlignment="Left">
<Paragraph>For ease of navigation, users can switch between the MDI and TDI
modes.
</Paragraph>

```

**<Paragraph>**Most users prefers keyboard to mouse navigation. Thus, to satisfy all users, syncfusion Document Container is boosted by Window Switcher(CTRL+TAB keys) for smooth, fine and flexible navigation between interfaced windows. syncfusion ships five different modes of window switchers.

**</Paragraph>**

**<Paragraph>**

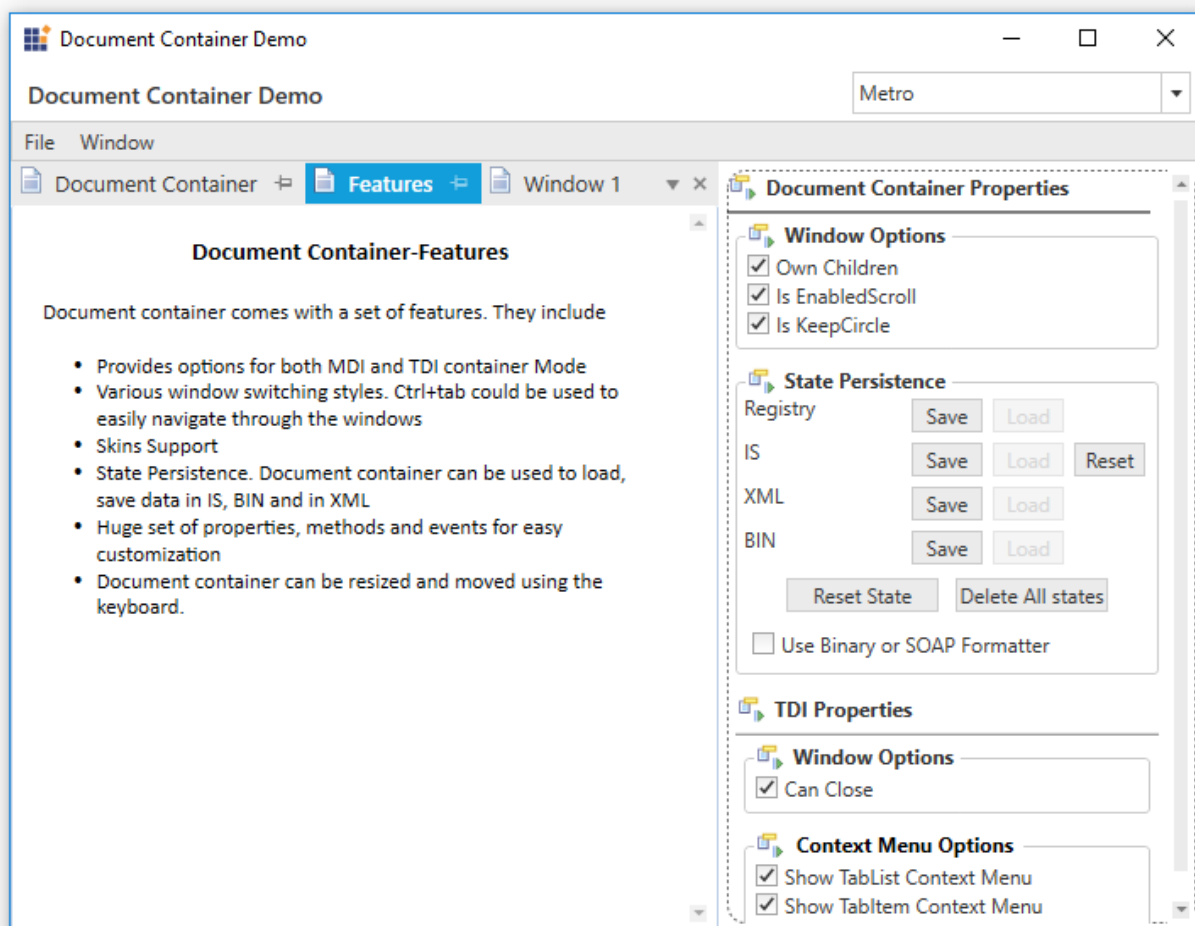
Hold down the CTRL key and use the TAB keystroke repeatedly to experience active switchers in effect.

**</Paragraph>**

**</FlowDocument>**

**</FlowDocumentScrollViewer>**

**</syncfusion:DocumentContainer>**



If the property [AllowPin](#) is true, the pin button will be enabled and visible. If the property [ShowPin](#) is true and [AllowPin](#) is false, the pin button will be displayed as disabled button.

#### Functionality of PinButton

When the pin button of the tab item is visible, the corresponding tab item can be pinned or unpinned from the Tabcontrol. When the corresponding tab item is pinned, it will be inserted at first position of the tab items collection(if the pinned tab item collection has zero count. Otherwise, the pinned tab item

will be added to the existing collection). When the tab item is unpinned, it will be removed from the pinned tab item collection and added to the first position of the unpinned tab item collection.

#### Pin and Unpin the tab items programmatically

Tab items can be pinned or unpinned from the DocumentContainer using [IsPinned](#) attached property of DocumentContainer. If the property [IsPinned](#) is set to true, the corresponding item will be added to respective index. Also, if the property [IsPinned](#) is set as false, the tab item will be removed from pinned collection and added to unpinned tab item collection. The default value of the property is False.

#### Pin and Unpin tab items through ContextMenu

The pin or unpin operations can be done through tab item's ContextMenu also. If the property [AllowPin](#) is true, and the tab item is not pinned, the "Pin Tab" option will be visible. If the tab item is pinned already, "Unpin Tab" will be visible.

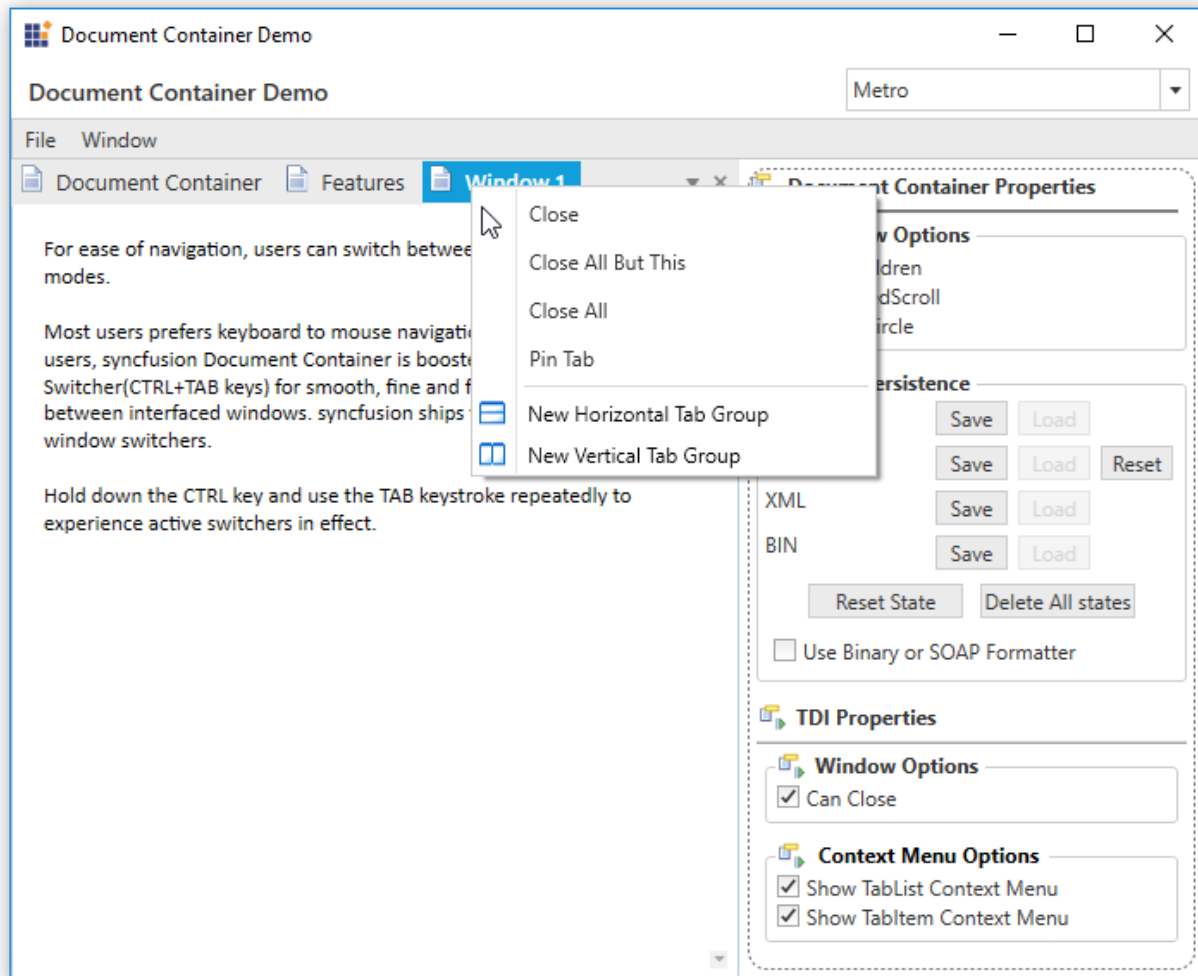
```
<syncfusion:DocumentContainer
Name="DocContainer"
Mode="TDI">
<syncfusion:DocumentContainer.Icon>
<ImageBrush ImageSource="document.png"/>
</syncfusion:DocumentContainer.Icon>
<!--TDI/MDI Children elements of the Document Container-->
<FlowDocumentScrollViewer
syncfusion:DocumentContainer.AllowPin="True"
syncfusion:DocumentContainer.Header="Document Container">
<FlowDocument FontFamily="Calibri" FontSize="13" TextAlignment="Left">
<Paragraph FontWeight="Bold" FontSize="15" TextAlignment="Center">
Syncfusion WPF Document Container</Paragraph>
<Paragraph>This sample exhibits the special features
of the Syncfusion Document Container Control for
Windows Presentation Foundation(WPF).
</Paragraph>
<Paragraph>View this document to experience the features of the
Document Container.Document Container supports both TDI and MDI.
</Paragraph>
<Paragraph>On certain occasions, users may need the data to be contained in
the traditional MDI(Multiple Document Interface) and others where constant
comparison needs to be made between the documents requires a
TDI(Tabbed Document Interface).</Paragraph>
</FlowDocument>
```

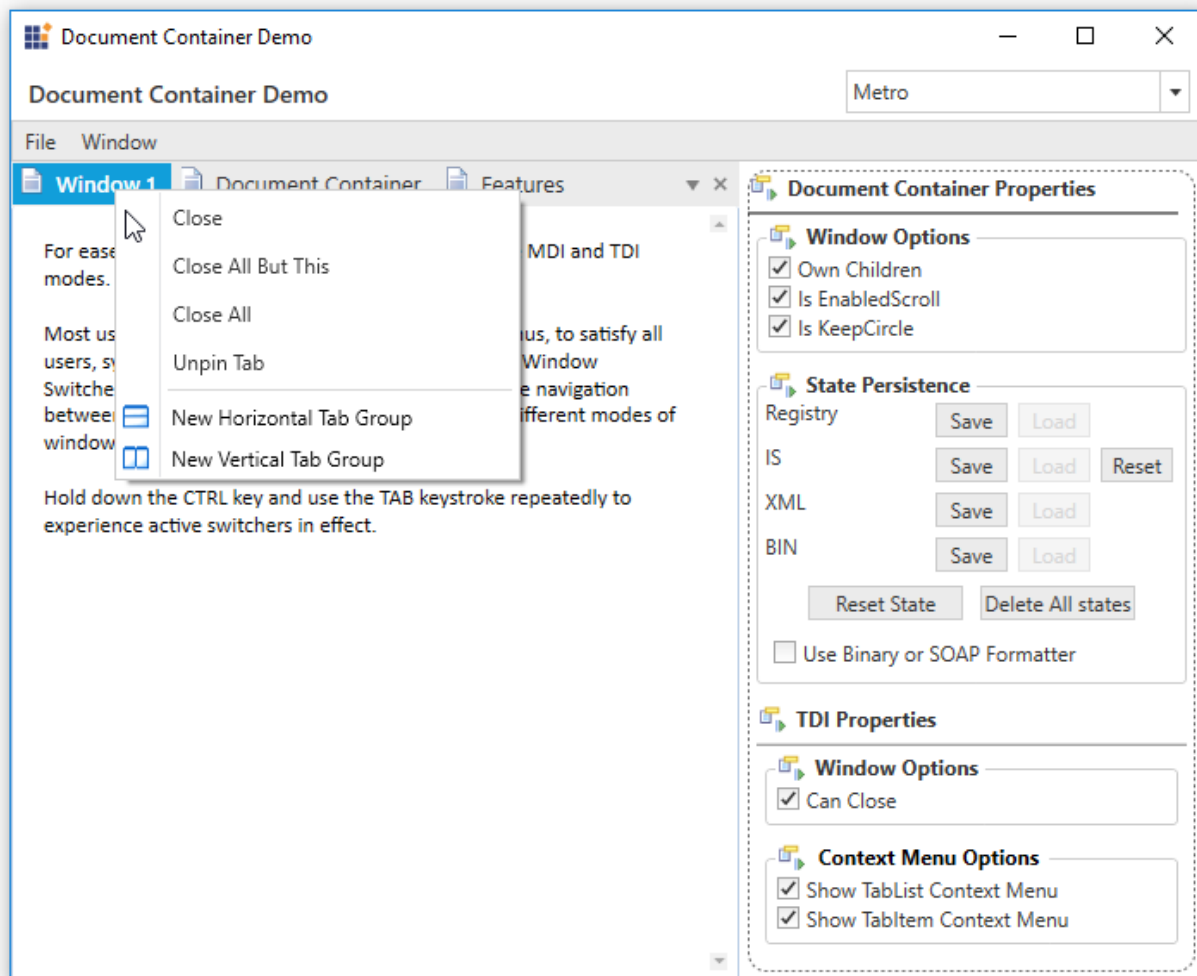
```
</FlowDocumentScrollViewer>
<FlowDocumentScrollViewer
syncfusion:DocumentContainer.AllowPin="True"
syncfusion:DocumentContainer.Header="Features" >
<FlowDocument FontFamily="Calibri" FontSize="13" TextAlignment="Left" >
<Paragraph FontWeight="Bold" FontSize="15" TextAlignment="Center">Document
Container-Features</Paragraph>
<Paragraph>Document container comes with a set of features. They include
</Paragraph>
<List>
<ListItem>
<Paragraph>Provides options for both MDI and TDI container Mode
</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Various window switching styles. Ctrl+tab could be used
to easily navigate through the windows</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Skins Support</Paragraph>
</ListItem>
<ListItem>
<Paragraph>State Persistence. Document container can be used to
load, save data in IS, BIN and in XML</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Huge set of properties, methods and events for easy
customization</Paragraph>
</ListItem>
<ListItem>
<Paragraph>Document container can be resized and moved using the keyboard.
</Paragraph>
</ListItem>
```

```
</List>
</FlowDocument>
</FlowDocumentScrollViewer >
<FlowDocumentScrollViewer
syncfusion:DocumentContainer.AllowPin="True"
syncfusion:DocumentContainer.Header="Window 1" >
<FlowDocument FontFamily="Calibri" FontSize="13" TextAlignment="Left">
<Paragraph>For ease of navigation, users can switch between the MDI and TDI modes.
</Paragraph>
<Paragraph>Most users prefers keyboard to mouse navigation. Thus, to
satisfy all users, syncfusion Document Container is boosted by
Window Switcher(CTRL+TAB keys) for smooth, fine and flexible navigation
between interfaced windows. syncfusion ships five different modes of window
switchers.
</Paragraph>
<Paragraph>
Hold down the CTRL key and use the TAB keystroke repeatedly to
experience active switchers in effect.
</Paragraph>
</FlowDocument>
</FlowDocumentScrollViewer>
</syncfusion:DocumentContainer>
```

The following images illustrates the same,







### Full Screen in DocumentContainer in WPF Tabbed MDI Form

TDIFullScreenMode is the property used to define the full-screen mode for TDI items. When a value is set for this property, the parent window will be displayed as a full screen and the tab item header will be visible only when the cursor passes over the top of the window. This property is an enum type with the following values:

- ControlMode—Makes the tab header visible on mouseover.
- WindowMode—Performs the full-screen operation and makes the tab header visible on mouseover.
- None—Does not do anything; it is the default value.

#### XML

```
<syncfusion:DocumentContainer Name="documentcontainer1" Mode="TDI"
TDIFullScreenMode="WindowMode" />
```

#### C#

```
documentcontainer1. TDIFullScreenMode=FullScreenMode.WindowMode;
```

**Note:** This feature also applicable for the TabControlExt.

### XML

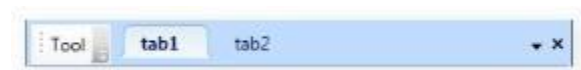
```
<syncfusion:TabControlExt Name="tabcontrol1" FullScreenMode="WindowMode"/>
```

### C#

```
tabcontrol1.FullScreenMode= FullScreenMode.WindowMode;
```

### Toolbar in DocumentContainer

A toolbar can be placed with headers in the header panel of TDI items in DocumentContainer; for this, the TDIToolBarTray property is used.



### XML

```
<syncfusion:DocumentContainer Name="documentcontainer1" Mode="TDI" >
  <syncfusion:DocumentContainer.TDIToolBarTray>
    <ToolBarTray>
      <ToolBar>
        <Button Content="Tool" />
      </ToolBar>
    </ToolBarTray>
  </syncfusion:DocumentContainer.TDIToolBarTray>
  <Grid syncfusion:DocumentContainer.Header="tab1" />
  <Grid syncfusion:DocumentContainer.Header="tab2"/>
</syncfusion:DocumentContainer>
```

### C#

```
ToolBarTray tooltray = new ToolBarTray();
ToolBar toolbar = new ToolBar();
toolbar.Items.Add(new Button{Content="Tool"});
tooltray.ToolBars.Add(toolbar);
documentcontainer1.TDIToolBarTray = tooltray;
```

This feature is also applicable to TabControlExt, as demonstrated in the following code.

### XML

```
<syncfusion:TabControlExt Name="tabcontrol">
  <syncfusion:TabControlExt.ToolBarTray>
    <ToolBarTray>
      <ToolBar>
        <Button Content="Tool" />
      </ToolBar>
    </ToolBarTray>
  </syncfusion:TabControlExt.ToolBarTray>
</syncfusion:TabControlExt>
```

### C#

```
ToolBarTray tooltray = new ToolBarTray();
```

```
ToolBar toolbar = new ToolBar();
toolbar.Items.Add(new Button{Content="Tool"});
tooltray.ToolBars.Add(toolbar);
tabcontrol.ToolBarTray = tooltray;
```

SizeToContent for MDI Window in DocumentContainer

SizeToContentInMDI is used to resize an MDI window to its child size. This is an attached property and can be applied to individual children inside DocumentContainer.

XML

```
<syncfusion:DocumentContainer Mode="MDI">
<Grid Name="grid1" syncfusion:DocumentContainer.SizeToContentInMDI="True"
Width="200" Height="200" />
</syncfusion:DocumentContainer>
```


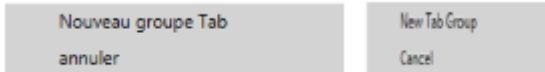
C#

```
DocumentContainer.SetSizeToContentInMDI(grid1, true);
```

Localization in WPF Tabbed MDI Form (DocumentContainer)

The following table describes how a DocumentContainer property can be localized to a specific culture. You can simply provide the string values in the resource file for a specific culture and set the culture in the application.

Property	Description
MDIRestore,MDIMove,MDIResize,MDIFloating,MDIDockable,MDIDocument,MDIMinimize,MDIMaximizeMDIClose	Sets the string for the context menu item in DocumentContainer.
<div><div><div><div>☐ Restore</div><div>☐ Move</div><div>☐ Resize</div><div>☐ Floating</div><div>☐ Dockable</div><div>☑ Document</div><div>☐ Minimize</div><div>☐ Maximize</div><div>☒ Close</div></div><div><div>☐ restaurer</div><div>☐ déménagement</div><div>☐ redimensionner</div><div>☐ flottant</div><div>☐ dockable</div><div>☑ document</div><div>☐ minimiser</div><div>☐ maximiser</div><div>☒ proche</div></div></div></div> <div>ContextMenu(en-US) ContextMenu(fr-FR)</div>	
MoveToNextTabGroup	Sets the string for MoveToNextTabGroup context menu item in DockingManager and DocumentContainer.

 <p>ges/Localizationimg4.png) <i>MoveToNextTabGroup(en-US)</i>  <i>MoveToNextTabGroup(fr-FR)</i></p>	
<p>MoveToPreviousTabGroup</p>	<p>Sets the string for MoveToPreviousTabGroup context menu item in DockingManager and DocumentContainer.</p>
 <p>ges/Localizationimg6.png) <i>MoveToPreviousTabGroup(en-US)</i>  <i>MoveToPreviousTabGroup(fr-FR)</i></p>	
<p>NewTabgroupMenuItemCancel</p>	<p>Sets the string for the Tab context menu item in DockingManager and DocumentContainer.</p>
 <p>ges/Localizationimg8.png) <i>NewTabGroup(en-US)</i>  <i>NewTabGroup(fr-FR)</i></p>	
<p>TabClose,CloseAllButThis,TabCloseAll,Floating,Document,Dockable,NewHorizontalTabGroup,NewVerticalTabGroup</p>	<p>Sets the string for the menu item in DocumentContainer and DockingManager.</p>
 <p>ges/Localizationimg10.png) <i>Menuitem(en-US)</i> <i>Menuitem(fr-FR)</i></p>	

